ISSN 0280-5316
ISRN LUTFD2/TFRT--5700--SE

Implementering och design av en intelligent och dynamisk fjärrkontroll i Palm OS

Henrik Fredriksson

Institutionen för Reglerteknik
Lund Tekniska Högskola
December 2002

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2002

Document Number
ISRN LUTFD/TFR--5700--SE

Author(s)
Henrik Fredriksson

Supervisor
Karl-Erik Årzén, LTH
Thomas Lindborg, Lindinvent AB

Sponsoring organization

Title and subtitle
Implementering och design av en intelligent och dynamisk fjärrkontroll i Palm OS. (Implementation and design of an intelligent and dynamic remote).

Abstract
This master's thesis involves design and implementation of an infrared link between an air diffuser and a Palm handheld computer. The diffuser, IDCC (Intelligent Diffuser for Climate Control), is developed by LindinVent AB. The application can download current controller variables and parameters and send new parameters to the diffuser. For adjusting parameters on the Palm an intuitive graphical user interface has been developed, that meets the look and feel of a standard Palm application. The infrared link is built on the commonly used IrDA standard and since it is working on the lowest level a packet system has been introduced to maintain a robust link. Since different diffusers may have different sensors, the controller variables and parameters will differ between diffusers. Therefore a dynamic graphical user interface has been developed by introducing device profiles. The introduction of device profiles gives the application a very dynamic behavior and the ability to operate against other devices. Every unit that can communicate with IrDA and meets the specifications regarding packets, interpretation of data and device profiles in this thesis can be administrated with the Palm application. The application on the handheld computer is a complete system for supervision and administration of an air diffuser and will be distributed along with IDCC starting 2003.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
Swedish

Number of pages
66

Recipient's notes

Security classification

Implementering och design av en intelligent och dynamisk fjärrkontroll i Palm OS

Henrik Fredriksson

Sammanfattning

Detta examensarbete innefattar design och implementering av en infraröd länk mellan LindinVent AB:s tilluftsdon IDCC (Intelligent Diffuser for Climate Control) och en Palm handdator. Applikationen kan hämta ner aktuella ärvärden och regulatorparametrar i donet samt skicka upp nya parametrar. För att ändra parametrar har applikationen ett intuitivt grafiskt gränssnitt. Länken bygger på den allmänt vedertagna standarden IrDA för infraröd kommunikation. Eftersom kommunikationen sker på lägsta nivå av IrDA-specifikationen har egen pakethantering implementerats för att få en robust kommunikationslänk. Eftersom olika don kan ha olika kringutrustning, t.ex. olika givare, kommer uppsättningen ärvärden och parametrar att variera från don till don, varför ett system för dynamiska grafiska användargränssnitt har utvecklats genom införandet av s.k. enhetsprofiler. Med enhetsprofiler får applikationen ett mycket dynamiskt uppförande och kan användas för kommunikation mot andra enheter. Varje enhet som har möjlighet till infraröd kommunikation via IrDA och som följer specifikationen av paket, tolkning av data och enhetsprofiler enligt detta examensarbete kan användas tillsammans med applikationen på handdatorn. Applikationen är ett komplett system för övervakning och administrering av ett tilluftsdon och kommer levereras tillsammans med IDCC fr.o.m. 2003.

Innehållsförteckning

1. Definitioner och akronymer

A/D-omvandlare	- Omvandlare mellan analoga och digitala signaler.
API	- Application Programmers Interface.
ASCII	- American Standard Code for Information Interchange
bps	- bitar per sekund, mått på överföringshastighet.
CAN	- Control Area Network.
CAV	- Constant Air Volume.
Chunk	- Del av allokerbart minne.
CMOS	- Complementary Metal-Oxide Silicon.
D/A-omvandlare	- Omvandlare mellan digitala och analoga signaler.
Debugger	- Felsökningsverktyg i samband med programmering.
Dragindex	- Uttrycker hur många procent av en grupp människor i ett rum som besväras av luftrörelserna i rummet.
Duplex	- Benämning för enkelriktad eller dubbelriktad kommunikation. Man talar därför om halv eller full duplex.
FIR	- Fast IR.
Flash-ROM	- Read Only Memory vars innehåll kan ändras.
Frame	- Benämning för all data i ett paket över en kommunikationslänk. En frame innefattar i det seriella fallet start-, stopp-, data- och eventuellt paritetsbitar.
IDCC	- Intelligent Diffuser for Climate Control.
IrDA	- Infrared Data Association.
ISO	- International Standardization Organization.
Kastlängd	- Det avstånd från ett tilluftsdon där lufthastigheten avtagit till 0.2 m/s.
LON	- Local Operating Network.
Offset	- Kompensation m.a.p. en bestämd nollnivå.
OSI	- Open Systems Interchange, referensmodel för utbyte av data mellan enheter.
PDB	- Palm Database.
POSE	- Palm Operating System Emulator.
PRC	- Palm Resources, format för t.ex. exekverbara filer på en Palm handdator.
PQA	- Palm Query Application.
RAD	- Rapid Application Development.
RAM	- Random Access Memory.
RTOS	- Real-Time Operating System.
SDK	- Software Developement Kit.
SIR	- Serial Infrared.
UART	- Universal Asynchronous Receiver Tranceiver.
USB	- Universal Serial Bus.
VAV	- Variable Air Volume.
VFIR	- Very Fast IR.
Vistelsezon	- Den del av en lokal som personer normalt vistas inom.

2. Inledning

I allmänhet innebär injustering av takmonterad ventilationsutrustning ofta dyra lösningar med väggmonterade inställningsdon eller oergonomiska alternativ där man med hjälp av stegar klättrar upp till enheter och gör injusteringar.

Möjlighet till infraröd (IR) kommunikation finns numera inbyggt i ett flertal elektroniska enheter, exempelvis mobiltelefoner och handdatorer. Kommunikationen är seriell och använder sig i stor utsträckning av den allmänt vedertagna IrDA-standarden (Infrared Data Association). För att underlätta installations- och injusteringsarbete av tilluftsdon ska därför en infraröd länk mellan en Palm handdator och ett tilluftsdon utvecklas. Injusteringar kommer att kunna ske betydligt snabbare än med dagens metoder. Relativt andra tekniker, som t.ex. Bluetooth, är IR idag ett billigt alternativ.

2.1 Bakgrund

På 60-talet gjorde den mekaniska ventilationen på allvar insteg i nybyggnationen. Även ett flertal nya byggnadsmaterial började användas som t.ex. golvbeläggningar, färger och tapeter. Från att man tidigare hade varit noga med kvaliteten på byggnadsmaterialet var det istället tid och pengar som i allt större utsträckning bestämde. För inomhusklimatet var detta nya byggnadssätt ofta till nackdel. Tidigare användes mer naturliga och öppna material som genom sina hygroskopiska förmågor absorberade fukt när det var fuktöverskott och avgav fukt när rumsluften blev torrare och därmed hjälpte till att balansera den relativa luftfuktigheten inomhus.

Med de nya materialen och färgerna tillfördes rumsluften nya typer av mer eller mindre skadliga emissioner som varierade beroende på kvalitet, ålder, rumstemperatur och relativ luftfuktighet. Emissionsavgivningen från dessa material ökas kraftigt vid förhöjd rumstemperatur, fuktskada eller förhöjd relativ luftfuktighet vilket kan få en tillväxt av alger och kvalster som konsekvens. Vid låga relativa luftfuktigheter får man istället problem med ökad statisk elektricitet hos personer som vistas i lokalen vilket medför att dammalstringen blir större vilket t.ex. kan irritera slemhinnor.

Allergier och astma har sedan 60-talet ökat kraftigt i Sverige. Ökningen sätts ofta i samband med inomhusklimatet som vi idag har i våra fastigheter byggda efter 1960. Klagomålen på inomhusklimatet ökade kraftigt i de nybyggda husen med mekanisk ventilation och nya byggnadsmaterial. Begreppet "sjuka hus" blev vanligare. I en utredning av Statens Folkhälsoinstitut [1] rekommenderades bl.a. kraftigt höjda luftomsättningar året runt i alla typer av lokaler för att på detta sätt minska emissionskoncentrationen och den relativa fuktigheten inomhus.

