Rana Ejtehadian

Developing Automatic Update Notification Software – Case Study Tekla

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 11.11.2011

Thesis Supervisor:

Prof. Jukka Manner

Thesis Instructor:

M.Sc. Lim Chau Ming

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY ABSTRACT OF THE

SCHOOL OF ELECTRICAL ENGINEERING MASTER'S THESIS

Author: Rana Ejtehadian

Title: Developing Automatic Update Notification Software – Case Study Tekla

Date: 11.11.2011 Language: English Number of pages: 10+62

Department of Communications and Networking

Professorship: Networking Technology Code: S-38

Supervisor: Prof. Jukka Manner

Instructor: M.Sc. Lim Chau Ming

With the growing amount of data on the Web and the increase in frequency of new information being published, it becomes more and more critical to deliver these updates to the users in a timely manner. Having an efficient update delivery system enables companies to distribute their valuable information to many more users and get more of them to visit their web pages and latest products.

Tekla, the case study company of this thesis aims to tackle this issue by developing an automatic update notification tool to periodically pull the updates from their various web pages and offer them to the users. The updates range from recent news and events to the latest software updates and extensions.

The company's web service, Extranet, holds valuable information to the company and therefore it is protected by an authenticated login form. Therefore, any communication with the server has to go through a secure channel and this prevents the use of conventional update delivery methods.

In this study, the Tekla Update Notification Tool is specified, designed and implemented. The application provides a communication channel between the software company and individual users. When a new update is detected, the application automatically delivers the update to the user's computer and sends notifications indicating that a new update is available. The retrieval and delivery of updates are performed according to the user's preferences and choice of settings.

This work relates generally to the distribution of updates ranging from software versions to latest news and more particularly to a method of establishing a secure communication channel between a client application and a web server acting as the source of information.

Keywords: Automatic update notification, software update distribution, client-server communication

AALTO-YLIOPISTO DIPLOMITYÖN

SÄHKÖTEKNIIKAN KORKEAKOULU TIIVISTELMÄ

Tekijä: Rana Ejtehadian

Työn nimi: Automaattisten päivitysilmoitusten sovelluksen kehittäminen - Tapaustukimus Tekla

Päiväys: 11.11.2011 Julkaisukieli: Englanti Sivumäärä: 10+62

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka Koodi: S-38

Työn valvoja: Prof. Jukka Manner

Työn ohjaaja: M.Sc. Lim Chau Ming

Internetin kasvavan tietomäärän ja lisääntyvän uuden tiedon julkaisutahdin myötä, tulee yhä kriittisemmäksi toimittaa käyttäjille tieto päivityksistä ajoissa.

Tehokkaan päivitysten jakelujärjestelmän avulla yritys voi jakaa heidän arvokasta tietoa useammalle käyttäjälle jolloin enemmän niistä vierailee yrityksen verkkosivuilla.

Tämän opinnäytetyön tavoitteena on puuttua tähän asiaan kehittämällä automaattinen päivityksen ilmoitusväline, joka ajoittain lataa päivitykset yrityksen Internet sivuilta ja tarjoaa niitä käyttäjille. Päivitykset vaihtelevat tuoreista uutisista ja tapahtumista aina uusimpiin päivityksiin ja laajennuksiin.

Yhtiön ekstranet sivuilla on arvokasta tietoa yhtiöstä ja se on sen takia suojattu sisäänkirjautumissivulla. Siksi kaikki tietoliikenne palvelimeen pitää mennä turvallista kanavaa pitkin ja tämä estää käyttämästä tavallisia toimitustapoja päivitysten hakemiseen.

Tässä tutkimuksessa, Tekla Update Notification työkalu määritellään, suunnitellaan ja toteutetaan. Sovellus tarjoaa viestintäkanavan ohjelmistotalon ja yksittäisien käyttäjien välillä. Kun uusi päivitys on havaittu, sovellus toimittaa automaattisesti päivityksen käyttäjän tietokoneelle ja näyttää ilmoituksen käyttäjälle tuloksista. Päivitysten haku tehdään käyttäjän valitsemien asetuksien mukaan.

Tämä opinnäytetyö käsittää yleisesti päivitysten jakelu aina ohjelmistoversioista uusimpiin uutisiin ja tarkemmin menetelmään jolla asiakassovelluksen ja web-palvelimen välillä suojattu viestintäkanava toimii tiedonlähteenä.

Keywords: Automaattisten päivitysilmoitusten, sovellus päivitysten jakelu, verkko viestintä

To the three pillars of my life:

My wonderful parents,incredibly supportive.

My amazing sisters, Sara and Lara, endlessly caring.

My life companion, Henrik, infinitely loving.

Acknowledgements

First and foremost, I would like to express my appreciation to my supervisor, Professor Jukka Manner and instructor, Lim Chau Ming for their professional insight and valuable guidance.

This thesis was written for Tekla Corporation. I wish to thank Tekla for giving me the opportunity and resources to work on this subject.

I am grateful to all of those with whom I have had the pleasure to work during this thesis, especially Tomi Hotarinen and Sarah Korhonen for giving me the original idea for this thesis and supporting me through the process, Päivi Palomäki, Marko Myllyma and Mattias Lindgren for their assistance.

In addition, I would like to thank my team leaders Erik Fallenius and Roger Carulla Bové for their interest in the progress of my thesis and encouraging me.

I wish to thank my family and friends for their support and encouragement in my studies and during the writing of the thesis.

Otaniemi, 11 November 2011

Rana Ejtehadian

Table of Contents

Table of figures

Abbreviations and Acronyms

API	Application Programming Interface
ASP	Application Service Provider
BIM	Building Information Modeling
DLL	Dynamic Link Library
DNS	Domain Name System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
MD5	Message-Digest Algorithm
RSS	Really Simple Syndication
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
XML	Extensible Mark-up Language

Chapter 1 Introduction

There are fundamentally two ways that users can remain up to date with what is happening on the Web. There is the old way, where the users have to frequently visit special web pages to check if there are any updates available or not and then there is the new way, where the updates from different web pages are delivered to the users automatically.

While most companies invest heavily in building data-rich web sites, a few focus the same effort on distributing that data. On the other hand, as the number of data updates increases, it becomes more and more critical that these updates are delivered to the users in a timely manner. The same holds true for software updates, where it can be a risky proposition to allow time to go by before users are aware of available version updates.

Without a good update delivery system, companies miss out on opportunities to distribute their valuable information to many more users and get more of them to visit their web sites and latest products. Therefore a strong automatic update delivery system is a very competitive and efficient way of communicating with the users and attracting them to the relevant web pages. In addition, it allows companies to gain access to new data about their user base including their geographical and system information.

When an update delivery system is configured so that updates are automatically downloaded and announced, the users do not need to visit a number of web pages or remember to periodically check for new updates. This is a valuable feature for occasionally connected users. It is sufficient for them to simply connect, update, and disconnect, to keep up to date with the latest news and version updates.

Furthermore, more and more companies are distributing software to their users across the Internet. Distributing and installing software over an existing network not only bypasses the costly overhead of producing DVDs but can be made intelligent so that only the correct version of software is pushed down to the user.

These issues are addressed by this thesis and Tekla Update Notification Tool is developed to allow the company to automatically notify users of updates according to users preference set to the application.

1.1 Problem Statement

At the case study company of this thesis, Tekla Corporation, the current methods of data distribution do not fully exploit the benefits of a modern communication system. Even though the company has a very good web service, there is no compelling infrastructure that will make the user aware that a specific web page has changed or let the user know automatically that a software update is available.

The company is attempting to establish a customer online community around its Extranet, the web platform for the community, to create value for its users and owners [1]. To do so, the company needs to attract more and more users to the Extranet. One way of achieving this goal is to frequently update users about the content of various web pages and attract them to the Extranet by providing messages that can be traced back to the corresponding web pages.

On the other hand, establishing a notable online community leads to creation of more Web content such as discussion forums and increases the overall frequency of updates. Tekla today, does not have a good infrastructure to notify users of the latest updates and the need for it is becoming more noticeable in time.

In this section a number of possible solutions to address the above challenges are presented and the advantages and disadvantages of each method are discussed.

One alternative is to use a more traditional method of communication such as E-mail. This is the method that Tekla currently uses to communicate latest updates. With the increasing amount of information, this method imposes difficulty for both the data publisher and receiving user. Although for many update channels there is the possibility to use automatic E-mail updates, for many more, the publisher of data has to constantly compose and send update messages to corresponding user groups, which is both time consuming and unnecessary. On the other hand, since the update distribution is initiated by the publisher and not by the user, the flexibility to adjust the time schedule or subscription to different mailing lists is being restricted.

Another popular alternative is to make use of Atom or RSS, both from a family of web feed formats used for automatic content distribution. All web feed formats are based on Extensible Markup Language (XML), a text-based computer language used to describe and distribute structured data and documents. The feeds are designed to provide content summaries of web pages. Although there are some differences between the technology of Atom and RSS, they both essentially perform the same functionality and therefore RSS is discussed in this section to represent this method.

RSS is defined as Really Simple Syndication or Rich Site Summary. RSS feeds can be read using web-based, desktop-based or mobile-device-based software called an RSS reader or aggregator. RSS is becoming increasing popular as a free and easy way to promote a site and its content without the need to advertise or create complicated content sharing partnerships [2]. It benefits publishers by letting them syndicate content automatically and benefits readers who want to subscribe to timely updates from many sites and view them in one place.

Although RSS has been set up for some pages of the Tekla web sites, it is not widely deployed. One obstacle is that the majority of data is secured behind an authenticated web service and it cannot be accessed by many aggregators currently available at the market. Although this limits the choice of aggregators for the user, nevertheless, it is possible for

the user to subscribe to existing RSS feeds and provides a solution to some of the mentioned problems. Using a conventional feed aggregator does not enable the company to have a customized software that can be tailored to the needs of the company and its specific user group, and cannot provide a unified user experience in information delivery.

In case of version updates, a common approach is to integrate the software update notification and installation with the software itself, so that it can automatically update itself. Although this is desired in case of Tekla Structures, the current architecture of the software imposes a great implementation complexity to allow this functionality. Even if this is currently not a feasible solution for the company, it has been considered during the progress of this work.

1.2 Objectives and Proposed Solution

The aim of this study is to provide Tekla with a communication channel to notify the users of the latest updates ranging from news to software versions.

Having a customized update software will accomplish the company’s specific requirements and will provide the users with the flexibility to adapt it to their preferences by selecting the update categories, frequency of checking and temporarily or permanently disabling the application. Also it will guarantee that the company’s update delivery system is attuned with its standards and brand.

For this purpose, the Tekla Update Notification is developed in this thesis. It is a background tool on the user's computer that checks the company web pages on a regular schedule for new updates. The check is performed by querying the server for information that is new since the last check. The Update Notification Tool then displays a message to the user informing them of new updates if there are any available. This ensures that the users are always up to date with the latest content.

In order to retrieve data from authenticated web sources, the application acquires user credentials and communicates with the server using the Hypertext Transfer Protocol. This way, a secure connection is established between the client and the server. This functionality is wrapped into a DLL package and can be used by any other software that requires a secure communication channel to the Extranet.

1.3 Thesis Structure

The content of this work is organized into 7 chapters.

Chapter 2 introduces the case study environment by providing an overview of the case study company and their public and private web services. A solution for modernizing the communication of the company is provided at the end of this chapter.

Chapter 3 focuses on the technical concepts related to this work. An in-depth study of the Hypertext Transfer Protocol and the authentication mechanisms provided by this protocol are discussed.

In Chapter 4 the software requirements of the application developed in this work are discussed. The requirements are divided into two broad categories of functional and non functional requirements and each requirement is explained by using case studies.

Chapter 5 provides an overview of the application structure, behaviour, and architecture. This is followed by introducing the application's classes, the behavior of each class and the relationship between these classes.

In Chapter 6 the implementation of the application is discussed step by step explaining the design decisions whenever applicable. The snapshots of the end product are also presented in this chapter.

Finally, a conclusion of the work is explained in Chapter 7 along with suggestions for future work.

Chapter 2 Case Study Environment

In this chapter Tekla Corporation which is the case study of this thesis is introduced. The company currently communicates with its customers, area offices and resellers through Tekla's public web site and private extranet service. An analysis of the current communication methods and a solution for modernizing them is provided later in this chapter.

