
Avdelning för datavetenskap

Martin Landälv

# Grafiska ioFTPD-verktyg

## Graphical ioFTPD Tools

Datavetenskap

C-uppsats (10 p)

| | |
|---|---|
| Datum: | 06-06-07 |
| Handledare: | Robin Staxhammar |
| Examinator: | Stefan Lindskog |
| Löpnummer: | C2006:12 |

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

# Grafiska ioFTPD-verktyg

**Martin Landälv**

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.

Martin Landälv

Godkänd, 2006-06-07

Handledare: Robin Staxhammar

Examinator: Stefan Lindskog

# Sammanfattning

Jag har under en period motsvarande 10 veckor på heltid utvecklat ett grafiskt konfigurationsverktyg för konfiguration av ftp-servern ioFTPD [4]. ioFTPD är en relativt ny ftp-server för Windows som är resurssnål och högpresterande [4]. ioFTPD utvecklas av iniCom Networks [4] som även utvecklar den kända ftp-klienten FlashFXP [4].

Idén till att utveckla ett grafiskt konfigurationsprogram kom till eftersom det inte fanns några grafiska verktyg för vare sig konfiguration eller övervakning av servern. Eftersom det inte fanns något användarvänligt grafiskt konfigurationsverktyg var användaren tvungen att editera ioFTPD:s konfigurationsfil manuellt. Manuell editering av en så stor fil gör det lätt att införa fel.

Mina delmål var att skapa tre grafiska applikationer för att hjälpa till att konfigurera och övervaka ioFTPD-servern.

- Ett grafiskt konfigurationsverktyg
- Ett övervakningsprogram
- En vfs-editor för skapande och redigering av det virtuella filsystemet

Huvudmålet var att implementera det grafiska konfigurationsverktyget, vilket jag gjorde. Jag är nöjd med slutprodukten även om det finns detaljer som kan förbättras.

Under projektets gång har jag bland annat förbättrat mina kunskaper i C++ och Win32-programmering, samt att jag insett vikten av att göra en genomtänkt design innan man påbörjar implementationen.

# Graphical ioFTPD Tools


## Abstract

I have, during a period of ten weeks, full time, developed a configuration utility, featuring a graphical user interface for the configuration of the ftp server ioFTPD [4]. ioFTPD is a relatively new high performance and resource efficient ftp daemon [4] for Windows. ioFTPD is developed by iniCom Networks [4], who also develop the popular ftp client FlashFXP. [4]

Because ioFTPD lacks graphical applications for the configuration and monitoring, the idea to develop a graphical configuration utility arose. Since there was no user friendly graphical configuration utility, the user manually had to edit ioFPTD's configuration file. Manually editing such a big file could easily result in error introduction.

My goals were to develop three graphical applications to aid in the configuration and monitoring of the ioFTDP server.

- A graphical configuration utility
- A monitoring application
- A vfs-editor to aid in the creation and editing of the virtual file system

The main goal was to implement the graphical configuration utility, which I did. I'm satisfied with the result even though there are details that could be improved.

During the project I have improved my knowledge with C++ and programming with the Windows API. I've also realized the importance of doing a thoroughly thought through design before one starts the implementation.

# Innehållsförteckning

# Figurförteckning

# 1 Inledning

## 1.1 Bakgrund

Jag har sen några år tillbaka använt mig av ftp-servern ioFTPD [4]. IoFTPD har funnits ett par år och numera utvecklas den av iniCom Networks [4], som även utvecklar den kända ftp-klienten FlashFXP [4].

IoFTPD riktar sig till lite mer avancerade användare, användare som är beredda att sätta upp en server ifrån grunden, där alla inställningar måste gås igenom. Förutom en ftp-tjänst erbjuder även ioFTPD möjligheten att logga in via telnet och http.

Http-tjänstens huvudanvändningsområde är att lätt sköta om administration av användare och grupper samt att lätt kunna ladda ner filer från servern utan tillgång till någon ftp-klient.

Telnet-tjänsten användes i början för att övervaka ftp-servern, för att se vad som hände på servern. Man kunde se vilka som loggade in respektive ut, skapade kataloger etc. Detta ändrades i en senare version och telnet-tjänstens användningsområde är nu mycket begränsat. Enda anledningen att använda den, som jag ser, är för att ändra inställningarna på användare och grupper utan att behöva ta upp en anslutning på ftp-tjänsten.

Ftp-tjänsten har endast de mest grundläggande egenskaperna en server bör ha inbyggt, såsom hantering av användare och grupper, filöverföring samt virtuellt filsystem. Resterande funktionalitet kommer i form av plugins eller så kallade skript. Således finns inga grafiska verktyg för vare sig konfiguration, övervakning eller dylikt. Detta ställer ofta till det för nya användare, då de inte riktigt vet hur servern ska konfigureras, vad de olika delarna i konfigurationsfilen gör o.s.v. Detta ska delvis underlättas med hjälp av det grafiska användargränssnittet, vilket gör det möjligt att ändra serverns inställningar utan manuell editering av konfigurationsfilen.

```
[FTP_Service]
Type              = FTP
Device_Name       = FTP-Listen
Port              = 7008
Description       = FTP SERVER STANDARD
User_Limit        = 100
Allowed_Users     = !6 *
Messages          = ..\text\ftp
### Encryption      ###
Require_Encrypted_Auth  = *
Require_Encrypted_Data  = !*
Certificate_Name    = fobban
Explicit_Encryption = True
Encryption_Protocol = TLS
Min_Cipher_Strength = 128
Max_Cipher_Strength = 128
### IDNT command handler###
Get_External_Ident  = True
### Traffic Balancing   ###
Data_Devices        = FTP-Data
Random_Devices      = False
```

*Figur 1.1: så här ser delar av konfigurationen för ftp-tjänsten ut vid manuell editering*

Det finns en ganska bra installationsguide på ioFTPD:s hemsida [4], som beskriver vad alla inställningar innebär, rekommenderade värden o.s.v., men allt måste ändras manuellt i konfigurationsfilen, vilket är både tidskrävande och krångligt, se Figur 1.1.

Det finns inte heller något bra, lättvindigt sätt att övervaka vad som sker på ftp-servern, vilka som är inloggade, vad de gör etc. Som det är nu måste man logga in på servern och skicka kommandon för att se vilka som är inne, vad den användaren gör o.s.v.

IoFTPD använder sig av ett virtuellt filsystem, som liknar det Linux använder. För att en användare ska kunna logga in krävs det att den har en så kallad vfs-fil associerad till sig; det kan antingen vara en specifik vfs för just den användaren, för gruppen användaren är medlem i eller en standard, så kallad *defaultfil*. Det manuella skapandet av en vfs-fil är i sig inte speciellt svårt, men för ovana användare som inte vet hur strukturen ser ut ställer det till vissa problem. Ett vanligt fel är att de inte inkluderar en rotkatalog. Med ett grafiskt verktyg för att skapa och manipulera vfs-filer underlättas detta arbete avsevärt, och man kan undvika bland annat rotproblemet.
Med en vfs-fil som i Figur 1.2 skulle vi få en katalogstruktur som motsvaras av bilden i figur 1.3. Utan rotkatalogen skulle det inte gå att logga in, utan man möts av ett felmeddelande.

```
"C:\ftp\rot"          /
"D:\Martin\Skolan"    /Skolan
"D:\Musik"            /Musik
"E:\Mp3"              /Musik
"E:\Hemligt"          /Privat
```

*Figur 1.2: en vfs-fil som bilden ovan skulle ge en katalogstruktur som representeras av Figur 1.3*


*Figur 1.3: katalogstrukturen som motsvaras av en vfs-fil som Figur 1.2 visar.*

## 1.2 Mål

Mitt huvudmål är att skapa ett grafiskt användargränssnitt för konfigurationen av ioFTPD, där alla inställningar som kan göras genom manuell editering av konfigurationsfilen lättare ska kunna utföras via ett grafiskt, lättförståeligt användargränssnitt.

Har jag tid över när jag är klar och nöjd med huvudmålet är mitt andra och tredje mål att göra en vfs-editor samt ett övervakningsprogram. Vilket av dem beror på hur mycket tid som återstår. Vfs-editorn kommer att gå snabbast att utveckla medan övervakningssystemet är ett betydligt större projekt.

# 2 Översikt av ioFTPD

IoFTPD är en högpresterande resurssnål ftp-server. I det här kapitlet beskrivs vad som är utmärkande för ioFTPD samt hur konfigurationen av servern sker.

## 2.1 Bakgrund

IoFTPD är en relativt ny ftp-server som försöker göra sig alltmer känd. Projektet med att starta utvecklingen av servern var mest en hobby för en finsk dataingenjörsstudent. Han fick relativt snabbt en grupp med anhängare som gav feedback då de testade hans produkt.

Målet med ioFTPD, som finländaren utvecklade, var att leverera en högpresterande, resurssnål ftp-server gratis. För varje ny betaversion växte antalet anhängare och många av de nya anhängarna var sådana som valt att prova den just för att den var gratis.

I och med populariteten produkten fick, såg utvecklaren sin chans att tjäna pengar. Servern fick en prislapp på tio dollar och för de som inte ville eller hade råd att betala fanns även en så kallad u-version (unregistered) som saknade stöd för säker inloggning och säker dataöverföring samt hade sämre minneshantering [4]. Vad sämre minneshantering innebar stod det ingenting om – kanske var det ett försäljningsknep?

Många oroade sig över ifall en ensam student skulle kunna hålla igång ett sådant stort projekt vid sidan av skolan och ifall han skulle fortsätta utveckla servern tills den blev det självklara valet för användare. Efter det att beta 5-8-5 kom, i slutet av 2004, blev det ett uppehåll i utvecklingen och många trodde att projektet var nedlagt, men IniCom Networks såg potentialen och köpte rättigheterna till programmet samt lovade finländaren ersättning ifall han fortsatte utvecklingen.

Allt gick dock inte som planerat och strax efter det att IniCom tagit över utvecklingen försvann utvecklaren spårlöst och utvecklingen stod återigen stilla. Företaget har nu anlitat en ny utvecklare som jobbar på att vidareutveckla ioFTPD [4].

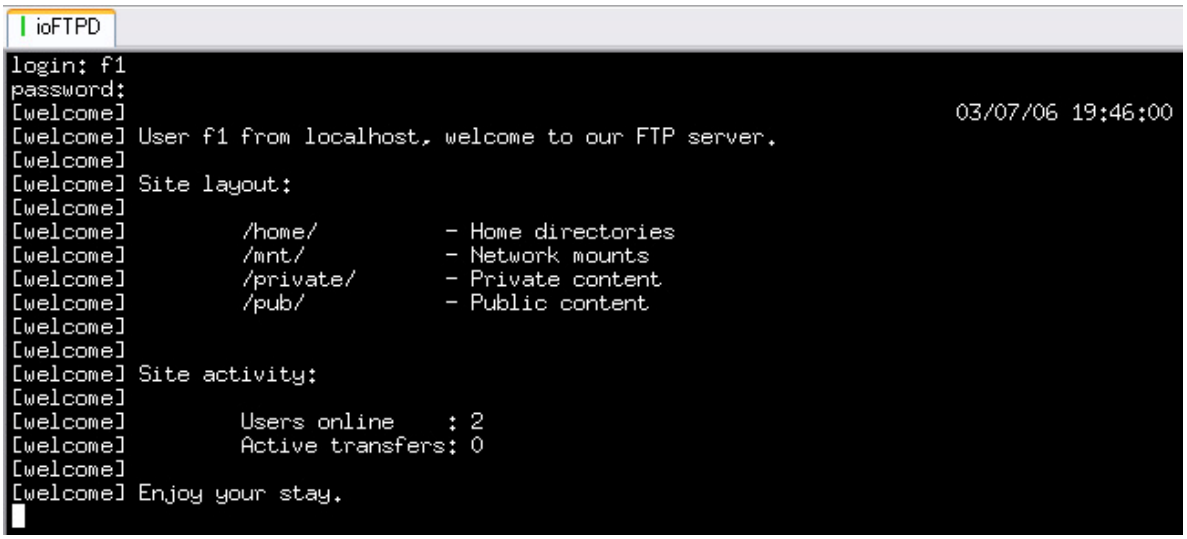
## 2.2 Specifikationer

Vad är det då som gör ioFTPD bättre än någon av sina konkurrenter, som exempelvis Serv-U, BulletProof eller RaidenFTPD? För det första skulle jag vilja säga att det inte finns något rätt svar; alla har sin egen favorit, precis som när det gäller operativsystem, bildhanterings-program eller vad det nu må vara. Att välja ”rätt” ftp-server beror helt på vad målet med servern är: ska man dela med sig av filer med sina kompisar eller ska servern användas av flera tusen användare samtidigt?

En stor fördel ioFTPD har jämfört med många av sina konkurrenter är flexibiliteten för utveckling av plugins, eller så kallade skript. Förutom att utvecklingen kan ske i valfritt programmeringsspråk har även ioFTPD inbyggt stöd för skriptspråket tcl (tool command language). Tcl är ett skriptspråk som körs på servern, vilket innebär att skripten kompileras ”on-the-fly” varje gång de ska användas och således behöver inte utvecklaren använda sig utav någon speciell utvecklingsmiljö eller kompilator. Nackdelen är att man förlorar prestanda eftersom kompilering sker vid varje användningstillfälle.

Läser man på ioFTPD:s hemsida kan man läsa att servern är designad för att användas på stora företag, där det förekommer flera tusen samtidiga användare, eller för privatpersoner som behöver en resurssnål, men ändå högpresterande server [4].

IoFTPD har förutom ftp-tjänsten även en telnet- och en http-tjänst. Telnet-tjänsten användes i tidiga betaversioner som övervakare, då man såg när användare loggade in/ut och när kataloger skapades; det gör man inte längre och tjänstens användningsområde är diskuterbart, vissa använder den för att administrera servern utan att blockera en plats på ftp-tjänsten. Ett exempel på telnet-tjänsten ses i Figur 2.1.




*Figur 2.1: Exempel på hur det ser ut efter man loggat in via telnet. Man kan sedan använda sig av kommandon för att kommunicera med servern.*

Http-gränssnittet är tänkt som hjälp för att lätt kunna hantera användarna på servern: lägga till nya, ändra rättigheter o.s.v. Tjänsten är långt från färdig, och även lågt prioriterad [4]. Som det är nu kommer man inte längre än till inloggningssidan innan man möts av ett felmeddelande. I Figur 2.2 visas hur inloggningen ser ut.