Genom att installera regulatorer på t.ex. tilluftsfläktar kan dessa programmeras för behovsstyrd och årstidsanpassad ventilation. I anläggningar med återluft kan denna reduceras väsentligt eller reduceras helt. Hur mycket luftflödena kan reduceras beror på värmeöverskottet i lokalerna och på anläggningens utformning beträffande fläktar, kanalsystem och tilluftsdon.

Med utgångspunkt från dagens krav på bättre inneklimat och minskad energianvändning har LindinVent AB utvecklat tilluftsdonet IDCC (Intelligent Diffuser for Climate Control), en produkt som bygger på styrning av undertempererad luft. IDCC är ett don med variabelt inblåsningsflöde, VAV (Variable Air Volume). Genom ett patenterat system kan IDCC hålla god omblandningseffekt samtidigt med hög inblåsningshastighet utan att alstra ljud eller skapa en lokal som känns dragig. IDCC är även energisnålt, vid en simulering utförd av Thorsell energi och klimatkonsult AB visades bl.a. att LindinVents tilluftsdon kan uppnå en energibesparing på 40% med VAV jämfört med CAV (Constant Air Volume) [2].

2.2 Omfattning och kravspecifikation

Målet med detta examensarbete är att utveckla ett program som via IR kan utbyta information, t.ex. titta på regulatorinställningar och ärvärden eller skicka ny data till regulatorn, mellan IDCC och handdatorn. Examensarbetet omfattar design och implementering av en infraröd länk mellan IDCC och en Palm handdator. Systemet ska inte fixeras vid en speciell version av operativsystem hos handdatorn utan ska vara fungerande på både äldre och nyare versioner. Dock finns en lägsta gräns på version 3.5. Design innebär allt från en säker datastruktur och ett garanterat robust program till ett intuitivt användargränssnitt och utveckling av kommunikationsprotokoll. Montering av kretsar och kringliggande elektronik samt programmering av donet utförs av annan personal.

Programmet ska ha möjlighet att kommunicera med ett don och avläsa dess ärvärden och skicka ny data i form av t.ex. regulatorparametrar. Det man skickar till ett don kommer i fortsättningen att kallas för inställningar. Palm OS ger möjlighet till en stor uppsättning grafiska komponenter för att skapa användarvänliga gränssnitt. Ett enkelt och intuitivt grafiskt gränssnitt som påminner användare med tidigare erfarenhet av Palm önskas. Eftersom IDCC beroende på kringutrustning, givare mm, innehåller olika parametrar och ärvärden ska applikationen vara generisk, d.v.s. en lösning för ett dynamiskt grafiskt användargränssnitt samt protokoll måste utvecklas. Utvecklingsarbete ska ske i programmeringsspråken C/C++.

2.3 Kapitelbeskrivning och läsanvisning

I kapitel tre ges en teoretisk bakgrund till seriell kommunikation, IrDA och hårdvaran som använts i utvecklingen. Kapitel fyra beskriver tilluftsdonet IDCC. I kapitel fem sammanfattas operativsystemet på vilket Palm handdatorer körs, Palm OS. Dessutom ges här en beskrivning av hur ett Palmprogram exekverar samt hur det speciella systemet med databaser används för datalagring. Kapitel sex handlar om implementeringsarbetet, vilket inleds med en kort sammanfattning av de använda utvecklingsverktygen. I kapitel sex ges även en beskrivning av paketdefinitioner vilka används vid kommunikationen. Kapitel sju presenterar resultat. I kapitel åtta ges en analys av implementeringen och resultaten. Kapitel nio diskuterar slutligen andra tekniker och framtida utveckling.

För att få en röd tråd genom hela rapporten bör kapitlen läsas i ordningen 2, 3, 4, 5, 7, 6, 8, 9, d.v.s. kapitel sju före kapitel sex. Kapitel sju kan läsas separat.

3 Seriell kommunikation och IrDA

Möjlighet till seriell kommunikation har funnits inbyggd i datorer och mikroprocessorer i tiotals år. Ett tidigt gränssnitt som lever kvar än idag är RS-232, vilket specificerar ett komplett system för seriell kommunikation. Förutom själva kommunikationsprotokollet definieras även signalspänningar, signalfunktioner och mekaniska anslutningar. Idag är den seriella kommunikationen bättre i många avseenden, inte minst vad gäller hastighet, men den är fortfarande bakåtkompatibel med RS-232. Seriell kommunikation sker i full duplex och går via ett antal ledningar, vilka har olika betydelse. Vanligt är en 9-polig mekanisk kontakt som således har nio ledare. Funktionen hos de viktigaste av dessa visas i tabell 3.1.

Ledning	**Funktion**
RX (Received data)	Ledare för inkommande data.
TX (Transmitted data)	Ledare för utgående data.

Tabell 3.1 – Ett par av ledningarna i en RS-232-kontakt och deras funktion

För att styra signalerna och ta emot och tolka inkommande data används vad som kallas för en UART (Universal Asynchronous Receiver Transmitter) vilket ibland är en fristående krets men som i många fall finns implementerad i själva processorn. Det är en UART som bestämmer vilken spänning som ska levereras över ledarna. Enligt RS-232 tolkas området +3-12V som en binär etta och motsvarande negativa område som en nolla. Området –3V till +3V är odefinierat och en signal i detta område är troligtvis brus. En byte kommer således att skickas som ett bitmönster. Förutom de åtta eller eventuellt sju databitarna skickas även extra bitar för att markera var data börjar respektive slutar, s.k. start- och stoppbitar. Utöver start- och stoppbit är det vanligt att en grov felkontroll implementeras m.h.a. en extra bit, en paritetsbit. Man brukar tala om udda eller jämn paritet där man med jämn och udda menar om antalet ettor och nollor i bitmönstret är udda eller jämnt. Är det t.ex. ett jämnt antal ettor och man använder jämn paritet skickas en paritetsbit som är ett. För en mer ingående beskrivning av seriell kommunikation, RS-232 och UART, se [3].

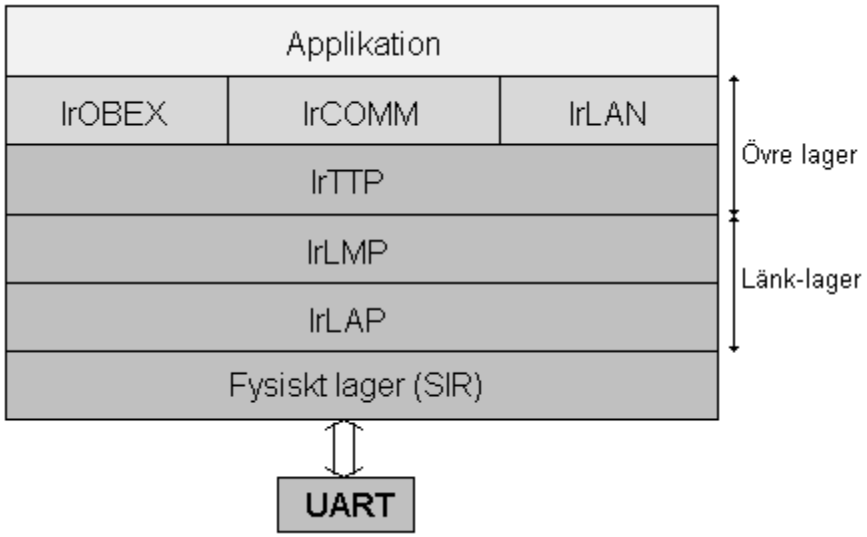
Kommunikation via infrarött ljus är seriell men sker här till skillnad från de flesta kabelbaserade system i halv duplex. Anledningen är att man annars får problem med att den sändande enheten själv tar emot det den skickade eftersom dioden för att sända och fototransistorn för att ta emot infrarött ljus ofta är monterade i samma kapsel, d.v.s. nära varandra. Eftersom det för ett tiotal år sedan fanns en hel del olika sätt att utföra kommunikationen skapades 1993 IrDA, vilket är en industribaserad grupp av mer än 150 företag. I september 1993 specificerade IrDA en standard för infraröd kommunikation. Standarden utvecklades kring följande fem punkter:

- Plattformsoberoende
- Låg kostnad
- Kort kommunikationsavstånd
- Punkt- till punktavstånd
- Möjlighet till ett flertal olika överföringshastigheter

IrDA-standarden för infraröd kommunikation är idag allmänt vedertagen och den senaste versionen utkom 2001. Protokoll som använder sig av IrDA finns idag i allt från mobiltelefoner och handdatorer till bärbara datorer och skrivare.

