Network investigation using SNMP

Christian Hilmersson

May 2, 2006
Master's Thesis in Computing Science, 20 credits
Supervisor at CS-UmU: Mikael Rännar
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

The result of this master thesis project is an application used for remote investigation of devices connected to a network. It can for example be used to list the softwares that are installed on different computers in the network or for measuring how much free memory there is in a router and much more. The solution is not limited to investigate only computers but can also investigate other network connected devices such as switches, printers and telephone switchboards. The solution is based on the Simple Network Management Protocol (SNMP) which is a standard protocol for remotely managing network connected devices. This master thesis report reveals how the solution was developed and it also gives a good general knowledge of SNMP.

Nätverksinventering med SNMP

Sammanfattning

Resultatet av detta examensarbete är en applikation som kan användas till fjärrinventering av enheter anslutna till ett nätverk. Man kan till exempel använda applikationen för att ta reda på vilka program som är installerade på en viss dator i nätverket eller för att mäta hur mycket ledigt minne det för tillfället finns i en router och mycket mer. Lösningen är ej begränsad till att inventera endast datorer, även annan utrustning som är kopplad till ett nätverk kan inventeras så som t.ex. switchar, skrivare och telefonväxlar. Lösningen är baserad på Simple Network Management Protocol (SNMP) som är ett standardprotokoll för att fjärrhantera drift av nätverksanslutna enheter. Denna rapport avslöjar hur lösningen utvecklades och den ger även en god allmän kunskap om SNMP.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The thesis work was done at a company, hereafter called the company, that is performing different kinds of consulting services and application adaptations. They are also developing their own softwares. One of those softwares, hereafter called the inventory software, was subject to the improvements described in this master thesis project.

The inventory software is used to keep track on resources in a network. It uses a database to store information about the computers that are connected to the investigated network. The data is collected by a client application that runs on the investigated machines, it is assembled into xml files and imported to the database. In this manner one is able to keep track of almost anything concerning the computer resources that are connected in a network. All of the data can then be presented and printed as reports in the administrator application.

Though the support for investigating computers in a network was good in the inventory software before this thesis project was carried out, the application had no way to tell something about other entities connected to the network, such as routers, switches, UPSs, network printers etc. It is nevertheless interesting to know something about those entities as well.

This master thesis project is addressing the possibility to use the Simple Network Management Protocol (SNMP) to fetch information about those entities described above and more.

An interesting thing that will be discussed is how well SNMP supports dynamic configurations in terms of providing tools for making complex queries in a simple manner.

Other things that will be considered are what limitations SNMP sets and also some possibilities for improvements.

Chapter 2

Problem Description

This chapter explains in detail what this master thesis project is about. The intention is to describe the need of implementing the SNMP supported investigation tool described in this master thesis. This chapter also provides information about why the application was built in a certain way.

2.1 Problem Statement

Implement a solution for using SNMP to accomplish investigation of SNMP manageable devices in a network. Find a way to sieve through the fetched information so that it can be used in the inventory software.

2.2 Goals

1. Find all SNMP manageable devices in a network

   The application shall be able to find all SNMP manageable devices connected to the investigated network.

2. Parse MIB file(s)

   The application shall be able to parse standard MIB files, see section 2.4.3 for more info about MIB files, and also be able perform some kind of error detection.

3. Provide visual representation of MIB file(s)

   The application shall be able to provide a visual representation of the parsed MIB files, most probably as some form of tree.

4. Fetch values

   The application shall be able to fetch values from SNMP manageable devices in the network. It shall be possible to fetch standard values as well as specific MIB values.

5. Display values

   The application shall be able to display the values collected from the SNMP manageable devices in some form.

6. Adapt fetched data for inventory software import

   The application shall be able to adapt the fetched data in such way that it can be imported into the inventory software's database.

7. Integrate the SNMP application in the inventory software manager

   The SNMP administration shall be integrated with the inventory software manager's user interface.

2.3 Purposes

In the dawn of the inventory software's history there existed functionality to fetch information about computers running a Microsoft Windows operating system and not much more. Through the years of customer reviews and creative development the product has evolved to be even bigger and even more useful in many aspects. Today the inventory software system can fetch data from computers running various operating systems, such as Linux, Microsoft Windows, and Macintosh OSX. It will in a soon future be able to track usage of applications which will give the user an opportunity to handle software licenses in a better way than before. A help desk module is also under development. Altogether the inventory software gives a good overview of the network's computer resources, in terms of everything one can imagine could be fetched from a network connected computer. Nevertheless it cannot say anything about other devices connected to the network, it may be printers, routers, switches, phone switchboards and many other things. A computer is inventoried by executing a software on the specified computer which in turn delivers the requested data to another application that is importing the information into the database. A switch or a printer may not have that possibility to run applications and even if it had it would be a whole lot of different programs to write. This is where SNMP comes in handy. Since SNMP is a standard protocol, only one application is needed. This application then uses SNMP to communicate with the desired devices and fetch the needed data. So instead of having many applications running on the clients and communicating with the database, one has only one application communicating with many clients and the database, see figures 2.1 and 2.2.

2.4 Methods

This section describes in which way the solution was developed in terms of programming environment and tools used. Further this section also describes what benefits one gets from using an SNMP based method for fetching data.

2.4.1 Development environment

As stated in section 2.2, goal number 7 says that the administration of the SNMP application shall be integrated with the inventory software manager's user interface. The inventory software system is all written in Borland Delphi/Kylix and to get advantage of already written code and also to keep consistency in the system the SNMP solution must also be implemented in Delphi. Borland Delphi comes with many bundled components for various uses. One component suite that is included is the Indy project's Indy.Sockets components which handles Internet protocols such as SMTP, POP3, NNTP, HTTP and

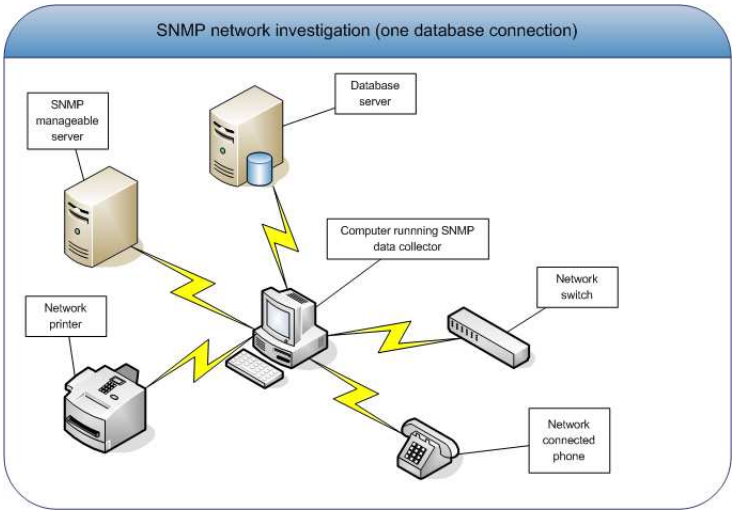

Figure 2.1: Network investigation in the inventory software using SNMP

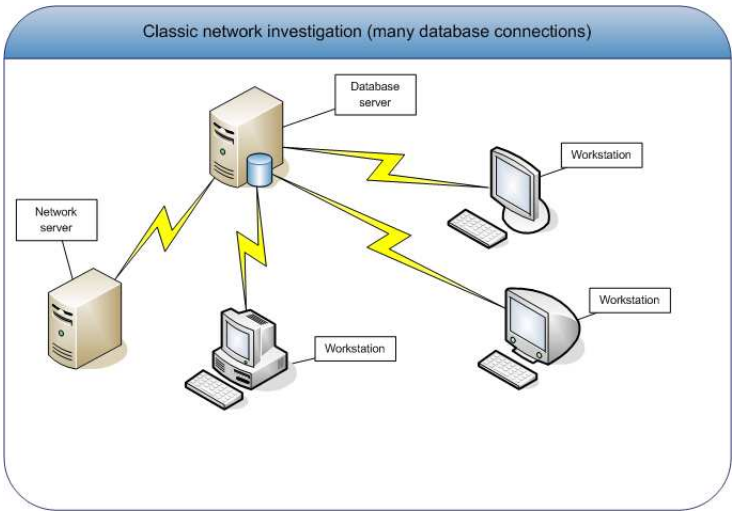

Figure 2.2: Classic network investigation in the inventory software

also SNMP. Indy's SNMP component will be used as a basis for the SNMP functionality, though it will be rewritten to fit in the application. SNMP is normally operating over the UDP protocol and to gain better control over what's happening at the UDP layer a new UDP sockets component will be written and used instead of Indy's original one. The new UDP component will operate by using calls defined in the Windows Socket 2 (WinSock2) API. To write a parser for MIB files, see section 2.4.3, an implementation of Lex/Yacc will be used together with a GUI (Graphical User Interface) front end called Glyd.

2.4.2 Possibilities with SNMP

SNMP can be used for many things, but at the bottom line it has three important functions. SNMP can be used to fetch values, the SNMP GET operation, from an SNMP manageable device, it can be used to store values, the SNMP SET operation, and it can also tell a device to send an asynchronous notification message on a specific event, the latter is called a TRAP message. Because the intended application will be used to fetch data only the first of the functions described above is important here, the SNMP GET operation.

2.4.3 Parsing MIB files

Most SNMP manageable devices follows a standard called MIB-II which is the second version of the Management Information Base described in RFC 1212 [11]. This standard is describing which values that can be fetched. These kinds of descriptions is usually stored in a file called a MIB-file. There also exists other standard MIB-files such as the Host Resources MIB described in RFC 2790 [13]. The Host Resources MIB describes SNMP manageable objects contained in a host system, such as a computer with Microsoft Windows or Novell NetWare etc. Because one cannot know anything about how devices will be designed in the future it would be impossible to cover all possible managed objects in standard MIB files. For that reason the manufacturers can also construct own MIB files which holds descriptions to support their own products. To make the application flexible it must thus be able to load and parse such files in order to give the user a presentation of the possible values to fetch. The parser will be done using TP Lex/Yacc, a Lex/Yacc implementation for Turbo and Borland Pascal, including Delphi [10].

2.5 Related Work

On the market there exists a lot of SNMP manager softwares with a wide diversity of purposes. Most of the available softwares are aimed at professional users using SNMP for maintenance and monitoring. Some softwares that use SNMP for various purposes have been tested but intentionally the peeks on softwares similar to the one developed was kept to a minimum. The solution presented in this thesis is intended to be original and easy to use for both the skilled and the not so skilled user. To be able to work in a free and open minded way no thorough investigation of related work were done prior to the implementation. There exists at least one software that somewhat resembles the one developed in this thesis work. The name of the software is TrackBird SNMP [16]. Another software that have been of good help when implementing the MIB browser, for testing and output comparisons, is Mibble MIB Parser. Mibble MIB Parser is an SNMP

MIB parser library developed in Java which includes tools for graphical representation of a MIB file [2].

