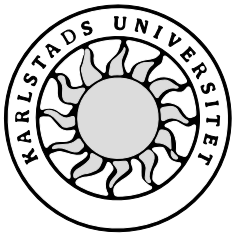


Datavetenskap

Jan Persson och Mathias Wagnsson

Filkonverteringsverktyg

NetViz till Count

Examensarbete, C-nivå

2004:04

Filkonverteringsverktyg

NetViz till Count

Jan Persson och Mathias Wagnsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Jan Persson

Mathias Wagnsson

Godkänd, 20040115

Handledare: Thijs Holleboom

Examinator: Stefan Lindskog

Sammanfattning

Detta dokument beskriver tillvägagångssättet för filkonvertering mellan netViz och Count med hjälp av ett fristående filkonverteringsverktyg som vi utvecklat. Verktyget är utvecklat på begäran av Ericsson Ungern och Tieto Enator i Karlstad. Filkonverteringsverktyget som fått namnet ”SG file converter” kommer vi referera till som ”konverteringsverktyget” eller ”vårt verktyg” för enkelhetens skull. NetViz och Count är två verktyg för att designa nätverk. NetViz har fördelen att det är lätt att designa sitt nätverk men nackdelen att det inte går att dimensionera nätverket. I Count kan man dock dimensionera ett befintligt nätverk, men designläget för att konstruera ett nätverk är svårarbetat. Tidigare har ett befintligt nätverk designats i netViz och skrivit in ”för hand” i ett Excelformulär hur nätverksstrukturen har sett ut, för att sedan importera denna fil i Count för att kunna dimensionera nätverket. Vi har med vårt verktyg gjort så att användaren slipper skriva in allt för hand i ett Excelformulär. Användaren kan nu designa sitt nätverk i netViz, konvertera designen med hjälp av vårt verktyg för att slutligen dimensionera det i Count. Hur allt detta går till, från design, vidare genom konvertering och hur slutresultatet ser ut kommer vi att beskriva närmare i detta dokument.

File converting tool

NetViz to Count

Abstract

This document describes the procedure of converting a network design between netViz and Count with the help of a standalone tool for file conversion which we have developed. The tool is developed on request by Ericsson Hungary and Tieto Enator in Karlstad Sweden.
The file conversion tool, which have been given the name ”SG file converter” will be referred to as ”konverteringsverktyget (the conversion tool)” or “vårt verktyg (our tool)” for the sake of simplicity. NetViz and Count are two tools for designing networks. NetViz has the advantage of an easy approach of designing networks but the disadvantage is that it can’t dimension the network. In Count one can dimension an existing network, but the design view to construct a network is a hard procedure. Earlier work with the dimension of a network included: designing the network in netViz, writing the information about how the network structure looks like “by hand” into an Excel form and finally importing the Excel form into Count for dimensioning. We have with our tool made it so that the user doesn’t have to write everything by hand into an Excel form. Now the user can design his or her network in netViz, convert the design with the aid of our tool and finally dimension it in Count. How all of this works, from the design, through the converter tool and how the end result will look like will be described in closer detail in this document.

Innehållsförteckning

1 Inledning 1

1.1 Bakgrund 1

1.2 Syfte 3

2 Detaljerad beskrivning 4

2.1 Beskrivning av netViz 4

2.2 Beskrivning av Count 7

2.3 Beskrivning av netViz exportfil 8

2.4 Beskrivning av Counts filformat 9

2.5 Beskrivning av vårt filkonverteringverktyg 10

2.5.1 Användargränssnitt
2.5.2 Konverteringsfunktionaliteten
2.5.3 Skriva och hämta innehåll till och från Excelfilen
2.5.4 Excelfilens innehåll efter parsning
2.5.5 Loggning
2.5.6 Nätverkstyper

2.6 Från netViz till Count 26

3 Problem och Summering 31

3.1 Problem 31

3.2 Summering 32

A Källkod 35

A.1 Huvudklassen ”Form1” 35

A.2 Parsningsklassen ”ParseNetViz” 40

A.3 Loggningsklassen “Logging” 51

A.4 Informationsfönstret “About” 54

Figurförteckning

Figur 1 Attribut till nätverkskomponent 5
Figur 2 Exempel på ikon med ett och fler attributvärden 5
Figur 3 Exempel på Counts Excelfil 10
Figur 4 Huvudfönster i verktyget 11
Figur 5 Aktivitetsdiagram för konverteringen 21
Figur 6 Inställningar för loggning 22
Figur 7 Aktivitetsdiagram över loggningsfunktionaliteten 23
Figur 8 netViz designläge 26
Figur 9 "Export Data" fönstret 27
Figur 10 Källfilsfönstret 28
Figur 11 Loggning till huvudfönstret 29
Figur 12 Counts importfunktion 29
Figur 13 Counts designläge 30

1 Inledning

1.1 Bakgrund

För att designa en nätverkstopologi finns ett verktyg som heter netViz, vad vi menar med nätverkstopologier och en närmare förklaring av vad netViz är går att läsa om längre fram i detta dokument. Vi kan dock redan nu säga att med nätverkstopologier menas att det innehåller nätverkskomponenter såsom noder och länkar mellan noder. NetViz används av företag över hela världen, däribland Ericsson och Tieto Enator. NetViz har dock en begränsning i att det inte kan dimensionera de nätverk som man designar. Med dimensionering menas att räkna ut kapacitet och länkkostnader i nätverket. Däremot har Ericsson Ungern utvecklat ett verktyg, kallat Count, för just detta. Count har dock begränsningen att designläget för nätverkstopologier är svårarbetat då designläget inte har ”drag and drop” funktionaliteten som netViz tillhandahåller (läs mer om detta i den detaljerade beskrivningen av netViz och Count). Vidare är submenyerna för att få fram formulären för att skapa noder och länkar svåra att hitta, samt att formulären för att sedan skapa noderna och länkarna är invecklade för den oinvigde. En lösning skulle då vara att först designa sitt nätverk i netViz för att sedan importera denna design i Count och slutligen dimensionera densamma. Tidigare har designen av nätverk gjorts i netViz, för att sedan skriva in de nätverkskomponenter som ingår i designen för hand i ett Excelblad tillsammans med dess attribut. Detta Excelblad läses sedan in i dimensioneringsprogrammet Count. Detta kan vara tidsödande och det är lätt att missa någon nätverkskomponent eller skriva attributen för nätverkskomponenterna fel. För att kunna importera designen skapad i netViz så behövdes ett konverteringsverktyg för detta, det blev slutligen vår uppgift att utveckla detta konverteringsverktyg som ligger till grund för denna uppsats.

Uppgiften utöver att skapa konverteringsverktyget, gick ut på att identifiera de nätverkskomponenter som ingår i Count och göra en s.k. palett i netViz som överensstämmer med de nätverkskomponenter som Count använder sig av. Denna palett i netViz är en grafisk representation av de nätverkskomponeter som användaren av netViz kan använda sig utav i sin nätverksdesign. Användaren av netViz kan då designa sitt nätverk m.h.a. paletten vi har

skapat, och använda sig av exportfunktionen som redan finns inbyggd i netViz. Exportfunktionen skapar en s.k. csv fil som är en textfil med kommaseparerade fält. Vi beskriver denna fil längre fram i detta dokument. Denna fil öppnas sedan i vårt verktyg som därefter översätter innehållet till Counts Excelfilformat vilket beskrivs senare under detaljerad beskrivning. Denna Excelfil kan då importeras i Count och således kan användaren dimensionera det nätverk som denne har designat i netViz. Vi skulle även i vårt verktyg inkludera funktionaliteten att konvertera Excelfilen tillbaka till netViz, om komplettering av designen var nödvändig. På grund av administrativa problem hann vi dock inte med detta.

Först valde vi att utveckla vårt verktyg i VBA i MS Access, då Ericsson tidigare utvecklat ett program kallat NDST i just Access. NDST används för att automatiskt generera nätverksdokumentation utifrån den nätverksdesign som skapats i netViz. Vi kan tyvärr inte säga så mycket om NDST då dokumentationen om vad det är för ett program är begränsad och ägs av Ericsson. Det enda vi kan säga är att användaren exporterar sitt nätverk som denne designat i netViz till en MS Accessdatabas, för att sedan använda NDST till att generera dokumentation över nätverket utifrån Accessdatabasen. Vår första idé var då att skapa en modul till NDST som även kunde exportera nätverksdesignen till Count. Detta opponerade sig dock Ericsson Ungern mot, då de ville ha ett fristående verktyg som skötte detta.

Vi valde därför att utveckla verktyget i VB .Net, då vi antog att läsa och skriva till Excelobjekt skulle vara enkelt eftersom Microsoft har utvecklat både VB .Net och Excel, samt att vi trodde det skulle vara lätt att utveckla ett grafiskt användargränssnitt i VB .Net. Fördelarna med ett grafiskt användargränssnitt tror vi beror på att:

- de flesta användare är vana med en Windows liknande miljö
- användaren slipper skriva in långa kommandon i ett konsolfönster
- det är smidigt att presentera information till användaren, om verktyget och loggning, i ett fönster

Vi har själva inte satt oss in i detalj hur varken netViz eller Count fungerar, mest på grund av att vi inte behövde kunna detta för att utveckla vårt verktyg. Den enda funktionalitet i netViz som vi använt oss av, är att rita upp och designa ett nätverk med noder och länkar mellan noderna. Vi har inte undersökt vad varje nod eller vad varje länk är för slags komponent, det vill säga att vi inte har undersökt funktionaliteten hos de olika nod- och länktyperna. Vi har

endast konstaterat att noder och länkar som bygger upp ett nätverk kan vara av olika fabrikat, ha olika namn och olika funktionalitet. Men innebörden av detta är inte relevant för oss i utvecklandet av vårt verktyg. Den enda funktionaliteten vi använt oss av i Count är att importera vår konverterade fil för att se att nätverksdesignen ser likadan ut i Count som den gjorde i netViz. Det innebär att alla noderna och länkarna som ritades i netViz skall vara med i Count och även att noderna och länkarna har samma namn, är av samma typ och ligger i samma vy (i samma bild) som de gjorde i netViz. Detta innebär alltså att vi ej bryr oss om vad de olika attributen som komponenterna har (noderna och länkarna) innebär i praktiken, utan endast att de ”hänger med” från netViz till Count och ej förloras i konverteringen. De nätverk vi har jobbat med för att försöka konvertera från netViz till Count har bestått av nätverk för data- och/eller teletrafik. De som är någorlunda insatta i t.ex. datanätverk vet att ett datanätverk kan bestå av noder såsom persondatorer, routrar, switchar, hubbar och servrar. Länkar kan då tex vara koaxialkablar eller fiberoptiska kablar och hela, eller delar av nätverket kan exempelvis vara paketkopplat eller kretskopplat[1]. Vi hoppas därmed att läsaren fått en förståelse för vad vi menar när vi talar om nätverkstopologier och designandet av dessa.

Även fast vi själva icke behövde veta något nämvärt hur varken netViz eller Count fungerar har vi dock inkluderat en mer detaljerad beskrivning av både netViz och Count senare under den detaljerade beskrivningen som finns av netViz och Count (kapitel 2.1 respektive 2.2) i vår uppsats.

1.2 Syfte

Syftet med verktyget är att underlätta arbetet vid design av nätverkstopologier och dimensioneringen av dessa. Genom att använda vårt verktyg slipper användaren en massa dubbelarbete genom att inskrivningen av nätverket denne ritat i netViz icke behöver skrivas in för hand i Counts importfil, d.v.s Counts Excelfilformat. Verktyget konverterar på ett snabbt sätt netViz designen till att överenstämma med Counts nätverksdesignupplägg.

2 Detaljerad beskrivning

Vi kommer här att närmare beskriva de delar som ligger till grund för vår lösning. I de två nästkommande underkapitlena kommer vi att beskriva de två verktyg som används av Ericsson Ungern och Tieto Enator, nämligen netViz och Count.

I kapitel 2.3 kommer vi beskriva vad den fil som netViz genererar vid dess exportfunktion innehåller och vilken struktur den har.

Kapitel 2.4 beskriver vad som Counts importfilformat behöver innehålla och hur den behöver vara strukturerad för att Count skall kunna importera den.

I kapitel 2.5 kommer vi att närmare beskriva vårt verktyg med information om hur vi löste filkonverteringen mellan netViz och Count. Detta kapitel är indelat i underkapitel som tar upp de delar som vårt verktyg innehåller som:

- användargränssnitt
- konverteringsfunktionaliteten
- skriva och hämta innehåll till och från Excelfilen
- excelfilens innehåll efter parsning
- loggning
- nätverkstyper

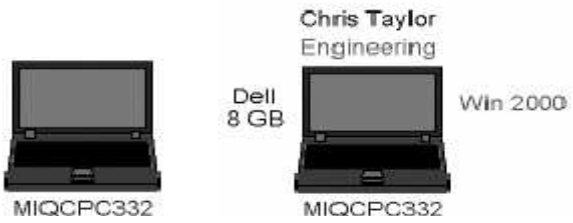
Vi kommer att avsluta detta kapitel med att visa ett exempel på förfarandet från att skapa en design i netViz, exportera designen till en csv-fil, konvertera denne med vårt verktyg, importera den genererade filen i Count och slutligen visa hur designen ser ut i Count.

2.1 Beskrivning av netViz

IT system som WAN och LAN, men även arbetsflödesmodeller och konceptuell design har alla en sak gemensamt; de är komplicerade samlingar av data och relationer. De kan vara svåra att förstå, men kanske ännu mera att behandla. NetViz är, enligt utvecklarna själva: ”*designad för att lätt kunna behandla stora mängder data. NetViz integrerar sömlöst information och grafik för att skapa en ”visuell databas”, som gör det lätt att se objekt*

tillsammans med deras unika karaktär och deras förhållande till ett annat väldigt, komplext system"[3]. När användaren skapar en nod eller en länk i netViz, kan denne välja vilka datafält eller attribut, som vi valt att kalla dem, som skall vara kopplade till noden eller länken (se Figur 1). Användaren kan i princip skapa hur många attribut som helst vilka innehåller sådan information som kan tyckas vara relevant för elementet (nod eller länk). För datautrustning, t.ex. en PC, så kan kanske attributfält som CPU-typ, tillverkare och hur mycket RAM som finns vara relevant. För en länk kanske användaren vill kunna lagra information om vilken relation mellan noder den representerar, vart länken tar vägen och vem som ansvarar för den.