För att ett protokoll för kommunikation ska fungera är det en hel del saker som måste tas i beaktning, därför delar man ofta upp protokoll i mindre enheter vilka brukar kallas för lager. Lager ordnas i en hierarkisk struktur där högre lager använder sig av funktioner i de lägre lagren. När man staplar alla lager ett protokoll innehåller erhåller man en s.k. stack.

Protokollstacken som IrDA specificerat bygger på en referensmodell definierad av ISO, kallad OSI (Open System Interchange), vilket är den ”ideala” arkitekturen för ett kommunikationssystem. Precis som OSI startar IrDA-stacken längst ner med ett fysiskt lager. Det fysiska lagret grupperas enligt IrDA-specifikationen beroende på kommunikationshastighet i tre olika undergrupper: SIR (Serial IR), FIR (Fast IR) och VFIR (Very Fast IR). SIR har en maximal hastighet av 115kbps, FIR 4Mbps och VFIR 16Mbps [4]. Stacken byggs sedan på med ett länklager och slutar efter ännu ett antal lager på applikationsnivå. För mer information om OSI-modeller och protokollstackar, se [5]. I figur 3.1 visas en figur över protokollstacken som IrDA specificerar.

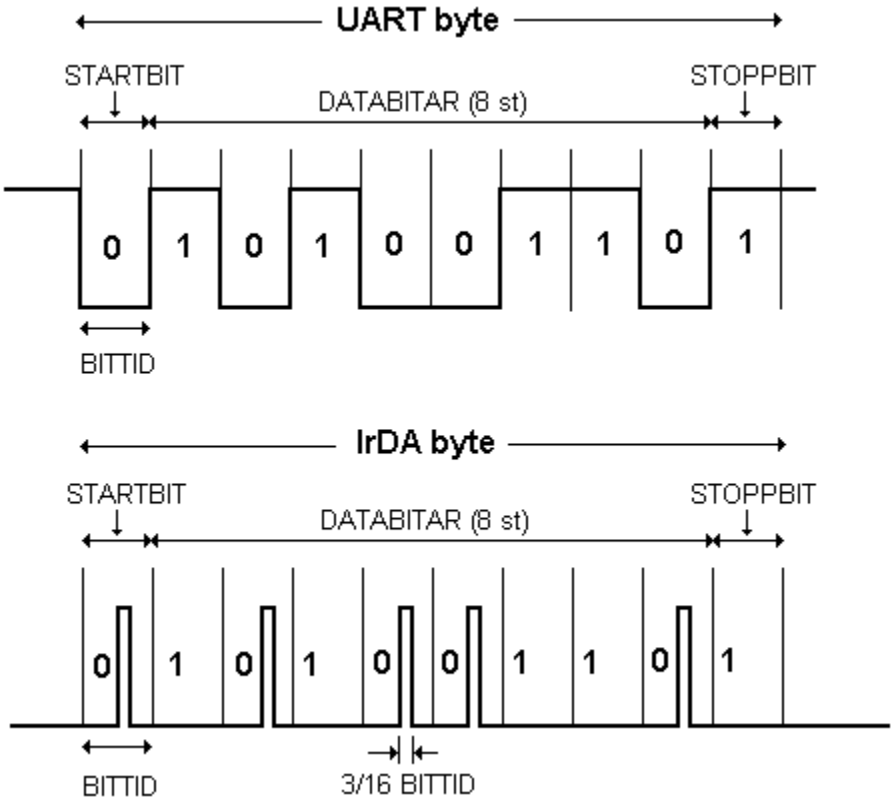


Figur 3.1 – IrDA protokollstack

Enheter som erbjuder IR-kommunikation är troligtvis mindre batteridrivna enheter (t.ex. Palm handdatorer). För att minska batterikonsumtionen fungerar IrDA så att vanliga bitmönster översätts, eller moduleras, till ljuspulser vars pulslängd är mindre än bittiden. Detta har flera fördelar. Dels drar det mindre ström från batteriet att endast blinka med en diod, dels kan högre strömmar skickas genom dioden för bättre effekt utan att riskera att den går sönder. Dessutom får man en grov felkontroll då man kan sålla bort brus, d.v.s. indata som inte har rätt pulslängd.

Stacken är som tidigare nämnts uppbyggd av ett antal lager som hanterar olika uppgifter. Längst ner finns det fysiska lagret, som hanterar översättningen av UART-bitmönster till pulser. Generellt kodas, eller moduleras, en puls som 3/16 av en UART-bitlängd [4]. Om man använder hastigheten 19200 bitar/sekund medför detta att en bitlängd är 52 μs, således kommer en IrDA-

puls att vara 9.75 µs. T.ex. kan man titta på det decimala talet 166 vilket binärt motsvaras av 10100110. I figur 3.2 nedan visas hur detta bitmönster ser ut dels i en UART, dels som IrDA-pulser. I denna figur används en startbit, en stoppbit och ingen paritet.



Figur 3.2 – Kodning av bitström till IrDA-pulser

Specifikationen för IrDA SIR tillhandahåller hastigheter från 300 bitar per sekund till 115000 bitar per sekund med ca en meters maximalt kommunikationsavstånd. Vidare har en sändare/mottagare för infraröda pulser kodade enligt IrDA ett optiskt aktivt område ±15° från den optiska axeln [4]. Eftersom relativt låga hastigheter kommer att användas i fortsatt implementering av den infraröda länken mellan Palm och IDCC refereras till [4] för den inramning (eng. framing) av data som sker av det fysiska lagret vid högre kommunikationshastigheter via FIR och VFIR.

3.1 IrDA övriga lager

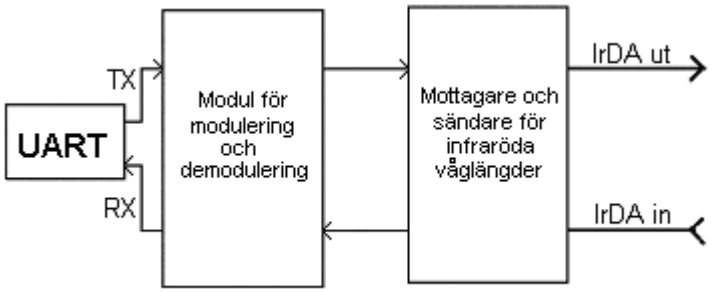
Utöver det fysiska lagret innehåller IrDA-stacken en del andra lager, se figur 3.1, där ett flertal baseras på motsvarande nivå i OSI-modellen. Dessa lager ger möjlighet till t.ex. flödeskontroll, överföring av hela objekt, nätverksemulering eller emulering av seriella och parallella portar.

- *IrLAP* - Infrared Link Access Protocol [6]
- *IrLMP* - Infrared Link Manager Protocol [7]
- *IrTTP* - Infrared Tiny Transport Protocol [8]
- *IrOBEX* - Infrared Object Exchange Protocol [9]

- *IrCOMM* [10]
- *IrLAN* - Infrared Local Area Network [11]
- *IrDA Control* - Tidigare *IrBUS* [12]
- *IrDA Lite* - minimal implementering [13]

3.2 Hårdvara med stöd för IrDA

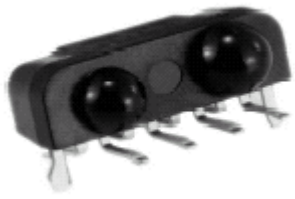
En enkel bild över hur det kan se ut för en sändande/mottagande enhet som kommunicerar via IR, d.v.s. från UART till fysiskt ljus, visas i figur 3.3 nedan. Här skickas data från en UART till en krets som tar hand om modulering eller eventuellt demodulering (avkodning) av data. Från denna krets skickas en signal till en diod som opererar i det infraröda området, eller så kommer en signal från en fototransistor in till den avkodande kretsen vilken efter avkodning skickas vidare till UART:en.