Chapter 3

More about SNMP

This chapter is intended for the reader that wants a deeper look at SNMP. It will describe how and when SNMP was created and also how the protocol has evolved over time. Some things are said about which technologies that are SNMP's current contestants and what the future may hold. The internal building blocks of SNMP and how they are functioning will also be explained in terms of transport protocols, agents and managers, message formats, operations and more.

3.1 SNMP history

SNMP has been around for quite a good while now and the fact that it is still not replaced as the standard for network management vouches for its robustness. The protocol now appears in three different versions simply referred to as SNMP Version 1, SNMP Version 2 and SNMP Version 3 where SNMP Version 1 is the common standard version of SNMP. SNMP was first defined in 1988 in RFC-1067 [4], then the development went on with a re-release as RFC-1098 [5] in 1989 and again in 1990 the document was re-released as RFC-1157 [6] which is the current definition of SNMP Version 1. More information about the different SNMP versions is available in the section RFCs and SNMP Versions at page 3 in the book Essential SNMP [12].

SNMP was developed with an earlier developed protocol as a basis. The earlier protocol, SGMP (Simple Gateway Management Protocol) described in RFC-1028 [7], was developed in the early Internet days to manage variables stored in gateways. The main difference between the two protocols is that SNMP can be used to manage a lot more things than just gateways. In fact all network connected devices that are able to run a software responding to SNMP calls can be managed by SNMP. A managed device in SNMP does not even have to be a physical device, it may also be a function embedded in a network connected device, such as a web or database server.

SNMP was from the beginning not meant to stay as long as it really has done. It was rather developed as something to have in the meantime while other, more complex protocols were developed and introduced. As seen at [15] the solution that was said to be replacing SNMP, further known as CMOT, were based on the CMIS (Common Management Information Services)/CMIP (Common Management Information Protocol) technologies, but it was never accepted in such way as SNMP.

3.2 SNMP today

SNMP is still today the biggest protocol used for network management and monitoring. The reason for the success of SNMP may lie in its relatively easy and understandable nature when dealing with simple tasks. As a protocol with only very few operations and also few limits SNMP is hard to beat even today. Another thing that is worth taking into consideration when thinking about the success of a protocol such as SNMP is how well it is supported by hardware manufacturers. The fact that SNMP already is implemented in many devices also makes the task to knock it down from the scene harder.

3.3 Competitors

As pinpointed in the previous section SNMP is, although it is old, even today hard to beat. It has had its contestants such as the CMOT solution based on CMIS and CMIP as described in section 3.1 but has always survived as the leading protocol for network management.

Some things that SNMP lacks which would make the life easier for network technicians dealing with remote management is found in the following list. Hopefully the future standards will address at least some of these issues.

- It should be standard for the agent, see section 3.5.2 for more information about agents, to be able to report which MIBs that it is implementing.
- Maybe an SNMP manageable device should be able to send a message representing the MIBs that it implements.
- It would be good to have the agents sending results in other formats, such as XML, if wanted.
- Some way to make complex queries simpler.

One main contestant that has been seen on the Internet community for some time now is the newly developed WS-Management which stands for Web Services for Management. WS-Management is intended to challenge SNMP and take over the role as the leading solution for network management. Behind the new technology lies amongst others big companies such as Microsoft, Novell, Sun, Intel, AMD and Dell. With support from those giants on the computer market it should mean that WS-Management may become a successful alternative by time, though it is not widely used yet. See [3] to read more about WS-Management.

SNMP may not be the ultimate protocol for network management but many people think that it is going to be with us for quite some time even ahead of today.

3.4 Uses for SNMP

Traditionally SNMP has been used to monitor and manage networks by making queries and setting values more or less one by one.

SNMP can be used to fetch certain information about network attached SNMP manageable devices. As an example of an interesting case where SNMP may be used we can take network map creation. Because an implementation of standard MIB RFC

1213 (MIB-II) amongst a lot of other information can be queried for ARP tables, used for translating between IP and MAC addresses, SNMP can be used as a help when dealing with automatic generation of network maps. For some devices there also exists complementary MIB files for gathering such kind of data.

Because SNMP can be used to both read and set values in a managed device it is much up to the implementor of the agent to impose the limitations of what SNMP is capable of. The agent can for example be programmed to execute an application when a certain value is set. Another big feature of SNMP is its possibility to send asynchronous trap messages when a predefined event occurs. This can be used to monitor the network and preventing it from malfunctioning, for example an agent can be programmed to be able to send a trap when the number of erroneous packages in a router suddenly increases, allowing the system operator to notice the problem before a possible break down.

Even though most of the SNMP management tools, so called NMS (Network Management System) softwares or simply SNMP managers, on the market today are big and powerful applications, they tend to rely on the users good knowledge in SNMP. This makes using SNMP relative hard for the non-initiated user.

When using SNMP as a tool to fetch data about a system, sometimes complex queries are needed to be made. In this case complex queries may be when an index is used to link two entries from different tables together. It is relatively easy to make the adequate queries to solve such a task in a conventional SNMP manager software. Nevertheless often there is quite some overhead work that is needed to be done by the operating user, by fetching one table column at a time and manually solving the intended query. When working a lot with this kind of queries the idea quickly arises that one would like to be able to do such lookups automatically in some sense.

In the inventory software's SNMP Configuration tool this problematics is solved by letting the user choose if an index shall be used, the configurator is also allowing the user to set some restrictions on the fetched values, such that it shall only fetch values that are equal to a certain given value etc. The user is also able to create some simple formulas. During the development of the configuration tool a quite interesting idea was brought up. It was noticed that the tool got more and more features that is commonly found in simple SQL queries. Because a MIB file, used to describe management information in SNMP, together with the corresponding implementing SNMP agent can be seen as a database of information, it would be interesting to see if it is possible to implement something like an ODBC database driver for MIB files. This would enable the functionality to use SQL expressions to fetch data from SNMP manageable entities which could be a way to approach users that do not want or have the time to learn about the inner structures of SNMP.

3.5 Technical description of SNMP

The intent of this section is to look deep into the internals of SNMP. It explains all the constituents important for the solution provided in this thesis and what their purposes are as parts of the protocol.

3.5.1 Supported transport protocols

SNMP does mainly communicate over UDP. At first it may seem strange to use a protocol with such little control mechanisms as UDP but it is just that property that

makes it a good choice. In an overloaded network UDP may have a bigger chance of reaching the destination due to its simpler nature than for example TCP. UDP does not have to carry around that much overhead data which in turn possibly would cause more congestion on the network than there already was. Nevertheless there are no restrictions for using other transport protocols than UDP and any suitable transport protocol providing the same functionality as UDP may be used.

3.5.2 SNMP Agents and Managers

There exists two types of devices described in SNMP, managers and agents. A manager, or network management station, is a device or software that is managing one or more agents. A network management station is used for fetching data from agents. It receives and handles traps and responses from agents.

An agent, on the other hand, is operating on the managed device. The agent is implemented to send traps on different events. It is also responding to requests from a manager. The implementation of an agent is based upon MIB files (see 2.4.3 for more information about MIB Files).

3.5.3 SNMP message format

All communications between agents and managers are done by encapsulating the protocol data at a form called SNMP message format, see figure 3.1.

In SNMP messages all data types are defined in Abstract Syntax Notation One (ASN.1) to get a platform independent view of data types. This means that all fields in an SNMP message shall be of a valid ASN.1 data type. To increase platform independence ASN.1 uses Basic Encoding Rules (BER) for encoding of data. This means that all SNMP data fields are valid ASN.1 types and must also be encoded according to BER. In short the basic encoding rules says that data is always prepended by its type and its length. Thus, every field in the SNMP Message, including the message itself, must hold not only the data but also the ASN.1 type and data length.

An SNMP message is divided into three blocks, the SNMP version field, the SNMP community string field and the SNMP PDU (Protocol Data Unit) block. The SNMP version field is an integer that tells the receiver about which version of SNMP that shall be followed. The SNMP community field is a string holding the community string used for access limitations, the community string can be seen much like a password. A PDU is in turn built out of four blocks, the request id field, the error field, the error index field and the variable binding list block.

More information about the SNMP message format can be found at the web page SNMP: Simple? Network Management Protocol [1].

3.5.4 Abstract syntax notation number One (ASN.1)

ASN.1 is a standard notation for describing data structures and encoding rules for communication between entities. The main purpose for using ASN.1 is that it is not dependent on the platform on which the described data structures originates. Another advantage is that it due to its support for many encoding rules can form very bandwidth efficient transmissions between entities. The standard has some pre-defined simple types like integers, boolean values, strings etc. ASN.1 also has support for describing more complex constructed data types such as structures, lists etc. Read more about the ASN.1 notation here [17].

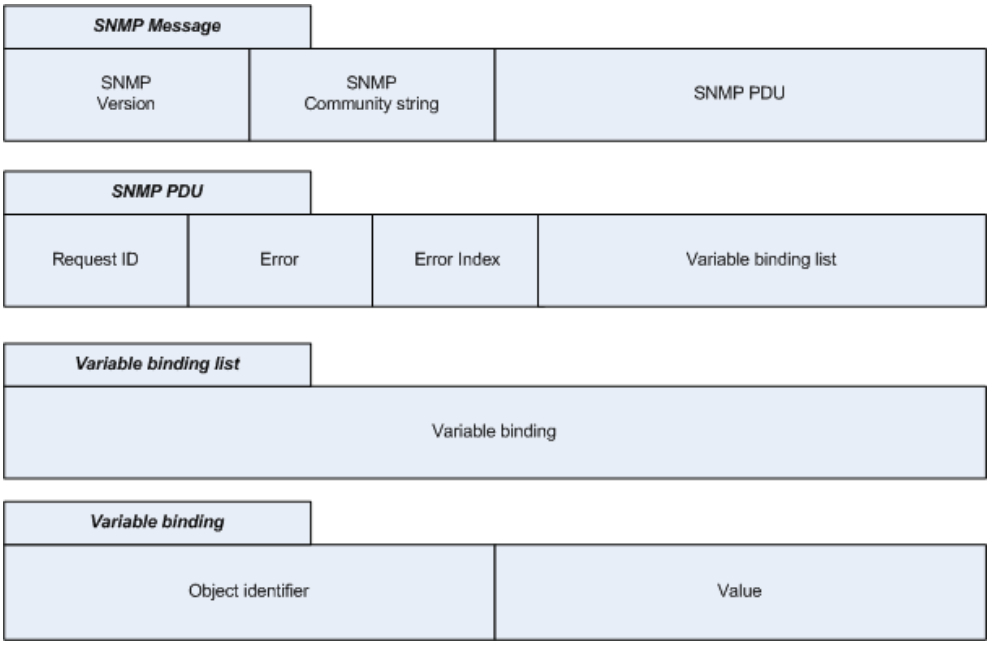

Figure 3.1: Overview of the SNMP message format

Figure 3.2: A block diagram describing the ASN.1 value encoded with BER

Encoding rules of ASN.1

According to [17], one of the main reasons for the success of ASN.1 is that it supports several different encoding rules. Two of these encoding rules are BER (Basic encoding rules) and PER (Packed encoding rules) which both are standard encoding rules that defines how the values defined in ASN.1 shall be encoded.