*Figur 2.2: så här ser den nuvarande inloggningssidan ut för http-tjänsten.*

### 2.2.1 Tcl

Tcl är ett kraftfullt dynamiskt programmeringsspråk som är lätt att lära sig, även för dem som inte tidigare programmerat; det är användbart för många applikationer, bland annat nätverksapplikationer. Tcl är ett öppen källkod-projekt som är lätt att både implementera och integrera i sin produkt. [11]

## 2.3 Konfiguration

IoFTPD:s konfigurationsfil har formen av en vanlig Windows-ini-fil, d.v.s. är uppbyggd av sektioner, där varje sektion har nycklar med värden (se Figur 2.3).

```
1  [Sektion]
2  Nyckel1=värde1
3  Nyckel2=värde2
4
```

*Figur 2.3: Ett exempel på hur en sektion ser ut. Denna har namnet ”Sektion” och två stycken nycklar med värden.*

De olika sektioner som finns i ioFTPD:s konfigurationsfil är följande:

- Threads
- File
- Locations
- Devices
- Services
- Network
- Ftp
- Telnet
- http
- Sections
- VFS
- Reset
- Scheduler
- Events
- Modules
- FTP_Pre-Command_Events
- FTP_Post-Command_Events
- FTP_Custom_Commands
- Telnet_Binaries
- FTP_Command_Permissions
- FTP_SITE_Permissions
- Telnet_Command_Permissions

- Telnet_Binary_Permissions
- Change_Permissions
- Http_Permissions

Som ses är konfigurationsfilen uppbyggd av ganska många sektioner, vissa större än andra. Jag skrev med *Devices*, enheter, och *Services*, tjänster, som sektioner, vilket är delvis sant. Det som är annorlunda med dessa är att de inte har ett specifikt sektionsnamn, som alla de andra har, utan har istället specifika nycklar som identifierar dem (se Figur 2.4). Hädanefter kommer jag använda de svenska orden enheter och tjänster när jag refererar till så kallade *devices* och *services* som finns i konfigurationsfilen.

```
1  [Any]
2  Host     = 1.2.3.4
3  Ports    = 1025-2048
4  Random   = True
5
```

*Figur 2.4: figuren visar ett exempel på en enhet med sektionsnamnet ”Any*

Nedan beskriver jag vad varje sektions uppgift är samt *några* inställningar som kan ställas in i dem.

### 2.3.1 Threads

Denna sektion har hand om inställningar som är specifika för ioFTPD-processen. Det är bland annat inställningar för processens prioritet, hur många trådar den skall använda sig av och fönsternamnet som används då man använder sig utav ioFTPD:s delade minne. Fönsternamnet används då man vill använda sig av ioFTPD:s delade minne.

### 2.3.2 File

File-sektionen hanterar inställningar relaterade till filuppladdning och hantering av skapade mappar. Exempelvis hur många mappar som ska lagras i ioFTPD:s interna minne eller hur många samtidiga skrivningar till disk som ska tillåtas.

### 2.3.3 Locations

Denna sektion innehåller sökvägar till olika filer och kataloger som ioFTPD använder, bland annat sökvägar till katalogerna där användare och grupper finns, där loggar sparas, etc.

### 2.3.4 Devices

Varje enhet har ett valfritt, unikt namn och har bland annat följande inställningar:

- Host: *IP-nummret servern körs på.*

- Ports: *Passive mode-portar som används vid dataöverföring.*
- Random: *Specificerar om man vill använda slumpvalda passive mode-portar inom intervallet.*
- Bind: *Binder servern till ett specifikt nätverkskort beroende på dess ip-nummer.*

Andra inställningar en enhet har är för begränsning av bandbredd, för både upp- och nedladdningar för kontroll- och/eller datakanal. Kontrollkanalen används bland annat för listning av filer och mappar samt meddelanden servern skickar. Datakanalen används för själva överföringen av data.

### 2.3.5 Services

Varje tjänst har en mängd olika inställningar som kan göras beroende på om man väljer att skapa en ftp-, telnet- eller http-tjänst. Största skillnaden mellan de olika tjänsterna är krypteringsinställningarna som, bland annat, utesluts vid skapandet av en telnet-tjänst.

Några av de viktigaste inställningarna en tjänst har är följande:

- Type: *Typ av tjänst: FTP/Telnet/HTTP.*
- Device_Name: *Namn på enheten som ska användas.*
- Port: *Port tjänsten lyssnar på.*
- User_Limit: *Maximala antalet användare tjänsten tillåter.*
- Allowed_Users: *En lista av de användare som tillåts logga in.*

IoFTPD stöder två krypteringsprotokoll: TLS (Transport Layer Security) samt SSL (Secure Sockets Layer). TLS är eftergångaren till SSL och standardiserades av standardiseringsorganet IETF [10].

### 2.3.6 Network

Network-sektionen specificeras vilka tjänster som skall vara aktiva samt lite andra inställningar som relaterar till anslutningar, exempelvis hur många gånger någon ska kunna logga in inom ett visst intervall innan det klassificeras som att användaren försöker logga in utan behörighet och blir avstängd en viss tid.

Under denna sektion ställer man också in uppdateringsfrekvensen för ioFTPD:s inbyggda schemaläggare samt hur länge ident och datornamn ska lagras i ioFTPD:s interna minne.

### 2.3.7 Ftp, Telnet och Http

Dessa sektioner har inställningar som är specifika för just de olika tjänsterna som de representerar. Man kan bland annat ställa in hur länge man får vara inaktiv på respektive tjänst innan man blir utkastad och storleken på olika buffrar för mottagning och sändning av filer (gället dock ej telnet, som inte tillåter filöverföring).

### 2.3.8 Sections

Denna sektion hanterar olika sektioner. Man kan ge olika sökvägar på ftp:n olika namn – sektionsnamn. Man kan också specificera olika kredit- och statistiksektioner för de olika namnen, det kan man göra för att kunna se vem som har laddat upp respektive laddat ned mest i de olika sektionerna.

Notera att sektion i detta avseende är namnet som virtuella sökvägar har på servern och inte en del av en Windows-ini-konfigurationsfil – som också kallas för sektion.

### 2.3.9 Vfs

Vfs-sektionen hanterar rättigheter för de olika sökvägarna som finns. Man kan bland annat ställa in vilka som har rättighet att ladda ner respektive ladda upp var, vilka som är tillåtna att ta bort filer och kataloger som tillhör någon annan användare o.s.v.

För att man ska slippa specificera alla rättighet under denna sektion har även ioFTPD stöd för oktala Unix-liknande rättigheter, som används som generella rättigheter medan vfs-sektionen specificerar specifika rättigheter för en viss sökväg, användare eller grupp. Generella rättigheter används då filer och kataloger skapas, då de inte omsluts av specifika rättigheter som är specificerade under Vfs-sektionen.

### 2.3.10 Reset

Reset-sektionen är den minsta av alla och innehåller i nuläget endast två nycklar. Under denna sektion ställer man in när vecko- och månadsstatistik skall nollställas.

### 2.3.11 Scheduler

Denna sektion hanterar schemalagda aktiviteter. Användare kan själva lägga till skript som denne vill ska köras vid vissa tidpunkter. När man specificerar minuter och timmar i schemaläggaren bör man veta att ioFTPD använder sig av GMT-tid.

### 2.3.12 Events

Events-sektionen hanterar specifika ioFTPD-händelser, exempelvis om något speciellt skript ska köras när någon loggar in eller ut, när någon filöverföring lyckas eller misslyckas eller dylikt.

### 2.3.13 Modules

Tanken med Modules-sektionen är att utvecklare ska kunna göra så kallade moduler som andra skript sedan kan använda sig av. Det har aldrig funnits någon dokumentation om hur man går tillväga för att skapa en modul, vilket troligtvis är en stor andledning till att inga moduler har skapats ännålänge.

### 2.3.14 FTP_Pre-Command_Events

Under denna sektion lägger man till skript som ska utföras innan ett ftp-kommando utförs på servern. Exempelvis kanske du inte vill att någon ska kunna skapa kataloger som innehåller ordet ”färjestad”; man specificerar då att ett visst skript, som kontrollerar katalogers namn, körs före mkdir-kommandot. Innehåller katalogen något förbjudet ord så blockeras skapandet.

### 2.3.15 FTP_Post-Command_Events

Till skillnad från föregående delkapitel, specificerar denna sektion skript som ska köras efter ett ftp-kommando har utförts. Exempelvis kan det vara så att du vill ha en lista över alla kataloger som har skapats eller alla filer som har laddats upp, och man lägger då till ett skript som gör just detta efter respektive kommando.

### 2.3.16 FTP_Custom_Commands

Denna sektion tillåter användaren att skapa egna kommandon som kan göra vad som helst.

### 2.3.17 Telnet_Binaries

I den här sektionen kan man specificera specifika kommandon för telnet-tjänsten. Tanken med denna sektion är att användaren ska kunna skräddarsy sin telnet-tjänst genom att lägga till egna kommandon som utför önskade operationer.

### 2.3.18 FTP_Command_Permissions

Denna sektion hanterar rättigheter för vanliga ftp-kommandon. Många ftp-klienter har inbyggda system för att förhindra att klienter blir utkastade på grund av att de är inloggade, men inte gör något. Detta görs oftast genom att klienten, med jämna mellanrum, skickar

kommandot noop. För att förhindra detta kan man alltså under denna sektion förbjuda alla att använda just noop-kommandot.

### 2.3.19 FTP_SITE_Permissions

Rättigheter för skräddarsydda kommandon som lagts in under sektionen *FTP_Custom_Commands* specificeras i den här sektionen.

### 2.3.20 Telnet_Command_Permissions

Den här sektionen är motsvarigheten till *FTP_Command_Permissions*, fast för telnet-tjänsten.

### 2.3.21 Telnet_Binary_Permissions

Denna sektion hanterar rättigheterna för eventuella kommandon man lagt till i sektionen *Telnet_Binaries.*

### 2.3.22 Change_Permissions

IoFTPD har implementerat change-kommandon för att kunna förändra läget på användare, grupper och även ftp-servern i sin helhet. Under den här sektionen kan användaren ställa in vilka som ska ha tillgång till de olika change-kommandona.

### 2.3.23 Http_Permissions

Den här sektionen specificerar vilka som har tillgång till olika kommandon genom http-gränssnittet. I nuläget fungerar dock inte http-tjänsten och således är denna sektion inte speciellt användbar.

## 2.4 Framtiden

Nu när utvecklingen av ioFTPD äntligen börjat gå framåt igen har IniCom kommit med relativt ny information om vad höjdpunkterna med den nya versionen som väntas komma i år är. Informationen är tagen från IniComs hemsida, och är översatt ifrån engelska.

- Unik flertrådsdesign

  *IoFTPD är designat för att kunna prestera under extrema förhållanden. Prioritets-baserade uppgiftsköer och smart användning av trådar ska garantera att demonen betjänar klienter lika även då den körs på ett lågprestandasystem. [4]*

- Integrerat skriptspråk

  *Integrerat skriptspråk tillhandahåller verktyg för att slutanvändaren ska kunna ändra uppförandet av demonens kommandon. Språket är utökat med kommando-*

*set, för att få tillgång till demonens olika resurser. [4]*

- Klientcertifikatsverifiering

  *Klientcertifikatbaserad verifiering gör det lättare att hantera konton på begränsade system. Administratören behöver inte längre oroa sig för eventuella ändringar av klientens IP-adress. [4]*

- Fjärradministration

  *Ett http-baserat administrationsgränssnitt gör det möjligt för administratörer att hantera och konfigurera sin ftp-server från fjärrsystem genom användning av en webbrowser. [4]*

# 3 Kravspecifikation

Kraven jag ställer på mina grafiska verktyg till ioFTPD är att de ska hålla hög kvalitet när det gäller både prestanda och funktionalitet, det vill säga att man genom användargränssnittet ska kunna ändra konfigurationsfilens alla sektioner på ett säkert sätt. Det innebär att man exempelvis inte ska kunna mata in strängar om det som förväntas är ett tal, eller att man får ett antal alternativ i en drop-down-lista istället för att behöva skriva alternativet själv.

Användargränssnitten ska vara lätta att förstå och använda, det vill säga att det ska vara lätt att orientera sig till det man vill konfigurera, men man bör komma ihåg att ioFTPD är ämnat för lite mer avancerade användare, personer som har en del kunskap inom området och således krävs det även viss kunskap för att förstå vad vissa uttryck betyder.

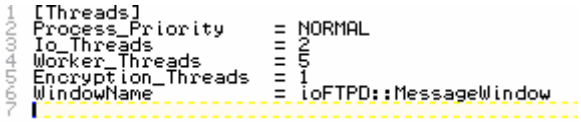
I det här kapitlet finns kraven jag ställde på konfigurationsverktyget inför utvecklandet samt kraven jag hade på VFS-editorn och övervakningsprogrammet.

## 3.1 Konfigurationsverktyget

Som det går att läsa i avsnitt 2.3 består ioFTPD:s konfigurationsfil av en mängd sektioner. Första delmålet när det gäller konfigurationsverktyget är att användaren genom det grafiska användargränssnittet ska kunna ändra de mest generella inställningarna, inställningar som krävs för att man ska få en fungerande server. Med de mest generella inställningarna menar jag de som behövs för att få igång en fungerande server, vilket omfattar följande sektioner: *Threads*, *File*, *Locations*, *Devices*, *Services*, *Network* och *Ftp*.

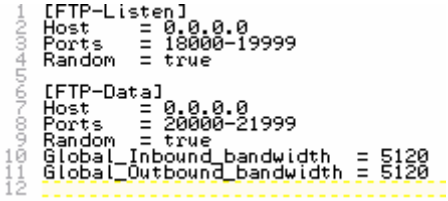
För att programmet ska få något genombrott och användas av majoriteten av ioFTPD:s användare krävs det att alla inställningar som kan göras manuellt även kan göras via användargränssnittet och det är detta är andra delmålet.

Alla inställningar i konfigurationsfilen ska på något sätt kunna redigeras. Vissa inställningar har både unika sektions- och nyckelnamn – exempelvis de under sektionen Threads, se Figur 3.1.

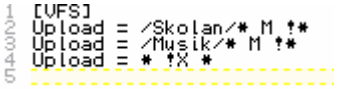



*Figur 3.1: inställningarna som kan göras under Threads-sektionen i konfigurationsfilen*

Andra inställningar är sådana som det antingen finns flera stycken av, antingen med olika sektionsnamn (se Figur 3.2), eller med flera likadana nycklar under samma sektion (se Figur 3.3).




*Figur 3.2: ett exempel på två stycken enheter som har, delvis, samma nycklar, men olika sektionsnamn*




*Figur 3.3: ett exempel på en sektion som har flera nycklar med samma namn, men med olika värden*

Utformningen av olika dialogrutor är det också viktigt att man ställer höga krav på [8]. Om man använder sig av flera dialogrutor, ska de då grupperas genom att använda sig av flikar eller ska man använda en knapp till varje dialog – eller kanske en kombination? Det finns ingen regel som avgör när man ska göra på ett visst sätt, men man bör noga fundera på vilken lösning som är den bästa i varje enskilt fall [8]. Konstruktionen för konfigurationsverktyget består av grupperingar på båda visen: flikar och enskilda. Hur gruppering gått till går att läsa i avsnitt 4.1.

När man konstruerar GUI:et är det viktigt att man väljer ett typsnitt där olika bokstäver inte kan förväxlas med varandra. Använder man exempelvis Arial, 8 punkter, så ser både litet ”L” och stort ”i” likadana ut, vilket kan göra det svårt att upptäcka var ett fel finns. [1]

## 3.2 VFS-editor

VFS-editorn är inte ett så stort projekt som varken konfigurationsprogrammet eller övervakningsprogrammet. Kravet på editorn är att man genom ett lättförståeligt användargränssnitt ska kunna skapa och redigera vfs-filer som ioFTPD använder sig av.