2.1 Tekla Corporation

Tekla Corporation is an international company established in 1966 in Finland. Tekla operates in two business areas: Building and Construction, which provides software products for the construction industry, and Infrastructure and Energy, which focuses on software products for energy distribution companies and infrastructure management. The company has a strong global presence with 15 own offices, a world-wide partner network and a customer base spread to nearly 100 countries [3]. With its headquarters located in Espoo, Finland, the company currently employs 500 people of whom almost 200 work at the area offices. Tekla offers model-based software products that support and streamline their customers' core processes in Building and Construction and Infrastructure and Energy. Tekla is driving the evolution of moving from traditional two-dimensional (2D) ways of working to information-model-based three-dimensional (3D) processes in both its target business areas [4].

2.2 Tekla Structures

Tekla Structures is Building Information Modeling (BIM) software that enables the creation and management of accurately detailed, highly constructible 3D structural models regardless of material or structural complexity. Tekla models can be used to cover the entire building process from conceptual design to fabrication, erection and construction management [4].

Tekla Structures is a single product available in different configurations and localized environments that provide specialized set of functionality to suit the segment and culture-specific needs of the construction industry.

2.3 Tekla Structures Extranet

Tekla provides a public web site consisting of 15 mirror sites worldwide and a private extranet service offered to the company's selected customers. Tekla's public web site contains information about the company's software solutions and products, financial information and careers as well as the latest news and events [1]. Figure 2.1 illustrates the content of Tekla's public web site. The news category is marked with a star to indicate that there is a need to notify the users when a new item is posted on this page.

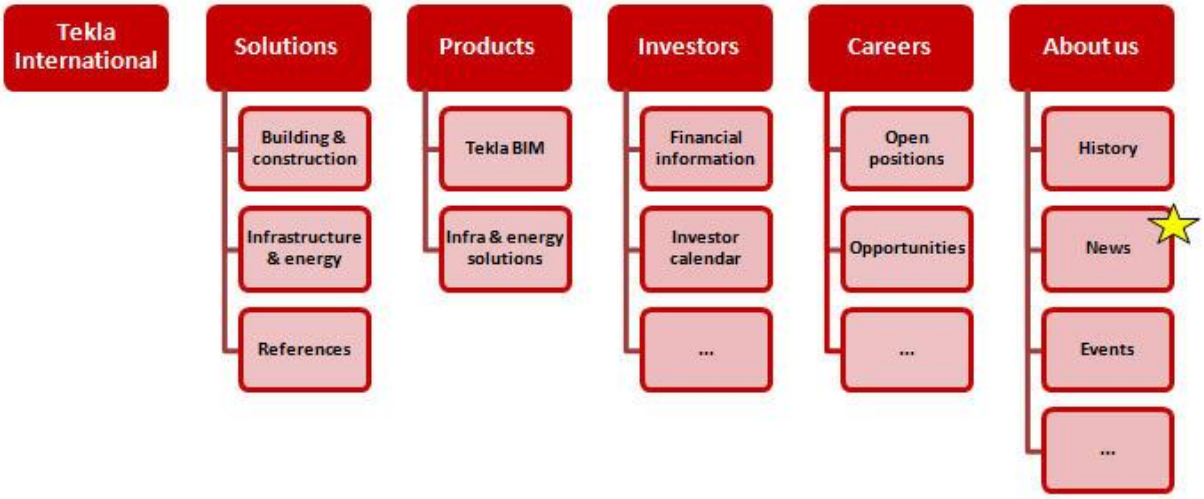

Figure 2.1 - The content of Tekla's public web site

Tekla's extranet provides the company with a platform for distributing products and sharing information. Using the extranet, customers are able to download products, use self-learning material, find out about latest news and events and share their questions and suggestions using discussion forums [5]. Figure 2.2 demonstrates the content of Tekla's extranet. The stars in the figure signify the sections where the content is updated regularly and it is beneficial to inform the users about the new content.

Part of Tekla's extranet that is targeted for Building and Construction customers is called Tekla Structures Extranet. It is referred later in this study simply as Extranet. Extranet is available in five different language versions, of those this study is limited solely to the English language.

Since Extranet hosts valuable, confidential and proprietary information, its security is of foremost importance to Tekla. Extranet is available only to Tekla employees, resellers and customers who are a member of Tekla maintenance policy. Furthermore, as it can be seen from Figure 2.2, Extranet includes an internal section that is not accessible to customers

and is mainly used for sharing material to area offices and resellers. This selective access is enabled by authentication mechanisms on a login page [6].

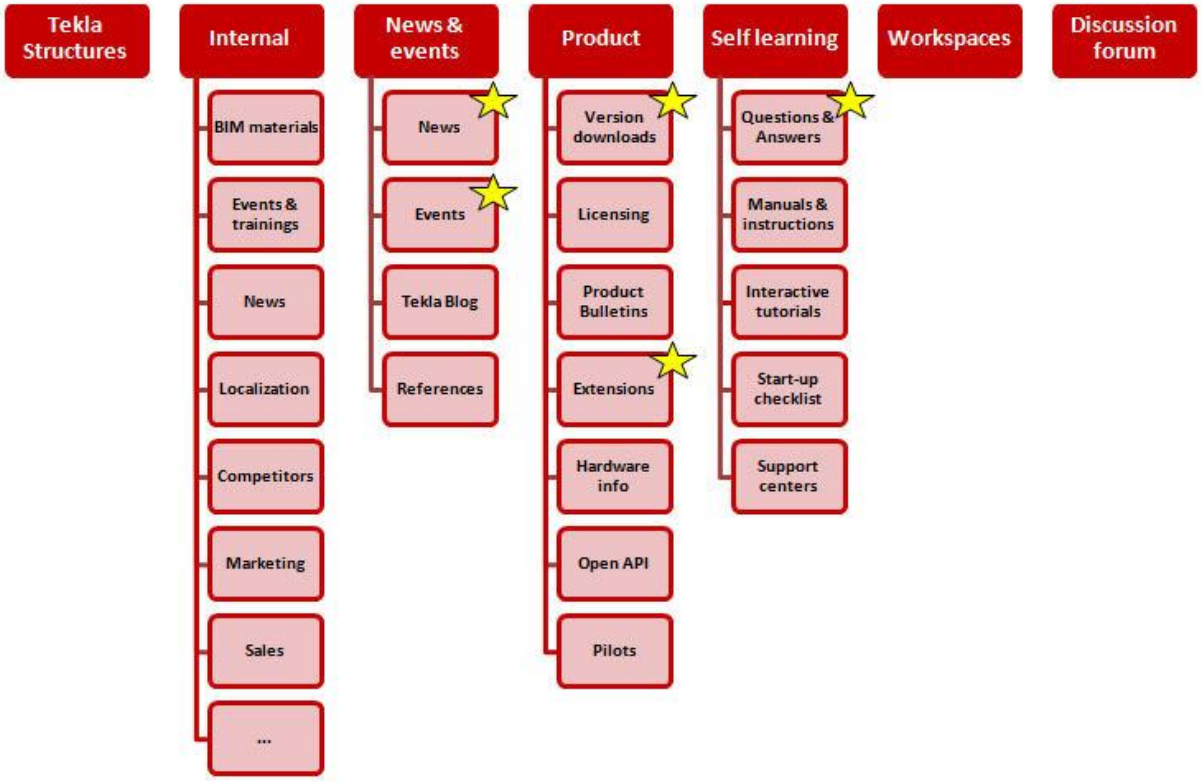

Figure 2.2 - The content of Tekla Extranet web site

2.4 Tekla Update Notification Tool

Apart from E-mail, currently Tekla uses its public web site and the Extranet to communicate and share information with its clients. As a result, the content of these sites are frequently renewed with the latest updates and it is up to the user to retrieve this information when the need arises. Tekla is aiming at enhancing this way of communication by eliminating the use of traditional methods such as E-mail and replacing them with more

modern method of automatic notification. This will provide a reliable communication channel for the company which will ensure secure and orderly delivery of information to the target audience and in turn will result in attracting more customers to Extranet. On the other hand, it provides customers with the assurance that they can be notified of updates as soon as they are available, on a timely manner and according to their own preferences.

Establishing a connection to Extranet is far more complicated than Tekla's public site and once this communication is established, the same mechanism can be used for connecting to Tekla's public web site. Therefore at this study, the Extranet is the focal point of establishing a connection and the public Web is only considered as a special case which is mentioned merely when the occasion arises.

Figure 2.3 offers a view of the system model displaying the optimum flow of data from the source of information to the server and subsequently to the end user computer. The update notification tool under study in this research is responsible for periodically checking the server for new data, automatically transferring any new data from server to user's computer and notifying the user of the latest update.

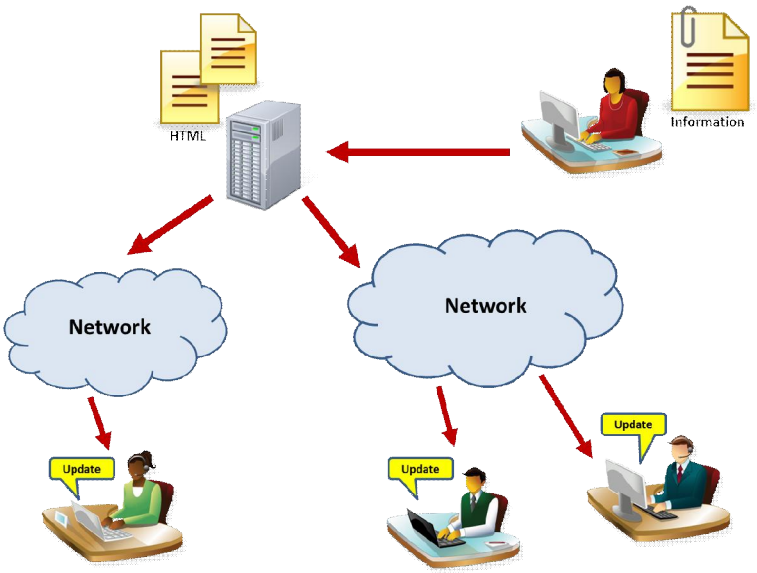

Figure 2.3 - The automatic update delivery system model

In summary, the application capable of achieving the above mentioned functionality is required to perform these main tasks and functionalities:

- User authentication
- Establishing a secure connection to the server
- Checking the server for new information on a regular basis
- Retrieving all new information and downloading them to user's computer
- Delivering update notifications to user at a specified time interval
- Providing an interface for viewing update descriptions
- Upon request, redirecting the user to relevant pages on the server for further information

Chapter 3 Review of Hypertext Transfer Protocol

In this chapter an in-depth study of the Hypertext Transfer Protocol (HTTP) is provided. This is followed by a discussion of the common authentication protocols used by web servers. Finally, it is described how to make programmatic HTTP requests to a resource that requires authentication.

3.1 Hypertext Transfer Protocol – HTTP

The Hypertext Transfer Protocol is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990 [7]. It was initially a very simple protocol used to request pages from a server. The client would connect to the server and send a GET request for a specific page and the server would respond with the contents of the requested file. This protocol was documented as HTTP/0.9 and supported no methods other than GET, no request headers and the response had to be a HTML document. In order to support new features such as additional request methods and request headers, HTTP/1.0 emerged and this was followed by the new version 1.1 including a lot of new features.

HTTP communication usually takes place over TCP/IP connections but this does not prevent HTTP from being implemented on top of any other protocol on the Internet, or on other networks. In fact, HTTP only presumes a reliable transport, therefore any protocol that provides such guarantees can be used [8].

HTTP was originally used in the clear on the Internet and the increased use of HTTP for sensitive applications required security measures where data transferred between client and server are always secured. Hypertext Transfer Protocol Secure (HTTPS) is the secure version of HTTP protocol used to establish very secure connections. This is achieved by use of ordinary HTTP over an encrypted Secure Socket Layer (SSL) or its successor Transport Layer Security (TLS) connection which were designed to provide channel-oriented security. HTTPS Uniform Resource Identifier (URI) is differentiated from HTTP URI by using the 'https' identifier in place of the 'http' protocol identifier [9].