In SNMP the BER encoding rules are used which means, as previously mentioned in section 3.5.3, that every value, or field in the SNMP message format, is encoded as an array of three fields. All BER encoded ASN.1 values are encoded in the Type Length Value (TLV) format. First in every BER package one has to inform the receiver about the ASN.1 type. Because for example an integer not alway is the same size on all platforms the next information that will be sent is how long the value transmission is. The rest of the package is the actual value that shall be transmitted. See figure 3.2 for a block diagram describing the TLV structure used in ASN.1 with BER encoding.

3.5.5 Structure of Management Information (SMI)

The Structure of Management Information (SMI) is an adapted subset of ASN.1 which defines the language used for writing MIB files. The current version, SMIv2, is described in RFC 2578, see [8].

SMI supports three different types of structure definitions, which are module definitions, object definitions and notification definitions. A module definition defines something called an information module. An information module is a structure that encapsulates various objects and notifications into a module. An object definition defines, just as the name suggests, SNMP managed objects. These objects can then be read and/or fetched from an SNMP agent that is implementing the encapsulating MIB module. An object definition not only defines the syntax of a managed object but also the semantics. Notification definition are used for describing unsolicited SNMP messages such as traps. Just as in the case with object definitions, notification definitions not only describes the syntax of the notification but also the semantics.

The syntax used in SMI to describe notifications and managed objects allow a subset of the ASN.1 primitive types, so called non-aggregate types, to be used. SMI also allows for compounded types, so called constructor types or aggregate types, to be created.

The primitive types available in SMIv2 are INTEGER, OCTET STRING, OBJECT IDENTIFIER and NULL. Constructor types can be lists or tables where lists are named SEQUENCE and tables SEQUENCE OF. The SEQUENCE type definition is followed by the types of the list entries as follows, SEQUENCE <type1> ... <typeN>. The SEQUENCE OF type definition is followed by a defined list to form a table as the following shows, SEQUENCE OF <list>.

There also exists two other constructs which need some attention here. Enumerated integers and defined types. To start with enumerated integers, one may define a textual description of each number. Say for example if one has a managed object describing the state of a network interface, the response from an agent shall be an integer between 1 and 3. Then one could use the following syntax {up(1), down(2), testing(3)} to get a description of the integers as well. The defined types are based upon legally defined SMI types. Some predefined types are IPAddress, based on OCTET STRING, TimeTicks, based on INTEGER, and Counter32 which is also based on INTEGER.

3.5.6 SNMP operations

SNMP is mainly capable of three things, to set values in a manageable device, get information from a manageable device and to deliver unsolicited notifications when certain events have occurred.

The list below holds all possible operations in SNMP. As one can see there exists only a few operations and they are all variants of the three main functionalities of SNMP, get values, set values and receive asynchronous notifications on events.

- get
- get-next
- get-bulk (from SNMPv2)
- set
- get-response

- trap
- notification (from SNMPv2)
- inform (from SNMPv2)
- report (from SNMPv2)

Below the GET and the GET-NEXT operations will be described in detail. Other operations are left out because they are of no practical use in this master thesis project.

SNMP GET

The GET operation is used to fetch single values from an agent.

Data flow:
Manager (GET) Agent (GET-RESPONSE) Manager

A manager starts by sending a get request to an agent. It enters the name of the object it wants to fetch in the variable binding list. When the get request is received by the agent it is given one of the states in table 3.1.

Table 3.1: Error states in a GetResponse PDU when responding to a GET request

Error status	**Error index**	**Appearance**
noSuchName	The index of the said object name	If any object requested in the variable binding list does not exist in the current MIB view or if any object requested in the variable binding list is of aggregate type.
tooBig	0 (Zero)	If the size of the response would exceed local limitation.
genErr	The index of the said object name	If any other error causing that the object cannot be retrieved.
noError	0 (Zero)	If no errors occurred.

To read more about the error states in a GetResponse PDU when responding to a GET request, see [6].

If no errors occurred the entity that received the request sends a GetResponse PDU back to the originator. The get response is composed so that each name-value pair in the variable binding list is filled in with the name and value of the wanted variable. The request-id field of the response is set to be the same as the received message.

For more information about the GET operation, see [12, 6].

SNMP GET-NEXT

The GET-NEXT operation is used to fetch a group of values, for example a list of IP addresses.

Data flow:
Manager (GET-NEXT) Agent (GET-RESPONSE) Manager (GET-NEXT) ... Agent (ERROR) Manager

The manager sends a GET-NEXT PDU to the agent, if the agent can respond it responses with a GET-RESPONSE PDU. The manager again sends a GET-NEXT PDU and receives the next object in the list. The manager continues to ask for more values with new GET-NEXT PDUs until the agent responds with an error saying that there is no more elements to fetch.

Table 3.2: Error states in a GetResponse PDU when responding to a GET-NEXT request

Error status	**Error index**	**Appearance**
noSuchName	The index of the said object name	If any object requested in the variable binding list does not lexicographically precede the name of some object available for get operations in the current MIB view.
tooBig	0 (Zero)	If the size of the response would exceed local limitation.
genErr	The index of the said object name	If any other error causing that the object cannot be retrieved.
noError	0 (Zero)	If no errors occurred

To read more about the error states in a GetResponse PDU when responding to a GET-NEXT request, see [6]. For more information about the GET-NEXT operation, see [12, 6].

Chapter 4

Accomplishment

This chapter explains the procedure of how the solution was accomplished. It also describes what the outcome of the application has been, in terms of technical details, such as a rough system design and coding choices.

4.1 Initial planning

At the beginning of the project a detailed plan for how the work would be done was made. A less detailed version of the planning is available in table 4.1 below. The plan was held at first, but because the project actually were bigger than first expected it took longer time at the Implementation 2 phase. This led to less testing than originally planned and the start of the report was also postponed for some days. At first the idea was to document each step of the plan in separate documents but this was also somewhat rationalized away due to lack of time.

Table 4.1: The initial planning

Activity	**Duration**	**Start**	**End**
Prerequisites	5 w.days	2005-08-01	2005-08-05
In-depth study	27 w.days	2005-08-01	2005-09-06
Requirements	8 w.days	2005-09-07	2005-09-16
Implementation 1 (Study SNMP)	10 w.days	2005-09-05	2005-09-16
Analysis and Design	15 w.days	2005-09-21	2005-10-11
Implementation 2	20 w.days	2005-10-12	2005-11-08
Testing	55 w.days	2005-09-19	2005-12-02
Report	31 w.days	2005-12-05	2006-01-16
Oral presentation	20 w.days	2006-01-02	2006-01-27

4.2 How the work was done

The work was done at a company in Gothenburg, Sweden. To start with it was said that there should be an existing prototype with some functionality finished within three months.

Having almost no previous knowledge at all about SNMP it seemed like a fully possible thing to do. What was not known then was that SNMP contained much more than just some send and receive functionalities. It pretty soon became evident that a parser for MIB files was needed to be built. To do the parser a special implementation of Lex/Yacc called Turbo Pascal Lex/Yacc was used. The parser took a great deal of time to finish, a lot of which was the time for learning the structures of the MIB files.

When the parser was done an internal structure for the MIB files existed and it was then possible to present the MIB files graphically as trees. This is a much more convenient way of reading MIB files than looking at the textual representation. Alongside with the construction of the MIB parser a component for fetching SNMP data was rewritten to fit the needs of the intended application.

When so both the MIB parser and the SNMP component where ready they were used together and suddenly it became very interesting. Now there was a neat way of pinpointing a certain object in the MIB tree and just fetch it, given that one would know what IP to fetch from. This lead the work into the next step which was to construct a way of automatically/semi-automatically discover all SNMP manageable devices in the network.

The first attempts of doing some kind of automatic device discovery was with a broadcasted query to all devices in the current subnet. This method is fast and does not congest the network that much. However if one wants to know something about a device that is located on another subnet the whole thing gets more difficult. Because most routers does not allow for retransmitting broadcasts from one network to another the broadcasted discovery message would stop at the first router. The way this problem is solved is by setting up certain IP ranges that shall be scanned for devices. This way of working means that the same message is sent once for each IP address in the given IP range, which in large ranges may become an awful lot of messages sent.

The first attempt, which of course was doomed from the beginning, was to send a message to each and every IP address in the range and before sending the next one wait for a reply. Because often only a small part of the range is actually used and able to respond to SNMP messages this led to severe slowness, waiting for the receives to time out.

The discovery procedure was somehow needed to be done asynchronously. That is, the application must be able to continue with sending the next discovery message instead of waiting for the first to be responded to. This was solved with support from the WinSock2 API which allows for setting up user defined windows messages at certain events, in this case on the receipt of a UDP datagram. With this functionality the application could now send out messages by iterating through the whole range and then wait a little while for the results to arrive, handling the devices asynchronously as their responses arrive. That is how the device discovery is done.

Now the application were able to find all wanted SNMP manageable devices and fetch certain values found in MIB files. This is were the configurator steps into the scene. One wants to be able to set up a list of values that shall be fetched from all devices that is discovered. One may also want to split the devices into groups to be able to have different configurations depending on the device type. For example it is not interesting to ask a printer for how big its hard disk drive is because it has none. Every SNMP manageable device has something called an Object ID which is one of the minimum values that the device should respond to. The Object ID is a string which makes it possible to identify the manufacturer of the SNMP agent and sometimes also the exact device type. By always fetching this number at discovery one is able to split the devices

into groups depending on what their Object ID is. This gives the opportunity to fetch different things depending on which group the device belongs to.

In short terms the fetch configuration is applied to a group of devices which is constructed with respect to the devices Object IDs.

Now there existed separate lists of values that should be fetched for the different device groups. For each entry in those lists there also exists a specification on where the value shall end up in the inventory software. To make the lists more flexible two new types of entries were added. The first one consists of constants which gives the user possibility to insert a non fetched constant value such as the device group name for every device in the group. The second one consists of formula entries where two values can be added, subtracted, multiplied or divided to achieve correct information. For example, disk and memory sizes shall always be reported in megabytes in the inventory software but sometimes an SNMP manageable device responds with kilobytes or something else, then this value could be scaled by using a formula.