Figur 1 Attribut till nätverkskomponent



Figur 2 Exempel på ikon med ett och fler attributvärden

Sedan kan användaren välja vilka attribut som skall visas då noden eller länken ritas upp genom att dra och släppa de attributfält från Figur 1 ovan till Figur 2 till vänster. I vänstra bilden i Figur 2 ser vi noden, i detta fall en representation av en laptop, enbart med dess namn. I högra bilden i Figur 2 har vi dragit och släppt några ytterligare attributfält som då lagts till. En samling elementtyper med fördefinierade attribut och med en ikon kallas i netViz för en palett. Denna palett kan användaren spara undan för att sedan kunna öppna och antingen använda i ett befintligt designprojekt eller ett nytt. Paletten representeras av ikonerna som är kopplade till elementen, och när användaren skall rita upp ett element tar denne helt enkel och drar och släpper elementet från paletten till ritytan. När vi skulle skapa vårt verktyg så fick vi av Ericsson Ungern, de nodtyper och länkar som de använder sig av när de designar nätverk i Count (mer om Count senare), samt vilka attribut dessa noder och länkar har. Tillsammans med en ikon i stil med laptopen i Figur 2 ovan, kunde vi då skapa en palett i netViz som överensstämmer med de noder och länkar som Count stödjer. NetViz stödjer även olika vyer av ett nätverk. T.ex. om användaren drar och släpper en nod, kallad ”MyNod1”, till ett tomt designfönster så hamnar den noden i den översta vyn, vilken kallas för ”Top Level”. Om

användaren sedan dubbelklickar på ”MyNod1” som denne har ritat upp, öppnas ett nytt tomt designfönster. Detta designfönster befinner sig under ”Top Level” och kallas för ”Top Level=>MyNod1”. En nod kan således representera en grupp noder med eventuellt tillhörande länkar som sammankopplar noderna. Detta är praktiskt om användaren designar t.ex. flera LAN som skall kopplas ihop. Då kan varje LAN representeras av en nod vardera och när användaren klickar på en av noderna så kommer en representation av det LAN:et fram med alla länkar och noder som LAN:et innehåller. På detta sätt går det att undvika en design som är svåröverskådlig och förbryllande. En ytterligare funktion i netViz som vi använt oss av i utvecklandet av vårt verktyg, är en av dess exportfunktioner. När användaren har designat sitt nätverk kan denne välja att exportera sin design till ett antal filformat. Det filformat som var intressant för oss var csv formatet, då detta kan representeras som en vanlig textfil. Hur denna textfil är uppbyggd berättar vi om senare i detta dokument.

Då vi själva inte är insatta i vilka ytterligare funktioner netViz tillhandahåller, p.g a. att detta inte var relevant för vår uppgift, utöver hur ett nätverk ritas upp, samt ger elementen attribut och hur nätverksdesignen sedan exporteras till en csv fil, kan vi enbart berätta vad de som utvecklat netViz själva har att säga om det[4].

NetViz är ett verktyg för att rita upp nätverkstopologier och netViz kan grafiskt visa den fysiska och logiska relationen mellan element i ett nätverk, system eller en process. NetViz kan även i sitt grafiska läge, koppla textinformation som beskriver de grafiska elementen, med detta menas att användaren kan sätta attribut och information som beskriver de grafiska objekten. NetViz kan användas på persondatorer med något av operativsystemen Windows 95/98/NT/2000.

NetViz kan användas till att rita ett flertal varianter av affärs och informationssystem såsom:

- data- och telekommunikationsnät
- affärsprocesser
- transport- och logistiksystem
- andra komplexa nätverk, system och processer av allehanda slag

Inom dessa ramar kan man med netViz:

- designa

- modellera
- analysera
- dokumentera
- behandla
- planera
- felsöka

NetViz är lätt att använda för att den är byggd på ”drag and drop” metoden. Tack vare att netViz har den metoden så behöver användaren av programmet inte någon speciell utbildning för att använda netViz, endast en viss förståelse för endera ovan nämnda fysiska och logiska relationer.

Vi kommer inte att gräva djupare i de övriga funktionaliteterna som netViz tillhandahåller och knapphändigt beskrivs av netViz utvecklarna ovan. Vi kommer däremot att koncentrera oss på att beskriva hur vårt verktyg fungerar. Vi kommer att beskriva i ett exempel hur processen går till från att designa ett väldigt enkelt nätverk i netViz, exportera designen till csv fil, konvertera det till Counts Excel format och slutligen hur slutanvändaren importerar designen i Count.

2.2 Beskrivning av Count

Count är utvecklat av Ericsson Ungern och är inte en kommersiell produkt. Vi är inte heller helt insatta i hur Count fungerar, då det ej låg på vår uppgift att sätta oss något djupare in i vilka funktionaliteter utöver nätverksdesign som finns i Count. Det vi kan, och får, berätta är att Count är, i likhet med netViz, ett verktyg för att rita upp och beskriva nätverkstopologier. Den stora skillnaden gentemot netViz, är att med Count så kan användaren dimensionera nätverket genom att räkna ut länkkostnader samt länkkapacitet. Själva designläget av nätverk anses dock som undermåligt varför det är aktuellt med en filkonvertering mellan netViz och Count. Några anledningar till varför det anses som undermåligt enligt Ericsson Ungern är att:

- count har ingen ”drag and drop” funktionalitet vid skapandet av nätverkskomponenter, det vill säga att det inte finns någon palett som det gör i netViz, där användaren kan ta en komponent från paletten och släppa den i designfönstret.

- vid skapandet av en koppling mellan noder i Count, får användaren först fylla i informationen om noderna i ett formulär för att sedan i ett annat formulär välja två noder, välja lämplig länktyp och slutligen trycka på en knapp för att generera kopplingen mellan noderna. Detta är ett omständigt och tidskrävande arbete.

Vi kommer inte att visa hur det går till att designa ett nätverk i Count, då detta skall ske automatiskt vid importen av den fil som genereras från vårt verktyg. Det är ett av skälen till att vi inte satt oss in i hur Count fungerar. Vi har inte intresserat oss i *hur* Count fungerar, utan förlitat oss på *att* det fungerar. Vi kommer inte heller att göra någon jämförelse av netViz och Count, då vi inte är tillräckligt insatta i någon av produkterna, och en jämförelse skulle därför kunna vara missvisande. Vi förlitar oss på kundens uppgifter om att designfunktionen är undermålig och ifrågasätter inte kundens integritet om varför kunden ville ha ett konverteringsverktyg.

Vi kommer däremot, som vi tidigare sagt, visa i ett exempel på hur nätverksdesignen ser ut i Count efter det att filen som genererats av vårt konverteringsverktyg har importerats i Count.

2.3 Beskrivning av netViz exportfil

NetViz har en exportfunktionalitet vilket genererar en csv-fil. Denna fil innehåller information om varje element i nätverkstopologin. Filen är uppbyggd på sådant sätt att elementen (noderna och länkarna) är ordnade efter typ. På första raden står attributnamnen på de olika attributen som elementet/elementen har, på raden under följer en lista med själva elementets attributvärden. Alla attribut oavsett namn eller värde är kommaseparerade och det är två tomrader mellan olika typer av element.

En beskrivning av en länk mellan två noder kan se ut se ut enligt följande i filen:

```
$type,$ancestry,$xy,$sn,$en,Name,capacity,mode
CircuitLink(CS),Top Level,"(2902,3222), (5070,4647)",APN-1,APN-2,
CircuitLink(CS)-1,-1,PCM
```

Den första raden, som börjar med tecknet ”$”, är en kommentarrad. Den beskriver de olika attributnamnen som ett visst nätverkselement har. De efterföljande två raderna innehåller själva informationen om de nätverkselement som innehåller de attributvärdena vilka är

kopplade till attributnamnen. Dessa attributnamn och attributvärden är de samma som elementet hade i designläget i netViz (minns Figur 1).
Exemplet ovan visar hur en ”Circuit link” skapad i netViz kan se ut i exportfilen.
De olika attributen för Circuit linken är:

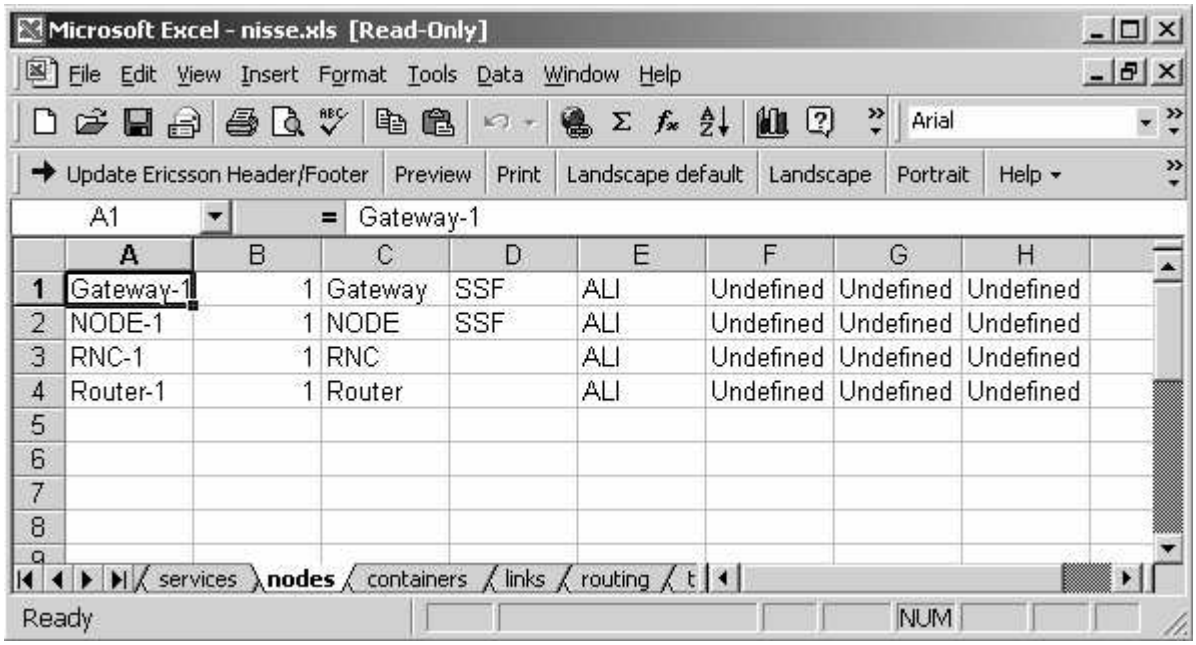
- $type – talar om vilken typ av länk det är, i detta fall en ”CircuitLink(CS)”. Notera att vi inte vet något om vad en ”CircuitLink(CS)” är för en typ av länk, utan alla typer som vi skall stödja, oavsett om det är en nodtyp eller en länktyp, har blivit definierade av Ericsson Ungern.
- $ancestry – anger i vilken vy som länken befinner sig i, denna länk befinner sig i högsta nivån av vyhierarkin, d.v.s. ”Top Level”.
- $xy – detta fält talar om på vilka koordinater som länken ritas upp på i netViz designfönster. De första koordinaterna talar om var den första noden som länken är kopplad till befinner sig, varpå de andra koordinaterna anger dess slutnod.
- $sn – står för namnet på startnoden, d.v.s. vilken nod som länken utgår ifrån. I detta fall heter noden APN-1.
- $en – står således för namnet på slutnoden där länken slutar (APN-2).
- Name – är namnet på länken, CircuitLink(CS)-1.
- capacity – beskriver kapaciteten på länken, -1 anger i detta fall att användaren inte har specificerat kapaciteten, så ett standardvärde har angivits som anger oändlig kapacitet.
- mode – anger på vilket sätt länken skall kommunicera, i detta fall PCM kodning.

Det är inte intressant att veta vad attributvärdena innebär för varken läsaren eller vid utvecklandet av verktyget. Detta beror på att vi kontrollerar i vårt konverteringsverktyg att attributvärdet, oavsett vad det är, skrivs in i Counts Excel fil. Vi struntar alltså i om t.ex. mode är satt till en giltig kodningsstandard som PCM eller inte. Du behöver alltså inte som läsare veta vad de olika attributvärdena innebär för att förstå denna uppsats, utan endast att ett attributnamn är kopplat till ett attributvärde.

2.4 Beskrivning av Counts filformat

Filformatet som Count använder sig av för att spara information om nätverkstopologier är i Excel kalkylbladsformat, alltså i xls format. Varje flik i kalkylbladet representerar olika delar av ett nätverk, men det som är intressant för oss är ”nodes”, ”containers”, ”links” och ”areas”. ”nodes” fliken innehåller en lista med alla noder i nätverksmodellen, ett exempel på hur det

kan se ut i ”nodes” fliken kan skådas i Figur 3 nedan. Vi kommer att förklara varför innehållet är vad det är senare i detta dokument under kapitel 2.5.4.



	A	B	C	D	E	F	G	H
1	Gateway-1	1	Gateway	SSF	ALI	Undefined	Undefined	Undefined
2	NODE-1	1	NODE	SSF	ALI	Undefined	Undefined	Undefined
3	RNC-1	1	RNC		ALI	Undefined	Undefined	Undefined
4	Router-1	1	Router		ALI	Undefined	Undefined	Undefined
5								
6								
7								
8								

Figur 3 Exempel på Counts Excelfil

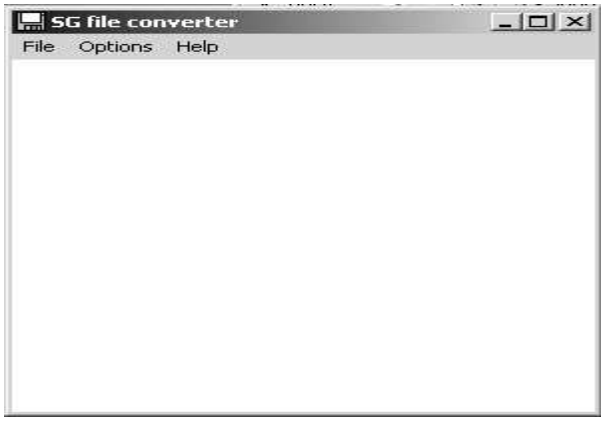
Under fliken ”containers” återfinns en beskrivning av hierarkin i nätverket och grupperingen av noderna. Fliken ”links” innehåller en lista av fysiska länktyper, samt både de logiska och fysiska länkarna i nätverket. Här listas även vilka noder en eller flera länkar är kopplade till. ”Areas”-fliken innehåller de olika områdena som kan undersökas på skärmen och motsvarar då de olika vyerna som finns representerade i netViz. Dessa vyer har vi beskrivit innan under beskrivningen av netViz.

2.5 Beskrivning av vårt filkonverteringverktyg

2.5.1 Användargränssnitt

Vi har skapat verktyget så att det skall vara lätt att använda, genom att göra användargränssnittet enkelt och med endast ett fåtal funktioner. Hur huvudfönstret ser ut, som visas när verktyget startas, visas i Figur 4 nedan. Användaren kan välja att konvertera en fil genom att välja File->Convert, då öppnas ett fönster för att först välja en källfil och sedan en målfil. När målfilen valts startar själva konverteringen och eventuella fel som stöds av loggningsfunktionaliteten rapporteras antingen i huvudfönstret, i en fil eller båda. Huruvida

detta rapporteras kan ställas in av användaren under Options->logging options. Under menyn Help kan man få information om programversion och en enkel användarmanual.



Figur 4 Huvudfönster i verktyget.