Knowing how HTTP works is essential for understanding the operation of a web server and figuring out the details of how the server responds to individual questions. Knowing that it is possible to make programmatic HTTP requests to that server. With this aim, the following section provides a more detailed study of the HTTP protocol.

HTTP is a request/response protocol where the client sends a request to the server and the server responds to the client’s request. The client is an application program, such as web browser or a desktop application that establishes connections for the purpose of sending requests. Most HTTP communication is initiated by a client. The server is an application program, such as an application hosting a web site that accepts connections in order to service requests by sending back responses.

3.1.1 HTTP Message Structure

Requests from client to server and responses from server to client are sent in the form of HTTP messages. Messages can be Full-Request and Full-Response, which include optional header fields and an entity body or Simple-Request and Simple-Response, which do not allow the use of any header information and are limited to a single request method GET. Use of the Simple-Request format is discouraged because it prevents the server from identifying the media type of the returned entity [10]. Therefore, most HTTP messages use

the generic message format consisting of a start-line, optional header fields, an empty line indicating the end of header fields and possibly a message body.

The HTTP header fields include General-Header which has applicability for the request and response messages, Request-Header, Response-Header and Entity-Header fields. The message body of an HTTP message is used to carry the information associated with the request or response. The following section illustrates an example of a request message from a client to a server.

```
GET / HTTP/1.1
Host: www.tekla.com
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; de-de)
AppleWebKit/523.10.3 (KHTML, like Gecko) Version/3.0.4 Safari/523.10
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no-cach
```

As it can be seen above, the first line of the request message contains the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

After receiving and interpreting a request message, a server responds in the form of an HTTP response message. The following section illustrates an example of a response message.

```
HTTP/1.1 302 Found
Date: Sun, 30 Oct 2011 18:52:43 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Location:http://www.tekla.com/_layouts/tekla/error.htm?aspxerrorpath=/Pag
es/default.aspx
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 196
```

The first line of the response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase. The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user [7].

The first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. The first digit can take 5 values:

- **1xx: Informational** - Not used, but reserved for future use.
- **2xx: Success** - The action was successfully received, understood, and accepted.
- **3xx: Redirection** - Further action must be taken in order to complete the request.
- **4xx: Client Error** - The request contains bad syntax or cannot be fulfilled.
- **5xx: Server Error** - The server failed to fulfill an apparently valid request.

The list of Status-Codes and their corresponding Reason-Phrase's [7] defined for HTTP/1.1 is presented in Appendix A.

HTTP applications are not required to understand the meaning of all registered status codes although they have to recognize the class of any status code as indicated by the first digit. In this manner, they can treat any unrecognized response as being equivalent to the x00 status code of that class.

3.1.2 HTTP Method Definitions

HTTP defines a set of methods indicating the action to be performed on the identified resource. Some of the methods that are closely related to this work are outlined in this section.

3.1.2.1 GET Method

The GET method is used to retrieve information that is identified by the Request-URI. If the request message includes an If-Modified-Since header field, the semantics of the GET method changes to a "conditional GET". A conditional GET method requests that the identified resource be transferred only if it has been modified since the date given by the If-Modified-Since header. The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client [7].

3.1.2.2 HEAD Method

The HEAD method is identical to GET method except that the server must not return any Entity -Body in the response message. This method is useful for obtaining meta-information about the resource identified by the Request-URI without transferring the Entity-Body itself. The information contained in the HTTP headers in response to a HEAD request should be identical to the information sent in response to a GET request. This method is often used for testing hypertext links for validity, accessibility, and recent modification [7].

3.1.2.3 POST Method

The POST method is used to submit data to be processed by the destination server identified by the Request-URI. The entity to be submitted is enclosed in the request body. The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. If as a result of a POST request, a resource has been created on the server, a 201 response is sent back indicating successful creation of resource and referring to this new resource. On the other hand, response with Status-Code 200 or 204 might be sent back as a result of a POST request if the action performed by the POST method does not result in an URI identified resource [7].

3.1.2.4 Idempotent Methods

The choice of methods to be used by an application making HTTP requests is important since the application represents its users in their interactions and any action that the user takes might have an unexpected significance. Methods such as GET and HEAD are intended not to take an action other than retrieval and therefore they are considered to be safe. By contrast, methods such as POST are intended for actions which may cause side-effects and the application should inform the user of the fact that a possibly unsafe action is being requested.

In the HTTP specification, the idempotent property of methods is specified so that the side-effects of $N > 0$ identical requests are the same as for a single request. Methods GET and HEAD are intended to share this property while the POST method is not necessarily idempotent and therefore sending an identical POST request multiple times may cause further side-effects.

HTML specification summarizes this topic by stating that if the processing of a form is idempotent i.e. it has no lasting observable effect on the state of the world, then the form method should be GET. On the other hand, if the service associated with the processing of a form has side-effects, the method should be POST [11].

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a request, so the important distinction here is that the user did not request the side-effects and therefore cannot be held accountable for them [12].

3.1.3 HTTP Cookies

This section describes the HTTP Cookie and Set-Cookie header fields which will be referred to in Section 3.3. The HTTP cookie is a way for an origin server to send state information to a user agent, and for the user agent to return the state information to the origin server.

To initiate a session, a HTTP server includes a Set-Cookie header in a response message. Using the Set-Cookie header, the origin server can store name/value pairs and associated metadata called cookies at the user agent. The user agent returns a Cookie request header to the origin server if it chooses to continue a session. The Cookie header contains a number of cookies the user agent received in previous Set-Cookie headers. The origin server may ignore the Cookie header or use the header to determine the current state of the session. Also, the origin server may send the user agent a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all. When the user agent makes subsequent requests to the server, the user agent uses the metadata to determine whether to return the name/value pairs in the Cookie header. Although the cookie protocol appears simple, it has a number of complexities.

Servers may return a Set-Cookie response header with any response. User agents should send a Cookie request header with every request. An origin server may include multiple Set-Cookie header fields in a single response [13].

3.2 HTTP Authentication Framework

The authentication provided by HTTP is a simple challenge-response mechanism which can be used by a server to challenge a client request and by a client to provide authentication information to a server. When an unauthenticated request is received by the server, the server challenges the authorization of the client by returning a 401 (Unauthorized) response message, prompting the client for its credentials. A client that wishes to authenticate itself with the server may do so by including an authentication header field with the request. The message must include a WWW-Authenticate header field containing at least one challenge applicable to the requested resource [14].

The Authorization field value consists of credentials containing the authentication information of the client for the realm of the resource being requested. The realm directive is required for all authentication schemes that issue a challenge. The realm value, in

combination with the canonical root URI of the server being accessed, defines the protection space. The protection space determines the domain over which credentials can be automatically applied. If a prior request has been authorized, the same credentials may be reused for all other requests within that protection space for a period of time determined by the authentication scheme, parameters, and/or user preference.

These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value should be considered an opaque string which can only be compared for equality with other realms on that server. The server will service the request only if it can validate the credentials for the protection space of the request URI and there are no optional authentication parameters [15].

3.2.1 Basic Access Authentication

HTTP provides a number of authentication mechanisms which can be used by a client to provide authentication information to a server. The simplest mechanism is called Basic authentication. The Basic authentication scheme is based on the model that the client must authenticate itself with a username and a password for each realm. The credentials are encoded as a sequence of base-64 characters. The base-64 encoding does not encrypt or protect the credentials; it merely ensures that the characters sent are in a format that will not conflict with any reserved characters [16]. Since the username and password are passed over the network in an unencrypted form, the Basic authentication scheme is not considered to be a secure method of user authentication.

3.2.2 Digest Access Authentication

Like Basic access authentication, the Digest scheme is based on a simple challenge-response paradigm which verifies that both parties to a communication share a secret. Unlike Basic authentication, using Digest scheme this verification can be done without

sending the password in the clear, which is Basic's biggest weakness. The Digest scheme challenges the client using a nonce value. A valid response contains a checksum (by default, the MD5 checksum) of the username, the password, the given nonce value, the HTTP method, and the requested URI. In this way, the password is never sent in the clear [14].

3.2.3 Comparison of Digest and Basic Authentication Schemes

Although both Digest and Basic Authentication are very much on the weak end of the security strength spectrum, a comparison between the two points out the utility, even necessity, of replacing Basic by Digest.

The greatest threat to the type of transactions for which these protocols are used is network snooping. This kind of transaction might involve, for example, online access to a database whose use is restricted to paying subscribers. With Basic authentication an eavesdropper can obtain the password of the user. This not only permits him/her to access anything in the database, but, often worse, will permit access to anything else the user protects with the same password. By contrast, with Digest Authentication the eavesdropper only gets access to the transaction in question and not to the user's password. The information gained by the eavesdropper would permit a replay attack, but only with a request for the same document, and even that may be limited by the server's choice of nonce [14].

Although Digest is the clear choice of authentication mechanism in comparison with Basic scheme, by modern cryptographic standards Digest Authentication is weak. It solves some, but not all, weaknesses of Basic Authentication. Its strength may vary depending on the implementation; in particular the structure of the nonce may affect the ease of mounting a replay attack. For a large range of purposes Digest is valuable as a replacement for Basic Authentication but it is not as secure as Kerberos, and not as secure as any client-side private-key scheme [14].

3.3 Cookie-based HTTP Authentication

Authentication on the Web can be done either at the HTTP level, as discussed above, or using SSL certificates. Among other issues already listed in User Agent Authentication Forms [17], the former suffers from a poor user experience while the latter can quickly become expensive. That is why the most common authentication mechanism is based on HyperText Markup Language (HTML) forms [18] and HTTP cookies [19]. However, form-based authentication is almost always implemented with an HTTP redirect to the login form, making it impossible for non-browser applications to detect a protected resource. User Agent Authentication Forms [17] tried to overcome this with an amendment to HTML forms making them HTTP authentication aware [20].

This section describes an HTTP authentication scheme for use when credentials are validated by an out-of-band mechanism and later communicated to the server through the use of HTTP cookies. It is common practice that this mechanism is an HTML form, sending the user's credentials with the use of a HTTP POST request to a tier URL which will set a cookie in response. The 401 (Unauthorized) response determines which out-of-band mechanism should be used and how [14].

3.3.1 Cookie Authentication Scheme

The cookie authentication scheme tries to reconcile the current practice of many web sites and web development frameworks of using HTML forms and cookies to authenticate users, and the HTTP access authentication framework described in Section 3.2.

When the origin server sends a 401 (Unauthorized) response containing a WWW-Authenticate header with a cookie authentication scheme, the message body gives instructions on how to create the appropriate cookies. This is generally done by issuing another, preferably POST, request to a distinct URL. In most current web sites and web applications, the response body would be an HTML document containing a form; when the

form is submitted, the server checks the form data provided by the user and upon validation sends the appropriate Set-Cookie response header fields within a 303 response redirecting back to the protected resource. Since the user credentials are being passed through cookies, the Authorization request headers are therefore not used [14].

3.3.2 Security Considerations

As with any use of cookies, it is important to prevent unexpected cookie sharing on the client side and avoid cookie spoofing on the server side [19]. Using HTTP over TLS or other means of encrypting the conversation is generally sufficient to mitigate most threats, although some additional measures might be required to be taken [9].

Chapter 4 Software Requirements

The software requirements discussed in this chapter are discovered by this thesis study through elicitation sessions with users, stakeholders, and other experts within the organization. The official draft of the requirements specification document was approved by the software architect and the product owners at Tekla.

In this chapter the functional requirements of the application are presented. The objective is to provide a complete description of the behavior of the software by dividing it into substantially small pieces. This is followed by introducing the non-functional requirements. These are the constraints imposed on the design or implementation of the software.

During the requirements collection phase, the user-interface of the application was specified by a team of experts in usability and design of Tekla products, so that it would be apparent in design and implementation. However, it is known that in practise the user-interface often changes during design and implementation to suit the real situation better. That is why in this chapter emphasis is on specifically depicting functions and only the important parts and principles of the user-interface worth mentioning are described.

4.1 Functional Requirements

This section contains the functional requirements, sufficient to define the complete behavior of the program. The external behavior of the application is described by defining a set of use cases. Use cases are simple tools for describing the functionality of the software in a specific interaction that a user may have with the software. Use cases

included in this section are a textual description of the ways that a user could work with the application through its interface. The aim here is not to describe internal workings of the software, nor to explain how the software will be implemented. It is simply to show the steps that an intended user follows to use the software.