Figur 3.3 – Uppsättning för kommunikation UART ↔ IrDA

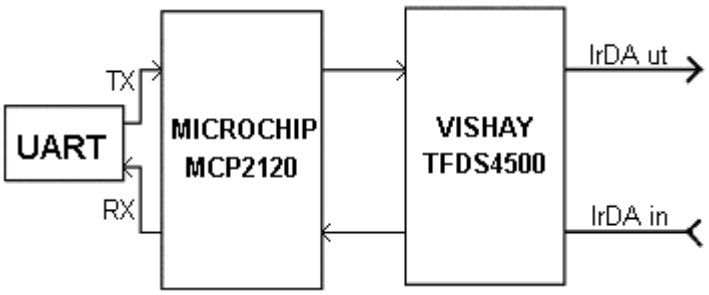
Det finns ett flertal kretsar på marknaden som kan koda och avkoda mellan IrDA och RS-232, t.ex. Telefunkens TOIM3000 [14]. Ett annat alternativ är en krets från Microchip, MCP2120, en krets som sköter både avkodning och kodning enligt IrDA SIR. Kretsen är av CMOS-teknologi, energisnål och har variabel sändnings- och mottagningshastighet. Se datablad från Microchip för mer information angående kretsens prestanda och karakteristik [15].

En komponent för att skicka och ta emot infrarött ljus behövs också. TFDS4500 från VisHay [16] är en kombinerad emitter och fotodiod anpassad för IrDA SIR, se figur 3.4.



Figur 3.4 – VisHay TFDS4500 [F1]

Figur 3.3 ovan skulle efter detta val av kretsar få utseendet enligt figur 3.5 nedan.



Figur 3.5 – Uppsättning för kommunikation IDCC↔IrDA

Microchip levererar även kretsar som kan konvertera högre lager av IrDA-specifikationen men eftersom kommunikationslänken mellan Palm och IDCC kommer att ha ett relativt sett litet datautbyte där inga större hastigheter behövs har vi använt den enklaste varianten MCP2120.

4. IDCC

Som tidigare nämnts är IDCC ett tilluftsdon för takmontering som bygger på styrning med undertempererad luft. Att styra med undertempererad luft har tidigare varit svårt eftersom det medför problem med höga ljudnivåer och dragiga miljöer. Genom ett patenterat utförande kan donet hålla en hög inblåsningshastighet med god medinjektion utan att varken skapa höga ljudnivåer eller skapa en miljö som känns dragig.

Donets kärna är mikroprocessorn PIC16F från Microchip [17] vilket är en enkel 8-bitars processor. Intressant att nämna är att processorn har en 2-nivåers seriell FIFO-buffert (First In First Out) och 320 byte internt minne. Ett reglerprogram skrivet i assembler som generellt arbetar med interrupts från externa givare eller från t.ex. processorns UART, exekveras då donet inte opererar med insignal från extern regulator. Programmet reglerar en motor som är kopplad till och styr avståndet mellan ett antal lameller som sitter i donets underkant.

Donet är monterat till en tilluftskanal och beroende på avståndet mellan lamellerna kan ett variabelt luftflöde erhållas. När motorn öppnar kommer således spaltvidden mellan lamellerna att öka. Lamellerna kan genom sin area fånga upp det tryck som bildas när luft med höga hastigheter strömmar ut genom dem. Ett tryck som annars kan skapa oönskade ljud. Systemkortet till IDCC är monterat under lamellerna och följer därmed med dessa upp och ned efter den till motorn levererade styrsignalen. Syftet med donet är att styra klimatet i en lokal med en optimerad energibesparing som följd. I figur 4.1 visas en bild på ett fristående don och i figur 4.2 är donet installerat i ett undertak.

Figur 4.1 – IDCC fristående [F2]

Figur 4.2 – IDCC monterad i undertak [F3]

Donets utveckling har i huvudsak baserats på följande fyra kriterier:

- Hög prestanda
- Kostnadseffektivitet
- Låg ljudnivå
- Enkla installations- och injusteringsmöjligheter

Regulatorn levererar en styrsignal till motorn genom indata från ett antal givare. I sitt maximala utförande kan regulatorn få indata från flödesgivare, temperaturgivare och CO_2-givare. För att kunna minimera energikostnaderna ytterligare är IDCC även försett med en närvarodetektor vilken kan anpassas att stänga av reglering om ingen har vistats i en lokal inom en bestämd tid. Utöver detta sitter en brandvarnare monterad i donet.

IDCC kan anslutas till andra varianter av överordnade styrsystem vilka kan leverera styrsignal. T.ex. kan protokollet enligt LON (Local Operating Network) nämnas. Vidare är tilluftsdonet förberett för nätverkskommunikation via CAN (Control Area Network). På detta sätt kan man enkelt via ett web-baserat gränssnitt kommunicera med donet. Genom en MCP2120 tillsammans med en TDFS4500 får IDCC möjlighet till kommunikation via IR. Uppställningen liknar figur 3.5. Under lamellerna sitter en bottenplatta för att täcka systemkortet. Bottenplattan har hål för närvarodetektor, temperaturgivare och TDFS4500

Grundidén vid utvecklingen av IDCC är enkelheten, donet ska vara enkelt att installera, enkelt i drift och enkelt att underhålla. Möjlighet till nätverkskommunikation via fältbussystemet CAN är ett bra steg på vägen, men det är möjligt att lokalen i fråga inte möjliggör detta alternativ eller att en kund inte väljer detta tillval varför kommunikationslänken via IR blir aktuell. Alternativet att genom en mer eller mindre avancerad fjärrkontroll ha möjlighet att kontrollera aktuella ärvärden men även att ställa om donet via ett antal inställningar, som t.ex. regleringsintervall eller maximalt flöde, är en både smidig och kostnadseffektiv lösning. Mer information om donet finns på [18].

5. Palm och Palm OS

5.1 Introduktion

Palm var från början ett företag som utvecklade textigenkänningssystem till andra tillverkares handdatorer. Deras system Graffiti lever kvar än idag i de senaste Palmenheterna. 1996 lanserades deras första handdator, den fick benämningen Palm Pilot m1000 och arbetade mot operativsystemet Palm OS 1.0. Till skillnad från den första enheten m1000 är dagens Palmenheter (m500-serien) utrustade med Palm OS4, ett operativsystem som har många nyheter jämfört med den första versionen. I skrivande stund kan man läsa på Palms hemsida om OS 5.0 som är under utveckling. Av de största nyheterna i OS 5.0 kan nämnas stöd för en ny processor (ARM-processorer), möjlighet till högre upplösning (320x320 bildpunkter) och stöd för samplade ljudströmmar.

Idag har en Palm handdator av standardmodell en 33MHz Motorola-processor, tryckkänslig display (160x160 bildpunkter), möjlighet att spela upp enklare ljud, ca 16Mb RAM och möjlighet till kommunikation med omvärlden i form av t.ex. IR, vilket baseras på stacken specificerad av IrDA. Utöver de obligatoriska lägsta lagren i stacken (SIR, IrLAP, IrLMP) implementerar Palm också t.ex. IrCOMM och IrOBEX. Precis som på de flesta handdatorer används en liten penna (eng. stylus) tillsammans med ett textigenkänningssystem för inmatning. Utöver detta finns inga andra möjligheter till inmatning förutom de sex tangenter som brukar sitta i nedre delen av handdatorn eller att man extrautrustar med ett tangentbord. En heltäckande guide till olika versioner av Palm OS, specifikationer och prestanda hos olika modeller ges i [19].

Precis som de flesta större operativsystem finns det även i Palm OS en handfull standardprogram som kan göra allt från att skriva anteckningar och skicka e-post till att rita bilder och föra kalender. Speciellt kan programmet Memopad nämnas, ett program för att skriva just anteckningar vilket kommer användas i en del av implementeringen.