2.5.2 Konverteringsfunktionaliteten

Användaren får, som vi tidigare nämnt, välja källfilen som skall konverteras. Formatet undersöks då genom att undersöka filändelsen, så att inte själva konverteringen behöver sätta igång i onödan om filformatet är annat än csv. Därefter får användaren välja målfil, vilken måste vara i xls format. Om målfilen redan existerar så får användaren en förfrågan om huruvida filen skall skrivas över eller ej. Om användaren icke tillåter överskrivning av existerande fil, får användaren göra om valet av målfil. När målfilen har valts läser verktyget rad för rad från csv filen och ignorerar de rader som börjar med $, då dessa rader enbart är kommentarer. I de rader som behandlas, undersöks det första attributet, som antingen identifierar typen av en nod eller en länk. Typen jämförs med en lista som innehåller de typer som stöds av Count. Det finns två filer som innehåller en lista vardera, den ena innehåller typer av noder som stöds och den andra av länkar som stöds. Finns ej typen i någon av filerna, så rapporteras detta till loggen, samt att hela raden ignoreras. Om typen finns i någon av listorna läses hela raden in till en sträng, varpå denna skickas till matchande parsningsmetod. Nedanstående metoder kallade för ”worker”, ”isNode” och ”getComponentTypeName” visar hur det vi beskrivit ser ut kodmässigt. Vi visar inte hur ”isLink” fungerar då den är snarlik ”isNode” metoden. Notera att vi skapat ett parserobjekt som vi valt att kalla ”ps” vars metoder vi anropar i ”worker” metoden.

```
Private Sub worker()
```

```
Dim aString As String

While Not st.Peek() = -1
aString = st.ReadLine 'gets row in csv file

If Not aString.Equals("") Then 'looks if string is empty
 If aString.StartsWith("$") Then
  aString = st.ReadLine 'if string starts with $, then read next row in
                         file
 End If
 If ps.isNode(aString) Then 'checks if the content is a node
   ps.nodeParser(aString) 'calls parser for nodes
   logIt(ps.getText()) 'gets eventual generated errors/warnings from the
                        parser

 ElseIf ps.isLink(aString) Then
   ps.linkParser(aString) 'calls parser for links
   logIt(ps.getText())
 Else
  'if it's neither a supported node or link, logs that it's skipped!
  aString = ps.getComponentTypeName(aString)
  logIt("The program don't support " + aString)
 End If

End If

End While
ps.addLinkTypes() 'adds linktypes if there is transport links
ps.closeExcel() ' closes excel broker
ps.quitExcel() ' kills excel
End Sub
```

Som vi ser i ”worker” metoden så delegeras arbetet för att se om det är en nod eller länk till parserobjektet ”ps”. Följande metod undersöker om det är en nod eller ej:

```
Function isNode(ByVal theString As String)
 'checks if it's a supported node
 Dim st As String
 Dim i As Integer
 Dim test As Boolean = False

 st = getComponentTypeName(theString)
 While (i < nodeLength)
  If st.StartsWith(nodeVector(i)) Then
   test = True
  End If
  i = i + 1
 End While

 Return test
End Function
```

Det vi kan se här som kan vara lite förvirrande är att det finns en variabel kallad “nodeLength” och en vektor kallad ”nodeVector”. ”nodeLength” är global och korresponderar

till antalet noder som vi enligt Ericsson Ungern skulle ha stöd för i vårt filkonverteringsverktyg. ”nodeLength” är en globalt känd vektor som innehåller typnamnet på alla de noder som det finns stöd för. ”nodeVector” läser in varje rad från nodfilen och lägger typnamnet på en separat position i vektorn. Hur nodfilen är uppbyggd går att läsa om under ”nätverkstyper” under kapitel 2.5.6. ”nodeLength” sätts genom att värdet på den ökas med ett(1) varje gång en nodtyp läggs till i vektorn. Hur detta ser ut rent kodmässigt kan läsas i den bifogade källkoden.

```
Public Function getComponentTypeName(ByVal aString As String)
 'returns a substring left from the first occurence of ","
 Return Left(aString, aString.IndexOf(","))
End Function
```

”getComponentTypeName” är ganska enkel. Det enda den gör är att returnera nodtypen från strängen. Nodtypen står alltid först i strängen varpå metoden helt enkelt returnerar vad som står innan första förekomsten av ett kommatecken (minns att csv filens attributfält var kommaseparerade och att varje rad läses in till en sträng).

Vi har, som kan skådas i ”worker” metoden, delat upp så att det finns en parsningsmetod för noder och en för länkar kallade ”nodeParser” och ”linkParser”. De fungerar dock på liknande sätt, varför vi inte beskriver båda separat utan endast beskriver ”nodeParser” lite närmare. Parsningen fungerar på sådant sätt att strängen delas upp så att varje attributvärde läggs i en strängvektor. Sedan skrivs varje position i vektorn till korresponderande flik i Excelfilen. Parsern ser till att de noder som ligger i samma vy (area), skrivs på samma rad under vy-fliken med namnet på vyn först på raden. Därefter följer vilka länktyper som finns i vyn och slutligen namnet på noderna tillsammans med deras koordinater.

```
Sub nodeParser(ByVal st As String)
 Dim theString As String
 Dim myArray(12) As String
 Dim column As Integer = 0

 theString = st

 While theString.Length() > 0
  'gets the x-coordinates
  If column = 2 Then
   myArray(column) = Mid((Left(theString, theString.IndexOf(")"))), 3,
                     theString.Length - 3)
   theString = Right(theString, theString.Length()
               (theString.IndexOf(")") + 1))
   column = column + 1
```

```
   'gets the y-coordinates
  ElseIf column = 3 Then
   myArray(column) = Mid((Left(theString, theString.IndexOf(")"))), 4,
                     theString.Length - 4)
   theString = Right(theString, theString.Length()
               (theString.IndexOf(")") + 3))
   column = column + 1
   'gets last field
  ElseIf column = 12 Then
   myArray(column) = theString
   theString = ""
   column = column + 1
   'gets any other field
  Else
   myArray(column) = Left(theString, theString.IndexOf(","))
   theString = Right(theString, theString.Length()
               (theString.IndexOf(",") + 1))
   column = column + 1
  End If
 End While

 nodeRow = nodeRow + 1
 Call fillNodeExcel(myArray, nodeRow)
End Sub
```

Även i “nodeParser” används en global variabel kallad ”nodeRow”, denna variabel håller reda på hur många rader som skrivits till Excelbladet och denna variabel växer i takt med att en rad skrivs till Excelbladet. Detta görs p.g.a. att verktyget skall kunna hålla reda på vilken rad som är den nästa lediga raden som verktyget kan skriva till. I övrigt kan vi se i ”nodeParser” metoden att verktyget systematiskt hämtar attributvärde för attributvärde från strängen. Strängens innehåll kommer, som ni kanske minns, från csv-filen. Attributvärdena skrivs sedan, ett och ett, till Excel filen m.h.a. ”fillNodeExcel” metoden. Vi ser även att vid något vi kallat för ”column = 2” och ”column = 3”, har vi två specialfall. Dessa specialfall grundar sig på att ”column” refererar till vilket kommatecken verktyget befinner sig på i parsningen av strängen, samt att ”column = 2” och ”column = 3” innehåller koordinater som i sin tur är omslutna av paranteser. Dessa paranteser tas senare bort i ”fillNodeExcel” för att det skall vara möjligt att enbart kunna behandla koordinaternas värde. ”column = 12” är också det ett specialfall. Det innebär att om det finns ett tolfte kommatecken så är det ”Nothing” efter sista kommatecknet. ”Nothing” kan jämföras med ”void-pekare” för de som förstår sig på C och C++, man kan säga att ”Nothing” är ett värde som ej kan behandlas i VB .Net. På grund av detta måste vi explicit sätta attributvärdet till att det skall vara tomt för att undvika att komplikationer skall uppstå, såsom att programmet kraschar då det försöker behandla ett attributvärde som är ”Nothing”.

2.5.3 Skriva och hämta innehåll till och från Excelfilen

Följande utdrag från källkoden visar hur verktyget skriver en nods attributvärden till Excelfilen som användaren har skapat eller valt att skriva över:

```
Private Sub fillNodeExcel(ByVal myArray() As String, ByVal rows As Integer)
 Dim firstXCoord As String
 Dim firstYCoord As String
 Dim secondXCoord As String
 Dim secondYCoord As String
 Dim xCoord As String
 Dim yCoord As String
 Dim temp As String
 Dim i As Integer = 0
 Dim j As Integer = 0

 toLabel = "" 'truncates the log string
 firstXCoord = Left(myArray(2), myArray(2).IndexOf(","))
 firstYCoord = Right(myArray(2), myArray(2).Length()
               (myArray(2).IndexOf(",") + 1))
 secondXCoord = Left(myArray(3), myArray(3).IndexOf(","))
 secondYCoord = Right(myArray(3), myArray(3).Length()
               (myArray(3).IndexOf(",") + 1))
 'translate coordinates to Counts coordinate format
 xCoord = (secondXCoord - ((secondXCoord - firstXCoord) / 2)) * RESOLUTION
 yCoord = (secondYCoord - ((secondYCoord - firstYCoord) / 2)) * RESOLUTION
 'delete spaces
 If myArray(1).IndexOf(" ") > -1 Then
  myArray(1) = Replace(myArray(1), " ", "")
 End If
 temp = myArray(6)
 If myArray(6).IndexOf(" ") > -1 Then
  myArray(6) = Replace(myArray(6), " ", "")
 End If
 'delete special characters
 If myArray(6).IndexOf("/") > -1 Then
  myArray(6) = Replace(myArray(6), "/", "")
 End If
 If myArray(6).IndexOf("(") > -1 Then
  myArray(6) = Replace(myArray(6), "(", "")
 End If
 If myArray(6).IndexOf(")") > -1 Then
  myArray(6) = Replace(myArray(6), ")", "")
 End If
 'write eventual character changes to log string
 If Not temp.Equals(myArray(6)) Then
  toLabel = toLabel + "changed " + temp + " to " + myArray(6).ToString &
            vbNewLine
 End If
 'write node fields to excel (first time)
 If (getExcel("nodes", 1, 1) Is Nothing) Then
  Call SetExcel("nodes", 3, rows, myArray(0)) 'Type
  Call SetExcel("nodes", 1, rows, myArray(6)) 'Name
  Call SetExcel("nodes", 2, rows, myArray(8)) 'Level

  If Not myArray(7) = "" Then
   Call SetExcel("nodes", 4, rows, myArray(7)) 'Functionality
  End If
```

```
  Call SetExcel("nodes", 5, rows, myArray(9)) 'Transport capabilty
  Call SetExcel("nodes", 6, rows, myArray(10)) 'HW platform
  Call SetExcel("nodes", 7, rows, myArray(11)) 'HW type
  Call SetExcel("nodes", 8, rows, myArray(12)) 'SW release
  'write node fields to first empty row in excel
 Else
  Dim k As Integer = 1
  Dim nodeTest As Boolean = False
  While Not (getExcel("nodes", 1, k) Is Nothing)
   If getExcel("nodes", 1, k) = myArray(6) Then
    nodeTest = True
    Exit While
   End If
   k = k + 1
  End While
  If Not nodeTest Then
   Call SetExcel("nodes", 3, k, myArray(0)) 'Type
   Call SetExcel("nodes", 1, k, myArray(6)) 'Name
   Call SetExcel("nodes", 2, k, myArray(8)) 'Level
   If Not myArray(7) = "" Then
    Call SetExcel("nodes", 4, k, myArray(7))
   End If
   Call SetExcel("nodes", 5, k, myArray(9))
   Call SetExcel("nodes", 6, k, myArray(10))
   Call SetExcel("nodes", 7, k, myArray(11))
   Call SetExcel("nodes", 8, k, myArray(12))
  End If
 End If
 myArray(1) = Replace(myArray(1), "=>", "__") 'darn special character in
 area identifier

 'Writes level to areas sheet
 While 1        'Writes name and coordinats
  If getExcel("areas", 1, getAreaLevel(myArray(1)) + i) = "" Then
   Call SetExcel("areas", 1, getAreaLevel(myArray(1)) + i, myArray(1))
   Call SetExcel("areas", 3, getAreaLevel(myArray(1)) + i, myArray(6) _
       + " " + xCoord + " " + yCoord)
   Exit While
  ElseIf getExcel("areas", 1, getAreaLevel(myArray(1)) + i) =myArray(1)Then
   While 1
    If getExcel("areas", 3 + j, getAreaLevel(myArray(1)) + i) = "" Then
     Call SetExcel("areas", 3 + j, getAreaLevel(myArray(1)) + i, myArray(6)
         + " " + xCoord + " " + yCoord)
     Exit While
    End If
   j = j + 1
   End While
  Exit While
 End If
 i = i + 1
 End While
End Sub
```

Metoden ”fillNodeExcel” börjar med att tömma den globala strängen (”toLabel”) som skall innehålla information om fel och varningar som kan uppstå vid parsningen, t.ex. om noden innehåller tecken som Count inte klarar av och att dessa har bytts ut. När verktyget är klar med att ta bort tecken som Count inte klarar av så utförs en ny beräkning genom att netViz

koordinatsystem räknas om så att designen ser likadan ut i Count. Där använder vi oss av en konstant som heter ”RESOULUTION” för att ändra värdet på koordinaterna så de överensstämmer med Counts koordinatsystem. Denna konstant har i skrivandets stund värdet 5 då vi tyckte att genom att ha det värdet, överensstämde designen av nätverken med varandra både i netViz och Count. Sedan tar verktyget bort mellanslag, paranteser och specialtecken om dessa påträffas i nodens namn. Om det förekommer mellanslag, paranteser eller specialtecken så loggas detta genom att ändringen skrivs till ”toLabel”.

Verktyget fyller i Excelbladet genom att först anropa ”getExcel” metoden, som beskrivs nedan, för att undersöka vilken plats i angiven flik som är ledig. Därefter anropas ”setExcel” med parametrar som anger flik, kolumn och rad samt vad som skall införas i Excelcellen. Det sista metoden gör är att skriva in nodens namn och koordinater i ”area” fliken. Detta görs genom att först undersöka om noden redan finns i det ”area” den tillhör. Då Count inte klarar av nodnamn som förekommer mer än en gång i samma ”area”, så måste denna kontroll utföras så att dubbletter inte införs.

```
Private Function getExcel(ByVal xlWorksheet As String, _
 ByVal xlCol As Integer, ByVal xlRow As Integer) As String
 'gets cell contents in a specific sheet on column xlCol, row xlRow
 Dim strCellContents As String
 ' Get the Cell Contents
 oApp.Sheets(xlWorksheet).Select()
 strCellContents = oApp.ActiveCell(xlRow, xlCol).Value
 getExcel = strCellContents
End Function
```

”getExcel” metoden returnerar innehållet i en cell. Innehållet fås genom att värdet på parametern ”xlWorksheet” anger vilken flik i kalkylbladet som värdet finns i, ”xlCol” och ”xlRow” anger i vilken cell värdet finns. Här finns även ett objekt kallat ”oApp”, detta objekt refererar till Excelfilen och dess metoder är fördefinierade av VB .Net. Genom ”oApp” objektet kan vi manipulera innehållet i Excelfilen som skapades av vårt verktyg och där har vi en fördel och anledning till att vi valde just VB .Net som vår utvecklingsmiljö.

```
Private Sub setExcel(ByVal xlWorksheet As String, _
 ByVal xlCol As Integer, ByVal xlRow As Integer, _
 ByVal xlCellContents As String)
 'sets cell contents in a specific sheet on column xlCol, row xlRow
 'if string is empty/nothing, set "Undefined"
 If (xlCellContents Is Nothing) Then
  xlCellContents = "Undefined"
 ElseIf xlCellContents.Length() = 0 Then
  xlCellContents = "Undefined"
 End If
```

```
 oApp.Sheets(xlWorksheet).Select()
 oApp.ActiveCell(xlRow, xlCol) = xlCellContents
End Sub
```

"setExcel" metoden fungerar snarlikt som "getExcel". Den stora skillnaden ligger i att "setExcel" sätter ett värde i en cell i stället för att hämta värdet som man gjorde i "getExcel". Vi kan även se att innehållet i cellen sätts till "undefined" då inget värde finns i strängen som inkommer till metoden. Detta beror på att Count ej godtar tomma celler i vissa fall. Varför Count ej godtar tomma celler får utvecklarna av Count svara för och vi godtar att sådant är fallet med Count.