4.1.1 Application Launch

Since the Tekla Structures software is only supported in Windows operating systems, the Tekla Update Notification tool will be also supported in the same operating system. The software will be available for download on Extranet as a standalone application. Therefore it will be distributed only to the target user group that is Extranet account holders. Once installed, the application will act as a background tool checking for updates on a set schedule and advertise new updates. The users will be able to access the application from the system tray and open the application's main window. This window will be referred to as the Update Center throughout this work.

4.1.2 User Preferences

Once the application has been installed and launched, the users have the possibility to customize it and apply their own preferences. The options available to users fall under three categories: Subscription, Notification and General Options.

4.1.2.1 Subscription Options

This set of options enables the users to subscribe to different update channels. These channels primarily relate to sections on the Extranet where automatic update notification can be enabled. The number of channels available for subscription will be subject to alteration; nevertheless channels offered within the scope of this work are marked with stars in Figure 2.1 and Figure 2.2. Apart from the Extranet update channels; it is also

required to provide a selection of news channels available on Tekla public web sites so that the user can subscribe to any number and combination of these channels.

Once the selection is made, the user shall be notified only about the updates available on the subscribed channels. In addition, this selection will determine the category of items that will be displayed on the front page of the Update Center. Nonetheless, updates from all available channels will be retrieved from the server regardless of user's selection.

4.1.2.2 Notification Options

This set of options enables the users to apply their own personal preferences related to how they want to be notified of the latest updates. The user shall be able to choose the frequency of notification (Always, Daily or Weekly) or turn the notification off altogether (Never). In case of Always notification, the user will be informed about new updates as soon as they are available. In practice this means that the notification schedule is the same as update checking schedule. In case of Daily notification, the user can select the time of day he/she would prefer to get the updates. In case of Weekly notification, the user can select which days of week (Monday-Sunday) he/she would like to receive the updates and at what time during the selected days. The users will also be able to apply their personal sound and visual preferences by turning the notification sound on/off or adjusting the visibility of the notification message box.

4.1.2.3 General Options

This set of options relates to general working of the application. The user will be able to decide how often the application shall check for updates. As explained in Section 4.1.2.2, this interval will not directly affect when the user is notified about updates but rather relates to how recent the available updates are.

In addition, the user will have the option to enable or disable the running of the application on each system startup. If this option is disabled, the user will have to launch the application manually following each system startup.

4.1.3 Connection Establishment

A secure connection to Extranet needs to be established in order to periodically check for updates, retrieve updates when available and direct the user to relevant pages when it is requested. Generally, the access to Extranet is limited to authorized users and therefore the application needs to perform authentication to set up a secure connection between the user's computer and the web server. To carry out authentication actions, the user shall be prompted by a login dialog to provide his/her Extranet credentials. Subsequently, the application shall attempt to establish a HTTP connection to the server using these credentials. If the authentication fails using the provided credentials, the user shall be prompted to enter his/her credentials again.

Furthermore, the login dialog should provide the user with an option for remembering the credentials for future connection establishment. When this option is enabled, the application will encrypt the credentials and store them. At the current scope of the project, the credentials will be stored on a file residing on user's computer.

4.1.4 Update Retrieval and Notification

The content of this section describes the process of checking for updates, sorting which updates to advertise and finally informing the user of these updates.

Once a secure connection to Extranet is established, the application will periodically check for new updates from all existing channels. The checking period will be either determined by a schedule set by the user in subscription options, or in case of no user preferences, by a default value.

If there are no new updates available, the application shall continue checking for updates periodically. If a new update is detected on a channel, the application will download the update to the user's computer and will proceed to the next task which decides if the user should be notified of the update or not.

In order to determine whether to notify a user when an update is available or not, the updates are divided into two categories: news updates and software updates.

The news and software updates are differentiated by the way a new update is verified. In case of news updates (basically all update channels except software versions); a new item will be notified to the user if the user has subscribed to the specific channel that the item belongs to. In case of version updates, the process of verifying an update for notification is more complicated as explained below.

A distinction needs to be made between different version numbers associated with a product. The first version is the installed version. The installed version is that version that is actually available on user's machine. The second version is the update version which is the version that is currently available via the software update channel. Whenever a version update is available, the application needs to compare the version number of this update with the software versions installed on user's computer. If the update version is greater than the installed version, then there is fresh information to bring to the user's attention and the message box is displayed [21].

This will make it possible to inform the user only about the updates available for the versions of software installed on his/her computer.

To find the latest version of the software installed on a certain computer, the application has to perform a backwards search on software version information available in Registry. Figure 4.1 illustrates the Registry entries related to the case study software, Tekla Structures, and Appendix C contains a reference to these Registry entries.

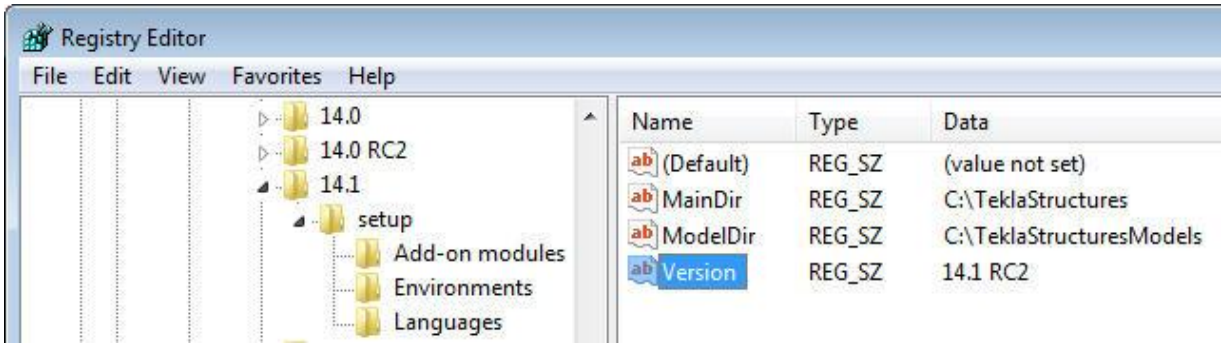

Figure 4.1 - Tekla Structures registry information

It should be noted that relying on the Registry information to determine whether a software version is installed on a machine or not might not provide accurate information in all cases. There is a known threat that uninstalling a version of software does not always remove all related information from the Registry.

4.1.5 Viewing the Updates

The user will be able to access the updates from two ways, directly opening the application's main window from the system tray or responding to a notification message by clicking on it. Both methods will direct the user to the Update Center where the new update items are available and categorized based on their channels.

The main window of the application will be a selection of tab pages where each tab page corresponds to an update channel showing update items related to that channel, even if the user has not subscribed to that channel. However, there shall be a tab page dedicated to user's subscribed channels, namely the Update Center. Different icons can be used to indicate different update channels. Therefore if the user has subscribed only to version updates channel, only the items corresponding to this category are displayed on the Update Center, even though the user can still access the latest items from all other channels by visiting their relevant tab pages.

On each page, the user will have the possibility of viewing the short description of update items along with their publication dates. Each item will be accompanied by a hyperlink pointing to its corresponding page on Extranet or Tekla public web site where the user will be directed to read the full description of that item.

4.1.6 Application Termination

When the user exits the application, the system tray icon of the application will disappear. Subsequently one of the following scenarios shall take place according to the user's selection of startup option.

Startup option enabled: In this case, the application will exit for the current Windows session. The application will run upon the next startup.

Startup option disabled: The application will be disabled and will not run on the next startup. The user has to run the application manually the next time.

4.2 Non-Functional Requirements

Non-functional requirements are global restrictions that must be considered when designing the software solution. These include performance requirements, reliability, security and design constraints. Non-functional requirements are difficult to model since they are usually stated informally and often it is very difficult to describe them as a measurable requirement. This section provides some of the main non-functional requirements of the application.

4.2.1 Modularity

The software architecture shall consist of different modules and interfaces which allow further development of the application without re-building the entire system. This

modularity allows adding new functionality to the software model later on. Furthermore, it is required to design and implement the application to accommodate its possible integration with an add-on management tool which has been under development parallel to this work [22]. Both applications require a connection to the Extranet, ability to check for new information from a specific page and downloading this information. Therefore there is need for an independent identity that can be integrated with both applications to facilitate performing the above mentioned functionality. The implementational aspects of this will be discussed in detail in Section 5.2.

4.2.2 Security

The system model shall address the security between the interacting parties. It does not need to consider issues such as key lengths, key generation, key management, certificates or security model implementation. However, the system model should provide security for the specified functionality and the following criteria will have to be addressed:

- No unauthorized instance shall gain access to the system or its data
- All system access shall be through specified interfaces
- The communication of data shall be encrypted
- All stored data shall be protected

Chapter 5 Software Specification

The design and architecture of the application is presented in this chapter. First an overview of the overall architecture is provided and after that the structure of the application is broken down into its components. Each component is explained by introducing its constructing classes, the relationship between these classes and the behavior of each class. The functions of the program are visualized using flowcharts or represented hierarchically using data flow diagrams to allow a clear understanding of the underlying processes.

5.1 Software Architecture

Figure 5.1 illustrates the overall architecture of the application.

Viewing the architecture from vertical direction reveals three distinct layers in the design of the application: the User-Interface layer, the Business layer and the Database layer. This layered architecture separates application functionality into three distinct abstractions achieving modularity and clarity in the design and implementation phases.

Handler classes are introduced to provide the necessary interface between the layers. All interactions with the User-Interface layer are handled by the DialogHandler class and all interactions with the Database layer are handled by the DatabaseHandler class.

Examining the architecture from horizontal direction shows that the application is consisting of two modules responsible for providing distinct functionality: The WebConnector module, which is responsible for establishing a secure connection with external web servers, and the UpdateManager module, which supports the rest of the functionality including scheduling and notification.

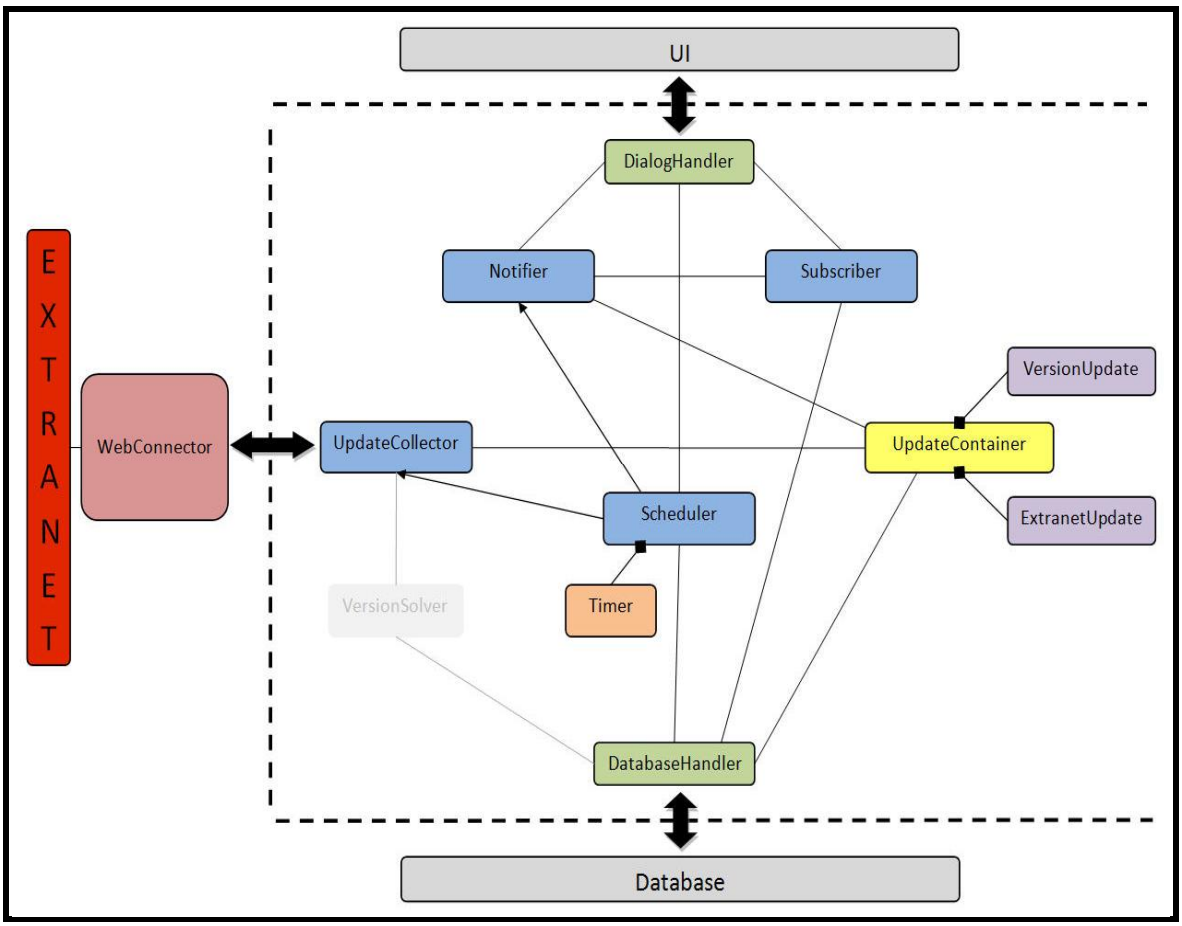

Figure 5.1 - Software Architecture

5.2 The WebConnector Module

The WebConnector is a dynamic linking library (DLL) that handles all communications to the Tekla web servers including the Extranet and Tekla public web sites. It provides an interface which can be used to establish a secure HTTP connection and request content of a specified URI.