## 3.3 Övervakningsprogrammet

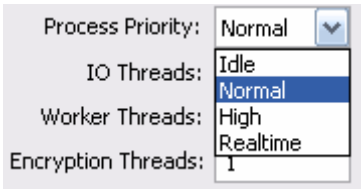
Övervakningsprogrammet delar jag upp i två delar:

- Övervakning
- Konfiguration

När ioFTPD-processen är igång kan man inte ändra användare- och gruppfiler eftersom de filerna låses. För att ändra egenskaperna på en användare eller grupp använder man ioFTPD:s delade minne och således ska övervakningsprogrammet dels visa vad som sker på servern, samt göra det möjligt att ändra användares och gruppers inställningar då ioFTPD-processen är aktiv.

# 4 Konstruktionslösning av konfigurationsverktyget

För att felaktiga inmatningar inte ska kunna inträffa är programmen konstruerade på ett sådant sätt att felaktiga inmatningar i så stor utsträckning som möjligt kan undvikas. Exempelvis är inmatningsrutor begränsade i antalet tecken de kan ta emot, för att försäkra sig om att man inte råkar ut för så kallade buffer overflows. Ett annat exempel är då man förväntas välja något specifikt värde från en mängd värden, exempelvis för nyckeln *Process_Priority* under sektionen *Threads*, som kan anta värdena *Idle*, *Normal*, *High* eller *Realtime*; valet i sådana fall görs från en dropdown-lista som i Figur 4.1. Dropdown-listan försäkrar användaren om att ett felaktigt alternativ inte kan ske [2].




*Figur 4.1: valet av ioFTPD:s processprioritet sker från en drop down-lista*

Det här kapitlet beskriver hur konstruktionen av konfigurationsverktyget är konstruerat, hur grupperingen av olika sektioner har skett samt hur det grafiska användargränssnittet har konstruerats och designats.

## 4.1 Gruppering

Konstruktionen av dialogrutorna är gjord på ett sådant sätt att information som relaterar till varandra finns lätt åtkomlig. Grupperingen har skett på två sätt: information inom samma dialogruta som relaterar till varandra har grupperats samt att dialogrutor som relaterar till varandra också är grupperade via så kallade flikar.

Sektionerna Threads, File och Locations har grupperats samman för att användaren lätt ska kunna ändra de inställningar som (oftast) behöver ställas in endast en gång. Dessa tre sektioner relaterar till varandra på grund av att alla sektioner har inställningar som påverkar ioFTPD:s grundläggande funktionalitet, bland annat sökvägar till kritiska filer och kataloger.

Inställningar för enheter, tjänster, nätverksinställningar samt specifika inställningar för de olika tjänsterna ftp, telnet och http har också grupperats samman. Det har gjorts för att användaren snabbt och enkelt ska kunna justera inställningar för de olika tjänsterna på ett och samma ställe.

Inställningar för sektionerna *Sections*, *VFS* och *Reset* har jag också valt att gruppera samman. Sections specificerar ftp-serverns olika sektionsnamn för olika virtuella sökvägar medan VFS-sektionen specificerar rättigheter för olika virtuella sökvägar. I VFS-sektionen ställs även kredit- och statistiksektioner in och det är där relationen till Reset-sektionen finns, eftersom man i den sektionen specificerar när vecko- och månadsstatistik skall nollställas.

Skriptsektionerna har också gupperats samman, vilket inkluderar sektionerna *Scheduler*, *Events*, *Modules*, *FTP_Pre-Command_Events*, *FTP_Post-Command_Events*, *Telnet_Binaries* och *FTP_Custom_Commands*.

Liksom för grupperingen av skriptsektioner har även alla sektioner för olika rättigheter grupperats samman, vilket inkluderar sektionerna *FTP_Command_Permissions*, *FTP_SITE_Permissions*, *Telnet_Command_Permissions*, *Telnet_Binary_Permissions*, *Change_Permissions* och *Http_Permissions*.

I avsnitt 4.3 finns grafiska exempel på hur grupperingarna ser ut.

## 4.2 Programmatisk konstruktionslösning

Det finns väldigt många inställningar som kan göras manuellt i ioFPTD:s konfigurationsfil och på något sätt ska alla dessa inställningar sparas. För att spara inställningarna som konfigurationsfilen består av har ett antal klasser skapats där varje klass, i regel, har hand om en sektion. Undantagen är enheter och tjänster som inte har egna sektionsnamn. Exempelvis har jag en klass för alla enheter, en för alla tjänster, en för sektionen *Threads* o.s.v. För att läsa mer om vilka klasser som finns och vad de heter se avsnitt 5.2.

## 4.3 GUI-konstruktionslösning

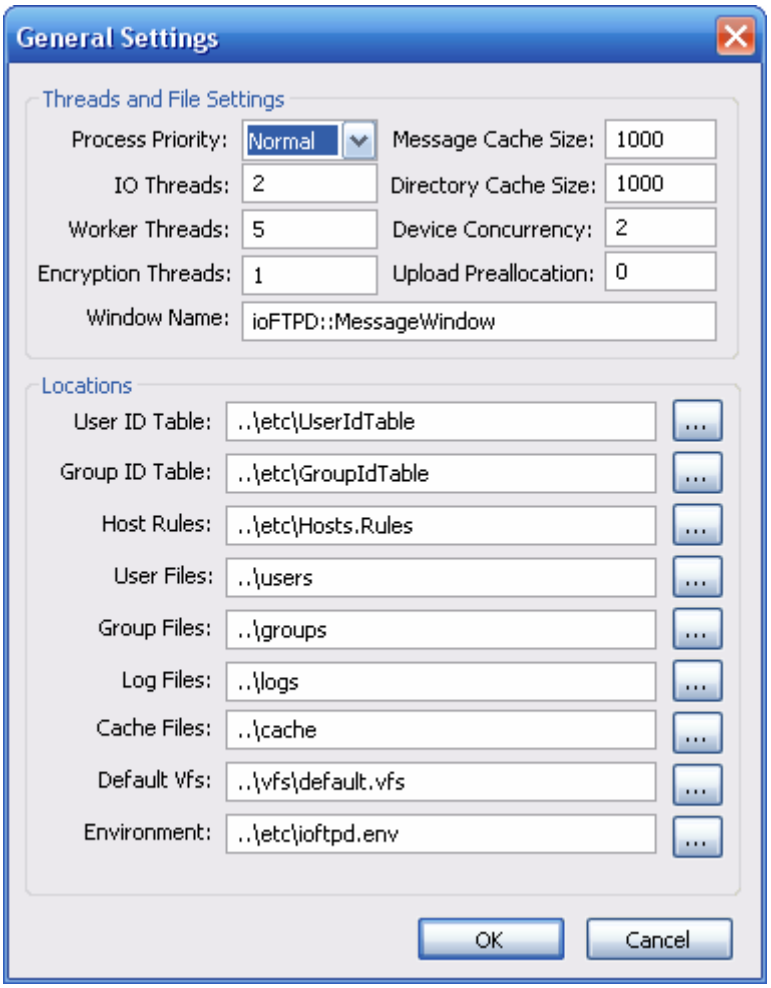
Inmatningen av värden, nya enheter etc., sker via olika dialogrutor som användaren kommer åt från huvudfönstret, se Figur 4.2. Det är tänkt att huvudfönstret ska visa en sammanfattning

om konfigurationsfilen till vänster om knapparna. Bland annat ska användaren kunna se hur många enheter och tjänster som finns och vilka tjänster som är aktiva.


*Figur 4.2: bild på huvudfönstret varifrån de andra dialogrutorna öppnas*

Dialogrutorna är utformade så att användaren lätt kan skriva in eller välja tillåtna värden. Dialogrutorna är också utformade på det viset att relaterade sektioner hanteras i samma dialogruta. I Figur 4.3 visas ett exempel på en dialogruta där man kan ändra inställningar för sektionerna *Threads*, *File* samt *Locations*.

Som man kan se i Figur 4.3 är inställningarna inramade med respektive sektionsnamn. Inramningen är gjord för att det ska vara lätt för användare att veta vilka inställningar som hör till vilken sektion, så de lätt kan lokalisera ändringarna vid eventuell manuell editering.




*Figur 4.3: dialogrutan ”General Settings” där inställningar kan ändras för de tre sektionerna Threads, File och Locations*

Förutom att dialogrutorna visar inställningar som relaterar till varandra, är de även utformade på ett sådant sätt att det är lätt för användaren att välja och mata in säkra värden, bland annat genom användning av ip-nummerrutor, se Figur 4.4.


*Figur 4.4: dialogrutan för inställningar av olika enheter*

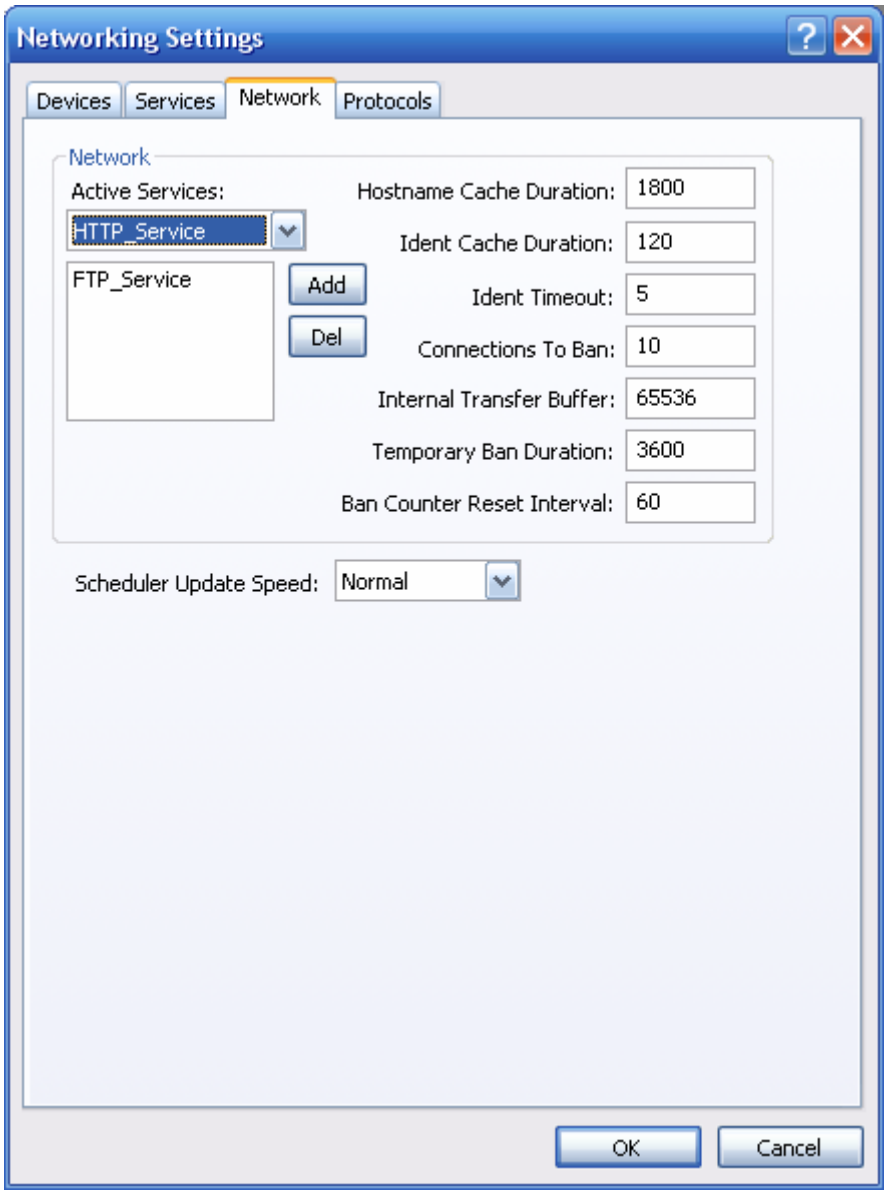
Fastän dialogrutan i Figur 4.4 endast hanterar enheter har vissa inställningar grupperats. Det har jag gjort för att jag anser att de olika inställningarna är olika viktiga: inställningarna i den vänstra rutan måste justeras, medan hastighetsbegränsningarna i den högra är valfria. Genom att gruppera dem förstår också slutanvändaren att inställningarna inom en ruta relaterar till varandra på ett speciellt sätt.

Som det går att läsa i avsnitt 4.1 är enheter, tjänster, sektionen Network samt protokollinställningarna relaterade till varandra. Det är lätt att komma åt de olika inställningarna via flikar.

En stor fördel när det gäller just grafiska applikationer är tydligheten; användaren kan mycket lättare se vad som ska ändras, vilket format texten som ska matas in är (ett ip-nummer exempelvis) o.s.v. [7]

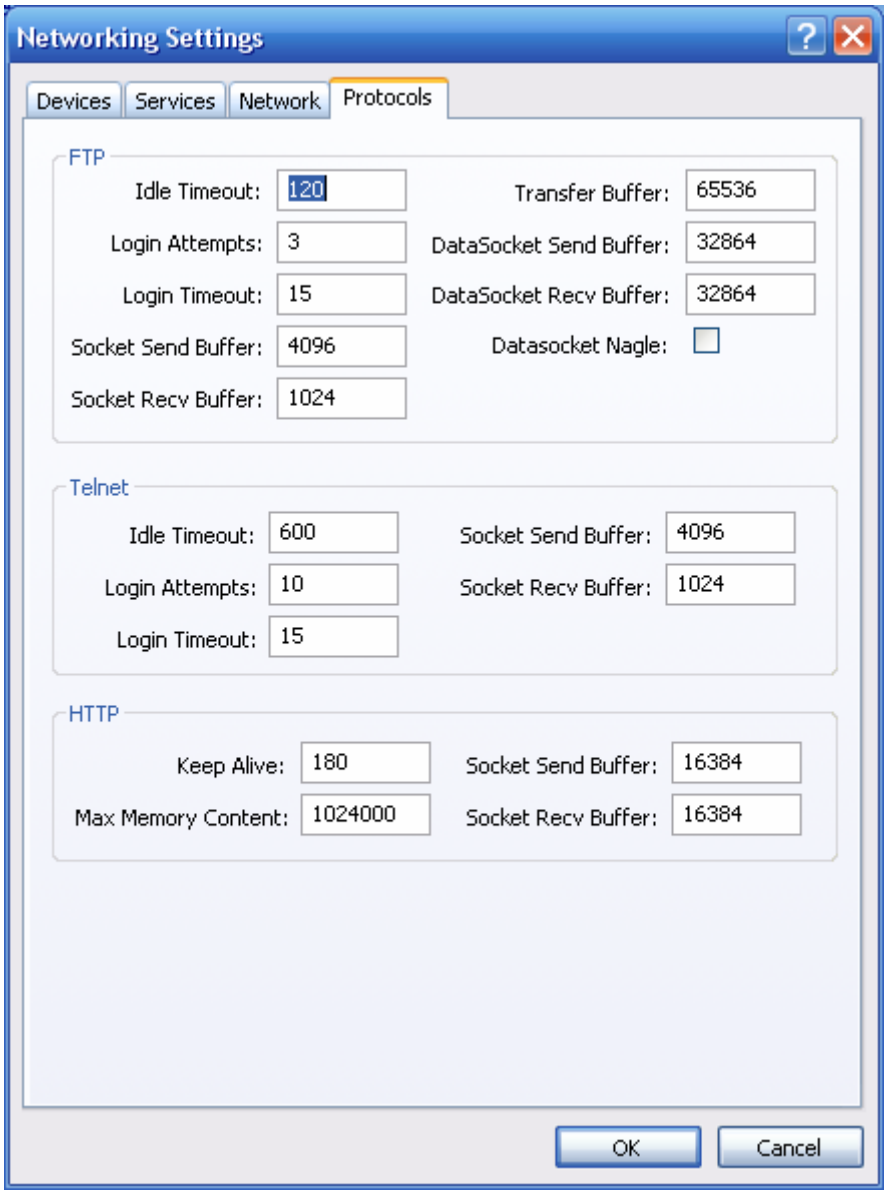

*Figur 4.5: dialogrutan för inställning av olika tjänster*

Grupperingen av dialogrutan i Figur 4.5 var den svåraste av dem alla. Först grupperades allt utom krypteringsinställningarna för sig men den lösningen blev inte bra: det blev stora tomma ytor och det såg ”kladdigt” ut. Bästa lösningen tycker jag är som i bilden ovan med krypteringsinställningarna i nedre vänstra hörnet av ramen och rättigheter och sökvägar, vars fält är långa i jämförelse med de andra, är längst ned. Om användaren väljer att skapa en telnet-tjänst, där krypteringsinställningarna inte används ska krypteringsfälten som inte används för en telnet-tjänst blir inaktiverade.