2.5.4 Excelfilens innehåll efter parsning

Om vi nu tittar på vad som skrivs i "area"-fliken i Excelfilen, så ser vi att det korresponderar till "vy" i netViz. I följande exempel så har den översta vyn "Top Level" i netViz, kallats för "TopLevel" i Counts "area". Subnätet till "Top Level" har kallats för "TopLevel_NODE-1" vilket korresponderar till "Top Level=>NODE-1" i netViz.

TopLevel	CS	Gateway-1 8480 7208	NODE-1 12818 11632
TopLevel_NODE-1	PS	RNC-1 15538 11250	Router-1 9077 13758

Konverteringsverktyget avlägsnar alla mellanslag och specialtecken såsom "=" och ">", då Count ej klarar av dessa tecken i varken nodnamn eller areanamn. "CS" och "PS" anger vilka typer av länkar som ingår i vyn, eller "area" som Count kallar den. "CS", som ni kanske känner igen, ingick ju i typnamnet för en CirciuitLink(CS) och "PS" står här för en länk av typen "PacketLink(PS)". En "area" kan innehålla flera länktyper och inte bara "CS" eller "PS" som i vårt exempel ovan. Om flera länktyper finns i samma "area" anges dessa separerade med ett mellanslag. Efter fältet för länktyp följer en lista på de noder "area" innehåller. I exemplet ovan innehåller "TopLevel" endast två noder, nämligen "Gateway-1" och "NODE-1". I "NODE-1" kan vi notera även det är ett subnät med noder och därför fått ett eget "area" namn("TopLevel_NODE-1"). Efter namnet på noderna följer två nummer, t.ex. "8480 7208" som står efter "Gateway-1". Dessa nummer motsvarar x och y koordinater för vart dessa noder skall ritas upp i Counts designläge.

Vi skall även i nästkommande exempel visa hur det ser ut under ”nodes” fliken och ”links” fliken med de noder och länkar som finns representerade ovan i ”area”.

Under nodfliken (nodes), skrivs i varje kolumn:

- namnet på noden
- vilken nivå i protokollstacken som noden befinner sig på (1-3). Vilken nivå siffrorna ”1-3” representerar på protokollstacken och vad en protokollstack är, är inte intressant för vårt konverteringsverktyg. Vi konstaterar endast att det finns något som heter protokollstack och att den har flera nivåer, samt att läsaren inte heller skall behöva veta vad en protokollstack är för att förstå uppsatsen
- vilken typ av nod det är
- vilken funktionalitet den har
- vilken transportförmåga noden har
- hårdvaruplattform
- hårdvarutyp
- mjukvaruutgåva

Ett exempel på hur det kan se ut i nodfliken efter att ett antal noder har lagts till (notera att varje nod har lagts till på en separat rad):

Gateway-1	1	Gateway	SSF	ALI	Undefined	Undefined	Undefined
NODE-1	1	NODE	SSF	ALI	Undefined	Undefined	Undefined
RNC-1	1	RNC		ALI	Undefined	Undefined	Undefined
Router-1	1	Router		ALI	Undefined	Undefined	Undefined

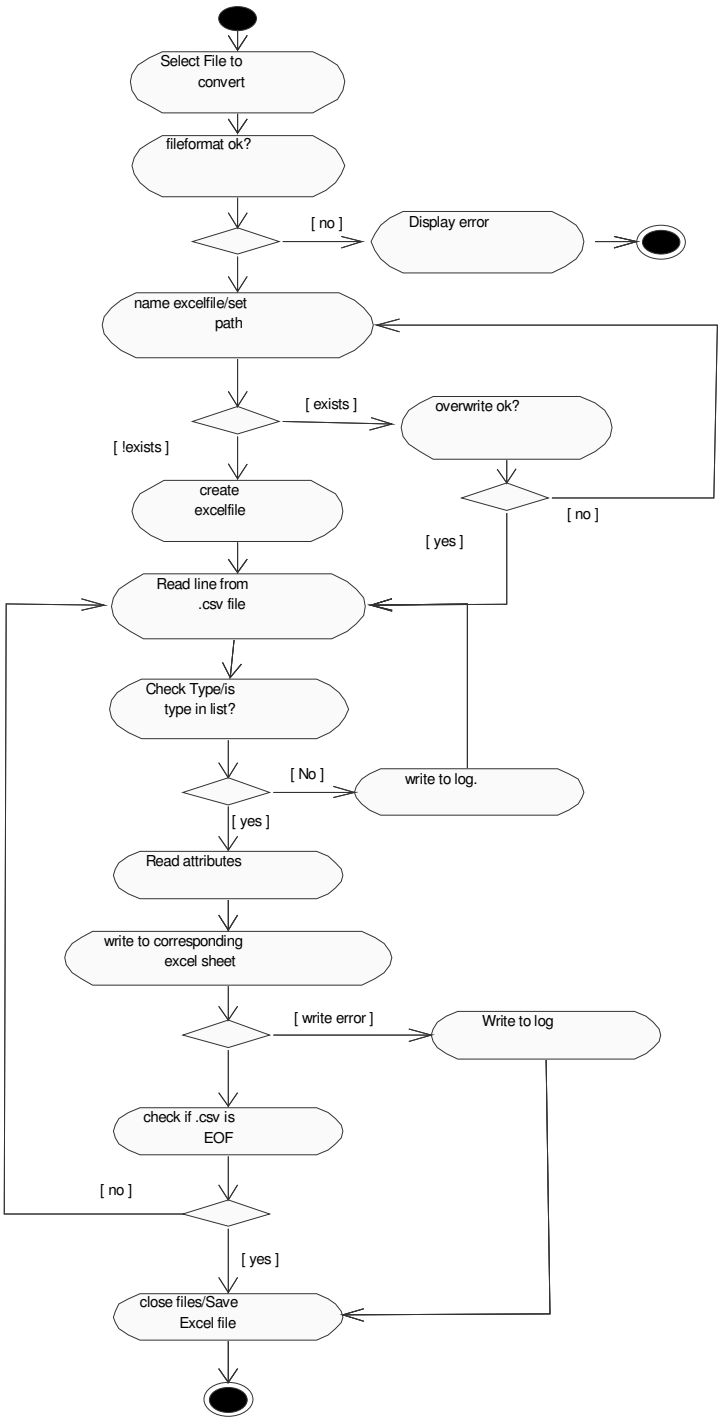
Under länkfliken (links) skrivs i varje kolumn:

- Länktyp
- Namnet på startnoden
- Namnet på slutnoden
- Kapacitet
- Mode (endast om länken är av typen CS eller Transport)

Innehållet i fliken kan då se ut enligt följande:

CS	Gateway-1	NODE-1	-1	PCM
PS	Router-1	RNC-1		

Om det av någon anledning skulle inträffa att det inte går att skriva till Excelfilen avbryts parsningen, då det är onödigt att fortsätta försöka att skriva till filen och verktyget återvänder till startläge, d.v.s. huvudfönstret. Parsningen fortsätter tills det inte finns några fler rader att läsa från csv filen, varpå verktyget återvänder till startläget. Verktyget avslutas då användaren antingen trycker på kryssymbolen i övre högra hörnet, eller väljer File->Exit.
På nästa sida följer ett aktivitetsdiagram som visar ovanstående nämnda moment (Figur 5).



Figur 5 Aktivitetsdiagram för konvertering.

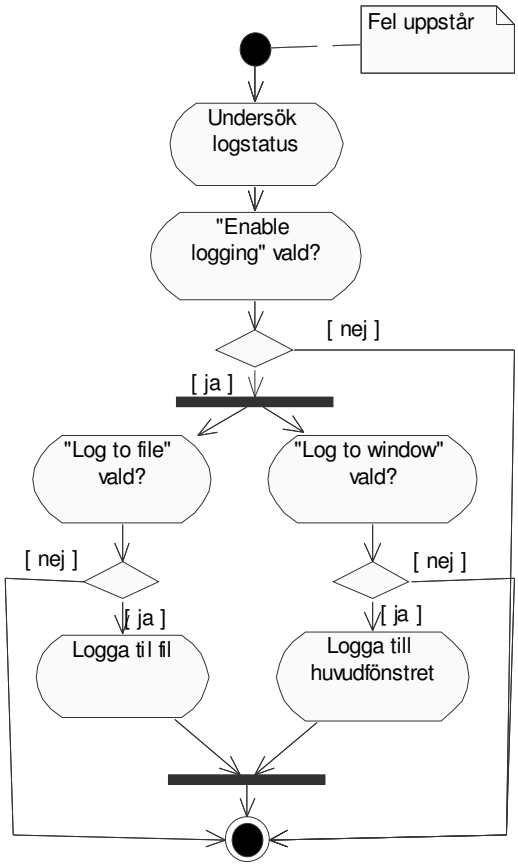
2.5.5 Loggning

Programmet har under menyvalet ”options” ett alternativ som heter ”logging”. När detta alternativ valts öppnas ett fönster där användaren kan välja tre olika alternativ att logga eventuella fel som kan uppstå. Fönstret är uppbyggt på följande sätt: tre stycken ”checkboxes” med valen ”enable logging”, ”log to window” respektive ”log to file” (Se Figur 6 nedan).
I startläget kommer den första och den andra vara förbockade (”enable logging”, ”log to window”). Men användaren kan även välja att logga till en fil som kommer att placera sig i programmets körkatalog med namnet log.txt. Om användaren har valt ”enable logging” så måste antingen ”log to window” eller ”log to file” vara vald, detta ser verktyget dock till genom att välja ”log to window” så fort användaren valt ”enable logging”. Om användaren väljer att bocka av ”log o window” kommer ”log to file” att automatiskt bli förbockad. Samma procedur gäller om ”log to file” bockas av.

Figur 6 Inställningar för loggning

De fel som verktyget rapporterar är de fel som eventuellt uppstår vid filmanipulation och när parsningsmetoderna finner nätverkselement som inte stöds av verktyget. Filmanipulationsfel upptäcks genom att vi lagt all filmanipulation inom s.k. ”try-catch satser”[4]. Try-satsen kastar ett undantag om ett fel uppstår vilket catch-satsen sedan tar hand om. Våra catch-satser identifierar vilken typ av undantag som kastats och skickar informationen om undantaget, omvandlat till en sträng, vidare till vår loggningsfunktion. Denna loggningsfunktion undersöker hur användaren vill att felet skall loggas och skickar därefter strängen med

informationen om felet vidare till lämplig presentationsfunktion (fil och/eller huvudfönstret). Upptäckten av att ett nätverkselement ej stöds av verktyget går till på följande sätt:
När en rad från csv filen läses in undersöks om nätverkselementets typattribut finns med antingen i en textfil där stödda nodtyper finns, eller i en textfil där stödda länktyper finns. Finns typen ej listad i någon av filerna skickas ett felmeddelande till loggningsfunktionen, hur detta ser ut rent kodmässigt går att läsa om i den bifogade källkoden. Hur loggningsfunktionen avgör hur felet skall presenteras visas nedan i Figur 7 och hur textfilerna är uppbyggda går att läsa om i nästa delkapitel.



Figur 7 Aktivitetsdiagram över loggningsfunktionaliteten

Ett utdrag från källkoden hur detta går till ser ut enligt följande:

```
Public Sub logIt(ByVal errorString As String)
 Dim state(3) As Integer
 state = log.getLogState() 'gets what kind of logging will take place

 If (state(2) = 1) Then
  logToFile(errorString) 'logs to file if it's chosen
```

```
End If
 If (state(1) = 1) Then
logToWindow(errorString) 'logs to window if it's chosen
End If

 End Sub
```

I loggningsobjektet ”log” finns en metod kallad ”getLogState” som hämtar en vektor innehållande information om vilken eller vilka ”checkboxes” som användaren kryssat för i loggningsfönstret (Figur 6). Vektorn är uppbyggd så att varje element i vektorn korresponderar till en ”checkbox”. Det första elementet refererar till den översta ”checkboxen”, det andra elementet till den mellersta ”checkboxen” och slutligen det sista elementet till den nedersta ”checkboxen”. Vektorn är deklarerad globalt i loggningsobjektet, och ser ut så här när man startar verktyget:

```
Private logState() As Integer = {1, 1, 0}
```

På så sätt kan vi hålla reda på vilket sätt vi skall logga som vi ser i ”logIt” metoden ovan.
Är Ni intresserade av hur loggningen fungerar ytterligare hänvisar vi Er till bifogad källkod.

2.5.6 Nätverkstyper

Vårt verktyg är gjort så att användaren kan lägga till mer nätverkselement efter hand som Count utvecklas, men då måste användaren först göra ett nytt nätverkselement i netViz som har samma uppbyggnad som de andra i den existerande Countpaletten vi tillverkat för netViz. Därefter får användaren lägga till typen av nätverkselementet antingen i filen där noder som stöds finns, om denne skapade en nodtyp, eller i filen för länkar, om användaren nu skapade en ny länktyp.
Filerna är uppbyggda på så sätt att det finns ett element per rad med namnet på typen. Dessa filer används, som vi tidigare nämnt, för att identifiera om Count stödjer de nätverkselement som skall översättas från netViz. Vi vet dock inte vad typnamnen på komponenterna innebär och vad de står för, dessa namn är som sagt något vi endast fått tilldelade genom Ericsson Ungern. Vad varje nod och länk har för funktionalitet ignorerar vi fullständigt då det varken var intressant för utvecklandet av vårt verktyg, eller fanns tid över att sätta sig in vad alla typer innebar.
Filerna med nätverkstyper som stöds, ser i skrivandets stund ut enligt följande:

Fil med nodtyper som stöds (nodes.txt):

NODE

PSTN

Gateway

BSC

RNC

MSC

MSCServer

MGW

SGSN

GGSN

CGSN

SGRX

GGRX

CGRX

APN

BG

HLR

EIR

AUC

FNR

SCP

IPSDP

SMS_C

MMS_C

STP

EXT_NW

ATM

FR

SDH

Router

IPSubnet

Fil med länktyper som stöds (links.txt):