The WebConnector module is integrated in the Tekla Update Notification tool and can be utilized by any other application that requires a secure communication channel with Tekla web servers.

Figure 5.2 shows the association between the main classes of the WebConnector module.

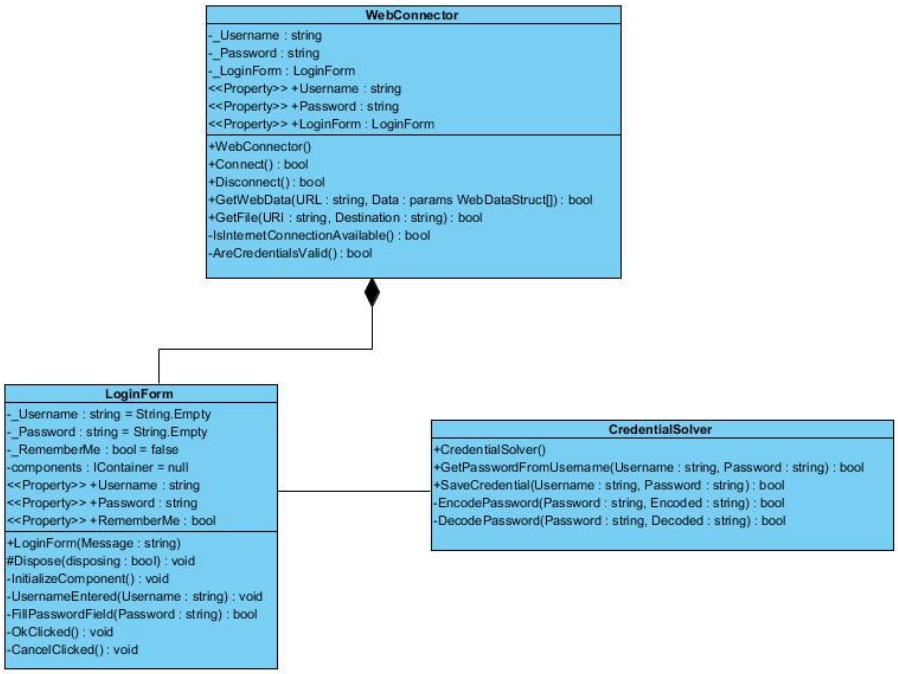

Figure 5.2 - The WebConnector class diagram

The Tekla Extranet web site is implemented using the Microsoft ASP.NET framework. In order to provide a secure channel, the client/server communications use the encrypted secure socket layer (SSL) protocol along with the HTTP protocol and authentication mechanisms. Therefore, to authenticate users, the client program will need to include the user credentials in a HTTP request and send it over a secure socket. The .NET framework provides the HttpWebRequest and HttpWebResponse classes that enable a HTTP communication with the Web. This solution is used by the WebConnector to communicate with the Extranet and will be explained in detail in Chapter 6.

The ExtranetLoginForm class presents a dialog for passing the user credentials. It also provides the functionality needed for encoding, decoding and storing passwords by calling the interface methods of the CredentialSolver class. If instructed by the user, the CredentialSolver class saves the username and encoded password in a UserAccounts.ini file under the %LOCALAPPDATA% directory. Figure 5.3 shows the process of getting user credentials depending on whether these credentials have been stored previously or not.

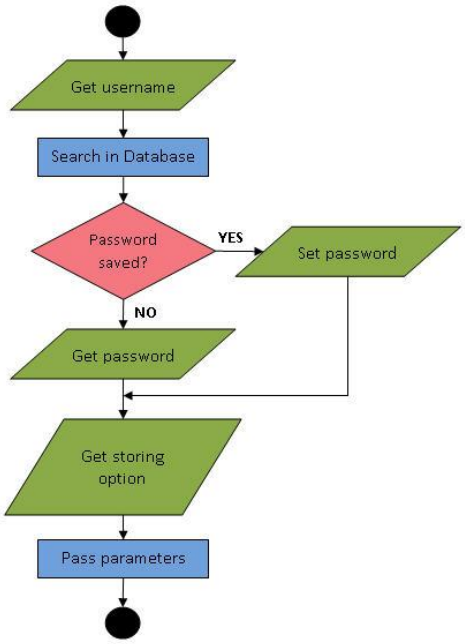

Figure 5.3 – The flow diagram of credential acquisition

Once the user credentials are passed from the ExtranetLoginForm class back to the WebConnection class, it performs basic credential validation (characters, Tekla account, etc.). If non-valid, the user will be prompted by another login dialog and if valid, it will try to establish HTTPS connection to Extranet server using the credentials.

The WebConnection class provides the interface of the WebConnector DLL. The details of this interface are presented in the following sections.

5.2.1 Interface Methods

This section introduces methods and data structures which are exposed by the WebConnector interface. These include methods to establish and tear down a connection, methods for data acquisition and the data structures used for transferring the data through the interface.

5.2.1.1 Connect

This method checks if there is an active Internet connection available before trying to connect to the Internet. This can programmatically be done at least in two ways, by pinging the server or by using the Windows Internet application programming interface (API) method InternetGetConnectedState. If an Internet connection is available, the user is prompted to provide his/her Extranet credentials by instantiating an instance of the ExtranetLoginForm class. As mentioned before, this class presents a dialog for passing the user credentials. Figure 5.4 illustrates the process of connecting to the Extranet using the Connect() method.

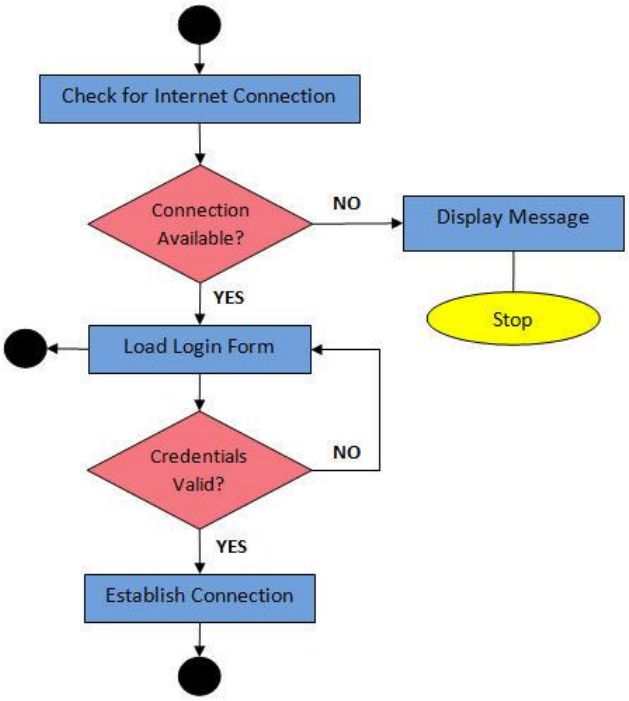

Figure 5.4 – The flow diagram of the WebConnector.Connect() method

5.2.1.2 IsConnected

This method can be called at any time to determine if an active connection to the Extranet is available. After successfully establishing an extranet connection using the Connect() method and before any more calls to the WebConnector, the state of the connection should be checked using the IsConnected() method.

5.2.1.3 GetWebDataStruct

This method can be called to retrieve the content of a web page. The URI of the requested web page is passed as a parameter to this method and if successful, the method returns the content of the page in WebDataStruct format.

Figure 5.5 shows the WebDataStruct which is the WebConnector interface data structure for passing the content of a specific XML format resource.

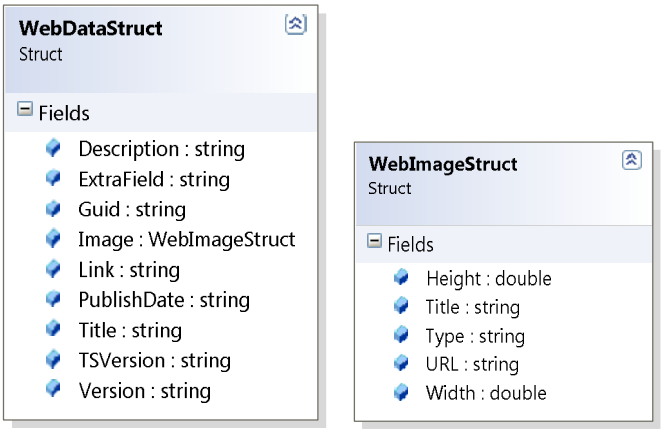

Figure 5.5 - WebConnector interface data structures

Figure 5.6 illustrates the process of retrieving the XML page specified in URI and extracting the relevant fields to fill the WebDataStruct.

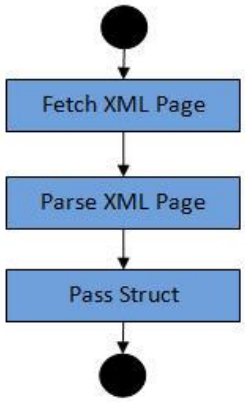

Figure 5.6 – The flow diagram of WebConnector.GetWebDataStruct() method

5.2.1.4 DownloadFile

As the name suggests, this method can be called to download a resource file identified by a given URI to a given destination location. The URI of the file and the path to the destination folder where the file needs to be saved are passed as input parameters.

5.2.1.5 Disconnect

The Disconnect() method will tear down an existing connection and gracefully release the resources used by the connection.

5.3 The UpdateManager Module

The UpdateManager module is the backbone of the software. It is responsible for co-coordinating the user-interface and database handling tasks as well as synchronizing the application's internal timers responsible for automatic update retrieval and notification. This module is a separate Windows application which communicates with WebConnector.dll through its interface.

5.3.1 Classes

The design of this application is discussed in this section by introducing its main classes and describing the relationship between these classes.

5.3.1.1 MainForm Class

This class is responsible for the functionality related to the applications main window and system tray icon menu. The main window consists of a main page called the Update Center containing the latest updates from user subscribed channels and a number of tab pages, each tab page corresponding to a specific update channel. Once each tab page is loaded,

updates corresponding to that channel are retrieved from the UpdateContainer via the DialogHandler class.

This class also provides the logic needed for changing an item’s status from Unread to Read when the user selects that item and storing these status in UpdateContainer for future references.

5.3.1.2 OptionsForm Class

This class represents the Options dialog. The selected options passed through this dialog affect the Subscriber and Scheduler classes. Once the dialog OK button is pressed, the option values are transferred via DialogHandler to these two classes. In parallel with this, the option values are saved in the database via DatabaseHandler. When the dialog is loaded next time, the default values or values set by the user are loaded from the database.