Överföring av filer mellan en stationär dator och handdatorn sker via en procedur som kallas Hotsync. Man synkroniserar data på handdatorn och den stationära datorn och ser på så sätt till att information som finns på handdatorn flyttas till den stationära datorn och tvärt om. Synkroniseringen sker mestadels via ett seriellt gränssnitt, t.ex. USB (Universal serial bus) eller vanlig serieport, men det är även möjligt att göra synkroniseringen via nätverk eller någon form av trådlös kommunikation. Hotsync är helt enkelt ett program som installeras på en stationär dator och som sköter kommunikationen med handdatorn.

Man har även möjlighet att synkronisera mot olika program, t.ex. kan nämnas att Hotsync kan konfigureras så att synkronisering sker med Microsoft Outlook. Detta innebär att bl.a. att e-post skickas över till handdatorns egen e-postklient. Tillsammans med Hotsync-programmet kan en s.k. Conduit installeras vilket är ett applikationsspecifikt program som installeras på den stationära datorn. Tanken med en Conduit är att den ska tala om för Hotsync vilken applikation på den stationära datorn de filer som synkroniseras ska lämnas till. De flesta kompletta palmprogram kommer dels med mjukvara för själva handdatorn, dels med en Conduit och en applikation på en stationär dator som ska användas för att bearbeta och administrera av handdatorn skapad data.

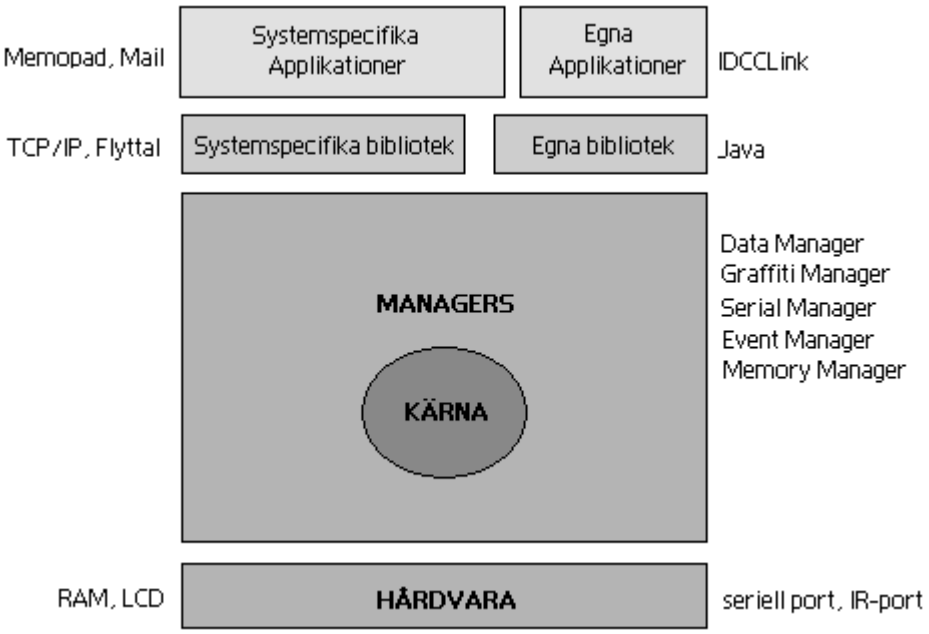
Palm OS har även stöd för att presentera html-dokument. Eftersom skärmen är liten har Palm OS definierat egna html-taggar vilka komprimerar innehållet i en html-sida något. En html-fil måste således översättas till det av Palm läsbara formatet, Palm Query Application (PQA). För att titta på filerna används programmet Clipper som levereras med en del versioner av Palm OS.

5.2 Kärna och minneshantering

Palm OS bygger i grunden på realtidskärnan Kadak [20]. Som ett realtidsoperativsystem (RTOS) har Kadak stöd för bl.a

- Möjlighet till kontroll över kritiska regioner m.h.a. semaforer
- Avbrottshantering
- Schemaläggning m.a.p. prioritet
- Säkerställande av korrekt kommunikation mellan trådar

Trots detta finns utifrån sett ingen möjlighet till realtidsprogrammering för en programmerare av Palm OS. Visserligen är programmering med jämlöpande exekveringspunkter möjligt, men Palm levererar inget stöd för detta. Är man däremot familjär med Kadak API är det fullt möjligt. Som exempel kan nämnas att Sonys Clié, en handdator som kör under Palm OS, har möjlighet att spela musik samtidigt som man kör andra applikationer. Däremot använder sig operativsystemet självt av Kadak:s realtidssupport. Detta krävs för den händelsedrivna utformningen av operativsystemet och samtliga applikationer. I figur 5.1 visas en bild över de väsentliga delarna i Palm OS.



Figur 5.1 – Palm OS, systemets uppbyggnad

En Palm handdator har både RAM (Random Access Memory) och ROM (Read Only Memory), vilket innehåller operativsystemet självt, de inbyggda applikationerna samt deras eventuella databaser (mer om databaser i kapitel 5.3). Men eftersom de flesta Palm handdatorer möjliggör uppgraderingar av operativsystemet är det snarare ett flash-ROM som används istället för ett

ROM. RAM-minnet är uppdelat i en dynamisk del och en lagringsdel. Den dynamiska delen delas av operativsystemet och den körande applikationen och innehåller bl.a:

- Systemets globala variabler
- Systemets dynamiska allokeringar
- Den körande applikationens globala variabler
- Den körande applikationens dynamiska allokeringar
- En stack med pekare till funktioner för den körande applikationen

Storleken hos den dynamiska delen varierar beroende på operativsystemets version och mängden tillgängligt RAM. I versioner tidigare än 3.0 var den dynamiska delen på endast 32kB och dessa handdatorer hade maximalt 512kB RAM. De lite senare versionerna, t.ex. version 3.5, har mer än 4 MB RAM och en dynamisk del runt 256kB.

Samtliga systemanrop görs av en programmerare genom en s.k. Manager, se figur 5.1. Palm OS innehåller Managers för att hantera allt från seriell kommunikation till hantering av textsträngar. I appendix 2 finns några av de mest förekommande Managers samt deras funktionalitet listade. Eftersom en applikation på en Palm har direkt tillgång till att skriva till RAM är den s.k. Memory Manager speciellt intressant. Vill man dynamiskt allokera nytt minne gör man detta genom att anropa `MemPtrNew()` vilket är en direkt motsvarighet till `malloc()` om man programmerar i C eller `new` i t.ex. C++. Memory Managern har således till uppgift att minimera risken för korrupt data, d.v.s. att man skriver över redan allokerade områden, samt att sköta fragmentering av minnet. För att detta ska fungera smidigt har Memory Manger alltid tillgång till att flytta runt allokerad data. Man arbetar därför med två olika dynamiska datatyper. Det ena är vanliga pekare vilka allokeras med `MemPtrNew()`. Det andra är handtag (eng. handles), vilket bäst beskrivs som en låst pekare. Till skillnad från pekare har Memory Manager ej möjlighet att flytta handtag i minnet. Så när man t.ex. ska skriva ner data i en dynamisk variabel krävs att man först låser denna med `MemHandleLock()`. När man har skrivit klart måste man låsa upp det låsta minnet med `MemhandleUnlock()` för att ge Memory Manger möjlighet att flytta detta data igen av anledningar nämnda ovan.

Som bekant har de flesta programspråk ingen inbyggd skräphantering, d.v.s. en rutin som rensar upp i minnet efter pekare utan referens. Eftersom Palm till skillnad från t.ex. en stationär dator har relativt lite minne är det viktigt att dynamiskt allokerat minne frigörs när det inte längre behövs. Att frigöra minne görs med `MemPtrFree()`, vilket motsvaras av `free()` i C eller `delete` i C++. För mer information om C-syntax hänvisas till [21], och [22] för C++-syntax.