CircuitLink

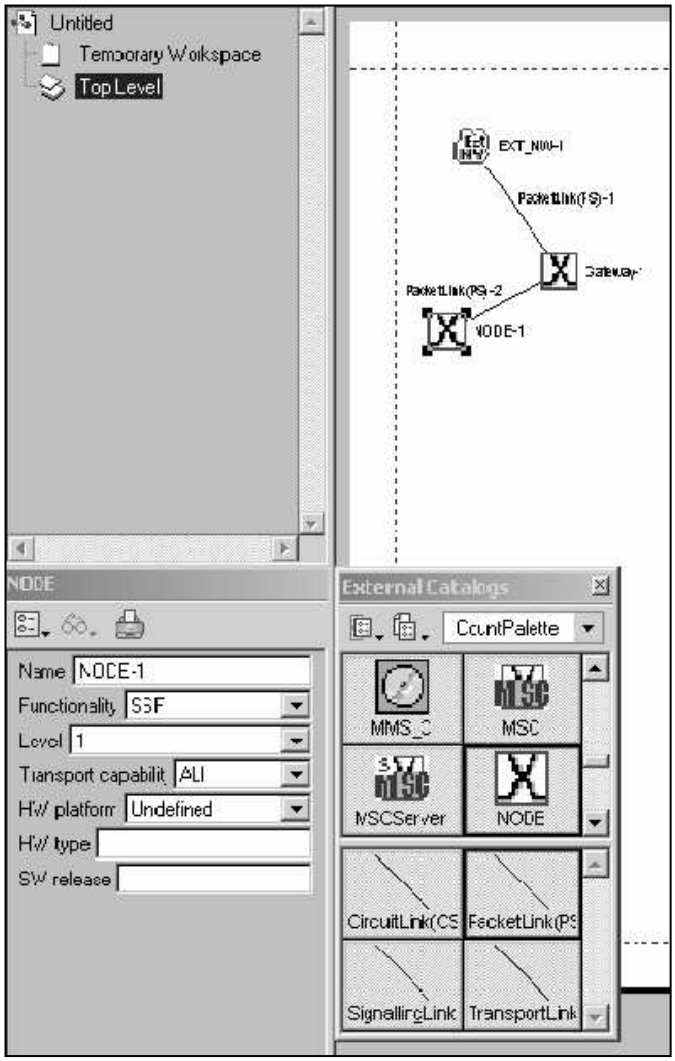
PacketLink

SignallingLink

TransportLink

2.6 Från netViz till Count

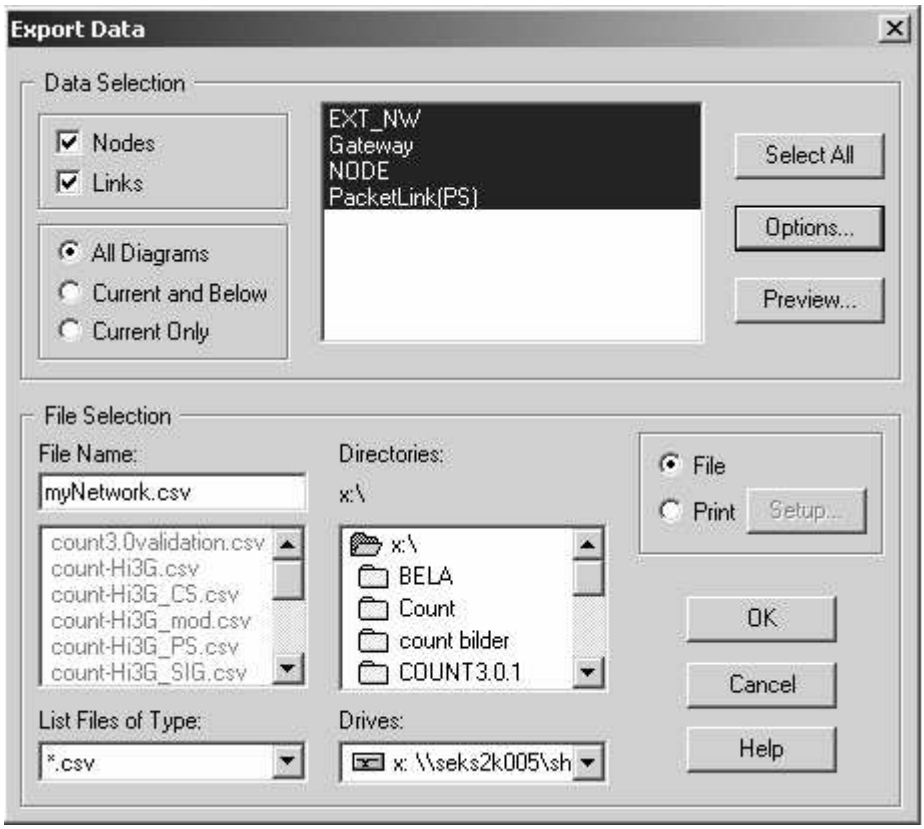
Här följer en kort beskrivning hur det går till att designa ett mindre nätverk i netViz, hur vårt verktyg konverterar designen och slutligen hur designen ser ut i Count efter konverteringen. Vi kommer alltså kort att beskriva hur det ser ut för användaren när denne använder sig av vårt konverteringsverktyg:



Figur 8 netViz designläge

Vi har här i figur 8 startat netViz och designat ett nätverk bestående av tre noder och två länkar. Noderna heter EXT_NW-1, Gateway-1 och NODE-1, samt länkarna heter PacketLink(PS)-1 respektive PacketLink(PS)-2. Dessa har vi skapat genom att dra deras ikoner från paletten (”External Catalogs” fönstret) och släppa dem i designfönstret. Fönstret längst upp till vänster visar vilken nivå eller ”vy” vi arbetar i. I detta fall befinner vi oss i den översta nivån av nodhierarkin, d.v.s. ”Top Level”. I fönstret nedanför visas attributnamnen och attributvärden för den nod eller länk användaren markerat, i detta fall har vi markerat noden ”NODE-1”. Därefter vill vi konvertera denna

designen så att vi kan visa den i Count. Detta gör vi genom att välja netViz exportfunktion i dess verktygsmeny. Vi väljer sedan att konvertera designen till en textfil (csv fil). Då visas nedanstående fönster (Figur 9), där vi får välja de nätverkskomponenter som skall konverteras samt ett filnamn. Vi har valt samtliga komponenter och döpt filen till ”myNetwork.csv”.



Figur 9 ”Export Data” fönstret.

Efter det att vi tryckt ”OK” i ”Export Data” fönstret (Figur 9), så skapas själva csv filen som då ser ut enligt följande:

```
$type,$ancestry,$xy,$container,$contained,Name,Functionality,Level,Transpor
t capability,HW platform,HW type,SW release
EXT_NW,Top Level,"(1302,1280), (1782,1760)",,,EXT_NW-1,,1,ALI,Undefined,,
```

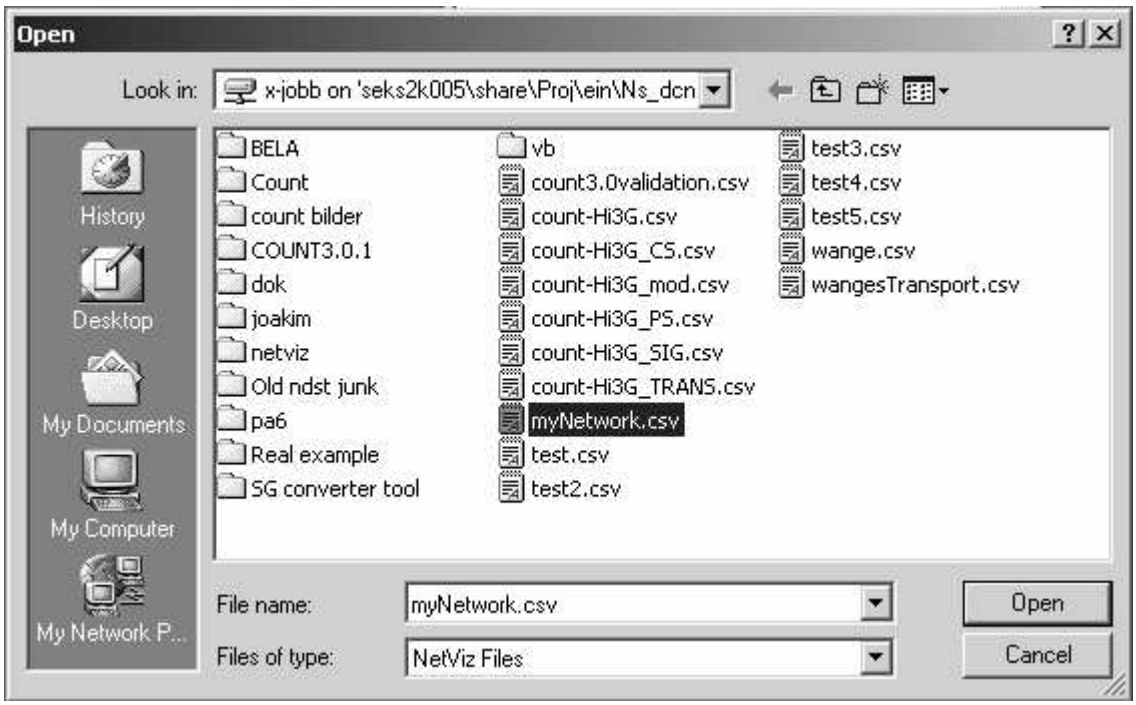
```
$type,$ancestry,$xy,$container,$contained,Name,Functionality,Level,Transpor
t capability,HW platform,HW type,SW release
Gateway,Top Level,"(2407,2833), (2887,3313)",,,Gateway-
1,SSF,1,ALI,Undefined,,
```

```
$type,$ancestry,$xy,$container,$contained,Name,Functionality,Level,Transpor
t capability,HW platform,HW type,SW release
NODE,Top Level,"(982,3578), (1462,4058)",,,NODE-1,SSF,1,ALI,Undefined,,
```

```
$type,$ancestry,$xy,$sn,$en,Name,capacity
PacketLink(PS),Top Level,"(1542,1520), (2647,3073)",Top Level=>EXT_NW-1,Top
Level=>Gateway-1,PacketLink(PS)-1,
PacketLink(PS),Top Level,"(2647,3073), (1222,3818)",Top Level=>Gateway-
1,Top Level=>NODE-1,PacketLink(PS)-2,
```

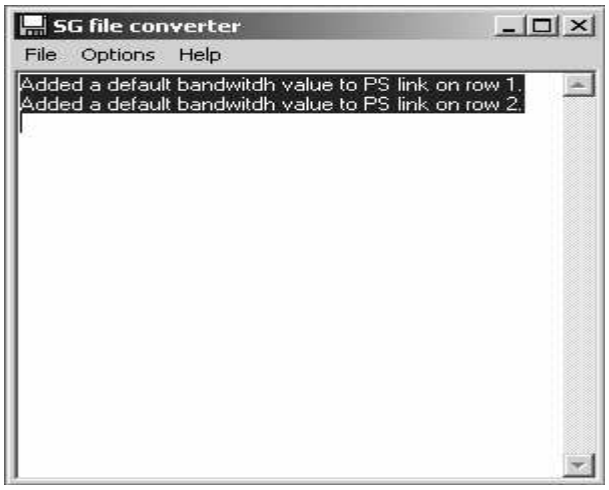
Denna fil finns närmare beskriven under sektion 2.3 i detta dokument. Det vi kan se i filen är att varje nodtyp och länktyp står separerade från varandra med två raders mellanrum vilket gör det lättare att kunna skilja på de olika typerna.

Efter exporteringen är det dags för vårt verktyg att konvertera filen till Counts Excelfilformat. Detta gör vi genom att välja ”File->Convert File” i menyn i vårt verktyg. Där får användaren först välja källfil (Figur 10) för att sedan välja målfil.



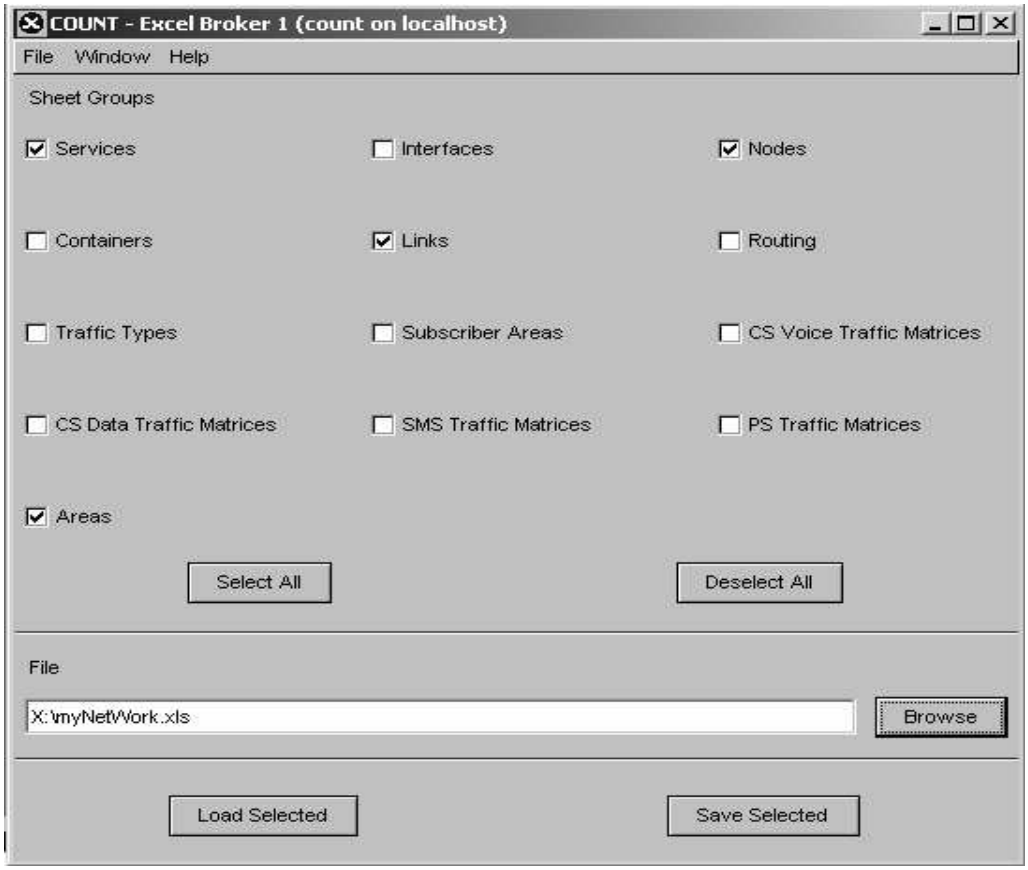
Figur 10 Källfilsfönstret

I nästa figur (Figur 11), ser vi hur verktyget rapporterar två varningar som säger att den har lagt till ett standardvärde på länkarnas bandbredd. Detta beror på att vi inte skrev in något attributvärde till attributet ”capacity” när vi skapade länkarna. Count kräver att det finns ett värde till detta attribut och vårt verktyg sätter standardvärden på de ställen där det behövs, det vill säga när användaren inte angivit något attributvärde.