5.3.1.3 DialogHandler Class

The DialogHandler class is responsible for handling interactions between the User-Interface layer and the Business layer by providing Getter/Setter methods to transfer dialog's structures between these two layers. It also sets dialog values to the database via the DatabaseHandler class when applicable.

Figure 5.7 shows the process of loading and setting the Options dialog values through the DialogHandler class as explained in Section 5.3.1.2.

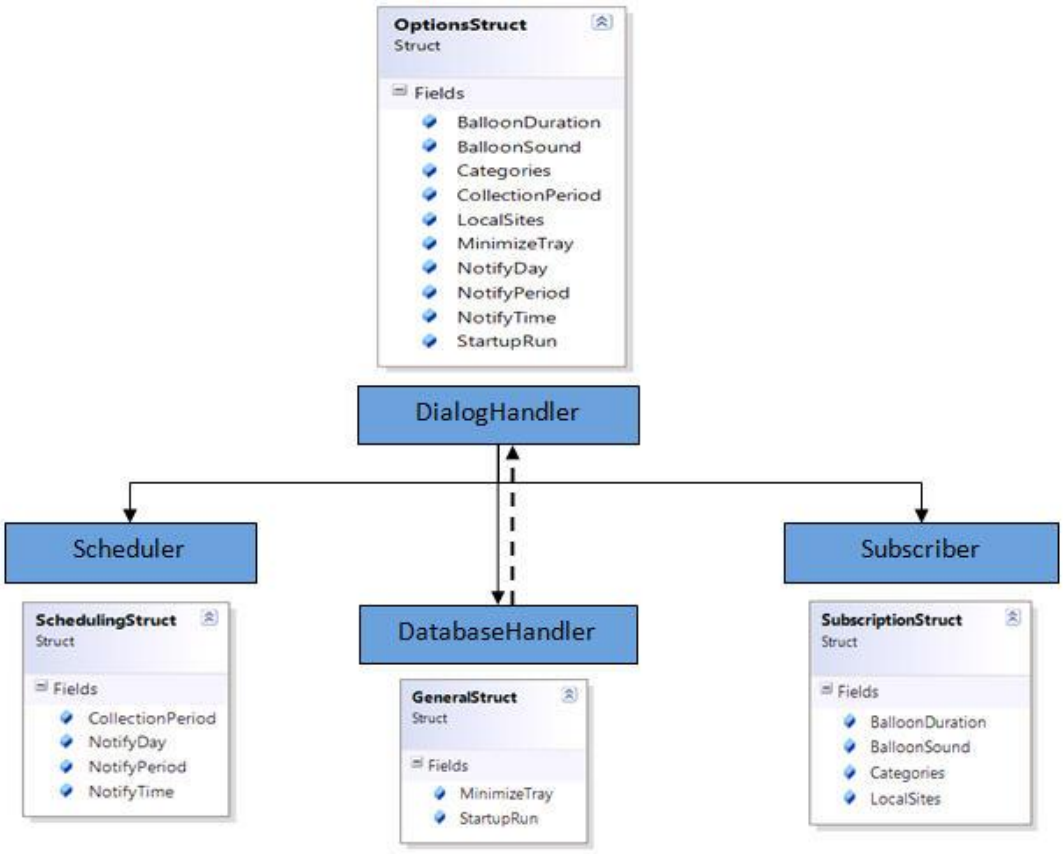

Figure 5.7 – The OptionsForm class diagram

As explained in Section 5.3.1.1, the DialogHandler also handles interactions with the MainFrom class by getting the updates to be displayed from the UpdateContainer class and also by changing the status of any item from Read to Unread in UpdateContainer class when the user clicks on an item.

5.3.1.4 Subscriber Class

The Subscriber class keeps track of user's choice of update channels and notification settings. If there is any change related to these in Options dialog, the DialogHandler class

informs the Subscriber. This class is called from the Notifier class to filter the updates fed to the user according to the selected settings.

5.3.1.5 Scheduler Class

The Scheduler is a singleton class that is responsible for controlling the program timers and co-coordinating update collection and update notification operations. In order to synchronize the program's internal processes the Scheduler class utilizes two timers: NotifyTimer for notification and CollectTimer for updates retrieval. These timers are instances of the .NET framework Timer class and are initialized using default or user set values to perform their assigned tasks. Figure 5.8 shows the data structure used by the Scheduler class to get or set the timer's interval.

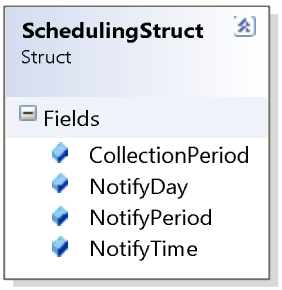

Figure 5.8 - Data structure used by the Scheduler class

Each timer raises an event once its designated time interval elapses. The Scheduler class is the listener to these events. The CollectTimer raises the OnCollectEvent to indicate that the collection time is up and therefore the Scheduler asks the UpdateCollector class to check for updates. Likewise, the NotifyTimer raises the OnNotifyEvent to indicate that the notification time is up. Consequently the Scheduler asks the Notifier class to inform the user of the latest updates. The latest collection and notification times are then stored in timestamp.ini file to be used as a time reference point.

According to the specification, there are three possible scenarios for the Scheduler to trigger notification: Always, Periodically and Never. Therefore, the Scheduler class provides a method to enable or disable the NotifyTimer according to one of these options. The timer is enabled in case of Weekly or Daily notification and disabled in case of Always and Never.

Furthermore, the Scheduler class offers the possibility to check for updates immediately and independent of the CollectTimer by raising the OnCollectEvent. This is a special case in a sense that the timers of the Scheduler class are not reset but the collection of data is triggered. From other aspects, this functionality is the same as normal update collection and notification. The mechanism is used when the user selects the Check Now option from the taskbar icon menu.

5.3.1.6 UpdateCollector Class

An instance of this class is created in OnCollectEvent() of the Scheduler class and the timestamp of the last collection is passed to it. The UpdateCollector class communicates with the WebConnector module through the latter's published interface methods. It collects the update items from all available Extranet channels as well as local Tekla public web sites. Once it has collected all the items it determines which one of them are new by comparing the update’s publish date to the timestamp of the last collection. The new items are added to the UpdateContainer class and UpdateCollector triggers a CollectionDone event to indicate a successful termination of the task. The Scheduler is the listener to this event as explained in Section 5.3.1.5.

Figure 5.9 illustrates the internal operation of this class for the update collection process.

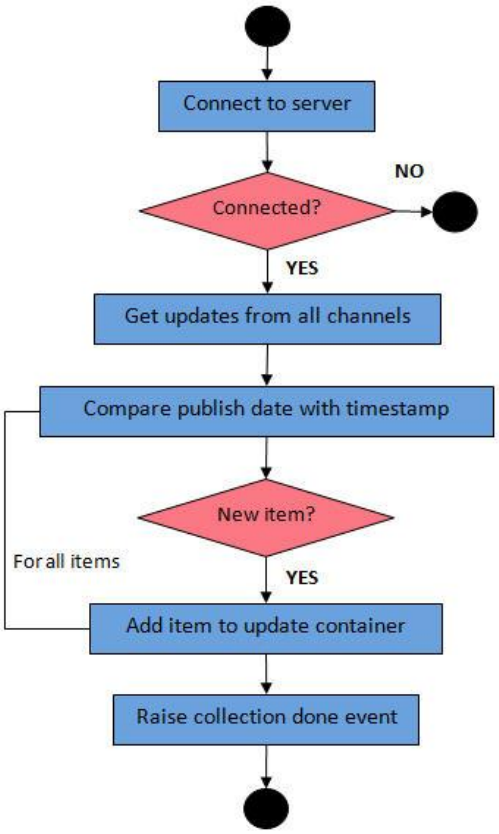

Figure 5.9 – The flow diagram of UpdateCollector.CollectUpdates() method

5.3.1.7 UpdateContainer Class

As the name suggests, this class is the container of update items including news and version updates. The UpdateCollector passes the newly retrieved updates to this class. UpdateContainer then passes these items to helper class XmlConverter to serialize the items to XML format and store them in Updates.xml file.

Figure 5.10 displays the structure of the update items handled by this class. The Status of an item can be either Read or Unread depending if the user has viewed the item or not. The

Category of an item maps to the update channel that the item belongs to, while the Office corresponds to one of the Tekla's public web site area offices.

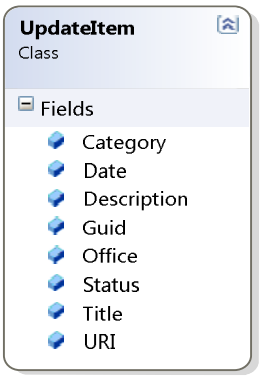

Figure 5.10 – The contents of an update item

The UpdateContainer class can be inquired to pass updates related to a certain category and with a certain status, for example Unread Version updates. This method is used by the Notifier class to retrieve the latest Unread updates belonging to user subscribed categories as explained in more details in the following section.

5.3.1.8 Notifier Class

Once the NotifyLatestUpdates() method of this class is called by the Scheduler, it gets the update items from the UpdateContainer and advertises them according to user preferences. In order to determine which updates to advertise, the Notifier class gets the subscribed channels from the Subscriber class and fetches the Unread updates belonging to those channels from the UpdateContainer. It then compares the date of these items with the last notify time stamp to inform the user of the latest updates only. Figure 5.11 depicts this process.

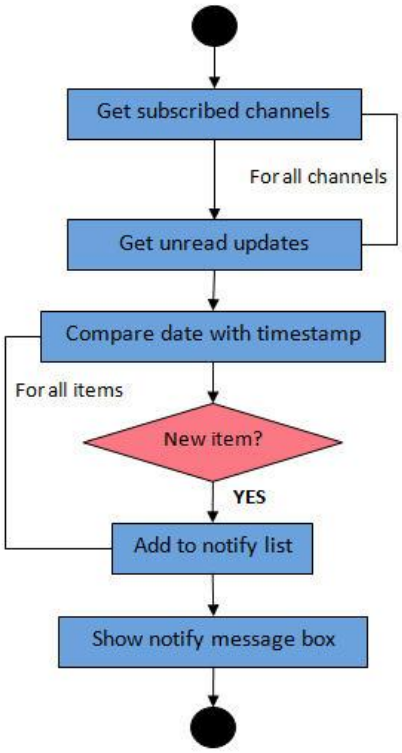

Figure 5.11 – The flow diagram of the Notifier.Notify() method

5.3.1.9 DatabseHadler Class

The DatabaseHandler class is responsible for interacting with database files and handling database operations. The application has two database locations: options.ini file for storing the Options dialog values and the timestamp.ini file for storing the latest collection and modification times. The WritePrivateProfileString and GetPrivateProfileString methods of KERNEL32.dll are used for database write-out and read-in.

Figure 5.12 shows a snapshot of the database files.

In future, there might be need for the application to have separate database files or to store some settings in the Registry instead.

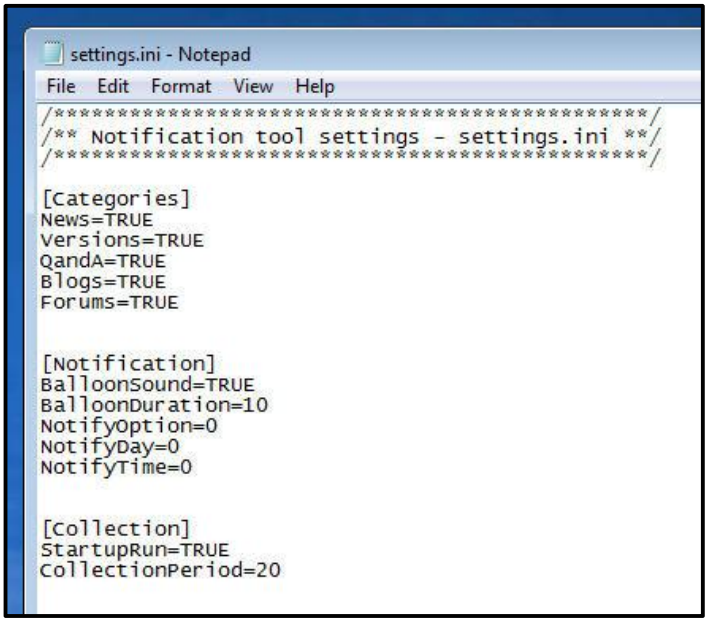

Figure 5.12 – The settings.ini database file

Chapter 6 Implementation

The application implemented in this work follows the client-service model of communication where client applications request information and server applications respond to requests from clients.