4.3 Design

Delphi is a Rapid Application Development (RAD) tool which in some sense lets the developer draw the applications with components, instead of writing them. Delphi is said to be a fully object oriented language [14] with support for the major principles of object oriented programming. There is still some amount of non objectiveness left in Delphi since the days of its imperative predecessor, Pascal. This may be compared with the way it is possible to mix C and C++ code in a C++ compiled application. The most of the application is therefore implemented with object oriented design but some parts are approached in a way that one may call imperative design.

4.3.1 Inheritance of the SNMP Manager

The most important line of inheritance is the one of the TdsSNMPManager class, see figure 4.1. The TdsSNMPManager inherits from TpdSNMPManager which in turn inherits from TpdSNMP and at last the TpdSNMP class inherits from TpdUDPSocket. To start from the bottom the TpdUDPSocket is a class that is based upon the WinSock2 API. TpdUDPSocket gives basic transport functionality to the TpdSNMP class which in fact is a changed version of Indy's SNMP component rewritten to function with the TpdUDPSocket instead of Indy's own UDP component. Some other things has also been rewritten in order to let the component support bigger representations of integers. TpdSNMP provides functionality for performing the simplest forms of SNMP operations. Next, the TpdSNMPManager which relies on TpdSNMP is a class that extends the SNMP functionality in terms of SNMP manageable device discovery in certain IP ranges, translations of fetched values with regards to their object syntax, support for storing discovered devices in different groups depending on their Object ID and also it provides some useful events that triggers when an SNMP PDU was received. The TdsSNMPManager utilizes TpdSNMPManager's functions and extends them by adding functionality specific to the inventory software. At this level the discovered devices also has possibility to store lists of fetched values that shall be imported into the inventory software.

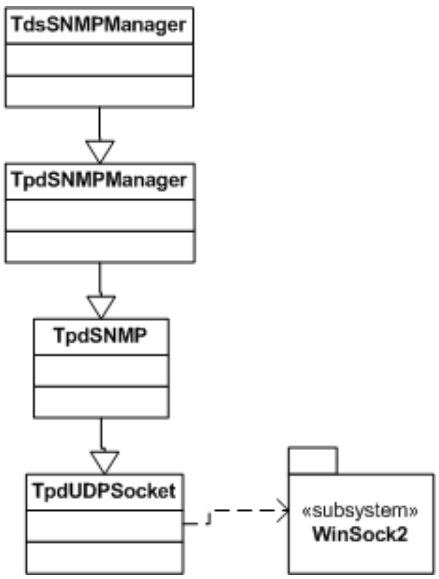

Figure 4.1: Inheritance line of the TdsSNMPManager class

4.3.2 Packages

The design is further explained in Appendix A as descriptions of the implemented packages.

Chapter 5

Results

This chapter describes this thesis's resulting software solution. The different parts of the software will be described with a brief description on how to use them.

5.1 SNMP Inventory Configurator

The SNMP Inventory Configurator is included as a view in the inventory software and is holding the configuration of what shall be fetched via SNMP and how it shall be done. It is possible to set up different configurations for different types of devices. The configurator main view is splitted into four panels, the group list, the import specification table, the device table and the device data table. See figure 5.1. In addition to those four main panels the configurator tool also contains a general settings dialog, a group settings dialog, an import row settings dialog, a MIB browser, a copy dialog and a settings dialog for device discovery.

5.1.1 Group list

The group list, see figure 5.2, holds a list of device groups. A device group is an abstraction for a collection of SNMP manageable devices. Because all SNMP manageable devices has an object identifier (OID) string that somewhat identifies the devices manufacturer and type, one is able to group the different devices on this string. The grouping is useful for getting correct icons after having finished an SNMP inventory in the inventory software. They are also useful for creating different configurations depending on what type of device is being investigated. As mentioned, the devices are grouped by different OIDs and this is set up in the group settings dialog, see figure 5.3. There is always at least two device groups present in this list and they are protected by the system and shall be treated a little different than the rest. The two default device groups are All Devices (protected) and Other Equipment (protected). The All Devices group is special in the sense that it cannot be deleted, nor shall it be seen as a real device group. The intention of the All Devices group is that it shall serve as a group holding a minimum inventory configuration, everything that is configured to be fetched in All Devices will always be fetched in an SNMP inventory no matter what device group. The Other Equipment group also cannot be deleted but different to the All Devices group it can be somewhat altered by the group settings dialog. This is because the Other Equipment can be more treated as a real device group holding all devices that did not

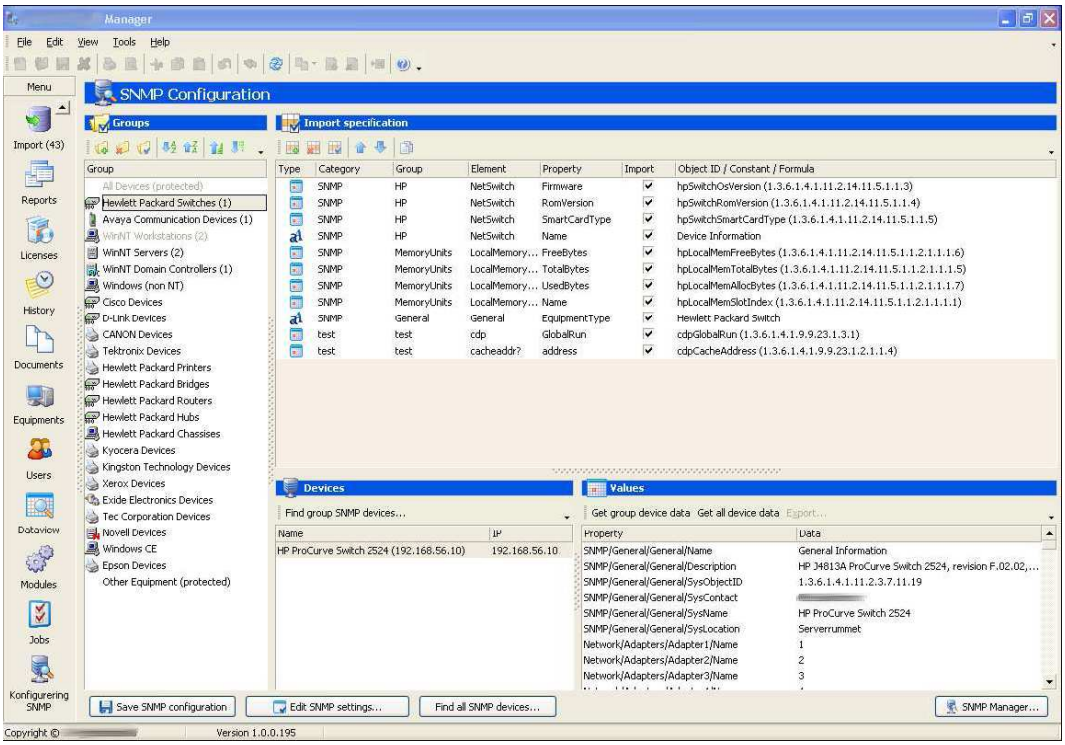

Figure 5.1: The SNMP configuration view in the inventory software showing the Hewlett Packard Switches group

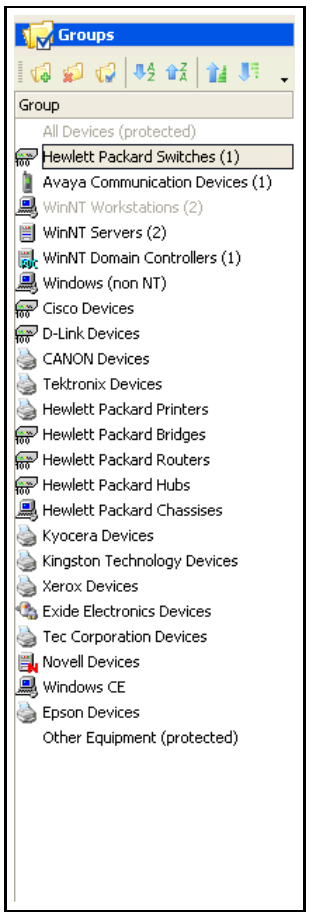

Figure 5.2: The device groups panel

fit in any other group. It is therefore not possible to change the OIDs that comprises this group nor is it possible to change the name but all other parameters can be edited just as in a normal user added group.

5.1.2 Device group settings

The device group settings dialog, see figure 5.3, is used to configure a device group. A device group consists of a name, an equipment type, a naming parameter, a variable for whether the group shall be included in an inventory or not and a list of object identifiers that shall belong to the group. The group name, equipment type and include in inventory choices are trivial to set up by typing or choosing the desired value. The naming parameter is using a way of replacing strings to achieve names for devices in the group in the inventory software. The string provided in the Naming text box will be parsed and all @N will be replaced by the devices SysName, in the same way @I will be replaced by the devices IP address. Take figure 5.3 as an example. If there exists an Hewlett Packard printer on IP address 192.168.0.5 with SysName Print01 and the naming parameter set up as @N (@I) then the device name will be Print01 (192.168.0.5).

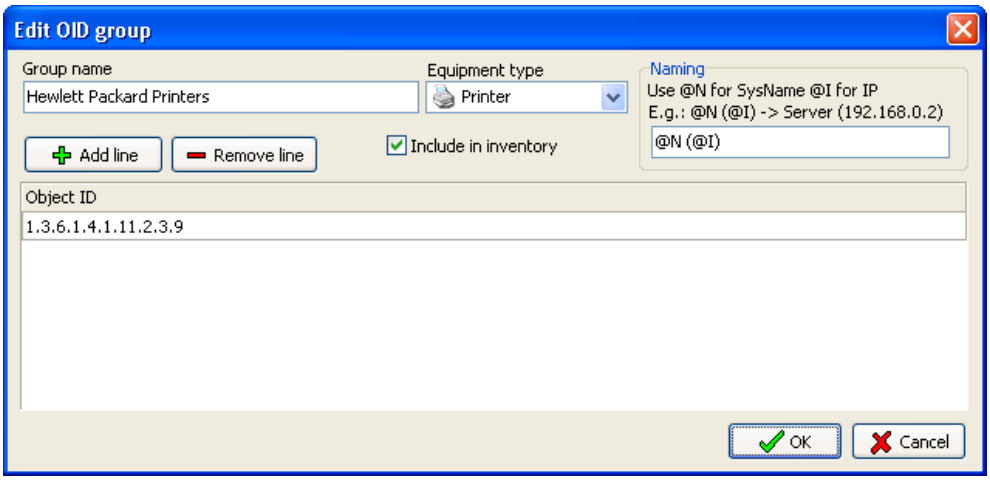

Figure 5.3: The device group settings dialog

The object identifier list holds the object identifiers that one wants to associate with this group. It may be one or more object identifiers, an example with more than one object identifier may be if one wants to group all kinds of computers running Microsoft Windows in one group, then the object id for each respective Windows system shall be added to the list. The application will compare the object identifiers in the list with the leftmost characters of the devices' object identifiers. Thus, in some cases another way of covering for example all Windows computers may be to find a position in the object identifier up until when all Windows computers respond the same and insert the part of the string that was identical in the list.