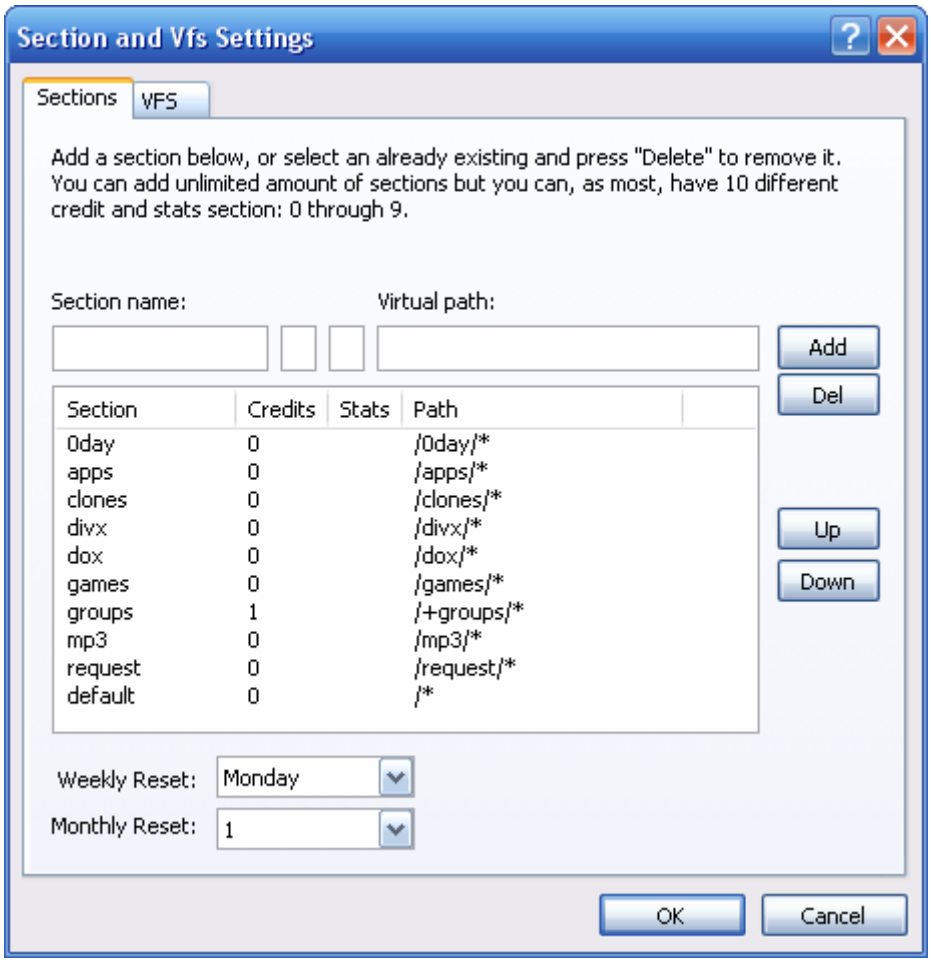

*Figur 4.6: dialogrutan för inställningar för sektionen Network*

Grupperingen för fönstret i Figur 4.6 är mest gjord för att det ska se kosmetiskt vackert ut. Alla inställningar är sådana användaren behöver ändra och således har de placerats ut på ett sätt så att det ser fint ut.



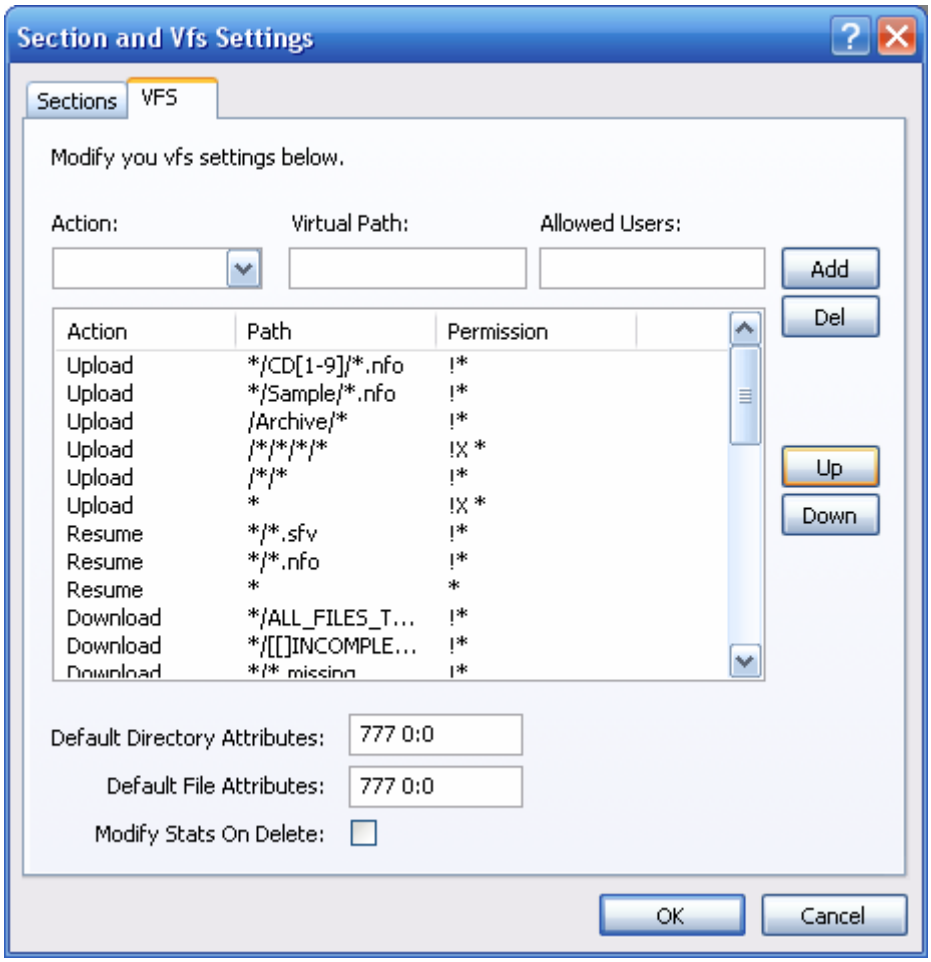

*Figur 4.7: inställningar för de olika protokollen*

I dialogrutan i Figur 4.7 ställer användaren in inställningar för sektionerna *Ftp*, *Telnet* och *Http*. Eftersom det är ganska få inställningar för de sektionerna har alla lagts in i samma dialogruta. Inställningarna för varje tjänst är separerade med ramar så att användaren lätt kan se för vilken tjänst inställningarna gäller.



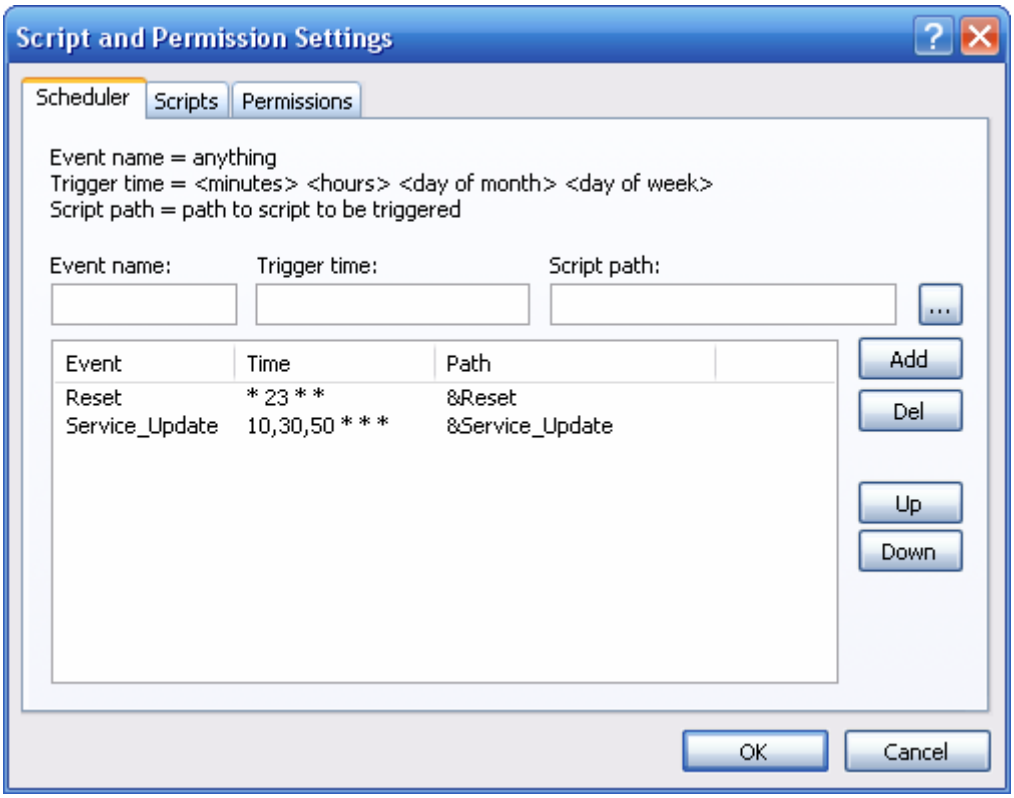

*Figur 4.8: inställningar för sektioner samt för när statistik skall nollställas*

I Figur 4.8, som visar inställningar för sektioner, har även inställningar för när nollställning av statistik lagts in. Eftersom det är här användaren ställer in alla sektionsnamn och deras kredit- och statistiksektioner är det ett lämpligt ställe att placera de inställningarna på.




*Figur 4.9: inställningar relaterade till virtuella sökvägar: specifika rättigheter samt standardrättigheter för nyskapade filer och mappar*

I fönstret som ser ut som i Figur 4.9 ställer användaren in alla inställningar som finns i *Vfs*-sektionen, alltså speciella rättigheter för vissa virtuella sökvägar, standardattribut på nyskapade filer och mappar samt om statistiken för en användare ska påverkas då denne raderar innehåll på servern.

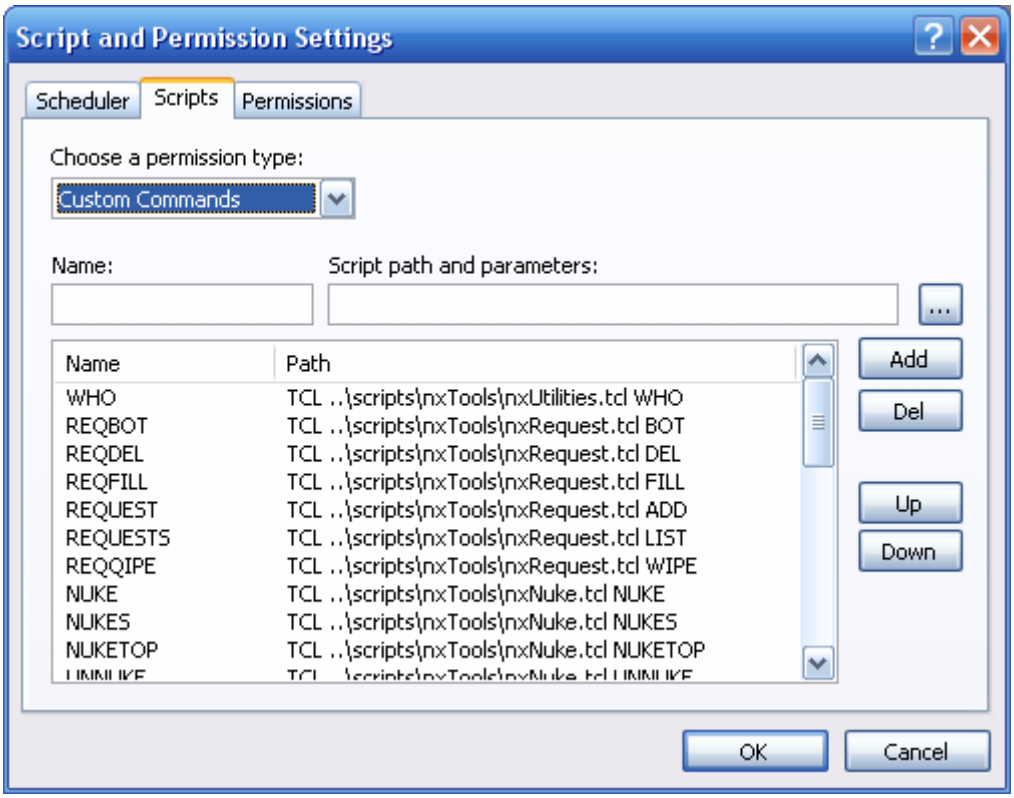



*Figur 4.10: inställningar för schemalagda aktiviteter*

Figur 4.10 visar dialogrutan för inställningar av schemalagda aktiviteter. Det finns en sak när det gäller den grafiska utformningen av denna dialogruta som det inte har lagts så mycket resurser på i början, utan som ska förbättras om det blir tid över, och det är inmatningen av tidpunkterna då schemalagda aktiviteter ska köras. I nuläget måste användaren använda sig av en viss syntax i *Trigger Time*-fältet, samma syntax som används i konfigurationsfilen. Användaren får i nuläget mata in en sträng, separerad av mellanslag, som specificerar vid vilken tidpunkt ett skript ska exekveras. Exempel på två schemalagda aktiviteter ser vi i Figur 4.11, där den första körs varje dag kl. 23.00 GMT och den andra var 20:e minut. &Reset och &Service_Update är inbyggda operationer i ioFTPD.