The client application makes a request by identifying the requested Internet resource and the communication protocol to use for the request and response. If necessary, the client also provides any additional data required to complete the request, such as proxy location or authentication information. Once the request is formed, the request can be sent to the server.

After the server has received the request and processed the response, it returns the response to the client application. The response includes supplemental information, such as the content type.

In this chapter, firstly the implementation of the server is examined. The aim is to identify how to successfully communicate with the server. Once this is clear, the client application implementation is explained in details including the step-by-step communication between the client and the server. Finally, the external implementation of the software, which is the finished product observed from the user perspective is presented.

6.1 Server-side Implementation

Understanding the implementation of a web server is essential to be able to communicate with that server. The Tekla Extranet service is an ASP .NET Web application. It provides user authentication through a login page. In order to programmatically authenticate users

and subsequently send requests to this server, the layout of the login page and pages containing information should be known. In this section, more detailed implementation of the Extranet is described in order to clarify decisions made in the client-side implementation. First, the structure of the Extranet login page is explained and this is followed by presenting the format in which data is stored on various Extranet pages.

6.1.1 Extranet Login Page

Viewing the source of the login page makes it possible to know the layout of the form used by this page. The form contains special elements called controls, and labels on those controls, as well as normal content and markup. In other words, the form element acts as a container and specifies:

- The layout of the form.
- The program that the form will be sent to.
- Character encoding acceptable by the server in order to handle this form.
- The HTTP method used to send the form data to the server. Possible values are GET and POST.

The following figure shows a section form that is to be processed by the Extranet server when submitted. The form will be sent to the server using the HTTP POST method. For simplicity a basic text editor is used for viewing the content.

```
<body onload="javascript:_spBodyOnLoadWrapper();">
    <form name="aspnetForm" method="post" action="MySignIn.aspx?ReturnUrl=%2fpages%2fdefault.aspx"
onsubmit="javascript:return WebForm_OnSubmit();" id="aspnetForm">
<div>
<input type="hidden" name="__SPSCEditMenu" id="__SPSCEditMenu" value="true" />
<input type="hidden" name="MSOWebPartPage_PostbackSource" id="MSOWebPartPage_PostbackSource" value="" />
<input type="hidden" name="MSOTlPn_SelectedWpId" id="MSOTlPn_SelectedWpId" value="" />
<input type="hidden" name="MSOTlPn_View" id="MSOTlPn_View" value="0" />
<input type="hidden" name="MSOTlPn_ShowSettings" id="MSOTlPn_ShowSettings" value="False" />
<input type="hidden" name="MSOGallery_SelectedLibrary" id="MSOGallery_SelectedLibrary" value="" />
<input type="hidden" name="MSOGallery_FilterString" id="MSOGallery_FilterString" value="" />
<input type="hidden" name="MSOTlPn_Button" id="MSOTlPn_Button" value="none" />
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
```

Figure 6.1 – A section of the Extranet Login form

Once the layout of the login page is known, the variables that are in input tags need to be specified in HTTP POST request. Among input controls there are some with hidden type. These controls are not rendered but their values are submitted with a form. This type of control is generally used to store information between client/server exchanges that would otherwise be lost due to the stateless nature of HTTP.

A property of ASP .NET server control is maintaining the view state, which is the gathering of all its property values. In order to preserve the submitted form values across HTTP requests, ASP.NET server controls use the view state. The view state of a page is, by default, placed in a hidden form field named __VIEWSTATE. It indicates the status of the page when submitted to the server [23].

There can be several number of elements and input fields available on a form and one way to determine what needs to be submitted to a web server is to use a debugging proxy/network sniffer like Fiddler or Firebug. In this study Firebug was used to debug the server's behaviour. Firebug is an extension that can be installed into Firefox browser. Once the cookie and content cache in Firefox is cleared, and login to the web site is established using Firefox, Firebug has all the requests sent by the browser, and the responses received. This information was used to compare the application's HTTP requests to those sent by a browser.

Using the above method, it was discovered that the cookie based authentication method is used by the Extranet login page and therefore, the user credentials need to be stored in a cookie container and sent to the server along with other required input fields through a HTTP POST message. Given that the correct credentials are provided by the user, the authentication process will succeed and from this point on, it is possible to programmatically browse to any Extranet page and retrieve the desired information.

6.1.2 Extranet Update Data Structure

After successful authentication through the login page, further HTTP GET requests can be sent to the server in order to receive the data content of various pages.

It is worth mentioning that the data structure of news items on Tekla public web sites also follow the above mentioned format.

6.2 Client-side Implementation

In this section, the implementation of the application is described in details. The aim is to provide an insight into the programmatic implementation of the client-side component according to the requirements and design specifications presented in previous chapters. The implementation of the application was carried out using thr Microsoft's Visual Studio and the .NET Framework. At the end of this chapter, the snapshots of the actual application are presented.

6.2.1 Internal Implementation

One of the functionalities of the application is to act as a client-side component and make HTTP requests to the server. This requires various features to be implemented including:

- Handling HTTP authentication
- Making requests using HTTP HEAD, GET and POST methods
- Handling redirection
- Handling cookies

The .NET Framework provides specific classes to provide the three pieces of information required to access Internet resources through a request/response model: the Uri class, which contains the URI of the Internet resources, the HttpWebRequest class, which

contains a request for the resource; and the HttpWebResponse class, which provides a container for the incoming response [24].

The HttpWebRequest class provides a rich set of features for making a HTTP request. Using this class, it is possible to perform very simple HTTP requests, or handle more complex scenarios by configuring the class properties.

The .NET Framework also provides the WebClient class, designed to simplify the HTTP request process [16]. Both of these classes can be found in the System.Net namespace. In this study, the HTTPWebRequest class has been used for making programmatic HTTP requests for no reason other than that it provides advanced properties to enable:

- If-Modified-Since support, which enables the HTTP request to check if the content has been changed since the last request and only download the complete content if it has been changed.
- A timeout value, which raises an exception if the request does not return within a specified number of milliseconds.
- Asynchronous HTTP request, which enables to start the HTTP request on a separate thread and receive notification when the request completes.

Once the appropriate .NET class for making HTTP requests has been selected, it is possible to connect and communicate to the web server.

The .NET Framework uses a Uniform Resource Identifier (URI) to identify the requested Internet resource and communication protocol. The URI consists of at least three, and possibly four fragments: the scheme identifier, which identifies the communication protocol for the request and response; the server identifier, which consists of either a Domain Name System (DNS) host name or a TCP address that uniquely identifies the server on the Internet; the path identifier, which locates the requested information on the server; and an optional query string, which passes information from the client to the server.

The first step is to authenticate the user on the web server through the login page. The user credentials are requested from the user the first time the user launches the application. Figure 6.1 shows the dialog which prompts the user for his/her credentials.

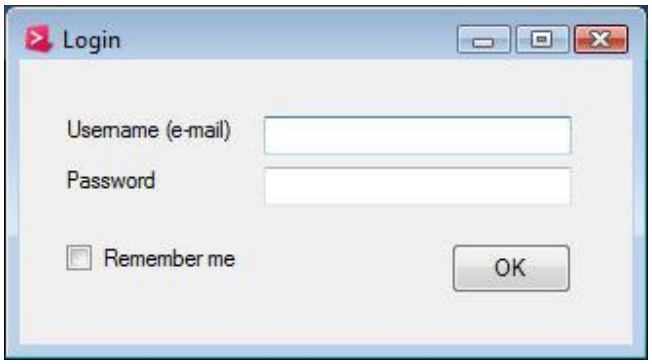

Figure 6.2 – The application's Login dialog

As soon as the credentials are inputted to the program, it automatically begins to retrieve and display the update information to the user. To do so, the following steps are taken:

1. Send HTTP GET request to the Extranet initial page. Since communication with any Extranet page requires forms authentication, this results in response with status code 302, Found, which is the most common way of performing a redirection. The response contains URI to the Extranet login page.
2. Send request to login page. This is done to collect the required collect VIEWSTATES and cookies.
3. Parse the response from step 2 and create response entity containing username/password to be used in the next post request to login page.
4. Make a HTTP POST request to the login page to post the form data. As explained in Section 6.1 it is possible to look at the HTML source of the login page to figure out the field values to POST. If successful, this should return a 302 with Set-Cookie and location header. Most form based authentication mechanisms use a HTTP cookie to

keep the user logged in. This can be done by examining the Set-Cookie header and storing the cookie value for subsequent GETs/POSTs to the web site.

5. Send request to the location pointed to in the last response which is the original page we requested in step 1. As explained in step 4, it is essential to supply the cookie value with any subsequent requests during all session. This way, the same session is used during the whole communication.

So far, the usage of HTTP GET and POST requests have been explained, however, HTTP HEAD request has also been used in implementation to increase the efficiency of data communication. HTTP HEAD request offers useful results, especially where speed and bandwidth conservation are at stake. It can become extremely useful in following cases:

1. Checking if a web page is accessible.
2. Checking if the content length has changed.
3. Checking the last modified date/time value.

The HTTP HEAD request is used in this work for the first above mentioned advantage, namely to determine if a page is available before making any other requests to that page. Based on this information, it is then possible to make a bandwidth conserving decision whether to issue a subsequent GET request to a web page or not [25].

The second property mentioned above, the Content-Length, can be compared with a database stored value for the URL to identify if the content of a certain page has changed. The Last-Modified date can be compared with a previously stored value as well, to determine if the content of a page is up of date. This is similar to the If-Modified-Since property of HTTP GET request and is used in the implementation of this work.

6.2.2 External Functionality

Although the user interface design of the application has been open to modifications for the duration of this work and possibly later on for integration with other applications, the

snapshots presented in this section show the final design by the author with an attempt to deliver a standalone tool which can be taken into use independently.

Tekla Update Notification Tool is a desktop based application. Once installed, it uses the system tray to display an icon that gives easy access to its features.

Right clicking on the icon opens up a popup menu, enabling the following functionalities depicted in Figure 6.2:

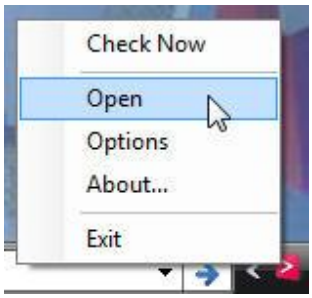

Figure 6.3 – The application's menu

By selecting the Options, the user will be directed to the application windows illustrated in Figure 6.3 where he/she can apply his/her own preferences. Each window corresponds to a certain category of options. The three available categories are Subscription, Notification and General options. The complete description of these options is presented in Chapter 4.

The user can modify and apply his/her preferences when initially loading the program or at a later time, in which case, the program runs with default options.