5.1.3 Import specification table

For each device group (see 5.1.1 for info on device groups) one is able to make a specific configuration of what shall be fetched and what the application shall do with fetched values. The import specification, see figure 5.4, holds an overview of the currently selected device group's configuration. Each row in the table represents one value, the value can either be of constant, fetched or formula type. The properties that can be read directly from the list are the value type, the inventory software's namespace, if the value shall be imported to the inventory software in an SNMP inventory and also a hint of how the value is retrieved.

5.1.4 Copy entry

To copy an entry in the import specification from one group to another select the source row, click the right mouse button and choose "Copy row to...". By doing so the Copy entry dialog will be shown, see figure 5.5.

Choose the destination group for the entry and click OK. If the "Jump to destination after copy operation" checkbox is checked the destination group and entry will be selected after a successful copy operation is performed.

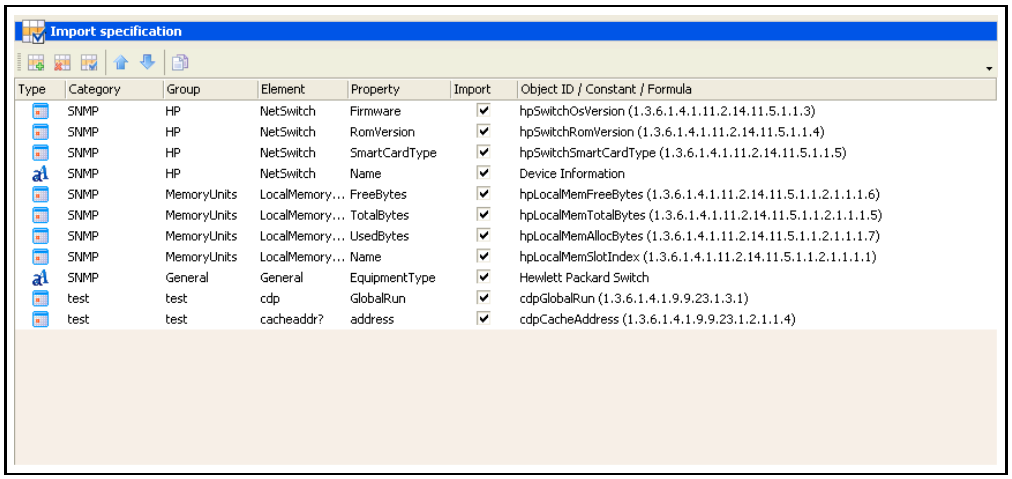

Figure 5.4: The import specification panel

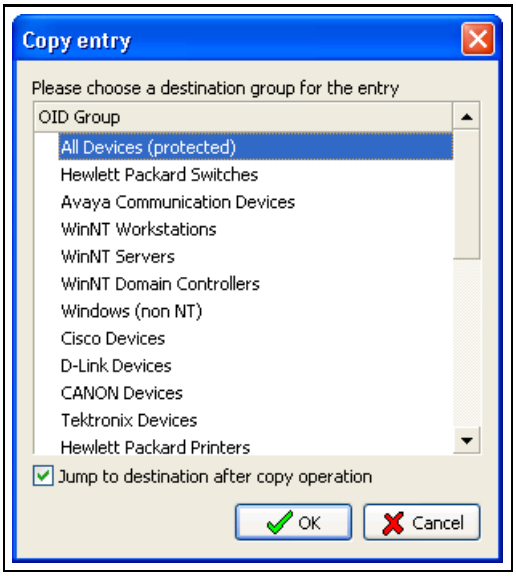

Figure 5.5: The Copy entry dialog copies entries in the import specification between groups

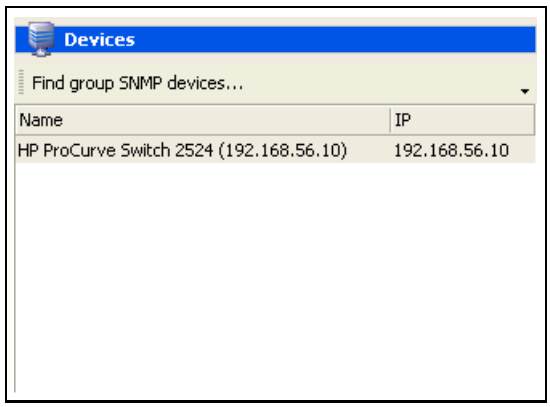

Figure 5.6: The device table

5.1.5 Device table

The device table, see figure 5.6, is always empty when the application starts. If the button "Find all SNMP devices..." or the button "Find group SNMP devices..." is clicked the Find SNMP devices dialog will appear to let the user scan IP ranges of the network for SNMP manageable devices, see figure 5.7.

The main purpose of the device table is to check that the current device group configuration functions as planned.

5.1.6 Device data table

This table, see figure 5.8, is intended for testing purposes. In this table values from the currently selected device can be presented. When the "Get all device data" or the "Get group device data" button is clicked the currently selected device is queried for the values in the current configuration. If the "Get all device data" button is clicked, all data configured for the group will be fetched, including the data configured in the All devices group. If the "Get group device data" button is clicked only the data specified in the current group, excluding the All devices group, will be fetched and displayed.

5.1.7 Import row settings dialog

The import row settings dialog, see figure 5.9, is used to edit the properties of a so called import row. The properties are divided into four categories, naming, formula settings, value and linking.

The first category, naming, holds the name of the current value. A name consists of a category a group, an element and a property text box, all of these must be filled in as they are used for accessing the value in different scenarios.

The formula settings shall be used if the current value is a formula. To use a formula value, the "Use formula" checkbox shall be checked. It is possible to do addition, subtraction, multiplication or division of numerical values and numerical vectors. If one of the operands is a vector, the mathematical functions are applied element pairwise so that the first element of the result vector is achieved by applying the operator on the first element in operand vector one and the first element of operand vector two and so on. If a constant value is used the constant is applied to all elements in the resulting

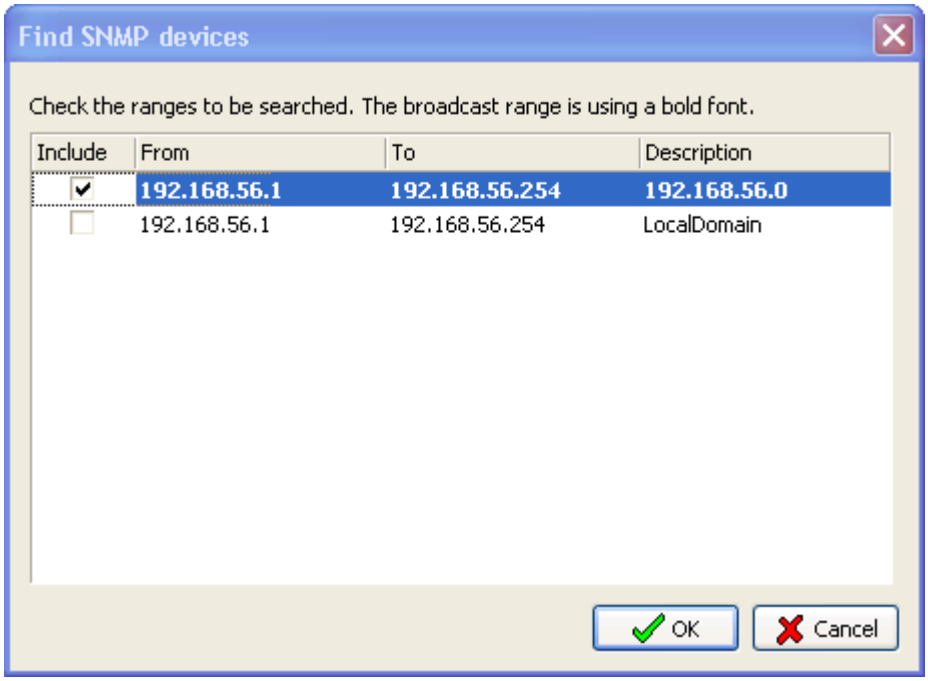

Figure 5.7: The Find SNMP devices settings dialog

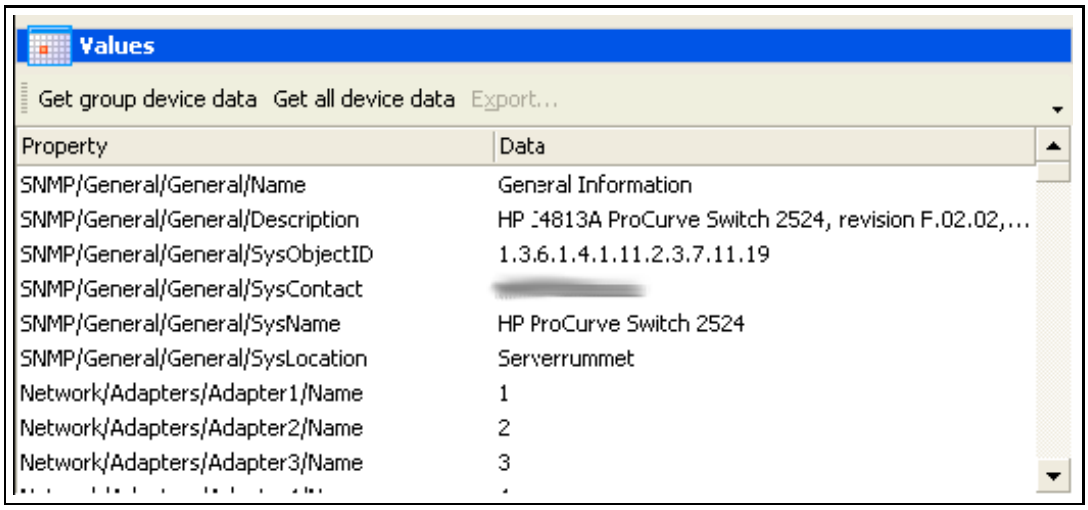

Figure 5.8: The devices data panel, displaying fetched data from an HP switch

vector. The index, read more on index below, is inherited from operand one if possible or else the index of operand two is used. In the future it may be a good thing to add more functionality such as string concatenation, different kinds of column operations etc.