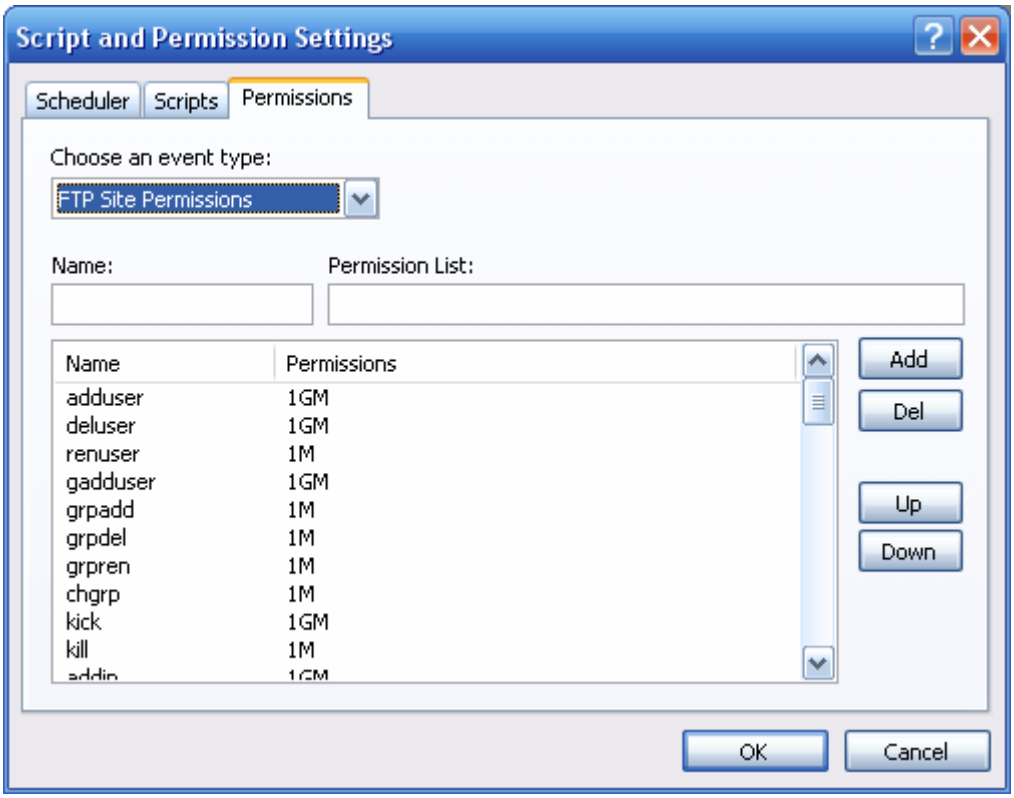
```
1  [Scheduler]
2  Reset           = * 23 * *      &Reset
3  Service_Update  = 10,30,50 * * *    &Service_Update
4
```

*Figur 4.11: exempel på två schemalagda aktiviteter*




*Figur 4.12: inställningar för olika skriptinställningar*

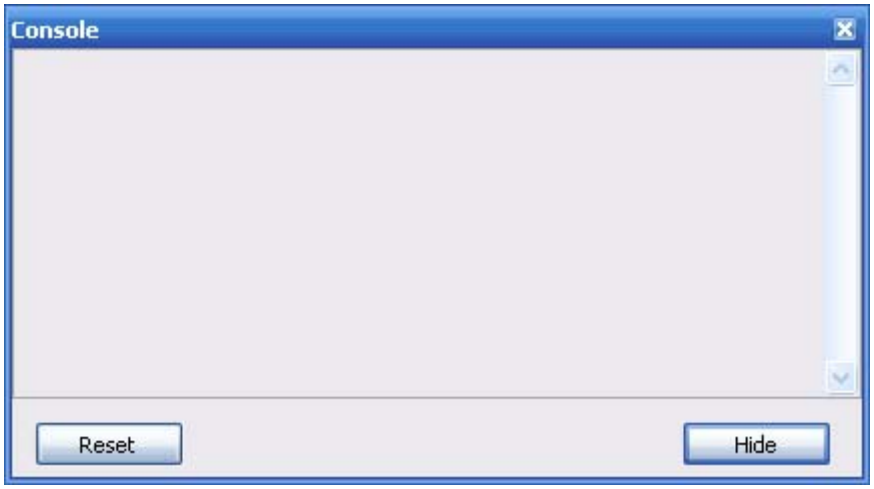
I den här dialogrutan (Figur 4.12) lägger användaren till alla skript som ska köras vid olika tillfällen, alla skript utom de schemalagda som ställs in i föregående flik. Genom rullningslistan längst upp väljer användaren vad det är för typ av skript som ska redigeras.




*Figur 4.13: inställningar för rättigheter för olika sektioner*

I detta fönster (Figur 4.13) ställer användaren in rättigheter för skript som lagts till på fliken *Scripts*. Valet av vilken sorts rättigheter som ska ändras görs via rullningslistan längst upp.

Ett icke-modalt fönster som körs i bakgrunden har också skapats. Syftet med det fönstret, som jag valt att kalla för en konsoll, är att visa meddelanden då fel uppstår, exempelvis vid problem av inläsning av en konfigurationsfil. Då ett felmeddelande visas blir fönstret synligt.




*Figur 4.14: konsolfönstret som visar felmeddelanden exempelvis vid inläsning av en felaktig konfigurationsfil*

# 5 Implementation

Implementationen av ioFTPD-programmen har skett i Windowsmiljö med utvecklingsverktyget Visual Studio 2005 och i programspråket C++. Visual Studio har jag ganska stor erfarenhet av och programmet erbjuder både en kraftfull resurseditor för skapande av dialogrutor samt en mycket bra debugger som underlättar felsökning. I resurseditorn skapas dialogfönster, menyer, ikoner etc. Visual Studios debuggern har använts för att felssöka rutiner som inte fungerat som önskat. Man kan, i Visual Studio, sätta så kallade breakpoints; när programmet kommer till en breakpoint stannas exekveringen och användaren kan stega vidare, rad för rad, för att felsöka, eller välja att fortsätta exekveringen som vanligt.

Språket C++ valdes för att det är ett kraftfullt språk som erbjuder bra prestanda och stora utvecklingsmöjligheter. C++ började utvecklas i början av 1980-talet av Bjarne Stroustrup och erbjuder ett antal egenskaper för förbättring i jämförelse med C – framförallt stöd för objektorienterad programmering [8]. Genom att använda sig av objekt är det möjligt att återanvända vanliga komponenter, vilket gör programmeringen mer effektivt [8]. Eftersom C++ bygger på C är det även möjligt att använda alla de egenskaper som C erbjuder.

Många böcker som jämför C++ med sin föregångare C nämner möjligheten till objektorienterad programmering som den största – och bästa – skillnaden.

Det här kapitlet beskriver, på en lägre abstraktionsnivå än kapitel 4, hur konfigurationsprogrammet är uppbyggt. Kapitlet beskriver bland annat hur inläsning och skrivning till fil går till, hur inläst data lagras och uppbyggnaden av varje dialogruta.

## 5.1 Översiktlig implementation av GUI:et

För att konfigurera de olika sektionerna i konfigurationsfilen har ett antal dialogrutor skapats. Dialogrutorna är skapade i Visual Studios resurseditor. För att tillhandahålla det tabbade gränssnittet, där användaren väljer dialogfönster via flikar, har så kallade *property sheets* använts. En *property sheet* består av minst en dialogruta – i den här implementationen användes dock inte property sheets då endast ett fönster visas, utan då visas det fönstret med

hjälp av *DialogBox*-funktion som tillhandhålls i API:n. Då en *property sheet* består av mer än en dialogruta kan användaren växla mellan dem via flikar, se Figur 5.1.


*Figur 5.1: En property sheet som består av fyra dialogrutor. Bilden visar endast flikarna.*

Eftersom *property sheets* skapas med hjälv av inbyggda API-funktioner underlättar det arbetet avsevärt jämfört med om det tabbade gränssnittet skulle implementeras med hjälp av så kallade *tab controls*, då man själv skulle behövt specificera vilken dialog som är kopplad till vilken flik, vilken rutin som ska hantera dess meddelanden o.s.v.

Varje dialogruta som användaren öppnar från huvudfönstret är modal, vilket innebär att användaren inte kan öppna flera likadana dialogrutor samtidigt. Genom att låta dialogrutorna som öppnas vara modala förhindrar det användaren från att eventuellt ändra samma inställningar från två eller flera fönster, vilket skulle kunna försätta programmet i ett inkonsistent läge.

## 5.2 Klasser

Alla inställningar sparas i olika klasser vilket innebär att jag har en klass för varje sektion. Vissa sektioner är små, exempelvis *Reset*, och det kan tyckas onödigt att spara de inställningarna i en egen klass. Genom att konsekvent använda en klass per sektion blir det lättare att underhålla, ifall det i framtida versioner tillkommer nycklar eller nya sektioner. För att det ska vara lätt att hålla reda på vilken klass som motsvarar vilken sektion heter de Io[sektionsnamn], med vissa modifikationer till de sektioner med väldigt långa namn som exempelvis *FTP_Pre-Command_Events* och *FTP_Post-Command_Events*. Eftersom sektionerna för skript och rättigheter har samma struktur är de instanser av IoScripts och IoPermissions respektive.

Alla klasser som inte har ett specifikt sektionsnamn med ett specifikt antal nycklar använder sig av länkade listor. Detta för att bland annat inte slösa med minnesutrymme i onödan. Exempel på sådana klasser är *IoDevices* och *IoServices*. Nedan följer en lista på klasser och de sektionerna de motsvarar. För en mer detaljerad beskrivning om vad varje sektion gör, se avsnitt 2.3.

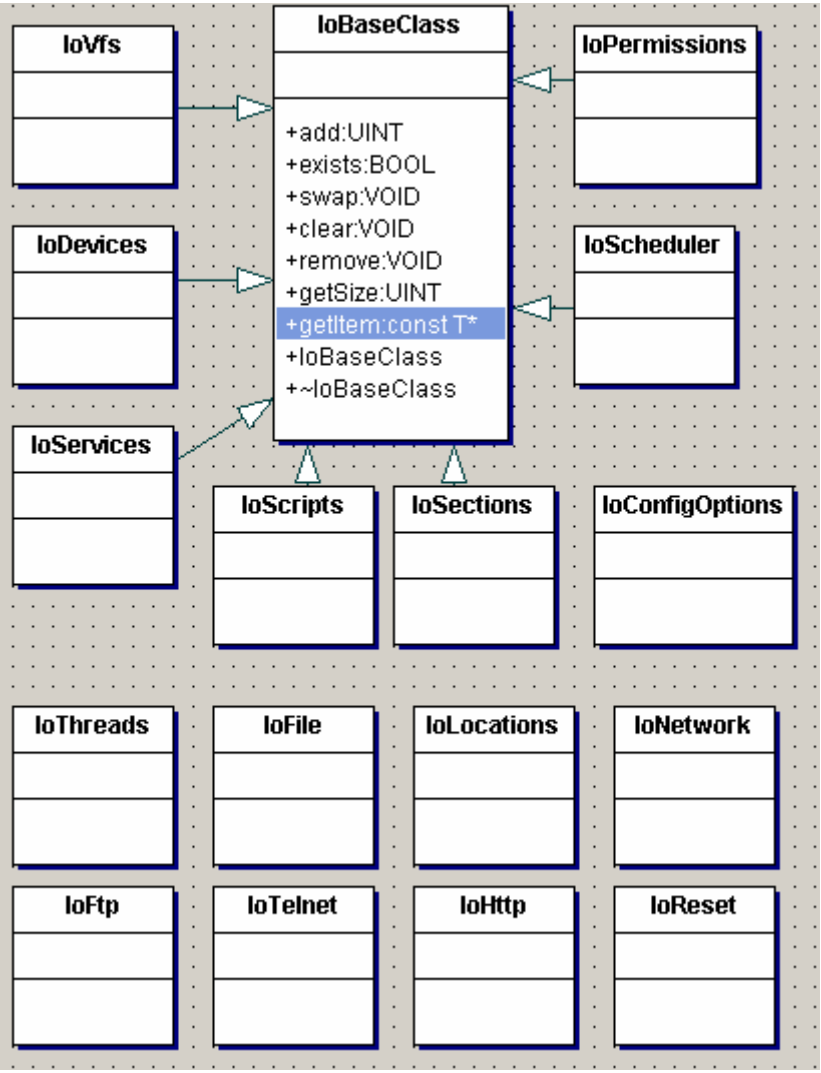
För att lagra alla inställningar har globala instanser av klasserna skapats. Att ha dem globala har underlättat åtkomsten av dem.

### 5.2.1 Klassdiagram

Alla klasser som använder sig av länkade listor ärver från klassen IoBaseClass som tillhandahåller den grundläggande funktionaliteten för manipulation av listorna, se Figur 5.2. Varje subklass tillhandahåller sina egna ”getters” och ”setters” vid behov. En getItem-funktion har implementerats i IoBaseClass för att klienten snabbt ska kunna få tillgång till en hel nod, utan att behöva använda sig av get-metoder för varje datamedlem i noden. I koden nedan visas deklarationen av basklassen IoBaseClass.

```
template <class T>
class IoBaseClass {
public:
   IoBaseClass();
   virtual ~IoBaseClass();

   UINT     add(T* newNode);
   UINT     getSize() CONST;
   BOOL     exists(UINT pos) CONST;
   VOID     swap(UINT nFirst, UINT nSecond);
   VOID     clear();
   VOID     remove(UINT pos);
   // getItem is used to return a constant pointer to the struct at position nPos
   CONST T* getItem(UINT nPos) CONST;

protected:
   T*   _head;
   UINT _size;

   // Only used internally
   T*   getNode(UINT pos) CONST;
};
```

Anledningen till att datamedlemmarna är protected är för att subklasser som ärver ska kunna implementera sina egna set-funktioner.




*Figur 5.2: klassdiagram över alla klasser*

### 5.2.2 IoBaseClass

IoBaseClass är en templateklass som utgör basen för många av de andra klasserna (se klassdiagrammet i Figur 5.2). Klassen är basklass för alla klasser som använder sig av länkade listor.

### 5.2.3 IoThreads, IoFile och IoLocations

Sparar information för deras respektive sektion. Inställningarna för dessa klasser matas in via General Settings-dialogrutan, se Figur 4.3.

### 5.2.4 IoDevices, IoServices, IoNetwork, IoFtp, IoTelnet och IoHttp

IoDevices sparar alla enheter som finns och IoServices sparar alla tjänster. De andra fyra klasserna sparar inställningar för deras respektive sektion. Alla inställningar för dessa klasser

kommer man åt via en dialogruta med flikar. IoFtp, IoTelnet och IoHttp har specifika inställningar för deras respektive tjänster, bl.a. bufferstorlekar, timeout-tider etc.