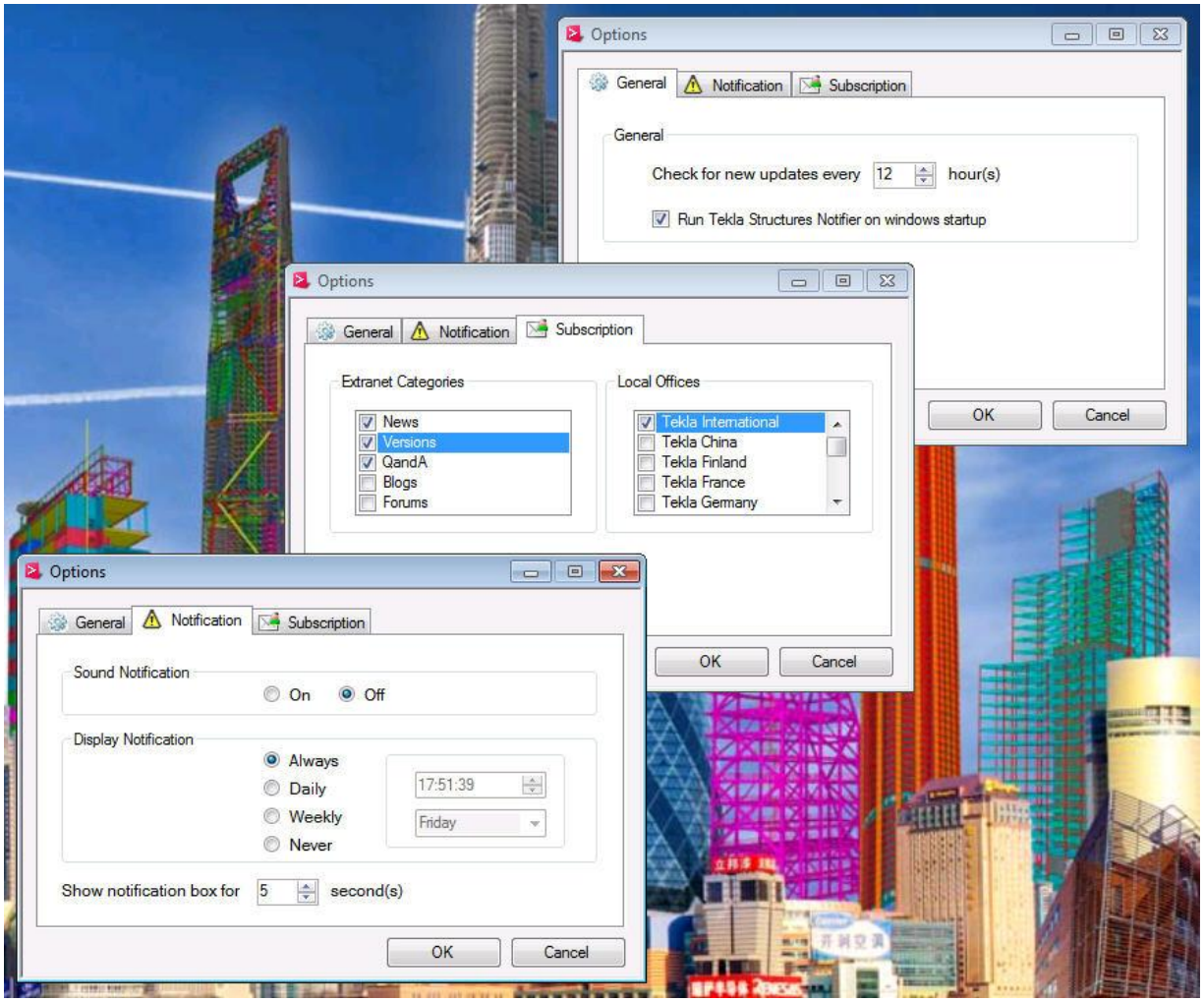

Figure 6.4 - Subscription, Notification and General Options

Once the application has been launched, it begins to create a secure communication channel with the Extranet, by prompting the user to enter his/her password. Figure 6.1 shows the corresponding Login dialog. The internal functionality of the application at this point is explained in Chapter 5.

Once a secure connection to Extranet is established, the application will periodically check for new updates. Whenever a new update is available the application stores it on the user's computer in an XML format document.

Depending on the values applied in Options dialog, the user will be notified about the available updates in his/her subscribed categories on intervals specified. The notification is done by a pop-up balloon message and a beep sound. The content of the balloon is the number of new items available and their categories. Figure 6.4 shows an example of an update notification message.

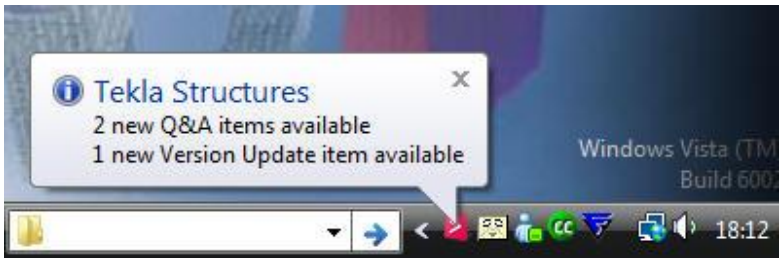

Figure 6.5 – An example of a notification message

6.2.3 Display of Errors

A complete record of all transaction information is by default stored in a log file located at:

%TeklaUpdateTool%\TeklaUpdate.log

If an error occurs, the user can utilize the information that is included in this log file to troubleshoot the problem or additionally report the issue.

Chapter 7 Conclusions and Future Work

In this chapter the summary of the work done, conclusions and some future work ideas are presented.

This Master's thesis has dealt with the software implementation of Tekla Update Notification tool. The aim of the project was to replace the traditional method of communications, in which the user needs to frequently visit the source of information to be aware of new updates, with the more modern method of periodic pull, in which the new updates are automatically and periodically retrieved from the source of information and stored on user's computer and the user is then notified about these updates based on his/her preferences.

Automatic update retrieval and notification enables the user to keep up to date with his/her desired source of information on a safe and timely manner. Furthermore, it enables the company to distribute news and software updates to its clients regularly and attract them to their web site with no extra cost or effort.

From technical point of view, the application is designed with clear architecture to support modularity and efficiency. Security has been a major consideration throughout the design and implementation, to protect both the user and the source of information.

From usability point of view, the application is designed to provide users with flexibility to apply their preferences from the update retrieval and notification schedule to options related to notification display and sound.

The end result is an efficient and flexible application which provides a positive step towards a more advanced communication method.

The future development suggestions presented in this section are based on ideas that were proposed at the duration of this work but were not implemented in the framework of this thesis.

Providing an offline mode of operation is one of the key features that enable the deployment of this application at the presence of network constraints. If the client's network administrator does not allow access to the Internet, there needs to be a method of notifying the user of new updates without Internet access. The author suggests introducing a common location on network where the latest updates are stored. In this case, the network administrator retrieves the updates periodically using the application's normal mode of operation and stores the latest updates at the common network location. The client's application should work in an offline mode of operation in which a pointer pointing to the common location fetches the update items stored at that location.

It is very useful for a software vendor company such as Tekla to collect data from their users. The easiest and most accurate way to gather information is through an automated system [26]. For this purpose, the Tekla update notification tool can be developed further to perform the metrics gathering tasks. The client-side data which can be collected include the version of Tekla Structures software installed on client's computer, along with client's IP and MAC addresses. This information needs to be collected periodically and stored on a server. User consent is required for the application to collect this kind of data from the client's computer, which can be gained during the installation process.

In summary, based on the results achieved, it can be concluded that the application created by this thesis work has successfully met the original requirements. Nevertheless, there is room for future improvements and possible integration of this application with other Tekla products.

Bibliography

[1] N. Korppi-Tommola, "Establishing a basis for a company online community: A case study", MSC thesis, Faculty of Information and Natural Sciences, Aalto University, Espoo, Finland, 2010.

[2] RSS Specifications, [Online]. Available: http://www.rss-specifications.com/rss-specifications.htm (cited Nov. 6, 2011).

[3] Tekla, "Facts and Figures", [Online]. Available: http://www.tekla.com/international/about-us/facts-and-figures/Pages/Default.aspx (cited Mar. 6, 2011).

[4] Tekla, "BIM software for building and construction", [Online]. Available: http://www.tekla.com/international/solutions/building-construction/Pages/Default.aspx (cited Nov. 6, 2011).

[5] Tekla, "Service, support and maintenance", [Online]. Available: http://tekla.com/international/products/tekla-structures/services-support/Pages/Default.aspx (cited Nov. 6, 2011).

[6] Extranet, "Login Page", [Online]. Available: https://extranet.tekla.com/user/Pages/MySignIn.aspx?ReturnUrl=%2fpages%2fdefault.aspx (cited Nov. 6, 2011).

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk Nielsen, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", IETF Request for Comments: 2616, Jun. 1999.

[8] Gary C. Kessler, "An overview of TCP/IP protocols and the Internet", [Online]. Available: http://www.garykessler.net/library/tcpip.html (cited Nov. 6, 2011).

[9] E. Rescorla, "HTTP over TLS", IETF Request for Comments: 2818, May 2000

[10] T. Berners-Lee, R. Fielding, H. Frystyk, Hypertext Transfer Protocol -- HTTP/1.0", IETF Request for Comments: 1945, May 1996.

[11] T. Berners-Lee, D. Connolly, " Hypertext Markup Language - 2.0", Sep. 1995, [Online]. Available: http://www.w3.org/MarkUp/html-spec/html-spec_toc.html (cited Nov. 6, 2011).

[12] J. Korpela. "Methods GET and POST in HTML forms - what's the difference?", IT and communication, [Online]. (cited Nov. 6, 2011).

[13] D. Kristol, L. Montulli, "HTTP State Management Mechanism", IETF Request for Comments: 2109, Feb. 1997.

[14] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", IETF Request for Comments: 2617, Jun. 1999.

[15] R. Fielding, J.Gettys, J. Mogul, H. Frystyk, T. Berners-Lee "Hypertext Transfer Protocol -- HTTP/1.1", IETF Request for Comments: 2068, Jan. 1997.

[16] S. Mitchell, Making Authenticated HTTP Requests from an ASP.NET Page, [Online]. Available: http://www.4guysfromrolla.com/articles/102605-1.aspx (cited Nov. 6, 2011).

[17] S. Lawrence, P. Leach, "User Agent Authentication Forms", W3C Note, Feb 1999.

[18] D. Raggett, A. Jacobs, I. L. Hors., "HTML 4.01 Specification", W3C Recommendation, Dec. 1999.

[19] D. Kristol and L. Montulli, "HTTP State Management Mechanism", IETF Request for Comments: 2965, Oct. 2000.

[20] A. Barth, "HTTP State Management Mechanism", IETF Request for Comments: 6265, Apr. 2011.

[21] S. Parthesarathy, R. Fink, S. L. Flynn, R. Sun, "Software update notification", United States, Microsoft Corporation, Patent number: 6353926, Mar. 2002.

[22] M. Myllymaa, "Implementing an Add-On Management Tool for Building Information Modeling Software", MSC thesis, Faculty of Information and Natural Sciences, Aalto University, Espoo, Finland, 2010.

[23] ASP .NET Tutorial, "ViewState", [Online]. Available: http://asp.net-tutorials.com/state/viewstate (cited Nov. 6, 2011).

[24] "How to use HttpWebRequest and HttpWebResponse in .NET", [Online]. Available: http://www.codeproject.com/KB/IP/httpwebrequest_response.aspx (cited Nov. 6, 2011).

[25] P. Bromberg, "The Lowly HTTP HEAD Request", [Online]. Available: http://www.eggheadcafe.com/tutorials/asp-net/2c13cafc-be1c-4dd8-9129-f82f59991517/the-lowly-http-head-request.aspx (cited Nov. 6, 2011).

[26] A. J. Perlis, "Software metrics: an analysis and evaluation", The MIT Press, Cambridge, Massachusetts, 1981. ISBN: 0262160838, 9780262160834.

Appendix A - The HTTP 1.1 status codes

The individual values of the numeric status codes defined for HTTP/1.1, and an example set of corresponding Reason-Phrase's, are presented below:

"100" : Continue
"101" : Switching Protocols
"200" : OK
"201" : Created
"202" : Accepted
"203" : Non-Authoritative Information
"204" : No Content
"205" : Reset Content
"206" : Partial Content
"300" : Multiple Choices
"301" : Moved Permanently
"302" : Found
"303" : See Other
"304" : Not Modified
"305" : Use Proxy
"307" : Temporary Redirect
"400" : Bad Request
"401" : Unauthorized
"402" : Payment Required
"403" : Forbidden
"404" : Not Found
"405" : Method Not Allowed
"406" : Not Acceptable

Appendix B - Tekla Structures registry information

The version information of Tekla Structures are stored in registry in the following order:

On 32 bit machines under:

HKEY_LOCAL_MACHINE\SOFTWARE\Tekla\Structures\%version %

On 64 bit machines under:

HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Tekla\Structures\%version%

Versions before and including 14.1

- Key: ..\setup
- Main version value: Version
- Service release value: ServicePack
- Progress release value: Update

Versions 15.0 and 16.0

- Key: ..\setup, ServiceRelease OR ProgressRelease
- Main version value: Version
- Service release value: ServiceRelease
- Progress release value: ProgressRelease

Version 16.1 and later

- Key: ..\setup
- Main version value: NA
- Service release value: NA
- Progress release value: NA