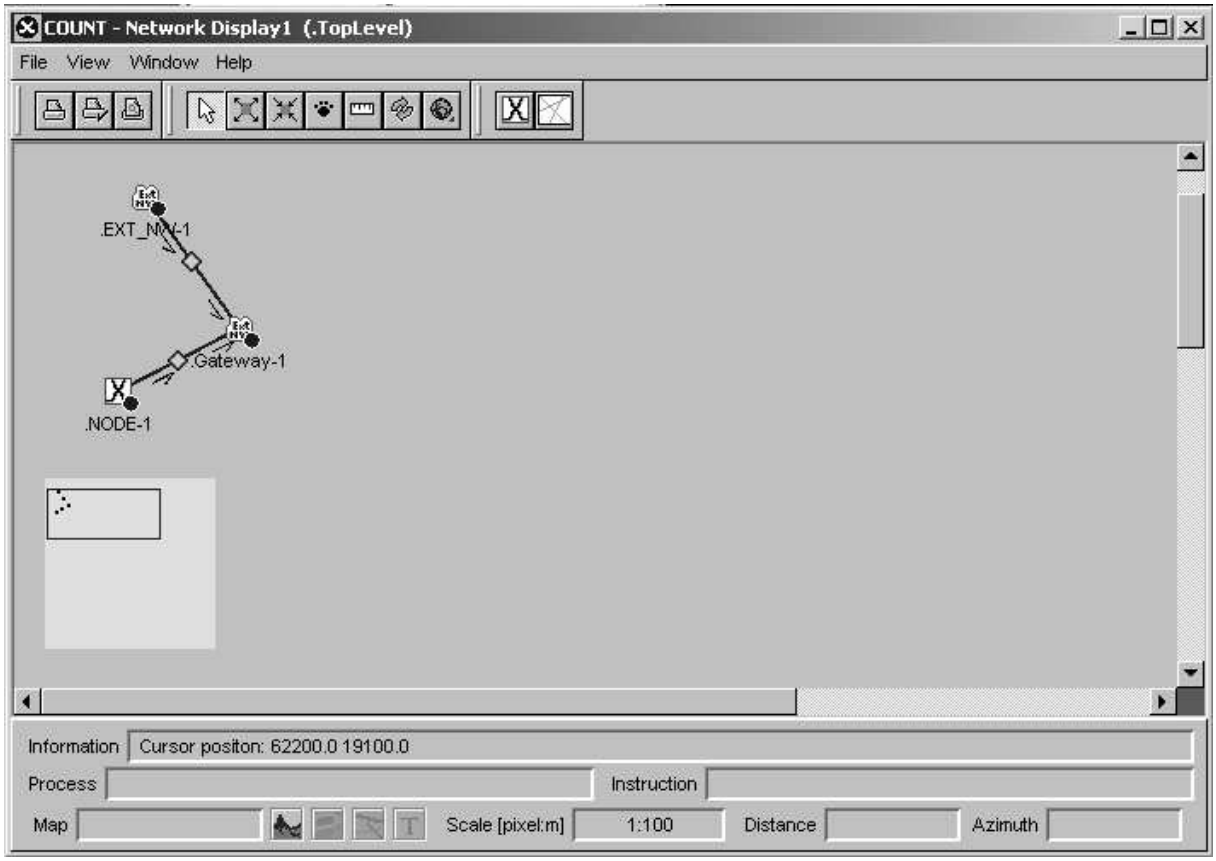
Figur 11 Loggning till huvudfönstret

Därefter återstår endast att importera filen som skapades efter att verktyget konverterat csv filen. Figur 12 visar hur Counts importfunktion ser ut. Där får man välja vilka flikar i excelfilen som skall importeras. I konverteringen mellan netViz och Count behöver endast de flikar som är markerade i Figur 12 importeras.



Figur 12 Counts importfunktion

Efter det att vi tryckt på “Load Selected” knappen i nedre vänstra hörnet, skapas nätverksdesignen och vi kan byta fönster till Counts designläge för att se hur nätverket ser ut. Som vi ser i Figur 13 så överensstämmer designen i Count adekvat med designen som vi gjorde i netViz.



Figur 13 Counts designläge

Som vi ser överensstämmer även ikonerna med de ikoner som fanns i vår palett i netViz, vilket beror på att vi fick ikonerna från Ericsson Ungern då vi skulle skapa paletten. Detta förhöjer intrycket av att det är samma design som visas i Count från den användaren gjorde i netViz.

3 Problem och Summering

I kapitel 3.1 så beskriver vi de problem som uppstod under arbetets gång, hur några av dessa löstes, varför några av problemen uppstod och hur dessa problem påverkade vårt arbetsflöde.

I kapitel 3.2 har vi gjort en summering på vår lösning av filkonverteringsproblemet. Vi har även tagit upp förslag på vad som skulle kunna förbättras med vårt verktyg och vi har också nämnt vad som varit positivt och negativt med arbetsprocessen vid utvecklandet av vårt verktyg.

3.1 Problem

Ett problem som gjorde att vi hamnade två dagar efter i planerningen, var att när vi skulle manipulera Excelobjekt fick följande felmeddelande vid exekvering: ”Error: 0x80028018 (-2147647512) Description: Old Format or Invalid Type Library”. Först trodde vi att de kommandon vi använde oss av för att manipulera Excelobjekten var utdaterade. Därför sökte vi högt och lågt på internet, och framförallt Googles nyhetsgrupper efter uppdateringar. Felet låg dock inte i att kommandona var utdaterade, utan det berodde på en bugg i Windows 2000[2]. Buggen grundar sig på att om användaren använder sig av en engelsk version av Microsoft Excel, samtidigt som operativsystemet är inställt för ett annat språk, så kommer Excel att försöka leta upp språkpaketet för det konfigurerade språket. Om språkpaketet icke hittas rapporteras ovanstående fel. För att kringgå detta problem, kan användaren installera korresponderande språkpaket på användardatorn. Detta tycker vi inte att användaren av vårt verktyg skall behöva bry sig om, så vi löste det genom att ändra språket på omgivningen för Excel i vår kod till ”US-English” när vi manipulerar Excelobjekt, för att sedan ändra tillbaka till den ursprungliga omgivningen när vi är klara med manipuleringen.

Ett annat problem som sände oss cirka en vecka tillbaka i planeringen, var att vi först tänkt att utveckla vårt verktyg som en modul i NDST genom att använda VBA i MS Access. Men eftersom Ericsson Ungern ville ha ett fristående verktyg utan koppling till NDST, så var vi tvungna att tänka om och börja på ny kula igen.

Då vi sedan valde att använda VB .Net innebar detta att vi var tvungna att försöka sätta oss in i ett nytt programmeringsspråk. Detta var dock ej något stort problem då vi genom Karlstads Universitet fick tillgång till litteratur om just VB .Net programmering. Inläsningen av VB .Net programmering tog ändå tyvärr upp ungefär två dagar av vår planering då det ändå är en del att läsa in om just detta.

Det största problemet som tog längst tid att åtgärda var de administritiva problem vi upplevde. De administritiva problemen var:

- att få bli lokala administratörer på våra datorer vi använde för att utveckla verktyget, det vill säga att få installera de program som vi var tvungna att använda oss utav för att kunna utveckla vårt verktyg
- att få tillgång till större hårddiskar så att de program vi var tvungna att ha, fick plats och således kunde installeras
- att överhuvudtaget få tillgång till datorer och någon plats att arbeta på
- att få passerkort så att vi kunde komma in på vår arbetsplats

Dessa administrativa problem gjorde att vi hamnade hela tre veckor efter i planeringen. Varför dessa administrativa problem uppstod, övergår vårt förstånd och kan ej förklaras utav oss. Vi har inte heller krävt någon närmare förklaring av vår uppdragsgivare då vi anser att detta kan uppfattas som stötande.

3.2 Summering

Problemet var att i de två verktyg som Ericsson Ungern och Tieto Enator använder sig av så innehåller båda verktygen brister. I det ena verktyget kallat netViz, kan en användare designa ett nätverk på ett snabbt och enkelt sätt, men netViz saknar funktionen för att dimensionera nätverket som har designats. Det andra verktyget som har namnet Count, är undermåligt i designläget men kan dimensionera existerande nätverk. Lösningen vi gjort var att skapa ett fristående verktyg som kan konvertera designen användaren skapat i netViz till att se likadan ut i Count, så att denne sedan kan dimensionera nätverket i Count.

Verktyget vi skapat kan genomföra konverteringen genom att läsa in information om nätverksdesignen från en exportfil som netViz skapat. Denna fil innehåller information om alla delarna som ingår i ett nätverk och importeras i vårt verktyg. Filen är en textfil i ett så kallat ”csv-format”. Det innebär att varje nätverkskomponent står på en separat rad och informationen om nätverkskomponentens attribut är separerade med kommatecken.

Vårt verktyg behandlar csv-filen på sådant sätt att den översätter alla delar i nätverket till att överensstämma med hur Counts importfilformat är uppbyggd. Denna fil är i Excelfilformat (xls) där information om alla nätverkskomponenter tillsammans med deras attribut finns lagrade. Varje nätverkskomponent konverteras genom att rad för rad läses i csv-filen och skrivs in i korresponderande ark tillsammans med dess attribut, till exempel nodarket ”nodes” om nätverkskomponenten är av typen nod.

Vi använde oss av VB .Net för att utveckla vårt verktyg, då vi från början hade färdig kod som var skriven i VBA till NDST. NDST är ett verktyg som redan existerar och vi ansåg att det vore smidigt att tillverka en modul till just NDST för att sköta konverteringen mellan netViz och Count. Då kunden istället ville ha ett separat verktyg för konverteringsprocessen, så kunde vi ta den kod vi skrivit till NDST i VBA för att använda i VB .Net då dessa programmeringsspråk är snarlika. Vi ansåg även att det skulle vara lätt att manipulera Excelobjekt då det fanns inbyggt stöd för detta i VB .Net.

Förbättringar som går att göra i vårt verktyg skulle bland annat kunna innefatta att vårt verktyg skulle kunna konvertera från Count till netViz. Detta skulle vara bra om användaren skulle vilja komplettera nätverksdesignen. Koden skulle även kunna göras mer strukturerad genom att ange kommentarer före varje metod som beskriver vad som krävs för att anropa metoden och vad metoden eventuellt returnerar. Även fler fördefinierade konstanter bör skapas så att den som läser koden inte blir förvirrad av alla siffror som förekommer.

Det som varit positivt vid utvecklingen av vårt verktyg var:

- vi har fått en djupare insikt i hur det går till ute i arbetslivet
- vi har fått lära oss ett nytt programspråk i VB .Net
- vi har fått mer programmeringsvana

Det enda vi upplevt som negativt under utvecklingsprocessen var att vid större företag tar administrativa uppgifter oftast lång tid.

Oavsett om det har funnits positiva och negativa moment i arbetsprocessen har det ändå varit nyttiga erfarenheter som vi kommer att ta med oss i vår fortsatta karriär.

Referenser

[1] James F. Kurose and Keith W. Ross. *Computer Networking*. Addison Wesley, 2nd edition, 2003

[2] http://support.microsoft.com/default.aspx?scid=kb;en-us;320369&Product=vbNET

[3] http://www.netviz.com

[4] NetViz Corporation. *User's guide.* Document No. NV-6.0, 2002

[5] John Cornell. *Coding techniques for Microsoft visual basic .net.* Microsoft Press, 2002

A Källkod

A.1 Huvudklassen "Form1"

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Const FILE_EXIST = 1
    Const FILE_NOT_EXIST = 0
    Private ps As ParseNetViz
    Private st As System.IO.StreamReader
    Private log As New Logging
    Private initFile As Integer = FILE_NOT_EXIST

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        System.GC.Collect()
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Public components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
```

```
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents MainMenu1 As System.Windows.Forms.MainMenu
    Friend WithEvents MenuItem1 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem2 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem3 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem4 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem5 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem6 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem7 As System.Windows.Forms.MenuItem
    Friend WithEvents MenuItem8 As System.Windows.Forms.MenuItem
    Friend WithEvents OpenFileDialog1 As
System.Windows.Forms.OpenFileDialog
    Friend WithEvents SaveFileDialog1 As
System.Windows.Forms.SaveFileDialog
    Friend WithEvents Label1 As System.Windows.Forms.TextBox
    <System.Diagnostics.DebuggerStepThrough()> Public Sub
InitializeComponent()
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(Form1))
        Me.MainMenu1 = New System.Windows.Forms.MainMenu
        Me.MenuItem1 = New System.Windows.Forms.MenuItem
        Me.MenuItem2 = New System.Windows.Forms.MenuItem
        Me.MenuItem3 = New System.Windows.Forms.MenuItem
        Me.MenuItem4 = New System.Windows.Forms.MenuItem
        Me.MenuItem5 = New System.Windows.Forms.MenuItem
        Me.MenuItem6 = New System.Windows.Forms.MenuItem
        Me.MenuItem7 = New System.Windows.Forms.MenuItem
        Me.MenuItem8 = New System.Windows.Forms.MenuItem
        Me.OpenFileDialog1 = New System.Windows.Forms.OpenFileDialog
        Me.SaveFileDialog1 = New System.Windows.Forms.SaveFileDialog
        Me.Label1 = New System.Windows.Forms.TextBox
        Me.SuspendLayout()
        '
        'MainMenu1
        '
        Me.MainMenu1.MenuItems.AddRange(New System.Windows.Forms.MenuItem()
{Me.MenuItem1, Me.MenuItem4, Me.MenuItem6})
        '
        'MenuItem1
        '
        Me.MenuItem1.Index = 0
        Me.MenuItem1.MenuItems.AddRange(New System.Windows.Forms.MenuItem()
{Me.MenuItem2, Me.MenuItem3})
        Me.MenuItem1.Text = "File"
        '
        'MenuItem2
        '
        Me.MenuItem2.Index = 0
        Me.MenuItem2.Text = "Convert file"
        '
        'MenuItem3
        '
        Me.MenuItem3.Index = 1
        Me.MenuItem3.Text = "Exit"
        '
        'MenuItem4
        '
        Me.MenuItem4.Index = 1
        Me.MenuItem4.MenuItems.AddRange(New System.Windows.Forms.MenuItem()
{Me.MenuItem5})
```

```
        Me.MenuItem4.Text = "Options"
        '
        'MenuItem5
        '
        Me.MenuItem5.Index = 0
        Me.MenuItem5.Text = "Logging "
        '
        'MenuItem6
        '
        Me.MenuItem6.Index = 2
        Me.MenuItem6.MenuItems.AddRange(New System.Windows.Forms.MenuItem()
{Me.MenuItem7, Me.MenuItem8})
        Me.MenuItem6.Text = "Help"
        '
        'MenuItem7
        '
        Me.MenuItem7.Index = 0
        Me.MenuItem7.Text = "About"
        '
        'MenuItem8
        '
        Me.MenuItem8.Index = 1
        Me.MenuItem8.Text = "How To"
        '
        'OpenFileDialog1
        '
        Me.OpenFileDialog1.Filter = "NetViz Files|*.csv|Count Files|*.xls"
        '
        'SaveFileDialog1
        '
        Me.SaveFileDialog1.CreatePrompt = True
        Me.SaveFileDialog1.DefaultExt = "xls"
        Me.SaveFileDialog1.Filter = "Count Files|*.xls"
        '
        'Label1
        '
        Me.Label1.Dock = System.Windows.Forms.DockStyle.Fill
        Me.Label1.Location = New System.Drawing.Point(0, 0)
        Me.Label1.Multiline = True
        Me.Label1.Name = "Label1"
        Me.Label1.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
        Me.Label1.Size = New System.Drawing.Size(292, 273)
        Me.Label1.TabIndex = 0
        Me.Label1.Text = ""
        '
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.Add(Me.Label1)
        Me.Icon = CType(resources.GetObject("$this.Icon"),
System.Drawing.Icon)
        Me.Menu = Me.MainMenu1
        Me.Name = "Form1"
        Me.Text = "SG file converter"
        Me.ResumeLayout(False)

    End Sub

#End Region
```

```
    Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem2.Click
        'If convert is selected, then open OpenFileDialog1
        OpenFileDialog1.ShowDialog()
    End Sub

    Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem3.Click
        'Exit program
        Me.Close()
    End Sub

    Private Sub OpenFileDialog1_FileOk(ByVal sender As System.Object, ByVal
e As System.ComponentModel.CancelEventArgs) Handles OpenFileDialog1.FileOk

        Dim sFileName As String = OpenFileDialog1.FileName

        'Getting around the "excel bug" in windows2000
        'Set enviroment to Us-English
        Dim oldCI As System.Globalization.CultureInfo = _
           System.Threading.Thread.CurrentThread.CurrentCulture
        System.Threading.Thread.CurrentThread.CurrentCulture = _
           New System.Globalization.CultureInfo("en-US")

        'ps is parser object(class1.vb)
        ps = New ParseNetViz

        'try to open selected file and show save window if success
        Try
            st = System.IO.File.OpenText(sFileName)
            SaveFileDialog1.ShowDialog() 'Open save dialog window
            st.Close()
        Catch ex As Exception
            logIt("could not open " + sFileName)

        End Try
        'Restoring from the "excel bug" in windows2000
        'set enviroment to the users language
        System.Threading.Thread.CurrentThread.CurrentCulture = oldCI