### 5.2.5 IoSections, IoVfs och IoReset

IoSections-klassen har hand om de sektionsnamn man get virtuella sökvägar på sin ftp-server. IoVfs har hand om rättigheter för virtuella sökvägar och IoReset specificerar när statistik för ska nollställas.

### 5.2.6 IoSecheduler, IoScripts

De här klasserna lagrar information om olika skript. IoScheduler lagrar inställningar för schemalagda aktiviteter medan instanser av IoScripts lagrar inställningar för sektionerna Events, Modules, FTP_Custom_Commands, FTP_Pre-Command_Events, FTP_Post-Command_Events och Telnet_Binarier.

### 5.2.7 IoPermissions

Det finns lika många instanser av IoPermissions-klassen som det finns av IoScripts-klassen (se föregående avsnitt). Instanserna av IoPermissions hanterar rättigheterna för kommandona som är specificerade i IoScripts-instanserna.

### 5.2.8 IoConfigOptions

IoOptions är en klass som hanterar inställningar för konfigurationsverktyget. Bland annat vilken fil som är öppnad och ifall användaren vill att den senast öppnade filen automatiskt ska laddas då konfigurationsprogrammet startar.

## 5.3 Filstruktur

Varje klass är deklarerad i *classes.h*. Varje klass är definierad i sin egen cpp-fil som heter Io[klassnamn].cpp. Det har också skapats egna källkodsfiler som har liknande namn som de olika dialogrutorna som finns. I dessa filer finns specifika funktioner för dialogerna de representerar definierade, bland annat för initiering av dialogrutan, sparning och laddning av dess värden etc. Delade funktioner, som används av flera olika operationer, är definierade i sin egen källkodsfil.

I filen *defines.h* är enumeratorer och konstanter definierade. Enumeratorerna representerar värdena som finns i olika rullningslistor, exempelvis prioritetsvalet i General Settings-

dialogen eller tjänsttyp och krypteringsprotokoll i Services-dialogen. Konstanterna som är definierade representerar bland annat inställningarna i konfigurationsfilen: nyckelnamn och standardvärde.

Filen *structs.h* innehåller strukturer som används i de länkade listorna.

## 5.4 Läsning från och skrivning till konfigurationsfil

Efter att användaren har valt vilken fil som ska användas för inläsning sker inläsningen med hjälp av funktioner inbyggda i Windows API.

Lagringen av all data som läses in lagras i de globala instanserna av klasserna som finns.

Det finns många funktioner skapats för inläsning av konfigurationsfilen, det finns en inläsningsfunktion per dialogruta. Funktionerna läser respektive sektioner i filen och lagrar datat i instanserna av klasserna som används. Att ha en inläsningsfunktion per dialog istället för sektion är något som gjordes i ett ganska tidigt skede och bryter lite mot principen när det gäller exempelvis klasserna – att ha en klass per sektion. Det är dock inget som har orsakat några problem, men det är ändå något som eventuellt kommer att förändras i framtiden.

Till skillnad från inläsning finns det funktion per sektion när det gäller skrivning till fil.

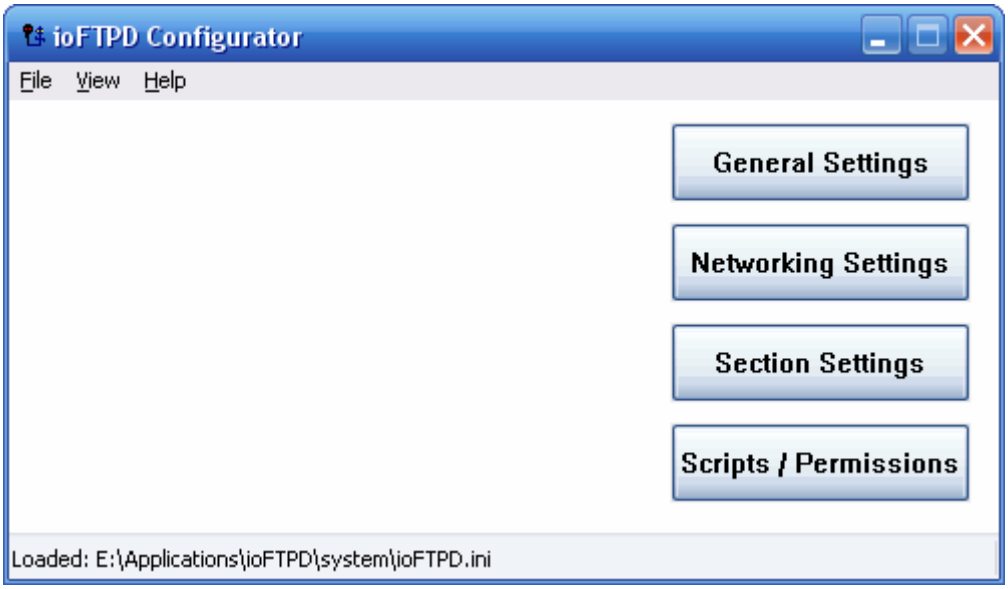
Användaren kan välja två sätt att skriva till fil: antingen genom menyvalet File-Save eller genom menyvalet File-Save As. Då användaren väljer *Save* sker skrivningen till den senast öppnade filen, som finns lagrad i instansen av IoConfigOptions-klassen, annars får användaren själv specificera sökväg och filnamn.

Skrivning av data till fil sker också genom funktioner som finns i Windows API. Det finns API-funktioner för uppdatering av nycklar samt för att skriva om hela sektioner, men ingen funktion för att ta bort hela sektioner, vilket medförde ett litet problem: eftersom användaren har möjligheten att ta bort enheter och sektioner behöver även deras motsvarande värden i konfigurationsfilen tas bort. Eftersom jag inte skrivit någon egen parser för skrivning till fil får användaren välja om den existerande filen ska uppdateras eller helt skrivas om. Väljer

användaren att helt skriva om filen försvinner eventuella kommentarer som fanns sedan tidigare. Väljer användaren att endast uppdatera filen innebär det att eventuella enheter och tjänster som tagits bort fortfarande finns kvar i konfigurationsfilen och även dyker upp i konfigurationsprogrammet då inläsning av konfigurationsfilen sker på nytt.

## 5.5 Konstruktion av dialogrutor

Som det står nämnt i avsnitt 4.3 öppnas alla dialogfönster från programmets huvudfönster, som i nuläget ser ut som i Figur 5.3. Implementationen av huvudfönstret är inte helt slutförd: bland annat saknas sammanfattningstext till vänster om knapparna som beskriver delar av den öppnade konfigurationsfilen, bland annat antal tjänster och vilka tjänster som är aktiva.

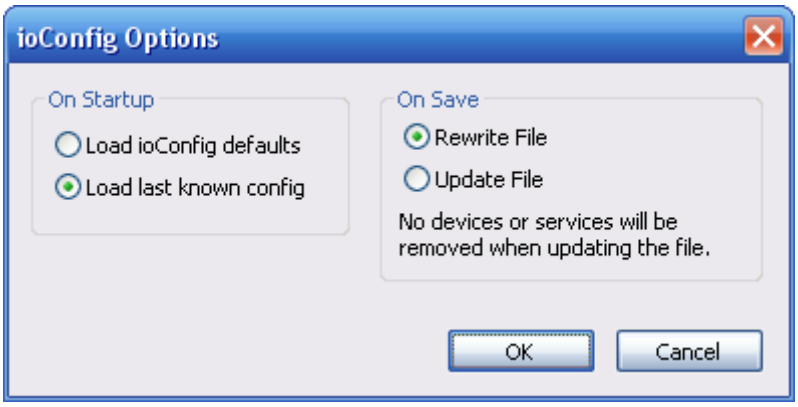



*Figur 5.3: bild på huvudfönstret*

Via de olika menyerna kan användaren läsa in eller spara en fil, ställa in inställningar för konfigurationsprogrammet samt se en *About*-ruta där det står vilken version av programmet det är samt vem som utvecklat det.

Längst ned har också en statusrad lagts till som visar relevant information, exempelvis vilken fil som har laddats eller till vilken fil användaren valt att spara inställningarna.

Via menyn View-Options kommer användaren åt inställningar för konfigurationsprogrammet, se Figur 5.4, där denne kan ställa in vad som ska ske när programmet startas och hur sparningen av filer ska ske, läs avsnitt 5.4.




*Figur 5.4: inställningar för konfigurationsprogrammet*

Då programmet startas körs bland annat följande kod som kontrollerar om den senast öppnade filen ska laddas, eller om programmet ska använda sig av standardvärden.

```
switch (Options.getStartupType()) {
   case STARTUP_LAST:
      if (ReadAllSettings(Options.getLastConfig())) {
         size_t len = wcslen(Options.getLastConfig()) + 9; // "Loaded: " = 9 chars
         LPWSTR wsText = new WCHAR[len];
         Options.setOpened(TRUE);
         wcscpy_s(wsText, len, L"Loaded: ");
         wcscat_s(wsText, len, Options.getLastConfig());
         SendMessage(hStatusBar, SB_SETTEXT, 0, (LPARAM)wsText);
         delete wsText;
      }
      break;
   // case STARTUP_DEF:
      // Do nothing since the constructors of the classes have set default values
      // break;
}
```

### 5.5.1 General Settings

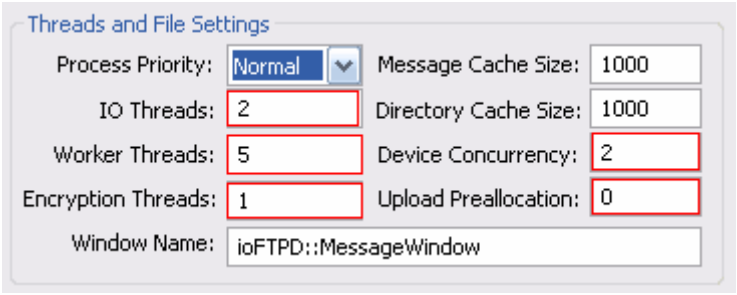
I Figur 4.3 kan man se hur dialogrutan för ”General Settings” ser ut. Dialogen är uppbyggd av en lista, ett antal inmatningsrutor samt ett par knappar.

Listan består av fyra alternativ: *Idle*, *Normal*, *High* och *Realtime*. Värdena representerar processprioriteten ioFTPD ska ha när den är aktiv och är representerade av enumeratorn *PriorityType*, se Figur 5.5. Standardalternativet för listan är *Normal* och det alternativet används, bland annat, då programmet vid inläsning av konfigurationsfilen inte hittar ett korrekt värde.

```
19  // ioFTPD's process priority
20  enum PriorityType {
21      PRIORITY_IDLE,
22      PRIORITY_NORMAL,
23      PRIORITY_HIGH,
24      PRIORITY_REALTIME
25  };
```

*Figur 5.5: figuren visar enumeratorn vars värden representerar de som användaren kan välja i rullningslistan*

Klasserna IoThreads och IoFile har några datamedlemmar av typen unsigned short, se Figur 5.6, d.v.s. osignerade 16-bitars heltal. Maxvärdet de datamedlemmarna kan hålla är 65535 och för att inte användaren ska mata in felaktiga värden, som inte ryms i datamedlemmarna har det implementerats kontroller som förhindrar detta. Överskrider värdet det maximala tillåtna ändras det till 65535. Kontrollen görs då ett EN_CHANGE-meddelande skickas. På detta sätt har användaren en garanti på att det är det angivna värdet som gäller.



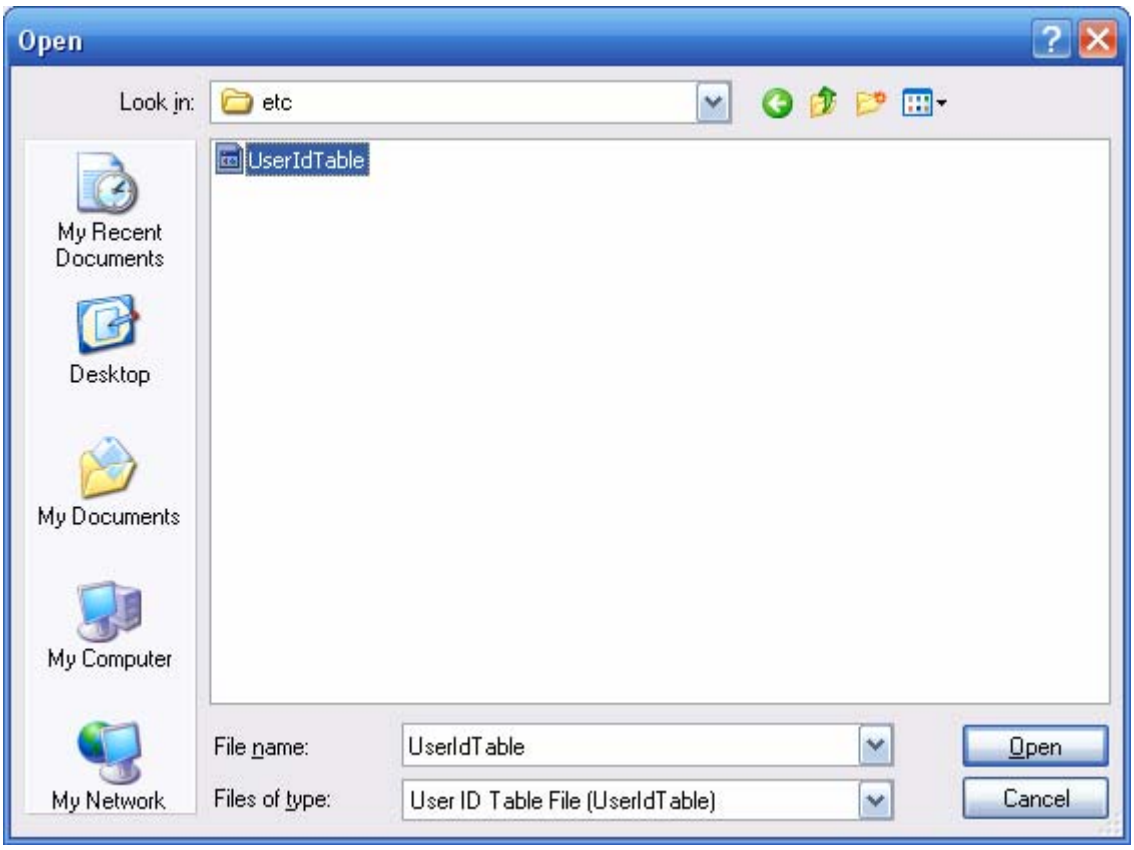

*Figur 5.6: i inmatningsrutorna som är markerade med rött tillåts endast osignerade 16-bitars heltal*

Nedan visas koden som körs då användaren ändrar värdet i en inmatningsruta.