The value settings is used when handling fetched or constant values. If one wants to import a constant value such as a group name etc. then the "Use constant value" checkbox shall be checked. If one wants to use an index together with a constant value one first has to choose a fetched column value to receive its column headers, to choose index from, and then click the "Use constant value" checkbox, the application will now fetch the whole chosen column object but will always write the constant as result and associate it with the chosen index. This procedure may be an object for improvements at a later point. If the value shall be fetched from a device rather than being set as a constant the "Use constant value" shall be unchecked and the object id of the value shall be entered in the "Object ID" text box. To get the correct object name, syntax and mib module it is recommended to use the MIB browser to pick the desired object id. The MIB browser is opened by clicking on the "Browse..." button. Read more about the MIB browser in section 5.1.8.

The last property category is the linking properties. Here one can set an index for the fetched value and also build SQL-like where expressions. The index is a value that replaces a ? in the Element string under the Naming category. This is useful when one has chosen to fetch an object that is a column. For example if one has chosen to fetch the ipAdEntAddr object (1.3.6.1.2.1.4.20.1.1), in the ipAddrTable contained by the RFC1213-MIB, the device will answer with a list of all its IP addresses, usually one per network interface. To keep track of the different IPs one can choose another column in the same table as ipAdEntAddr to use as an index, preferably one wants to choose ipAdEntIfIndex for this. There also exists a table called ifTable, containing amongst other information the MAC address of interfaces and also the same index number as found in the ipAddrTable which makes it possible to link the IP and MAC address together in the inventory software via the element tag.

In the linking category there also exists a where expression where one can choose from the values in the same context as the fetched value. This value can then be used together with an operator and a textual expression to form a rule for what values that shall be imported. For now, the available operations are just EQUALS and NOT EQUALS, they function as their corresponding operators in SQL. The list of operators would be possible to extend in the future to hold more operators such as the SQL LIKE or SQL IN operators. For an example of a where expression please consider the following scenario. One wants to fetch all IP addresses for a device but one does not want to receive the 127.0.0.1 address, which is the address of localhost and therefore not interesting outside the device. Then one would, just as in the example of indexes above, choose to fetch the ipAdEntAddr object and then choose ipAdEntAddr in the where expression combo box together with NOT EQUALS from the operator combo box and 127.0.0.1 in the comparison string text box.

At last there is the "Import to inventory software" checkbox. In some cases one wants to fetch or calculate values without importing them to the inventory software. For example if one wants to fetch a value to use as an operand in a formula value that shall be imported but does not want the operand itself to be imported. In that case the "Import to inventory software" checkbox shall be unchecked.

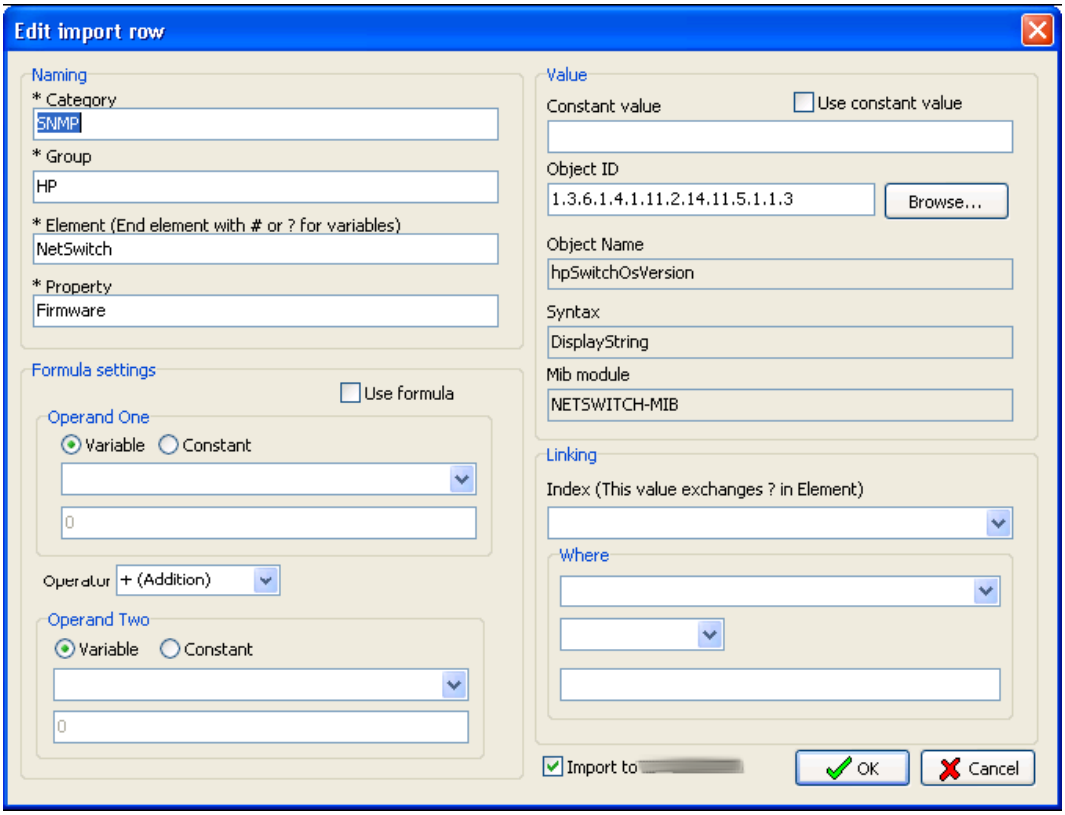

Figure 5.9: The import row settings dialog

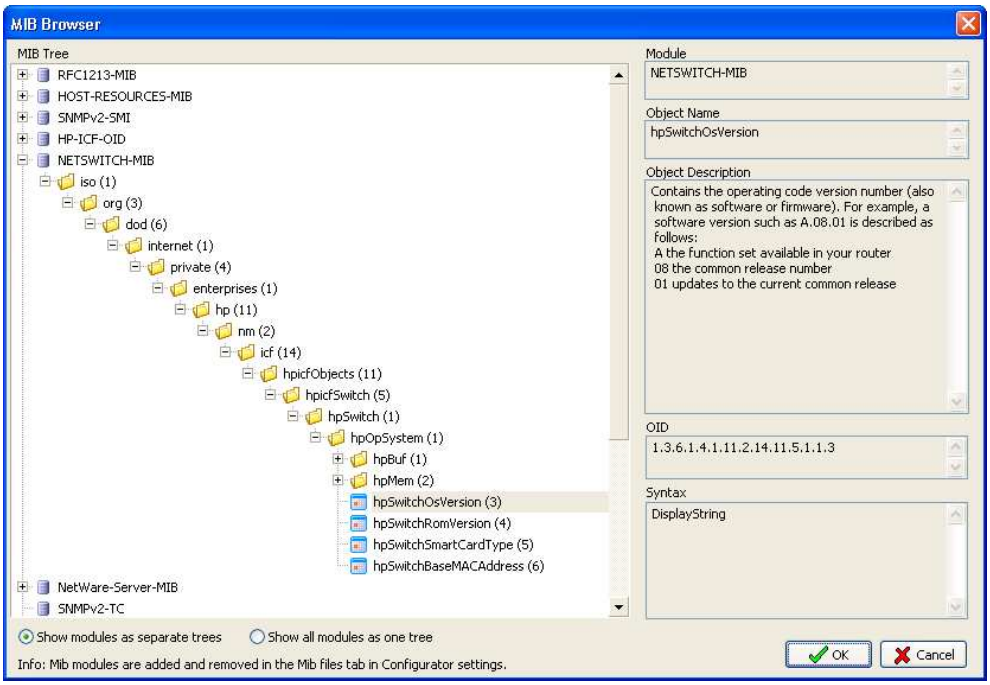

Figure 5.10: The MIB browser dialog used for choosing values to fetch

5.1.8 MIB Browser

The MIB browser, see figure 5.10, is used to browse through the MIB files included in the SNMP configurator settings, see section 5.1.9. It is used for choosing values for import and presents MIB files in forms of trees. Because all MIB modules are sub trees of the same big tree one can choose to present all MIB files as one big tree or as separate trees, one per MIB module.

While browsing around the objects in the MIB tree some helpful information is displayed in the text boxes at the right side of the tree. It tells from which module the value was fetched, what its object name is in the MIB tree, it gives a description of the current object if available and also displays the object id and syntax of the current object.

To choose a value, highlight the value in the tree and click the OK button.

5.1.9 SNMP configurator settings

The SNMP configurator settings is used for overhead configuration of both the configurator and the SNMP manager. Here one can set up what parts of the network that shall be searched for SNMP manageable devices, what SNMP specific properties, such as community strings, send delays etc., that shall be used and also which MIB files that shall be included in the MIB browser.

The IP ranges, see figure 5.11, are pretty simple to set up. For each range one provides a start of the range and an end of the range, one shall specify if the provided range shall be included or excluded and also a name of the range, which will be the network/domain name reported in the inventory software for a device in that specific

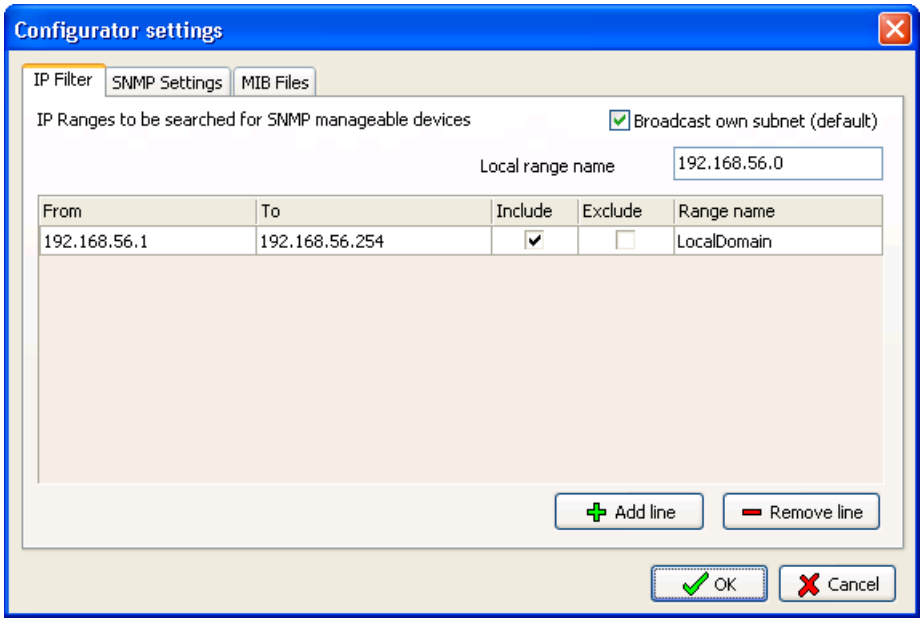

Figure 5.11: IP filter settings

IP range. In the IP filter ranges one can also choose to broadcast the own subnet. This is the fastest way of receiving information about devices in the local subnet. The range name for the own subnet can also be specified and defaults to the local subnet address.