5.3 Databaser

Applikationer på en Palm handdator bygger på filosofin att när man startar ett program återkommer man till det tillstånd som var aktivt då man senast lämnade programmet. Detta system bygger på att det måste finnas möjlighet att lagra data. Palm OS har inget filsystem i bemärkelsen sekundärminne, som hårddiskar, diskettenheter eller liknande. Istället erbjuder Palm OS vad som kallas för databaser. Detta har dock inget att göra med en databas i sin vanliga betydelse, t.ex. relationsdatabaser, utan är helt enkelt Palm OS sätt att lagra valfri data på flashminne vilket kommer att finnas kvar tills batteriet tar slut.

Palm OS har support för två olika typer av databaser, det ena är en resource-databas (prc filer), det andra en record-databas (pdb filer). Resource-databaser kan jämföras med en exekverande fil på ett konventionellt OS, en record-databas med en fil i sin vanliga bemärkelse. Detta medför att allt som lagras i handdatorns minne i avstängt läge (oavsett om det är applikationer eller applikationers datafiler) lagras som databaser. En record-databas har en uppbyggnad som består av tre olika block. Först ett huvud (PDBHeader) enligt nedanstående struktur.

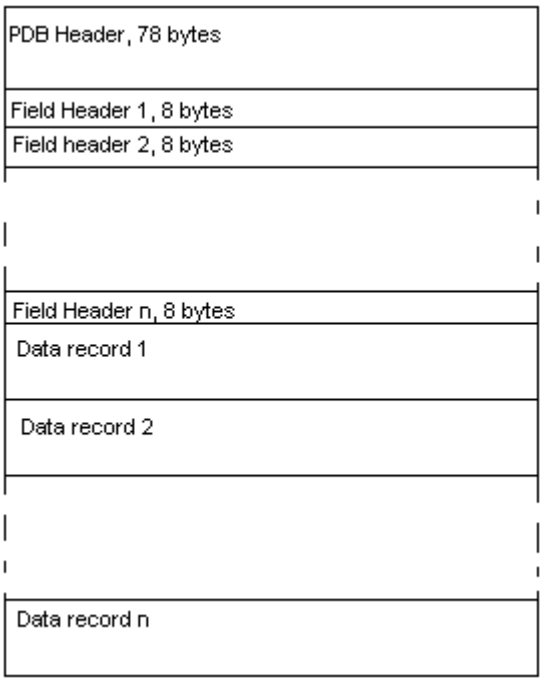
```
struct PDBHeader
{
            char name[32];
            unsigned short attributes;
            unsigned short version;
            long creationDate;
            long modificationDate;
            long lastBackupDate;
            long modificationNumber;
            long appInfoID;
            long sortInfoID;
            long type;
            long creator;
            long unqueIDSeed;
            long nextRecordistID;
            unsigned short numRecords;
};
```

Huvudet är 78 byte stort. Efter PDBHeader lagras en ny struktur, ett fälthuvud (FieldHeader), som beskriver varje enhet i det tredje och sista blocket. Ett fälthuvud innehåller framför allt en offset (localChunkID) till dess datablock. Till varje fälthuvud finns således ett datablock associerat.

```
struct FieldHeader
{
            long localChunkID;
            char attributes;
            char uid [3];
};
```

Sist ligger datablocken. Observera att här kan man lagra vilken slags information som helst, en godtycklig datastruktur. Funktioner som Palm OS Data Manager tillhandahåller för att skriva till och läsa ur databaser kan jämföras med de vanliga filfunktionerna `fwrite()` och `fread()` i C.

En record-databas består således av ett huvud (78 byte) för allmän beskrivning, därefter ett fälthuvud per datafält (8 x antalet datafält byte) och slutligen datafält, som är godtyckliga strukturer. Sammanfattningsvis har en komplett record-databas utseende enligt figur 5.2.



Figur 5.2 – Generellt utseende hos en record-databas

Field Header 1 innehåller en local chunk som är 78+8*n byte, d.v.s. en offset till *Data record 1*. På liknande sätt innehåller Field Header k en local chunk som är

$$local\,chunk(k) = 78 + 8 \cdot n + \sum_{i=1}^{k} sizeof\,(data\,record\,i)$$

där k är ett godtyckligt tal mellan 1 och n.

Känner man till denna strukturen kan man enkelt skriva konverterare mellan pdb-filer och valfritt format. De i Palm OS redan existerande programmen har sina egna databaser för att lagra information, som exempel kan nämnas att Memopad har en databas som heter ”MemoDb”. Det har tidigare nämnts att man exekverar endast ett program samtidigt på en Palm, men man har däremot möjlighet att byta mellan program, och när man byter tillbaka kommer man tillbaka till den punkt där man lämnade det första programmet, vilket kan liknas vid co-rutiner. För detta ändamål tillhandahåller Palm OS en speciell record-databas som lagrar programmets aktuella tillstånd (eng. state). Hur detta går till rent tekniskt är att när man avslutar ett program skriver programmet ner en av programmeraren definierad struktur, `struct preferences`, innehållande godtycklig data i en speciell databas.

```
struct preferences
{
            datatyp_1 data_1
            datatyp_2 data_2
```

```
                    ..
                    .
                    /* här kan godtyckliga datatyper läggas till */
        }
```

Här lagrar man bl.a. information som behövs för att återgå till samma punkt i exekveringen när man startar programmet igen. För att detta ska fungera krävs att varje program vid uppstart kontrollerar om det existerar en preferences-databas och om det gör det så är det upp till programmeraren att vidta åtgärder för att återställa applikationen till sitt föregående tillstånd. Med denna teknik får man känslan att när man väl har startat ett program så är man alltid i körande läge och om man tittar på redan existerande mjukvara för Palm OS verkar samtliga följa denna filosofi.

Palm OS har även en Manager för filhantering, File Stream Manager. Detta kan verka förvillande eftersom möjlighet till filer skulle eliminera behovet av databaser. Den stora skillnaden är att en fil öppnad med File Stream Manager lagras på RAM och kommer således att gå förlorad när handdatorn stängs av. Filer används därför endast för temporär lagring i situationer där en eventuell dataförlust spelar mindre betydelse eller för felsökning.

5.4 Typisk programstruktur

För att programmera Palm OS använder man sig med fördel av ett mjukvaruutvecklingspaket, SDK, vilket distribueras av Palm. En del av SDK:n är Palm OS API [23], ett gränssnitt mellan programmerare och Palm OS som bl.a. innehåller en uppsjö Managers för allt man kan tänka sig. Operativsystemet kan programmeras i ett flertal språk, men eftersom hela Palm OS API och exempelkod på de inbyggda applikationerna är skrivna i C är också nästan alla andra program skrivna i C eller syntaxmässigt liknande språk, t.ex. C++.

Palm OS tillhandahåller en variant av C som syntaxmässigt är identiskt med standard-C, men som typmässigt innehåller en hel del härledda datatyper och ett annorlunda sätt att komma åt hårdvara som t.ex. minne och externa portar, nämligen de ovan nämnda Managers .

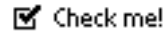
Ytterliggare skillnader är att ett Palm-program startas inte från en `main()`-funktion vilket program gör i standard-C utan ifrån funktionen `PilotMain()`. Skillnaderna är små.

```
Int main(int argc, char** argv)
{

}
```

```
UInt32 PilotMain( UInt16 cmd, MemPtr cmdPBP, UInt16 launchFlags)
{

}
```

Här förekommer ett par av Palm OS härleda typer, `UInt16` skulle kunna översättas till C:s datatyp `unsigned short`. `MemPtr` är en typdefinition av `void*`.

Eftersom en Palm handdator endast har en 160x160 punkters skärm och inmatning endast genom Grafittisystemet är Palmprogram alltid av grafisk karaktär. Visserligen finns möjlighet att ansluta externt tangentbord men eftersom dessa inte levereras som standard kan det ej förutsättas att samtliga användare har möjlighet till förenklad inmatning. En användare arbetar helt enkelt med en uppsättning grafiska inmatningsenheter. Vidare placeras de grafiska elementen i vad man kallar formulär (eng. form), vilket kan jämföras med ett fönster i andra grafiska miljöer. De mest förekommande grafiska komponenterna som finns tillgängliga genom Palm OS API visas i tabell 5.1.