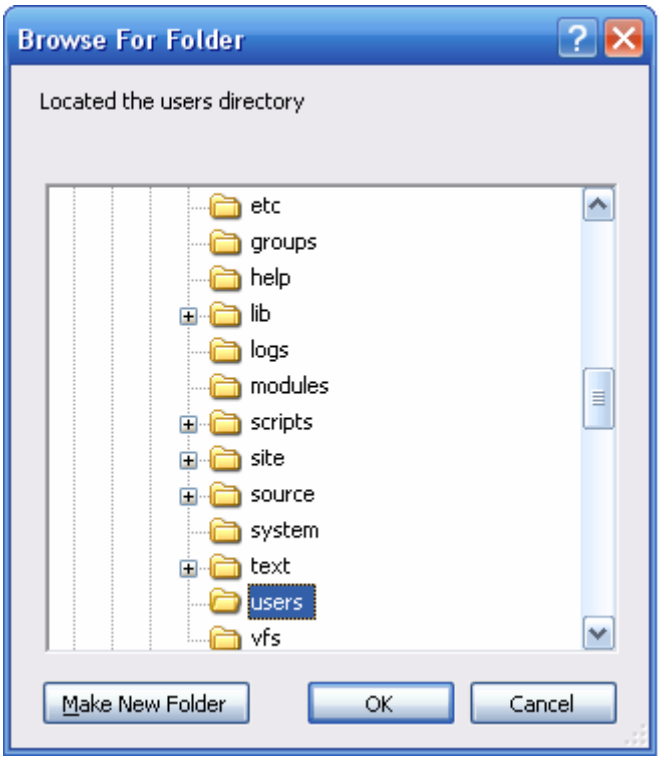
```
case EN_CHANGE: {
   WORD hId =  LOWORD(wParam);
   if (hId == IDC_EDIT_IOTHREADS || hId == IDC_EDIT_WORKERTHREADS
       || hId == IDC_EDIT_ENCRTHREADS || hId == IDC_EDIT_DEVICECONCURRENCY
       || hId == IDC_EDIT_UPLOADPREALLOC) {

      // Get current value
      UINT nValue = GetDlgItemInt(hDlg, hId, NULL, FALSE);

      // If the user has entered a number that is too big, then change to unsigned short's
      // maximum allowed, which is 65535
      if (nValue > 65535) SetDlgItemInt(hDlg, hId, 65535, FALSE);
   }
   break;
}
```

Inmatningsrutorna för de olika sökvägarna, som är inramade i ramen ”Locations” är begränsade till MAX_PATH, vilket är en Windows-begränsning för icke-Unicode-applikationer. Till höger om varje inmatningsruta för sökvägar finns en knapp som i sin tur

öppnar ett fönster där användaren kan klicka sig fram till eventuell fil eller katalog som efterfrågas, se Figur 5.7 och Figur 5.8.




*Figur 5.7: fönstret som öppnas då användaren trycker på knappen till höger om inmatningsrutan för User Id Table*




*Figur 5.8: fönstret som öppnas då sökväg till en katalog förväntas*

### 5.5.2 Networking Settings

Networking Settings heter fönstret med flikar för åtkomst av enheter, tjänster samt inställningar för sektionerna *Network*, *Ftp*, *Telnet* och *Http*.

För att spara *enheter* och *tjänster* används länkade listor. Det görs för att dels utnyttja minnet effektivt samt för att inte sätta in begränsningar i användargränssnittet som inte finns vid manuell konfiguration. Eftersom det inte finns någon begränsning vid manuell editering om hur många enheter eller tjänster man kan lägga till så känns det naturligt att inte heller ha någon sådan begränsning i konfigurationsverktyget – och därför används länkade listor.

Källkoden för deklarationen av template-klassen IoBaseClass, som är basklassen för de länkade listorna, finns att läsa i avsnitt 5.2.1.

### 5.5.3 Devices

Devices-dialogrutan, se Figur 4.4, är bland annat uppbyggd av en rullningslista, två ip-nummerrutor, två knappar samt ett antal inmatningsrutor.

I rullningslistan får användaren välja vilken enhet denne vill ändra värden för. Så fort värdet för någon inställning ändrats blir det möjligt att trycka på knappen ”Save”. Trycker användaren på knappen lagras de nya inställningarna.

Då något värde förändras blir knappen som gör det möjligt att spara aktiv:

```
EnableWindow(GetDlgItem(hDlg,IDSAVE), TRUE); // Enables the Save button
```

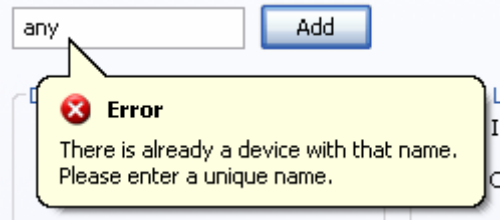
I min konstruktion av länkad lista har första elementet position 0, således är varje enhet som finns lagrad representerad av ett nummer större än eller lika med noll. När en enhet sparas då användaren trycker på sparaknappen kollar programmet upp vilket ID enheten med namnet som är valt i rullningslistan har och lagrar sedan inställningarna på den positionen. Nedanstående kod körs då användaren trycker på sparaknappen.

```
case IDSAVE: // Save settings and disable the Save button
   SaveDevice(hDlg); // Saves the settings
   EnableWindow(GetDlgItem(hDlg,IDSAVE), FALSE);
   break;
```

Inmatningsrutorna för hastigheter är inte nödvändiga att ge värden; sätts inga värden, eller om värdena är 0, skrivs inte de inställningarna ut i konfigurationsfilen. IoFTPD tolkar nämligen värdena 0 som ”ingen begräsning”, vilket även är resultatet då man utelämnar de nycklarna i konfigurationsfilen.

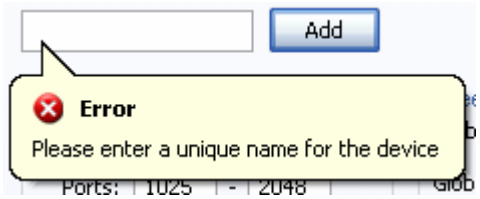
Då användaren väljer att skapa en ny enhet skrivs namnet i inmatningsrutan och användaren trycker på *Add*. Finns det redan en enhet med samma namn visas meddelandet i Figur 5.9 och om användaren försöker lägga till en enhet utan att ange namn visas meddelandet i Figur 5.10. Meddelanden visas med så kallade ”balloon tooltip”. Lite pseudokod på implementationen ses nedan:

```
if (name already exists)
   Edit_ShowBalloonTip(GetDlgItem(hDlg,IDC_EDIT_NEW), &tooltip);
```




*Figur 5.9: meddelandet som visas då man försöker skapa en enhet med ett namn som redan finns på en enhet*

En stor fördel med grafiska applikationer är möjligheten att ge användaren vettiga meddelanden till varför något inte fungerar, exempelvis i fallet ovan orsaken till varför det inte går att lägga till en enhet, istället för ett kryptiskt felmeddelande som exempelvis ”error 147” [7].




*Figur 5.10: meddelandet som visas då användaren försöker lägga till en enhet utan namn*

Då enheter sparas till fil skrivs endast relevanta inställningar ut. De hastighetsbegränsningar som inte är definierade skrivs inte ut.

### 5.5.4 Services

Eftersom en tjänst antingen är av typen ftp, telnet eller http finns det en lista där användaren får välja vilken typ av tjänst det är man arbetar med. Vissa inställningar gäller bara för vissa tjänster; exempelvis är säkerhetsinställningarna meningslösa vid skapandet av en telnet-tjänst. För att användaren inte ska kunna ändra onödiga inställningar, inaktiveras de inställningarna som inte är relevanta för en viss tjänst (se Figur 5.11 och koden efter detta stycke). Jämför med Figur 4.5 där man ser ett exempel på en ftp-tjänst där alla inställningar är möjliga att ändra. Det är bara telnet-tjänsten som inte använder sig av alla inställningar. Nedanstående kod ställer in Service-dialogrutan för om det är en telnet-tjänst eller inte.

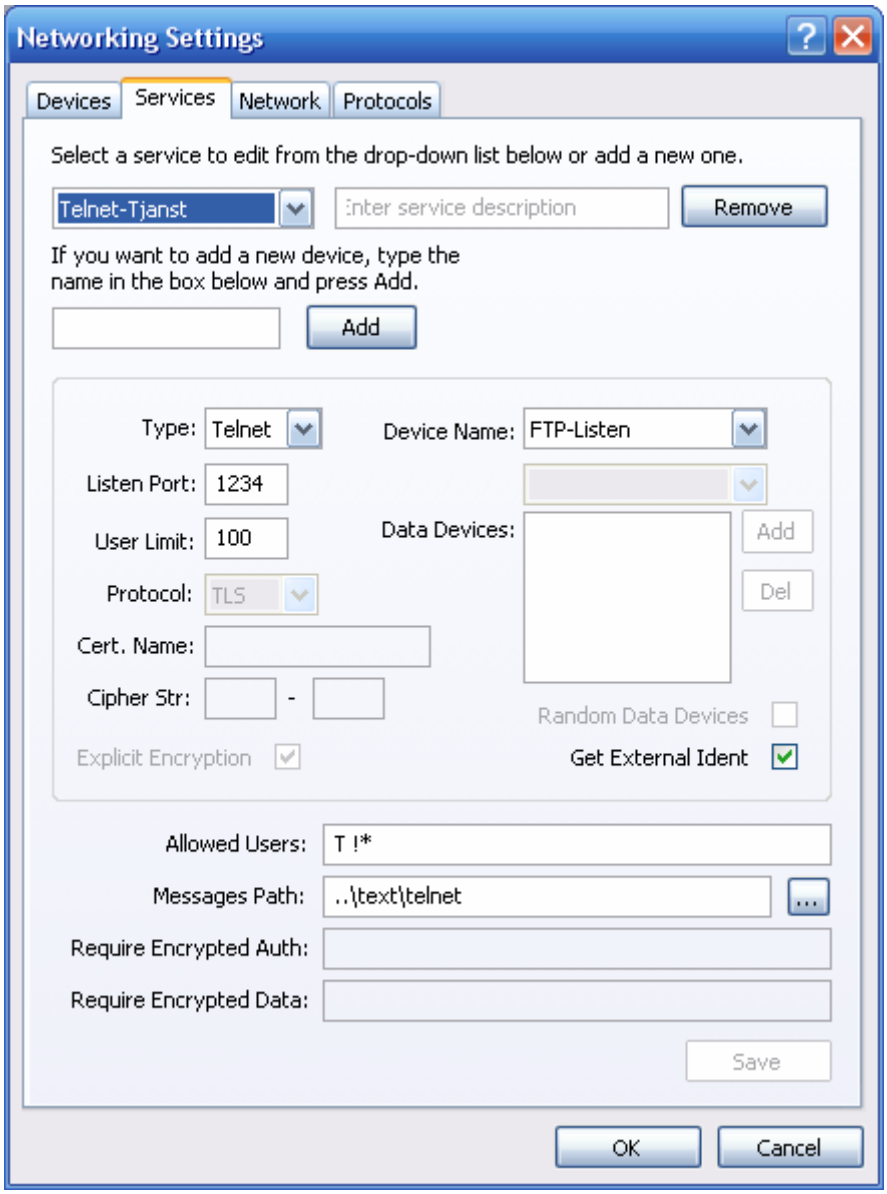
```
// If it's not Telnet, make sure the options are active
if (SendMessage(GetDlgItem(hDlg,IDC_COMBO_TYPE),CB_GETCURSEL,0,0) != TYPE_TELNET)
   setIsTelnetService(hDlg, FALSE);
else // If it is telnet, then disable encryption options
   setIsTelnetService(hDlg, TRUE);
```

Eftersom säkerhetsinställningarna inte är obligatoriska för ftp- och http-tjänster är det möjligt att utesluta dem; det gör man genom att lämna certifikatnamnrutan tom.

Då användaren väljer en tillgänglig tjänst laddas den tjänstens data in i dialogen. Är det en telnet-tjänst laddas dock inga säkerhetsvärden, utan de fälten sätts tomma och inaktiverade. När användaren ändrar alternativ i rullningslistan körs koden nedan.

```
case IDC_COMBO_LIST: // When the user selects a service from the drop-down list
   if (SendDlgItemMessage(hDlg, LOWORD(wParam), CB_GETLBTEXT, (WPARAM)nIndex,
        (LPARAM)buffer) != CB_ERR) {
      UINT nId = Services.getServiceIdByName(buffer);

      if (nId != INVALID_SERVICE) {
         // Update the dialog with the service's content
         updateServiceDialogContent(hDlg, Services.getItem(nId));

         // Since the functions above has loaded a service we disable the save button
         EnableWindow(GetDlgItem(hDlg,IDSAVE), FALSE);
      }
   }
   break;
```

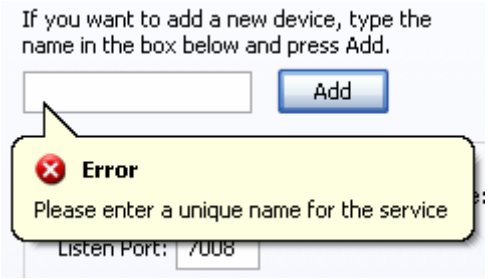
När tjänsterna sparas till fil kontrolleras vilken typ det är och ifall certifikatnamnet är satt. Nödvändiga inställningar skrivs sedan till konfigurationsfilen.




*Figur 5.11: dialogrutan för en telnettjänst*

Då användaren väljer att skapa en ny tjänst skrivs namnet på den nya tjänsten i inmatningsrutan och användaren trycker sedan på Add. Skulle det redan finnas en tjänst med samma namn informeras användaren om det genom ett meddelande (se Figur 5.12), och får sedan mata in ett annat namn. Trycker användaren på lägg till-knappen utan att ha matat in något namn visas också ett meddelande (se Figur 5.13). Koden för enheter är identisk med den för tjänster, det enda som skiljer är texten som visas.


*Figur 5.12: meddelandet som visas då användaren försöker skapa en tjänst med ett namn som redan existerar på en tjänst*




*Figur 5.13: meddelandet som visas då användaren försöker att lägga till en tjänst utan att ange ett namn*

### 5.5.5 Network

Network-dialogrutan, se Figur 4.6, motsvarar konfigurationsfilens *Network*-sektion. Användaren får, i en lista, lägga till eller ta bort tjänster som ska vara aktiva då ioFTPD-processen startas. I Figur 4.6 är det endast en ftp-tjänst som är aktiv.

Aktiva tjänster radas upp i konfigurationsfilen skiljda åt med ett mellanslag; det är på samma sätt aktiva tjänster lagras i konfigurationsprogrammet: när en användare sparar Network-inställningarna lagras de aktiva tjänsterna i en sträng.