In the SNMP settings tab one can specify community strings for SNMP. There are two community strings, one for reading and one for writing, they work as passwords giving access for reading and writing values on an SNMP manageable device. Default most SNMP manageable devices come with standard community strings, public for reading and private for writing. Because no writing, for the moment, is needed by the application the write string is not used and cannot be altered. The application currently uses the same community strings for all devices so it will be up to the network administrator to have the same community strings setup on all devices.

The port for getting and setting values, default 161, may be altered but since no support for traps are needed at the moment the trap port is not used and cannot be altered.

It is possible to configure the timeout value, which is the longest time the SNMP manager will wait for a single response, and also the number of retries before the query will be abandoned.

The discovery frequency properties are used when searching the network for SNMP manageable devices. The first value, Make pause every # IP, is a value for how many IPs that will be queried before making a pause, it may be necessary to make those pauses if large networks are searched. The next value, Pause [ms], is how long a pause shall last in milliseconds. The last value, Delay between each discovery message [ms] is the amount of time the manager shall wait between each discovery message is sent, this is also a good control mechanism when working with big networks. If these three values are improperly setup the network may be congested by SNMP messages and become slow. Because SNMP relies on UDP as transport protocol responses may be lost in

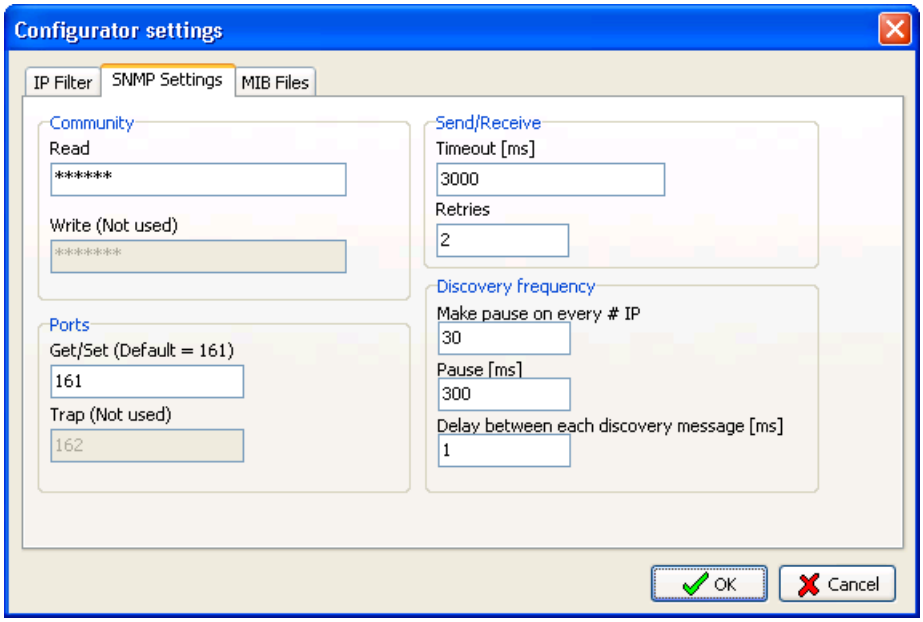

Figure 5.12: SNMP Settings

congested networks.

The MIB Files section of the settings dialog, see figure 5.13, is used for selecting which MIB files that shall be available in the MIB browser. The MIB files included in the list are parsed in the order they are listed. For example the hostmib.mib relies on the RFC1213-MIB.mib and must therefore be placed below RFC1213-MIB.mib. The MIB files can be rearranged with the Move up and Move down buttons. A MIB file may be needed for the parser in order to parse another MIB file even though it does not hold any interesting information. In that case one can choose to set it as inactive. It will then be parsed but not shown in the MIB browser.

5.2 SNMP Manager

The SNMP Manager, see figure 5.14, is the stand-alone application used for performing the actual network investigation based upon the configuration created in the SNMP Configurator in the inventory software. It first uses the settings for IP ranges to scan the network for SNMP manageable devices. During the network scan the SNMP manager asks the devices for their object identifiers which identifies the device group. When the the device discovery process is finished, the SNMP manager has knowledge of all available devices in the network and their respective device groups, and can start the inventory process. One by one the found devices are investigated. The result is sent to the inventory software for import. When the import is finished the data of the investigated devices are available in the inventory software.

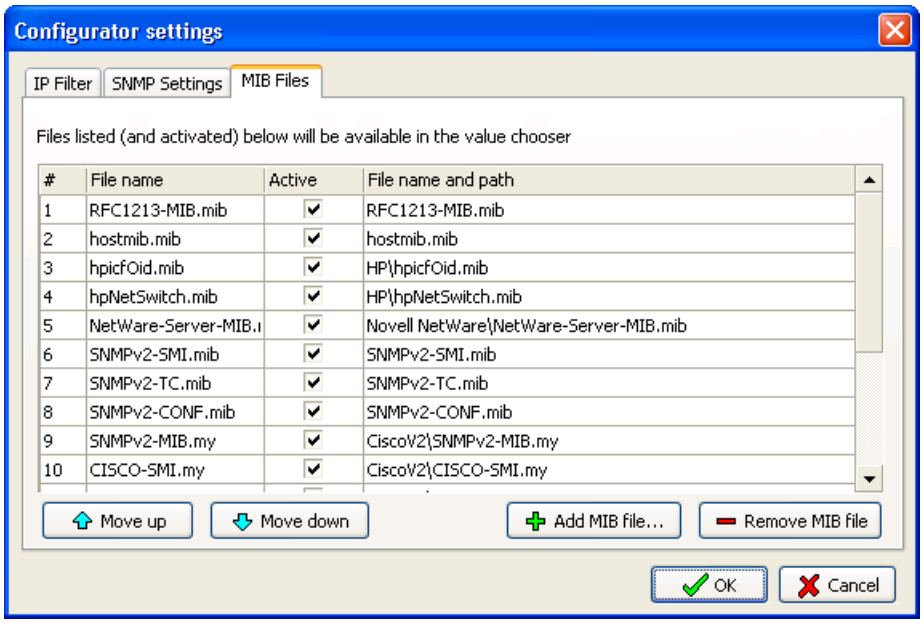

Figure 5.13: MIB Files settings

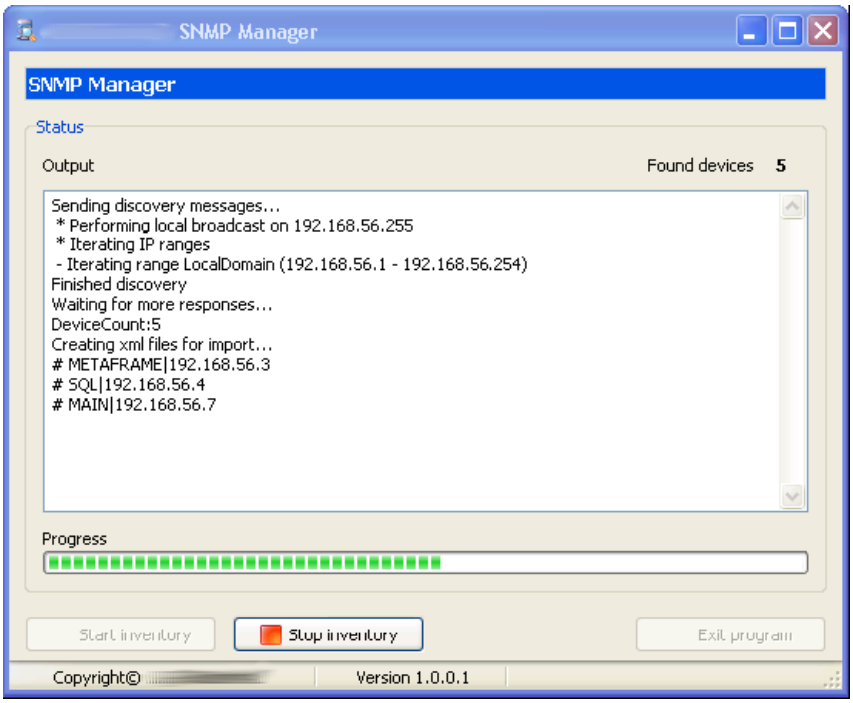

Figure 5.14: The SNMP Manager performing a network investigation

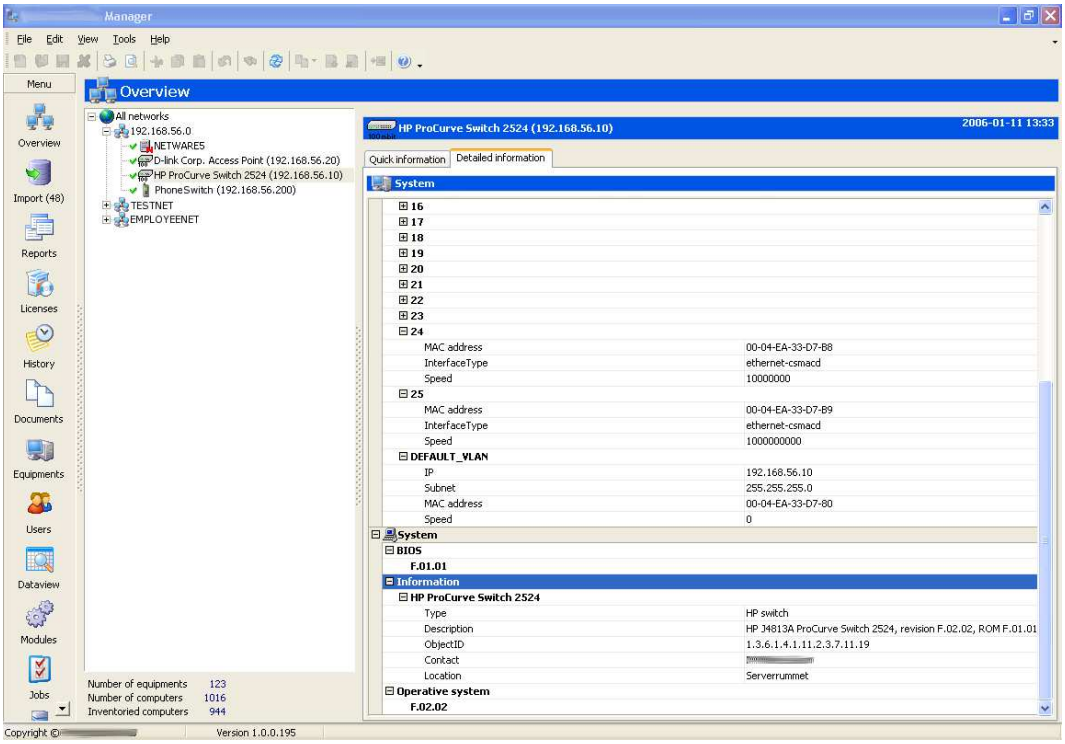

Figure 5.15: The inventory software manager's overview after having performed an inventory

5.3 The inventory software manager

The inventory software manager is a part of the company's inventory software suite. It is here that the user probably most often will be located when working with the inventory software. The SNMP Inventory Configuration part of master thesis project, see section 5.1, is integrated as a view in the manager. Apart from configuring SNMP inventories the manager is capable of showing all inventoried devices in different ways. It is also possible to manually add own entities such as desktops, mobile phones, cars or any gadget that needs documentation. On each device it is possible to connect not only users but also certain documents that is found to be interesting regarding the device. The manager also holds a powerful report generator for building dynamic reports from the database. This is some of the functionalities of the inventory software manager, of course there exists a lot more. After having performed a network inventory the investigated devices appear in the inventory software manager's overview, see figure 5.15.