Komponent	**Utseende**	**Funktion**
Button		En enkel knapp.
Field		Textinmatningsfält.
Checkbox		Komponent av typen av eller på.
Slider		Erhåll ett värde genom att placera slidern.
List		Valalternativ genom en lista.

Tabell 5.1 – De vanligaste grafiska komponenterna hos Palm OS

Efter uppstart kommer programmen att köra initieringskod i funktionen `AppStart()`. Här är det tänkt att man ska initiera variabler och ladda och initiera eventuell annan data. T.ex. så kontrolleras här om databasen preferences finns tillgänglig.

```
static Err AppStart(void)
{
    UInt16 prefsSize;

    /* initiering */
    myGlobalVariable=22;
    myLinkedList=NULL;

    /* Kontrollera om det finns en preferences databas. */
    prefsSize = sizeof(MyAppPreferenceType);
    if (PrefGetAppPreferences(appFileCreator, appPrefID, &g_prefs,
         &prefsSize, true) != noPreferenceFound)
    {
        /* Vidta åtgärder för att återställa applikationens
           tillstånd här */
    }
    return errNone;
}
```

Palm OS är ett händelsestyrt operativsystem, vilket påminner om t.ex. Windows. Vid varje händelse (eng. event) skickar systemet information om detta till den exekverande applikationen, detta kan vara allt från applikationsrelaterade händelser som t.ex. att användaren trycker ner pennan eller att applikationen just laddat en fil, till mer systemspecifika händelser som t.ex att minne måste flyttas. En Palm-applikation svarar helt enkelt på olika händelser genom att exekvera kod beroende på händelsens typ.

Efter den initierande funktionen `AppStart()` exekveras `AppEventLoop()`, vilket är den s.k. händelseslingan.

```
static void AppEventLoop(void)
{
      UInt16 error;
      EventType event;

      do {
      EvtGetEvent(&event, evtWaitForever);
            if (! SysHandleEvent(&event))
                  if (! MenuHandleEvent(0, &event, &error))
                              if (! AppHandleEvent(&event))
                                          FrmDispatchEvent(&event);

      } while (event.eType != appStopEvent);
}
```

`EventType` är ännu en fördefinierad typ, just `EventType` är en enorm stuktur med en stor mängd information om en eventuell händelse. `AppEventLoop()` använder en slags pollingteknik för att hämta events, `evtWaitForever` anger att programmet ska vänta tills systemet skickar applikationen en händelse.

Händelser kan vara av olika typ. I prioriteringsordning skickar man vidare händelsen till olika hanteringsfunktioner (eng. handlers). Först kontrolleras om systemet i sig tar hand om händelsen, i.o.m. `SysHandleEvent(&event)`. Det kan ju hända att en händelse inte alls har med den körande applikationen att göra. Det som är viktigt för en utvecklare är metoden `AppHandleEvent()`, hit kommer alla händelser som man kan tänkas vilja svara på. En liten tabell över de vanligaste händelserna samt deras förekomst presenteras i appendix 1. Väl inne i `AppHandleEvent()` kontrolleras vilket formulär som är aktuellt för närvarande och motsvarande händelsehanterare anropas.

```
static Boolean AppHandleEvent(EventType * eventP)
{
    UInt16 formId;
    FormType * frmP;

    if (eventP->eType == frmLoadEvent)
    {
        formId = eventP->data.frmLoad.formID;
        switch (formId)
        {
            /* Här kontrolleras vilken händelsehanterare som ska anropas
        case Form_A:
```

```
            FrmSetEventHandler(frmP, Form_A_HandleEvent);
            break;

        case Form_B:
            FrmSetEventHandler(frmP, Form_B_HandleEvent);
            break;

..

        default:
            break;

        }
        return true;
    }
    return false;
}
```

När program avslutas, vilket oftast sker i och med att en användare byter till ett annat program anropas programmets `AppStop`() funktion. Här är det lämpligt att avallokera minne man reserverat och att spara undan nödvändiga variabler i preferences-databasen för att kunna återställa programmet till befintligt tillstånd nästa gång det startas.

```
            static void AppStop(void)
            {
               /* spara undan värden på variabler och skriv sedan detta till
            preferences */
               PrefSetAppPreferences(
                    appFileCreator, appPrefID, appPrefVersionNum,
                    &g_prefs, sizeof(tessstPreferenceType), true);

                FrmCloseAllForms();
            }
```

Ett väldigt enkelt flödesschema över den statiska koden i de flesta Palmprogram visas i figur A3.1 i appendix 3. I tabell A3.1 visas en förklaring till symboler som används i flödesscheman.

6. Implementering

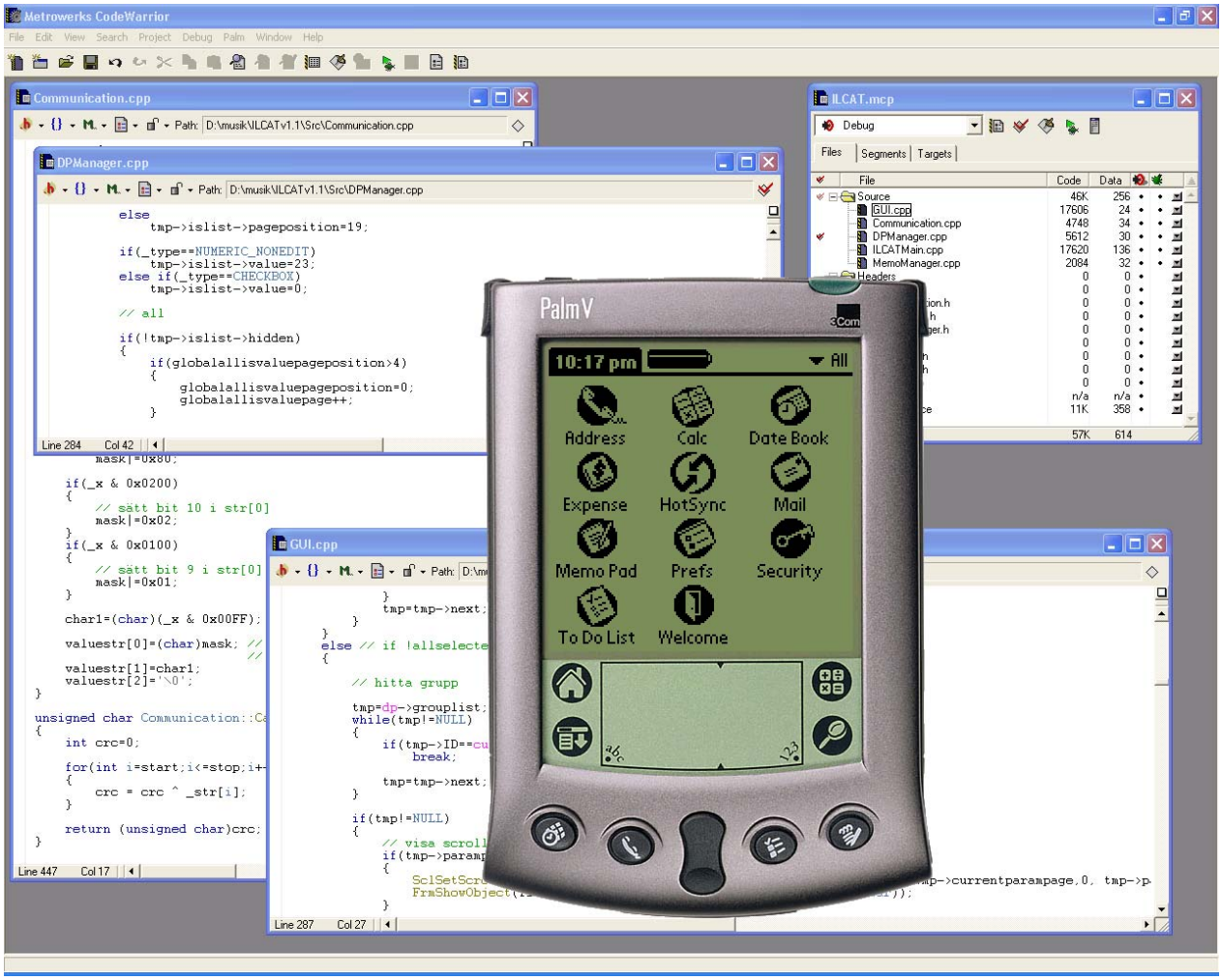
6.1 Utvecklingsverktyg

Som tidigare nämnts kan Palm OS programmeras i nästan valfritt språk, för att nämna några finns kompilatorer för Java, C, C++ och Basic. Program för Palm OS skrivs dock med fördel i C eller C++, kanske inte för att det är den bästa lösningen utan för att det är det språk som många redan befintliga Palm-applikationer är skrivna i, dessutom bygger hela referensmanualen till operativsystemet från Palm på C.