    End Sub

    Private Sub MenuItem7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem7.Click
        'About info window
        Dim ab As New About
        ab.Show()
    End Sub

    Private Sub SaveFileDialog1_FileOk(ByVal sender As System.Object, ByVal
e As System.ComponentModel.CancelEventArgs) Handles SaveFileDialog1.FileOk
        'Save dialog window
        Dim sFileName As String = SaveFileDialog1.FileName
        Dim myfile As IO.File

        'check if file already exists, tries to delete it if file is found
        'this is done instead of trying to truncate the existing file
        Try
            If myfile.Exists(sFileName) Then
                myfile.Delete(sFileName)
            End If
```

```
            ps.createExcel(sFileName) 'creates excel file
            ps.readNodeFile() 'reads node file
            ps.readLinkFile() 'reads link file
            worker() 'worker does the job! :)
        Catch ex As Exception
            logIt(ex.Message.ToString())
            ps.quitExcel() 'kills excel
        End Try
    End Sub
    Private Sub worker()
        Dim aString As String

        While Not st.Peek() = -1
            aString = st.ReadLine 'gets row in csv file
            If Not aString.Equals("") Then 'looks if string is empty
                If aString.StartsWith("$") Then
                    aString = st.ReadLine 'if string starts with $, then
read next row in file
                End If
                If ps.isNode(aString) Then 'checks if the content is a node
                    ps.nodeParser(aString) 'calls parser for nodes
                    logIt(ps.getText()) 'gets eventual generated
errors/warnings from the parser
                ElseIf ps.isLink(aString) Then
                    ps.linkParser(aString) 'calls parser for links
                    logIt(ps.getText())
                Else
                    'if it's neither a supported node or link, logs that
it's skipped!
                    aString = ps.getComponentTypeName(aString)
                    logIt("The program don't support " + aString)
                End If

            End If

        End While
        ps.addLinkTypes() 'adds linktypes if there is transport links
        ps.closeExcel() ' closes excel broker
        ps.quitExcel() ' kills excel
    End Sub
    Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem5.Click
        'log window
        log.Show()

    End Sub

    Public Sub logIt(ByVal errorString As String)
        Dim state(3) As Integer
        state = log.getLogState() 'gets what kind of logging will take
place

        If (state(2) = 1) Then
            logToFile(errorString) 'logs to file if it's chosen
        End If
        If (state(1) = 1) Then
            logToWindow(errorString) 'logs to window if it's chosen
        End If

    End Sub
```

```
    Private Sub logToFile(ByVal errorString As String)

        Dim path As String =
System.Reflection.Assembly.GetExecutingAssembly.Location()
        path = ps.getPath(path)
        Try

            If initFile = FILE_NOT_EXIST Then
                Dim logFile As IO.StreamWriter
                logFile = IO.File.CreateText(path + "log.txt")
                logFile.Write(errorString)
                logFile.Close()
                initFile = FILE_EXIST
            Else
                Dim fs As New IO.FileStream(path + "log.txt",
IO.FileMode.Append)
                Dim logFile As New IO.StreamWriter(fs)
                logFile.Write(errorString)
                logFile.Close()
            End If

        Catch ex As Exception
            logToWindow("Couldn't create logfile" & vbNewLine)
            logToWindow(errorString)
        End Try

    End Sub

    Private Sub logToWindow(ByVal errorString As String)

        Label1.Text = Label1.Text + errorString
    End Sub

End Class
```

A.2 Parsningsklassen ”ParseNetViz”

```
Imports System.Globalization
Imports System.IO
Public Class ParseNetViz
    Const STARTLEVEL = 1 'corresponds to "top level" in netViz
    Const RESOLUTION = 5 'multiplyer value to scale the coordinates between
netViz->Count
    'globals
    Private toLabel As String = "" 'string to be logged
    Private oApp As New Excel.Application
    Private linkRow As Integer 'what row next link will be written in excel
    Private nodeRow As Integer
    Private serviceRow As Integer = 1
    Private nodeVector() As String 'holds supported nodes
    Private linkVector() As String 'holds supported links
    Private nodeLength As Integer 'states how many nodes that are supported
    Private linkLength As Integer 'states how many links that are supported
    Private transportVector As New Queue 'holds transport link name and
speed alrady parsed
    Private serviceVector As New Queue 'holds transport protocol name
already parsed
```

```
    'transport link struct
    Structure transStruct
        Dim name As String 'name of transport link
        Dim speed As String 'the speed of the link
    End Structure
    Sub New()
        linkRow = 0
    End Sub
    Public Function getPath(ByVal sThePathAndName As String) As String
        ' Return a string containing the file's path from a fully qualified
file name.
        ' Return "" if no path
        Dim i As Integer                    'used in for/next loops
        Dim sTemp As String
        sTemp = Trim$(sThePathAndName)      'trim it
        If InStr(sTemp, "\") = 0 Then       'any backslash?
            Exit Function
        End If
        For i = Len(sTemp) To 1 Step -1     'find the right most one
            If Mid$(sTemp, i, 1) = "\" Then
                getPath = Mid$(sTemp, 1, i) 'now have just path
                Exit Function
            End If
        Next
    End Function

    Sub readNodeFile()
        'reads the supported node file (nodes.txt)
        Dim file As IO.StreamReader
        Dim counter As Integer = 0
        Dim path As String =
System.Reflection.Assembly.GetExecutingAssembly.Location()

        path = getPath(path)
        file = System.IO.File.OpenText(path + "nodes.txt")
        While Not file.Peek() = -1
            file.ReadLine()
            nodeLength = nodeLength + 1
        End While
        nodeLength = nodeLength - 1
        ReDim nodeVector(nodeLength)
        file.Close()
        file = System.IO.File.OpenText(path + "nodes.txt")
        While Not file.Peek() = -1
            nodeVector(counter) = file.ReadLine()
            counter = counter + 1
        End While
    End Sub

    Sub readLinkFile()
        'reads the supported link file (links.txt)
        Dim file As IO.StreamReader
        Dim counter As Integer = 0
        Dim path As String =
System.Reflection.Assembly.GetExecutingAssembly.Location()

        path = getPath(path)
        file = System.IO.File.OpenText(path + "links.txt")
        While Not file.Peek() = -1
            file.ReadLine()
```

```
        linkLength = linkLength + 1
    End While
    linkLength = linkLength
    ReDim linkVector(linkLength)
    file.Close()
    file = System.IO.File.OpenText(path + "links.txt")
    While Not file.Peek() = -1
        linkVector(counter) = file.ReadLine()
        counter = counter + 1
    End While
End Sub

Function isNode(ByVal theString As String)
    'checks if it's a supported node
    Dim st As String
    Dim i As Integer
    Dim test As Boolean = False

    st = getComponentTypeName(theString)
    While (i < nodeLength)
        If st.StartsWith(nodeVector(i)) Then
            test = True
        End If
        i = i + 1
    End While

    Return test
End Function

Function isLink(ByVal theString As String)
    'checks if it's a supported link
    Dim st As String
    Dim i As Integer = 0
    Dim test As Boolean = False

    st = getComponentTypeName(theString)
    While (i < linkLength)
        If st.StartsWith(linkVector(i)) Then
            test = True
        End If
        i = i + 1
    End While

    Return test
End Function

Sub linkParser(ByVal st As String)
    'parse the link
    Dim theString As String
    Dim myArray(9) As String
    Dim column As Integer = 0
    Dim columnCounter As Integer

    theString = st
    'sets the columnCounter to match corresponding link
    If theString.StartsWith("CircuitLink(CS)") Then
        columnCounter = 8
    ElseIf theString.StartsWith("TransportLink") Then
        columnCounter = 9
    Else
        columnCounter = 7
```

```
        End If


        While theString.Length() > 0
            'gets the x-coordinates
            If column = 2 Then
                myArray(column) = Mid((Left(theString,
theString.IndexOf(")"))), 3, theString.Length - 3)
                theString = Right(theString, theString.Length() -
(theString.IndexOf(")") + 1))
                column = column + 1
                'gets the y-coordinates
            ElseIf column = 3 Then
                myArray(column) = Mid((Left(theString,
theString.IndexOf(")"))), 4, theString.Length - 4)
                theString = Right(theString, theString.Length() -
(theString.IndexOf(")") + 3))
                column = column + 1
                'gets the last field
            ElseIf column = columnCounter Then
                myArray(column) = theString
                theString = "" 'truncates the string
                column = column + 1
                'gets any other field
            Else
                myArray(column) = Left(theString, theString.IndexOf(","))
                theString = Right(theString, theString.Length() -
(theString.IndexOf(",") + 1))
                column = column + 1
            End If
        End While

        linkRow = linkRow + 1
        Call fillLinkExcel(myArray, linkRow)

    End Sub

    Sub nodeParser(ByVal st As String)
        Dim theString As String
        Dim myArray(12) As String
        Dim column As Integer = 0

        theString = st

        While theString.Length() > 0
            'gets the x-coordinates
            If column = 2 Then
                myArray(column) = Mid((Left(theString,
theString.IndexOf(")"))), 3, theString.Length - 3)
                theString = Right(theString, theString.Length() -
(theString.IndexOf(")") + 1))
                column = column + 1
                'gets the y-coordinates
            ElseIf column = 3 Then
                myArray(column) = Mid((Left(theString,
theString.IndexOf(")"))), 4, theString.Length - 4)
                theString = Right(theString, theString.Length() -
(theString.IndexOf(")") + 3))
                column = column + 1
                'gets last field
            ElseIf column = 12 Then
```

```
                myArray(column) = theString
                theString = ""
                column = column + 1
                'gets any other field
            Else
                myArray(column) = Left(theString, theString.IndexOf(","))
                theString = Right(theString, theString.Length() -
(theString.IndexOf(",") + 1))
                column = column + 1
            End If
        End While

        'If column = 12 Then
        'myArray(column) = ""
        'End If

        nodeRow = nodeRow + 1
        Call fillNodeExcel(myArray, nodeRow)

    End Sub
    Private Sub fillLinkExcel(ByVal myArray() As String, ByVal rows As
Integer)
        'Dim temp As String
        Dim i As Integer = 1
        Dim test As String

        'remove all spaces
        If myArray(1).IndexOf(" ") > -1 Then
            myArray(1) = Replace(myArray(1), " ", "")
        End If
        toLabel = ""
        If myArray(0).StartsWith("CircuitLink") Then
            myArray(0) = "CS"
        ElseIf myArray(0).StartsWith("PacketLink") Then
            myArray(0) = "PS"
        ElseIf myArray(0).StartsWith("SignallingLink") Then
            myArray(0) = "SIGNALLING"
        ElseIf myArray(0).StartsWith("TransportLink") Then
            myArray(0) = "TRANSPORT"
        End If

        'gets name of start and end node
        myArray(4) = Right(myArray(4), myArray(4).Length() -
(myArray(4).IndexOf(">") + 1))
        myArray(5) = Right(myArray(5), myArray(5).Length() -
(myArray(5).IndexOf(">") + 1))

        Call SetExcel("links", 1, rows, myArray(0)) 'Link type

        myArray(4) = Replace(myArray(4), " ", "")

        'delete special characters
        If myArray(4).IndexOf("/") > -1 Then
            myArray(4) = Replace(myArray(4), "/", "")

        End If
        If myArray(4).IndexOf("(") > -1 Then
            myArray(4) = Replace(myArray(4), "(", "")

        End If
        If myArray(4).IndexOf(")") > -1 Then
```

```
            myArray(4) = Replace(myArray(4), ")", "")

        End If


        Call SetExcel("links", 2, rows, getAreaName(myArray(4))) 'Start
node

        myArray(5) = Replace(myArray(5), " ", "")

        If myArray(5).IndexOf("/") > -1 Then
            myArray(5) = Replace(myArray(5), "/", "")

        End If
        If myArray(5).IndexOf("(") > -1 Then
            myArray(5) = Replace(myArray(5), "(", "")

        End If
        If myArray(5).IndexOf(")") > -1 Then
            myArray(5) = Replace(myArray(5), ")", "")

        End If

        myArray(5) = Replace(myArray(5), " ", "")
        Call SetExcel("links", 3, rows, getAreaName(myArray(5))) 'End node

        If myArray(0).Equals("CS") Then
            'if the capacity field is empty, set default value
            If (myArray(7) Is Nothing) Or myArray(7).Equals("") Then
                Call SetExcel("links", 4, rows, "-1")
                toLabel = toLabel + "Added a default numberofcircuits value
to CS link on row " + rows.ToString + "." & vbNewLine
            Else
                Call SetExcel("links", 4, rows, myArray(7))
            End If
        ElseIf myArray(0).Equals("PS") Then
            If (myArray(7) Is Nothing) Then
                Call SetExcel("links", 4, rows, "{2000 2000} {}")
                toLabel = toLabel + "Added a default bandwitdh value to PS
link on row " + rows.ToString + "." & vbNewLine
            ElseIf myArray(7).Equals("") Then
                Call SetExcel("links", 4, rows, "{2000 2000} {}")
                toLabel = toLabel + "Added a default bandwitdh value to PS
link on row " + rows.ToString + "." & vbNewLine
            Else
                Call SetExcel("links", 4, rows, myArray(7))
            End If
        ElseIf myArray(0).Equals("SIGNALLING") Then
            If (myArray(7) Is Nothing) Or myArray(7).Equals("") Then
                Call SetExcel("links", 4, rows, 1)
                toLabel = toLabel + "Added a default value to SIGNALLING
link on row " + rows.ToString + "." & vbNewLine
            Else
                Call SetExcel("links", 4, rows, myArray(7))
            End If
            'special case for the bloody transport link
        ElseIf myArray(0).Equals("TRANSPORT") Then
            parseTransport(myArray, rows)
        End If
        'set link mode for CS or TRANSPORT links
        If myArray(0).Equals("CS") Or myArray(0).Equals("TRANSPORT") Then
```

```
            Call SetExcel("links", 5, rows, myArray(8)) 'Link mode
        End If

        myArray(1) = Replace(myArray(1), "=>", "__") 'changes area name

        'checks if a linktype already exists in a area!
        While Not getExcel("areas", 1, i) = ""
            If (getExcel("areas", 2, i) Is Nothing) Then
                test = ""
            Else
                test = getExcel("areas", 2, i)
            End If
            If getExcel("areas", 1, i) = myArray(1) And
test.IndexOf(myArray(0).ToString()) < 0 Then
                Call SetExcel("areas", 2, i, getExcel("areas", 2, i) + " "
+ myArray(0))
                Exit While
            ElseIf getExcel("areas", 1, i) = myArray(1) And
test.IndexOf(myArray(0).ToString()) >= 0 Then
                Exit While
            End If
            i = i + 1
        End While

    End Sub
    Private Sub fillNodeExcel(ByVal myArray() As String, ByVal rows As
Integer)
        Dim firstXCoord As String
        Dim firstYCoord As String
        Dim secondXCoord As String
        Dim secondYCoord As String
        Dim xCoord As String
        Dim yCoord As String
        Dim temp As String
        Dim i As Integer = 0
        Dim j As Integer = 0

        toLabel = "" 'truncates the log string
        firstXCoord = Left(myArray(2), myArray(2).IndexOf(","))
        firstYCoord = Right(myArray(2), myArray(2).Length() -
(myArray(2).IndexOf(",") + 1))
        secondXCoord = Left(myArray(3), myArray(3).IndexOf(","))
        secondYCoord = Right(myArray(3), myArray(3).Length() -
(myArray(3).IndexOf(",") + 1))
        'translate coordinates to Counts coordinate formate
        xCoord = (secondXCoord - ((secondXCoord - firstXCoord) / 2)) *
RESOLUTION
        yCoord = (secondYCoord - ((secondYCoord - firstYCoord) / 2)) *
RESOLUTION
        'delete spaces
        If myArray(1).IndexOf(" ") > -1 Then
            myArray(1) = Replace(myArray(1), " ", "")
        End If
        temp = myArray(6)
        If myArray(6).IndexOf(" ") > -1 Then
            myArray(6) = Replace(myArray(6), " ", "")
        End If
        'delete special characters
        If myArray(6).IndexOf("/") > -1 Then
            myArray(6) = Replace(myArray(6), "/", "")
        End If
```