När användaren sparar nätverksinställningar körs nedanstående kod för att lagra aktiva tjänster.

```
// Saving active services
lRes = SendDlgItemMessage(hDlg, IDC_LIST_SERVICES, LB_GETCOUNT, 0, 0);
if (lRes != CB_ERR) {
   buffer[0] = L'\0';
   for (LRESULT i=0; i<lRes; i++) {
      SendDlgItemMessage(hDlg, IDC_LIST_SERVICES, LB_GETTEXT, (WPARAM)i, (LPARAM)bufservice);
      if (i != 0) wcscat_s(buffer, MAX_PATH, L" ");
      wcscat_s(buffer, MAX_PATH, bufservice);
   }
   Network.setActiveServices(buffer);
}
```

### 5.5.6 Protocols

Denna dialog, se Figur 4.7, är endast uppbyggd av inmatningsrutor och en kryssruta. Varje protokolls inställningar är inramade och dess data lagras i sin respektive klass – som är en instans av IoFtp, IoTelnet eller IoHttp.

Vid skrivning till fil tas värdena från respektive klass och skrivs ut till deras sektioner.

### 5.5.7 Section and Vfs Settings

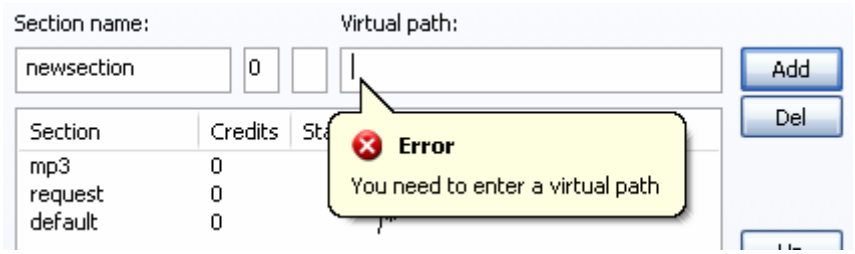
Den här property sheeten har hand om två stycken dialoger: Sections (Figur 4.8) och VFS (Figur 4.9). I Sections-dialogen kan användaren redigera sektioner som finns på servern samt ställa in när statistik ska nollställas.

De olika sektionerna lagras i en lista och användaren kan med hjälp av olika knappar ändra ordningen på sektionerna – ordningen är viktig eftersom det är första träffen som ioFTPD hittar som används för kredit och statistik.

Liksom enheter och tjänster visas ett antal felmeddelanden för användaren om denne försöker lägga till en sektion utan att fylla i nödvändiga detaljer, se Figur 5.14 och Figur 5.15. Koden nedan visar vad som sker då användaren lägger till en ny sektion:

```
case IDC_BUTTON_ADD: {
   EDITBALLOONTIP tooltip;
   BOOL bAdd = FALSE; // Whether or not we've got enough info to save the new section
   tooltip.cbStruct = sizeof(EDITBALLOONTIP);
   tooltip.ttiIcon = TTI_ERROR;
   tooltip.pszTitle = L"Error";

   if (SendDlgItemMessage(hDlg,IDC_EDIT_SECTION,WM_GETTEXTLENGTH,0,0) == 0) {
      tooltip.pszText = L"You need to enter a section name";
      Edit_ShowBalloonTip(GetDlgItem(hDlg,IDC_EDIT_SECTION), &tooltip);
   }
   else if (SendDlgItemMessage(hDlg,IDC_EDIT_PATH,WM_GETTEXTLENGTH,0,0) == 0) {
      tooltip.pszText = L"You need to enter a virtual path";
      Edit_ShowBalloonTip(GetDlgItem(hDlg,IDC_EDIT_PATH), &tooltip);
   }
   else bAdd = TRUE;

   // Save the new section
   if (bAdd) {
      AddSectionLViewItem(hDlg);
      // Clear the controls
      SetDlgItemText(hDlg, IDC_EDIT_SECTION, L"");
      SetDlgItemText(hDlg, IDC_EDIT_CREDIT, L"");
      SetDlgItemText(hDlg, IDC_EDIT_STATS, L"");
      SetDlgItemText(hDlg, IDC_EDIT_PATH, L"");
   } // Add section to list-view and class
   break;
}
```


*Figur 5.14: meddelandet som visas då användaren försöker lägga till en sektion utan att ange ett sektionsnamn*



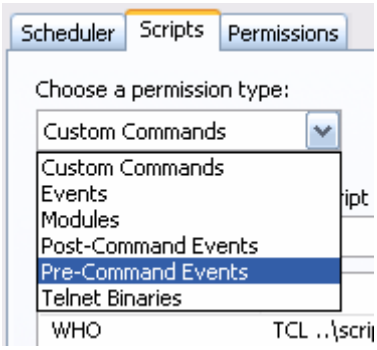

*Figur 5.15: meddelandet som visas då användaren försöker lägga till en sektion utan att ange en virtuell sökväg*

I VFS-dialogen ställer användaren in speciella rättigheter för olika virtuella sökvägar. Även här är ordningen viktigt och således går det att ändra ordningen på elementen i listan genom att trycka på knapparna *Up* och *Down*. Sections- och VFS-dialogen är i princip uppbyggda likadant så jag hänvisar till koden jag visat för Sections-sektionen.

### 5.5.8 Script and Permission Settings

Den här property sheeten har hand om schemalagda aktiviteter, olika skript samt rättigheter för skripten, se Figur 4.10, Figur 4.12 och Figur 4.13.

De tre dialogrutorna som ingår är väldigt lika de i ”Section and Vfs Settings”. De består av en lista med olika element, vars ordning går att ändra genom *Up*- och *Down*-knapparna. Det som skiljer dem åt är – förutom att de lagrar olika data – att man i Skript- och Rättighets-fliken väljer vilken sektion man arbetar med via en rullningslista, se Figur 5.16.




*Figur 5.16: användaren får välja vilken sektion den vill lägga till eller ta bort skript i via en rullningslista*

Nedan följer koden då användaren väljer en sektion i rullningslisten:

```
case CBN_SELCHANGE: {
   INT nType = GetScriptType((HWND)lParam);

   switch (nType) {
      case SCRIPT_CUSTOMCOMMANDS:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), CustomCommands);
         break;
      case SCRIPT_EVENTS:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), Events);
         break;
      case SCRIPT_MODULES:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), Modules);
         break;
      case SCRIPT_POSTEVENTS:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), PostEvents);
         break;
      case SCRIPT_PREEVENTS:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), PreEvents);
         break;
      case SCRIPT_TELNETBINARIES:
         LoadScriptSettings(GetDlgItem(hDlg,IDC_LIST_EVENTS), TelnetBinaries);
         break;
   }
   break;
}
```

Dialogrutan *Permissions* är uppbyggd på liknande sätt som *Scripts.*

### 5.5.9 Begränsningar

Begränsningarna som finns i programmet är i form av begränsade maxlängder på textsträngar: längden på sektionsnamn för enheter och tjänster samt längden på nycklar. Värdena är satta till sådana värden att de inte ska behöva påverka funktionaliteten eller namnvalen för slutanvändaren.

I Figur 5.17 nedan illustreras begränsningarna för sektionsnamn och nycklar.

```
1  [Device_Name]
2  Host   = 0.0.0.0
3  Ports  = 20000-21999
4  Random = True
5
6  [FTP_Custom_Commands]
7  KEY    = !..\help\key.msg
8  CRYPTO = !..\help\crypto.msg
9  HELP   = !..\help\help.msg
10 RULES  = !..\help\rules.msg
11
```

*Figur 5.17: ett exempel för att illustrera några begränsningar*

Den första sektionen har sektionsnamnet *Device_Name*. Sektionsnamnet är begränsat till en maxlängd av 32 tecken. I det andra exemplet har användaren skapat fyra stycken egna kommandon: key, crypto, help och rules. Längden på de nycklarna är också begränsade till 32 tecken. Längden, 32 tecken, är satt till ett så pass stort värde att det inte bör påverka användaren.

## 5.6 Sammanfattning

I det stora hela har implementationen gått mycket bra. Det som har skapat problem är programmets omfång, att det blivit så pass mycket källkod och det från början var dåligt kommenterat.

Eftersom projektet vuxit har också saker som behövts förbättras upptäckts, både strukturen av koden, exempelvis uppdelningen av funktioner i filer samt förändringar av funktioner.

Jag har dock inte stött på några direkta problem utan de flesta problemen har berott på bristfälliga kunskaper i C++ tillsammans med Windows API. Svaren har jag fått från vanliga sökningar via sökmotererna msdn [5] och google [3].

# 6 Resultat och rekommendationer

Av de tre delmålen jag hade så hann jag med ett av dem – konfigurationsverktyget. Jag trodde i början av projektet att jag även skulle hinna med åtminstone vfs-editorn, men tiden räckte inte till.

Jag är överlag nöjd med slutresultatet av det grafiska konfigurationsprogrammet. Det finns vissa saker som jag inte hann med helt och hållet, bland annat implementationen av sammanfattningen till vänster om knapparna på huvudfönstret. All funktionalitet finns dock med, vilket är huvudsaken.

Vfs-editorn och övervakningsprogrammet är två stycken applikationer som jag fortfarande vill utveckla och eftersom jag inte hann med dem under det här projektets tid blir det eventuellt något som jag kommer att utveckla i framtiden.

Nu med facit i hand kan jag säga att bra planering är mer värdefullt än man tror. ”Happy hacker”-varianten fungerar för mindre projekt, men när man gör något större är det väldigt lätt att man råkar göra något som senare, helt eller delvis, måste göras om.

Vad jag menar är att målet skulle kunna nås snabbare om man från början hade gjort en bättre planering. Jag har uppfyllt mitt mål och jag är nöjd över slutprodukten, men jag är dock ganska övertygad om att jag hade tjänat många timmar om jag exempelvis hade planerat strukturen, vilka klasser som skulle behövas etc., bättre från början.

Genom att använda STL-lista hade jag även kunnat spara tid eftersom jag då sluppit implementera min egna länkade lista.

## 6.1 Framtida arbete

Det finns en del inställningar som kan finjusteras i konfigurationsapplikationen, bland annat då schemalagda aktiviteter ska köras. Även filstrukturen behöver förbättras – en headerfil för varje klassdeklaration exempelvis.

Konfigurationsprogrammet i sig har inget krav på exceptionell prestanda, vilket är en anledning till att det inte har spenderats så mycket tid på optimering av funktioner. Optimering finns på min ”att göra”-lista.

# 7 Projektsammanfattning

Det här kapitlet sammanfattar projektet och beskriver vad jag lärt mig, vilka problem jag har stött på, hur lång tid projektet tog samt vad jag skulle göra annorlunda om projektet skulle göras om.

Det har varit mycket intressant att få programmera ett större projekt. När jag ser tillbaka på projektet inser jag att min tidsplanering i början inte blev som jag trodde, jag var tidsoptimist! Jag skyller detta delvis på att jag aldrig tidigare gjort ett projekt i denna omfattning; det tar mer tid än man tror.

## 7.1 Nya kunskaper

Självklart lär man sig något när man arbetar med ett så pass stort projekt. Lär man sig inte något nytt så förbättrar man redan befintliga. I mitt fall har jag fått nya kunskaper och utvecklat de gamla. Några av nyckelsakerna jag lärt mig mer i är:

- C++
- Win32
- Planering

I och med att jag samtidigt som jag skrivit mitt examensarbete läst kursen *Avancerad programmering i C++*, finns det saker jag lärt mig i den kursen som jag önskar jag använt mig av i min utveckling av konfigurationsverktyget, se avsnitt 7.4.

Jag har också lärt mig mycket om Windows API. Jag hade sen tidigare ganska knapphändiga kunskaper inom området men har nu lärt mig mer om hur det hela är uppbyggt: kontroller, hur de olika kontrollerna kommunicerar med varandra. Mycket lättförståeligt nu efteråt när man förstår uppbyggnaden.

”Experience is the name we give our past mistakes” sa Oscar Wilde och visst stämmer det. Även om min planering misslyckades har jag lärt mig av det, kanske främst har jag insett hur viktig en bra planering är.

## 7.2 Problem

Jag har inte stött på några direkta problem när det gäller implementationen, men däremot lite andra när det gäller uppsatsskrivningen. Det har varit svårt att hitta bra relevanta källor för mitt projekt. IoFTPD är ett relativt nytt program och det finns inte några böcker om programmet, vilket har lett till att de flesta källorna är referenser till uppbyggnad av grafiska program, programmering i allmänhet samt några webbreferenser.

## 7.3 Tidsåtgång

Jag har inte suttit med tidtagarur varje gång jag arbetat med projektarbetet utan försökt lägga ner så pass mycket en 10-poängskurs motsvarar, men min uppfattning är att det blivit mer. Jobbar man med något som man själv är intresserad av och som man vill jobba med är det lätt att man sitter länge och att tiden rinner iväg.

Jag vet inte hur det är tänkt att tiden ska prioriteras mellan implementationsdelen och uppsatsskrivandet. En vild gissning är att jag lagt ungefär fem gånger så mycket tid på programmerandet jämfört med skrivandet; mest programmering var det i början medan det i slutet blivit mest skrivande på uppsatsen.

Att jag kommit upp i tid som motsvarar en tiopoängskurs är jag övertygad om, frågan är hur mycket mer tid jag lagt ned; någonstans motsvarande 10-15 poäng är min gissning.

## 7.4 Vad skulle göras annorlunda?

Skulle jag göra om hela mitt projekt nu skulle jag lägga mer tid på bra planering i början. Jag skulle också använda mig av STL-biblioteken. Jag skulle till exempel använt mig av en -lista eller STL-vector istället för min egna implementation av en länkad lista [6]. Genom att använda mig av fördefinierade containrar för lagring av data hade jag kunnat spara en hel del tid.

Jag hade också valt att skapa en h-fil för varje klassdeklaration istället för att ha alla klassdeklarationer i en gemensam. Det hade blivit en enklare och mer överskådlig lösning.

Som följd av en bättre planering hade programuppbyggnadens struktur blivit bättre: bättre filstruktur, smartare klassuppdelning och mer kommentarer i källkoden.

# Referenser


[1] H. M. Deitel och P. J. Deitel. *C++ How To Program*. Pearson Education, Inc. 2003. 4 utg. ISBN 0-13-111881-1.

[2] W. O. Galitz. *The Essential Guide to User Interface Design*. John Wiley & Sons, Inc. 2002. 2 utg. ISBN 0-471-084646.

[3] *Google Inc*. <http://www.google.com/>.

[4] *IniCom Networks*. 23 mars 2006, 14.33. <http://www.inicom.net/>.

[5] *Microsoft Developer Network*. <http://msdn.microsoft.com/>.

[6] T. Müldner. *C++ Programming with Design Patterns Revealed*. Addison-Wesley, Inc. 2002. ISBN 0-201-72231-3.

[7] D. A. Ruble. *Practical Analysis & Design for Client/Server & GUI Systems*. Prentice Hall PTR. 1997. ISBN 0-13-521758-X.

[8] B. Shneiderman och C. Plaisant. *Designing the User Interface*. International Edition. Pearson Education, Inc. 2005. 4 utg. ISBN 0-321-26978-0.

[9] J. Spolsky. *User Interface Design for Programmers*. Springer-Verlag. 2001. ISBN 1-893115-94-1.

[10] "RFC 2246" *IETF Tools*. 11 juni 2006, 22.55. <*http://tools.ietf.org/html/2246*>.

[11] *Tcl Developer Site*. 23 mars 2006, 14.33. <*http://www.tcl.tk/*>.