Chapter 6

Conclusions

When I first started doing this master thesis project I did not know that there was so much to learn about the simple network management protocol. In fact it is not too simple all through and there emerged a lot of challenges to consider during the development process.

I was amazed about the originality of this application when I searched for information on the Internet. For some reason it does not seem to exist solutions that are alike or at least they are not very common on the market. I hope that this way of using SNMP for investigation of network devices may be further developed in the future. It would possibly open up for very exciting new application functionalities.

It feels very good to have completed this work. There is still a lot that can be done to improve the application but as a whole I am very satisfied with the outcome.

6.1 Goals revisited

This section contains comments on how well the original goals were reached.

1. Find all SNMP manageable devices in a network

   The application shall be able to find all SNMP manageable devices connected to the investigated network.

   Comment:

   The application has a way of automatically finding all SNMP manageable devices in the current subnet. For devices outside the current subnet there exists functionality for setting up IP ranges that shall be scanned for SNMP manageable devices. I'm satisfied with the outcome of this goal.

2. Parse MIB file(s)

   The application shall be able to parse standard MIB files, see section 2.4.3 for more info about MIB files, and also be able perform some kind of error detection.

   Comment:

   Because some time were taken from the testing phase this is not as thoroughly tested as it should have been. Nevertheless the application can parse all tested MIB files and during the development a whole lot of MIB files have been tested. The parser is able to give parser errors though these have been hidden from the user.

If something should go wrong during the parsing an exception would be generated and the user would receive some kind of information. For example if the parser cannot find a MIB file that the parsed MIB requires it gives information about that. I would say that this goal is reached, the parser does its job as it should.

3. Provide visual representation of MIB file(s)

   The application shall be able to provide a visual representation of the parsed MIB files, most probably as some form of tree.

   Comment:

   There exists a MIB browser which is capable of showing the MIB files all in one tree or as one tree per separate MIB module. The MIB browser also gives information about the different MIB objects while browsing around in the tree. In some very special cases an object may end up outside the tree, this happens when the parser for some reason fails to resolve the location of an object. I would say this goal is reached.

4. Fetch values

   The application shall be able to fetch values from SNMP manageable devices in the network. It shall be possible to fetch standard values as well as specific MIB values.

   Comment:

   It is possible to fetch values from any SNMP manageable device as long as it is not encrypted with support of the encryption standard of SNMP version 3. It was not required by the company that the application should be able to fetch encrypted values so I would say that this goal is reached.

5. Display values

   The application shall be able to display the values collected from the SNMP manageable devices in some form.

   Comment:

   It is possible to fetch and display the configured values of a device group in the SNMP Inventory Configurator. The values are also displayed in the inventory software manager's overview after an SNMP inventory has been performed. This goal is reached.

6. Adapt fetched data for inventory software import

   The application shall be able to adapt the fetched data in such way that it can be imported into the inventory software's database.

   Comment:

   Here there are three major things to consider. Network/domain names, device names and value names. The network/domain name is used in the inventory software manager's overview to categorize the devices into different networks/domains. This is solved by letting the user choose names for the IP ranges, which are used when deciding where to put the device. The device name is the name that is displayed in the inventory software manager's overview, because this name together with the network/domain name is used as a unique identifier it is important that the name is unique. When fetching the name of for example an HP switch one

will receive a name that is preset at the factory, therefore it will be the same on all HP switches of the same preferences. To avoid this, it is possible to use also the IP address in the device name. The third thing to consider, the value names, is set up in the value configuration. All the issues of importing the data to the inventory software's database has been considered and solved, therefore one can say that this goal is reached.

7. Integrate the SNMP application in the inventory software manager

   The SNMP administration shall be integrated with the inventory software manager user interface.

   Comment:

   The SNMP Inventory Configurator is integrated as a view in the inventory software manager. This goal is reached.

6.2 Restrictions

The testing has been performed only on small and middle sized networks. The solution should in practice work on very big networks too but that has not been confirmed.

6.3 Limitations

- The MIB parser cannot load files with multiple MIB modules.

  A MIB file can in rare cases contain more than one MIB module. The parser in the solution assumes that every MIB file holds one and only one MIB module. To rewrite the parser so that it supports multiple MIB modules in a single file may be something to work on in the future.

- All devices must share the same community strings.

  Today there is no way of setting up separate community strings for different networks. This is also something that can be done in future improvements. For example it would probably be a good idea to tie the community strings to the different IP ranges.

6.4 Future work

Besides from solving the issues discussed in section 6.2 above there is also some other things that may be addressed in the future in order to improve the system. One thing that could be simplified and improved for the users of the application is in my opinion the import value editor. That is the editor in which one sets the properties for the values, see figure 5.9. Maybe, as has been considered earlier in this document one could develop some kind of adapter to be able to make SQL-like queries to an SNMP agent.

Chapter 7

Acknowledgements

Some people deserves an extra thank you.

Mikael Rännar, for good help with the report
Stefan Johansson, for letting me do the work at the company.
and
Josefine Bjursell, for supporting me and for being a wonderful girlfriend.

Thanks!

References

[1] D. Bruey. Snmp: Simple? network management protocol. http://www.rane.com/note161.html (visited 2006-01-10).

[2] P. Cederberg. Mibble :: Mib parser. http://directory.fsf.org/network/tools /Mibble.html (visited 2006-01-02).

[3] A. Aror et al. Web services for management. http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-management.pdf (visited 2006-01-03).

[4] J. Case et al. Rfc 1067 - simple network management protocol. http://www.faqs.org/rfcs/rfc1067.html (visited 2006-01-02).

[5] J. Case et al. Rfc 1098 - simple network management protocol (snmp). http://www.faqs.org/rfcs/rfc1098.html (visited 2006-01-02).

[6] J. Case et al. Rfc 1157 - simple network management protocol (snmp). http://www.faqs.org/rfcs/rfc1157.html (visited 2006-01-02).

[7] J. Davin et al. Rfc 1028 - simple gateway monitoring protocol. http://www.faqs.org/rfcs/rfc1028.html (visited 2006-01-02).

[8] K. McCloghrie et al. Rfc 2578 - structure of management information version 2 (smiv2). http://www.faqs.org/rfcs/rfc2578.html (visited 2006-01-13).

[9] The Internet Corporation for Assigned Names and Numbers. IANA Home Page. http://www.iana.org/ (visited 2005-12-05).

[10] Albert Graef. Tp lex/yacc. http://www.musikwissenschaft.uni-mainz.de/ ag/tply/tply.html (visited 2005-12-07).

[11] M. Rose K. McCloghrie. Rfc 1213 - management information base for network management of tcp/ip-based internets:mib-ii. http://www.faqs.org/rfcs/rfc1213.html (visited 2005-12-07).

[12] D. R. Mauro and K. J. Schmidt. *Essential SNMP*. O´Reilly Media Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2001.

[13] P. Grillo S. Waldbusser. Rfc 2790 - host resources mib. http://www.faqs.org/rfcs/rfc2790.html (visited 2005-12-07).

[14] R. Spenc. Object oriented programming in delphi - a guide for beginners. http://www.webtechcorp.com/web-developer-training-delphi-article-oop.htm (visited 2006-01-12).

[15] DPS Telecom. Snmp tutorial. http://www.dpstele.com/layers/l2/snmp_l2_tut_part1.html (visited 2006-01-03).

[16] TrackBird. Trackbird s.n.m.p. http://www.trackbird.com/index.cfm/Solutions/Enterprise_Edition/S_N_M_P (visited 2006-01-02).

[17] Unknown. Asn.1 information site. http://asn1.elibel.tm.fr/ (visited 2006-01-11).

Appendix A

Packages

The following packages are realized in Delphi as units.

dsSNMPDevices

: A package containing classes used for storing inventory software specific information about SNMP manageable devices.

dsSNMPLib

: This package holds the extended SNMP manager specifically designed for use in the inventory software.

dsSNMPXML

: Contains a helper class for generating inventory software import files.

LexLib

: This package is a modified version of the one bundled with the GLYD 2.0 Lex/Yacc user interface.

pdASN1util

: A package that originates in Indy's component suite which comes bundled with Delphi. This version is modified to support bigger integer values.

pdMIBLib

: Holds classes used by the MIB parser to store parsed MIBs.

pdMIBParserLib

: Holds a class used for parsing MIB files.

pdMIBSyntax

: This package holds classes for storing information about object syntaxes found in MIB files.

pdParserLib

: This is a package that connects the output from Lex/Yacc to form a basis for the MIB parser. It also holds a class used for storing and loading parser states when temporarily interrupting a parse.

pdSNMPConstants

A package containing different kinds of pre-declarations used in many places in the application. It also defines some exceptions for global use.

pdSNMPDb

This package is used for communicating with a database that holds information about enterprise ids. An enterprise id is a unique number used for determining for example which company that has built a certain SNMP agent. Enterprise ids are assigned by the IANA organization, see [9].

pdSNMPDeviceList

Holds a class for storing a list of SNMP manageable devices.

pdSNMPDevices

This package holds classes for storing information about SNMP manageable devices.

pdSNMPLib

Holds an SNMP manager used for discovering and fetching data from SNMP manageable devices. Also holds a modified version of Indy's SNMP component that comes bundled with Delphi.

pdUDPLib

This package holds functionality for sending and receiving UDP datagrams on a UDP socket. The UDP socket is relying on WinSock2 which is a part of Microsoft Windows.

YaccLib

This package is a modified version of the one bundled with the GLYD 2.0 Lex/Yacc user interface.