```
        If myArray(6).IndexOf("(") > -1 Then
            myArray(6) = Replace(myArray(6), "(", "")
        End If
        If myArray(6).IndexOf(")") > -1 Then
            myArray(6) = Replace(myArray(6), ")", "")
        End If
        'write eventual character changes to log string
        If Not temp.Equals(myArray(6)) Then
            toLabel = toLabel + "changed " + temp + " to " +
myArray(6).ToString & vbNewLine
        End If
        'write node fields to excel (first time)
        If (getExcel("nodes", 1, 1) Is Nothing) Then
            Call SetExcel("nodes", 3, rows, myArray(0)) 'Type
            Call SetExcel("nodes", 1, rows, myArray(6)) 'Name
            Call SetExcel("nodes", 2, rows, myArray(8)) 'Level

            If Not myArray(7) = "" Then
                Call SetExcel("nodes", 4, rows, myArray(7)) 'Functionality
            End If

            Call SetExcel("nodes", 5, rows, myArray(9)) 'Transport
capabilty
            Call SetExcel("nodes", 6, rows, myArray(10)) 'HW platform
            Call SetExcel("nodes", 7, rows, myArray(11)) 'HW type
            Call SetExcel("nodes", 8, rows, myArray(12)) 'SW release
            'write node fields to first empty row in excel
        Else
            Dim k As Integer = 1
            Dim nodeTest As Boolean = False
            While Not (getExcel("nodes", 1, k) Is Nothing)
                If getExcel("nodes", 1, k) = myArray(6) Then
                    nodeTest = True
                    Exit While
                End If
                k = k + 1
            End While
            If Not nodeTest Then
                Call SetExcel("nodes", 3, k, myArray(0)) 'Type
                Call SetExcel("nodes", 1, k, myArray(6)) 'Name
                Call SetExcel("nodes", 2, k, myArray(8)) 'Level

                If Not myArray(7) = "" Then
                    Call SetExcel("nodes", 4, k, myArray(7))
                End If

                Call SetExcel("nodes", 5, k, myArray(9))
                Call SetExcel("nodes", 6, k, myArray(10))
                Call SetExcel("nodes", 7, k, myArray(11))
                Call SetExcel("nodes", 8, k, myArray(12))

            End If
        End If
        myArray(1) = Replace(myArray(1), "=>", "__") 'darn special
character in area identifier

        'Writes level to areas sheet
        While 1        'Writes name and coordinats
            If getExcel("areas", 1, getAreaLevel(myArray(1)) + i) = "" Then
                Call SetExcel("areas", 1, getAreaLevel(myArray(1)) + i,
myArray(1))
```

```
                Call SetExcel("areas", 3, getAreaLevel(myArray(1)) + i,
myArray(6) _
                                      + " " + xCoord + " " + yCoord)
                Exit While
            ElseIf getExcel("areas", 1, getAreaLevel(myArray(1)) + i) =
myArray(1) Then
                While 1
                    If getExcel("areas", 3 + j, getAreaLevel(myArray(1)) +
i) = "" Then
                        Call SetExcel("areas", 3 + j,
getAreaLevel(myArray(1)) + i, myArray(6) _
                        + " " + xCoord + " " + yCoord)
                        Exit While
                    End If
                    j = j + 1
                End While
                Exit While
            End If
            i = i + 1
        End While
    End Sub

    Function getAreaLevel(ByVal myArray As String)

        If myArray.IndexOf("__") > -1 Then '"__" seperates every level
(area)
            Return STARTLEVEL +
getAreaLevel(myArray.Substring(myArray.IndexOf("__") + 1))
        Else
            Return 1
        End If

    End Function

    Function getAreaName(ByVal myArray As String)
        'returns lowest component name in a area (node name)
        Dim index As Integer
        index = myArray.LastIndexOf(">")
        If index = -1 Then
            Return myArray
        Else
            Return myArray.Substring(index + 1)
        End If
    End Function

    'returns string containing eventual errors/warnings occured under
parsing
    Function getText()
        Return toLabel
    End Function

    Private Function getExcel(ByVal xlWorksheet As String, _
     ByVal xlCol As Integer, ByVal xlRow As Integer) As String
        'gets cell contents in a specific sheet on column xlCol, row xlRow
        Dim strCellContents As String

        ' Get the Cell Contents

        oApp.Sheets(xlWorksheet).Select()
        strCellContents = oApp.ActiveCell(xlRow, xlCol).Value
        getExcel = strCellContents
```

```
End Function
Private Sub SetExcel(ByVal xlWorksheet As String, _
ByVal xlCol As Integer, ByVal xlRow As Integer, _
ByVal xlCellContents As String)
    'sets cell contents in a specific sheet on column xlCol, row xlRow

    'if string is empty/nothing, set "Undefined"
    If (xlCellContents Is Nothing) Then
        xlCellContents = "Undefined"
    ElseIf xlCellContents.Length() = 0 Then
        xlCellContents = "Undefined"
    End If

    oApp.Sheets(xlWorksheet).Select()
    oApp.ActiveCell(xlRow, xlCol) = xlCellContents

End Sub

Sub createExcel(ByVal filename As String)
    'Creates a Excel object and open a excel file
    oApp.UserControl = True
    oApp.Workbooks.Add()
    Call makeSheets(oApp)
    oApp.ActiveWorkbook.SaveAs(filename)

End Sub

Sub makeSheets(ByVal xls As Excel.Application)
    'makes excel sheets in the excel document

    Dim i As Integer
    Dim SheetString As String
    SheetString = "Sheet"
    Dim MyVector(5) As String
    ' Sets name for the sheets

    MyVector(4) = "services"
    MyVector(3) = "nodes"
    MyVector(2) = "containers"
    MyVector(1) = "links"
    MyVector(0) = "areas"

    ' makes sheets

    For i = 0 To 4
        xls.Worksheets.Add()
        xls.Sheets(SheetString + CStr(i + 1)).Name = MyVector(i)
    Next i


End Sub

Sub parseTransport(ByVal myArray() As String, ByVal rows As Integer)
    'the special tranport link parser
    Dim myStruct As New transStruct
    Dim temp As String

    'set defaults if capacityy field is empty
    If (myArray(7) Is Nothing) Or myArray(7).Equals("") Then
```

```
            Call SetExcel("links", 4, rows, "ATM_PVC {LINEAR {}} LINKTYPE 1
{ATM}") 'AAL5_PVC {LINEAR {}} LINKTYPE 1 {AAL5}
            toLabel = toLabel + "Added a default value to TRANSPORT link on
row " + rows.ToString + "." & vbNewLine
            myStruct.name = "ATM_PVC"
            myStruct.speed = "1 1kb 100"
            'add name and speed to a queue if it's not already in the queue
            If checkTransport(transportVector, myStruct) Then
                transportVector.Enqueue(myStruct)
            End If
            'add default protocoltype ATM to a queue if it's not already in
the queue
            If checkService(serviceVector, "ATM") Then
                Call SetExcel("services", 1, serviceRow, "PROTOCOLTYPE")
                Call SetExcel("services", 2, serviceRow, "ATM")
                Call SetExcel("services", 3, serviceRow, "0")
                Call SetExcel("services", 4, serviceRow, "0")
                serviceRow = serviceRow + 1
                serviceVector.Enqueue("ATM")
            End If

            'if capacity field is non-empty
        Else
            Call SetExcel("links", 4, rows, myArray(7))
            temp = Left(myArray(7), myArray(7).IndexOf(" "))
            myStruct.name = temp
            temp = Right(myArray(7), myArray(7).LastIndexOf(" "))
            temp = Replace(temp, "{", "")
            temp = Replace(temp, "}", "")
            myStruct.speed = myArray(9)
            'add name and speed to a queue if it's not already in the queue
            If checkTransport(transportVector, myStruct) Then
                transportVector.Enqueue(myStruct)
            End If
            'add protocoltype to a queue if it's not already in the queue
            If checkService(serviceVector, temp) Then
                Call SetExcel("services", 1, serviceRow, "PROTOCOLTYPE")
                Call SetExcel("services", 2, serviceRow, temp)
                Call SetExcel("services", 3, serviceRow, "0")
                Call SetExcel("services", 4, serviceRow, "0")
                serviceRow = serviceRow + 1
                serviceVector.Enqueue(temp)
            End If
        End If
    End Sub

    Function checkTransport(ByVal checkVector As Queue, ByVal compStruct As
transStruct)
        'check if a transport link name is alrady in the queue
        Dim myStruct As New transStruct
        Dim checker As Boolean = True
        While checkVector.Count > 0
            myStruct = checkVector.Dequeue()
            If myStruct.name = compStruct.name Then
                checker = False
                Exit While
            End If
        End While
        Return checker

    End Function
```

```
    Function checkService(ByVal checkVector As Queue, ByVal compString As
String)
        'check if a protocoltype is already in the queue
        Dim myString As String
        Dim checker As Boolean = True
        While checkVector.Count > 0
            myString = checkVector.Dequeue()
            If myString = compString Then
                checker = False
                Exit While
            End If
        End While
        Return checker


    End Function

    Sub closeExcel()
        oApp.ActiveWorkbook.Save()
        oApp.Workbooks.Close()
    End Sub

    Sub quitExcel()
        oApp.Quit()
        GC.Collect()
    End Sub

    Public Function getComponentTypeName(ByVal aString As String)
        'returns a substring left from the first occurence of ","
        Return Left(aString, aString.IndexOf(","))
    End Function

    Public Function addLinkTypes()
        'adds all different linktypes listed in a queue to excel
        Dim myStruct As New transStruct
        While transportVector.Count > 0
            myStruct = transportVector.Dequeue()
            oApp.Sheets("links").Select()
            oApp.ActiveSheet.Rows("1:1").Select()
            oApp.Selection.Rows.Insert() 'insert blank row
            Call SetExcel("links", 1, 1, "LINKTYPE")
            Call SetExcel("links", 2, 1, myStruct.name)
            Call SetExcel("links", 3, 1, myStruct.speed)

        End While
    End Function
End Class
```

A.3 Loggningsklassen “Logging”

```
Public Class Logging
    Inherits System.Windows.Forms.Form
    Private logState() As Integer = {1, 1, 0} 'Start logstat
#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
```

```
        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'If disposing Then
        'If Not (components Is Nothing) Then
        'components.Dispose()

        'End If
        'End If
        MyBase.Hide()
        'MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox2 As System.Windows.Forms.CheckBox
    Friend WithEvents CheckBox3 As System.Windows.Forms.CheckBox
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.CheckBox1 = New System.Windows.Forms.CheckBox
        Me.CheckBox2 = New System.Windows.Forms.CheckBox
        Me.CheckBox3 = New System.Windows.Forms.CheckBox
        Me.SuspendLayout()
        '
        'CheckBox1
        '
        Me.CheckBox1.Checked = True
        Me.CheckBox1.CheckState = System.Windows.Forms.CheckState.Checked
        Me.CheckBox1.Location = New System.Drawing.Point(24, 16)
        Me.CheckBox1.Name = "CheckBox1"
        Me.CheckBox1.TabIndex = 0
        Me.CheckBox1.Text = "Enable logging"
        '
        'CheckBox2
        '
        Me.CheckBox2.Enabled = False
        Me.CheckBox2.Location = New System.Drawing.Point(24, 64)
        Me.CheckBox2.Name = "CheckBox2"
        Me.CheckBox2.TabIndex = 0
        Me.CheckBox2.Text = "Log to Window"
        '
        'CheckBox3
        '
        Me.CheckBox3.Enabled = False
        Me.CheckBox3.Location = New System.Drawing.Point(24, 128)
        Me.CheckBox3.Name = "CheckBox3"
        Me.CheckBox3.TabIndex = 2
        Me.CheckBox3.Text = "Log to File"
        '
```

```
        'Logging
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.Add(Me.CheckBox3)
        Me.Controls.Add(Me.CheckBox2)
        Me.Controls.Add(Me.CheckBox1)
        Me.Name = "Logging"
        Me.Text = "Logging"
        Me.ResumeLayout(False)

    End Sub

#End Region

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
        'sets window lock. if "enable logging" is checked then turn on "log
to window"
        If (Me.CheckBox1.Checked = True) Then
            Me.CheckBox2.Enabled = True
            Me.CheckBox2.Checked = True
            Me.CheckBox3.Enabled = True
            Me.CheckBox3.Checked = False
            logState(0) = 1
            logState(1) = 1
            logState(2) = 0
        Else
            'no logging
            Me.CheckBox2.Enabled = False
            Me.CheckBox2.Checked = False
            Me.CheckBox3.Enabled = False
            Me.CheckBox3.Checked = False
            logState(0) = 0
            logState(1) = 0
            logState(2) = 0
        End If
    End Sub

    Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
        'to keep logging to either window or file when eneable logging is
checked
        If (Me.CheckBox2.Checked = True) Then
            logState(1) = 1 'log to window
        Else
            If Me.CheckBox3.Checked = False Then
                Me.CheckBox3.Enabled = True
                Me.CheckBox3.Checked = True
                logState(2) = 1 'log to file
            End If
            logState(1) = 0 'don't log to window
        End If

    End Sub

    Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox3.CheckedChanged
        If (Me.CheckBox3.Checked = True) Then
            logState(2) = 1 'log to file
        Else
```

```
            If Me.CheckBox1.Checked = True And Me.CheckBox2.Checked = False
Then
                Me.CheckBox2.Enabled = True 'keep logging to window if
logging enabled
                Me.CheckBox2.Checked = True
                logState(1) = 1
            End If
            logState(2) = 0
        End If
    End Sub

    Public Function getLogState()
        Return logState
    End Function


End Class
```

A.4 Informationsfönstret “About”

```
Public Class About
    Inherits System.Windows.Forms.Form


#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents AboutLabel As System.Windows.Forms.Label
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.AboutLabel = New System.Windows.Forms.Label
        Me.SuspendLayout()
        '
```

```
        'AboutLabel
        '
        Me.AboutLabel.Location = New System.Drawing.Point(0, 0)
        Me.AboutLabel.Name = "AboutLabel"
        Me.AboutLabel.Size = New System.Drawing.Size(232, 136)
        Me.AboutLabel.TabIndex = 0
        Me.AboutLabel.Text = ("The SG File Converter" & vbNewLine &
"version 0.9" _
        & vbNewLine & "made by:" & vbNewLine & "Mathias Wagnsson and Jan
Persson" _
        & vbNewLine & "Copyright Tieto Enator 2003")
        Me.AboutLabel.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'About
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(232, 133)
        Me.Controls.Add(Me.AboutLabel)
        Me.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle
        Me.MaximizeBox = False
        Me.MinimizeBox = False
        Me.Name = "About"
        Me.Text = "About"
        Me.ResumeLayout(False)

    End Sub

#End Region


End Class
```