För att implementera ILCAT (Infrared Link Controller Administration Tool) har C/C++-versionen av utvecklingsmiljön CodeWarrior från Metrowerks [24] använts. Utvecklingsmiljön liknar t.ex. Microsoft Visual Studio. CodeWarrior är således en komplett miljö med editor, kompilator, länkare, avlusare (eng. debugger) och kommer dessutom med ett flertal extra verktyg som kan komma till nytta under utvecklingsarbete. T.ex. kan programmet Constructor nämnas, som påminner om ett enkelt ritprogram och används för att utveckla grafiska användargränssnitt. Tillbehör som Constructor brukar finnas inbyggda i de flesta större moderna utvecklingsverktyg för programutveckling. CodeWarrior finns med kompilatorstöd för ett flertal språk.

Ett verktyg som underlättar utvecklingsarbetet avsevärt är POSE (Palm Operating System Emulator), ett program som körs under t.ex. Windows och som näst intill exakt emulerar en verklig Palm handdator. Palm levererar via sin hemsida [19] kopior (eng. images) av de ROM som sitter i en fysisk Palm handdator och POSE har stöd för att ladda dessa. Man kan på så sätt enkelt testa program på olika versioner av Palm OS.

CodeWarrior har stöd för POSE, man kan direkt från editorn genom enkla tangenttryckningar kompilera sitt program och ladda över den körbara filen till POSE, med eller utan stöd för avlusning. Att kunna göra detta går betydligt snabbare än att först kompilera och sedan via HotSync ladda över sitt program till en fysisk Palmenhet. Figur 6.1 visar Metrowerks CodeWarrior tillsammans med POSE.



Figur 6.1 – Metrowerks Codewarrior tillsammans med POSE

För att skriva program för konvertering mellan olika filformat och dylikt har den fria C/C++ kompilatorn DJGPP [25] använts.

6.2 Användargränssnitt

Det grafiska användargränssnittet är skapat i Constructor. Applikationen innehåller sex stycken formulär.

- *UserDetails* - Visas vid uppstart och vid menyvalet användare, se figur 7.1.
- *MainForm* - För att presentera ärvärden, inställningar och den lista där användaren har möjlighet att välja grupp, se t.ex. figur 7.2 och 7.3. Detta formulär innehåller även menyn.
- *ManualValues* - För inmatning av manuella värden, se figur 7.5.
- *LoadSetFile* - Presenterar listan med inställningsfiler då användaren väljer att ”ladda setfil”, se figur 7.6.
- *EditString* - Används för att presentera en sträng, ger dessutom editeringsmöjlighet, se figur 7.9.
- *ViewString* - Presenterar en sträng, ej editeringsmöjlighet. Liknar *EditString*.

De grafiska komponenterna i t.ex. figur 7.2 eller för manuella värden i figur 7.5 skapas ej dynamiskt utan är redan placerade på formuläret. Således finns fem rader med olika kontroller utritade på formuläret *MainForm*. Vid uppstart är samtliga kontroller släckta, d.v.s. de visas ej, och beroende på vilken typ av värde som ska visas tänds kontroller. Mer om olika datatyper och tillhörande kontroller i 6.3.

6.3 Enhetsprofiler

För att kunna representera olika typer av ärvärden, inställningar och manuella värden har typerna enligt tabell 6.2 definierats.

Numeriskt värde	**Beskrivning**	**Möjlig för:**	**Storlek (byte)**
0, CHECKBOX	Checkbox/Booleskt värde. Kommer för ärvärden att presenteras som ”Ja” eller ”Nej”. För inställningar och manuella värden visas denna typ som en checkbox.	Ärvärden, inställningar och manuella värden.	2
1, NUMERIC_NON_EDIT	Ej editerbart numeriskt värde	Ärvärden.	2
2 , NUMERIC	Editerbart numeriskt värde. Visas med hjälp av knapparna plus och minus.	Inställningar och manuella värden.	2
3, LIST	Valmöjlighet. Visas grafiskt som en dropdown-lista med val angivna enligt enhetsprofil.	Inställningar.	2
4, MODE	”Mode selector”. Visas grafiskt som en dropdown-lista med val angivna enligt enhetsprofil.	Inställningar.	2
5, TEXT_NONEDIT40	Ej editerbar sträng, 40 tecken.	Inställningar.	40
6, TEXT40	Editerbar sträng, 40 tecken.	Inställningar och manuella värden.	40
7, TEXT_NONEDIT10	Ej editerbar sträng, 10 tecken.	Inställningar.	10
8, TEXT10	Editerbar sträng, 10 tecken.	Inställningar.	10
9, DECIMAL	Flyttal med två decimaler.	Ärvärden och inställningar.	2

Tabell 6.2 - Möjliga typer för att skapa enhetsprofiler

Dessa nio typer antas räcka för att skapa en tillfredsställande enhetsprofil. Typen MODE är speciell, denna används för att ställa om hela regulatorn till ett annat läge (eng. mode). T.ex. kan det finnas ett läge där regulatorn får sin insignal externt via ett överordnat styrsystem vilket

kanske kommer eliminera en del ärvärden eller inställningar. Detta är således ett sätt att ändra hela beteendet hos regulatorn med en enda datatyp.

En komplett enhetsprofil skulle enligt syntaxen i appendix 4 se ut enligt nedan.

```
G,Temperatur,1,0
G,Övrigt,2,0
G,CAN,3,1
A,Kanaltemp,1,1,Grader,0
A,Extern sensor,1,0,,0
A,Brandvarnare inst.,2,0,,0
I,Maxtemp,1,2,0,50,0,Grader,0
I,Mode,2,4,-1,-1,1,,3,ModeA,ModeB,ModeC
I,CANparam1,3,0,-1,-1,1,,0
I,CANparam2,3,3,-1,-1,1,,2,Can_A,Can_B
M,Fungerande don,0,-1,-1,,
M,Uppmätt maxflöde,2,10,30,l/s
```

Eftersom detta lätt blir oöverskådligt presenteras ovanstående enhetsprofil enligt nedanstående tabeller som troligtvis ska vara lite mer lättförståeliga. Tabell 6.3 visar grupperna, tabell 6.4 ärvärdena, tabell 6.5 inställningarna och slutligen tabell 6.6 de manuella värdena.

Grupper

Namn	id	Säkerhetsnivå
Temperatur	1	0
Övrigt	2	0
CAN	3	1

Tabell 6.3 – Grupper i exempelenhetsprofil

Ärvärden

Namn	Gruppid	Typ	Enhet	Säkerhetsnivå
Kanaltemp	1	1	Grader	0
Extern sensor	1	0	-	0
Brandvarnare inst.	2	0	-	0

Tabell 6.4 – Ärvärden i exempelenhetsprofil

Inställningar

Namn	Grupp-Id	Typ	Min	Max	Säkerhet	Enhet	Antal List-element	Elem1	Elem2	Elem3
Maxtemp	1	2	0	50	0	Grader	0	-	-	-
Mode	2	4	-	-	1	-	3	modeA	modeB	modeC
CANparam1	3	0	-	-	1	-	0	-	-	-
CANparam2	3	3	-	-	1	-	2	can_A	can_B	-

Tabell 6.5 – Inställningar i exempelenhetsprofil

Manuella värden

Namn	Typ	Min	Max	Enhet
Fungerande don	0	-	-	-
Uppmätt maxflöde	2	10	30	l/s

Tabell 6.6 –Manuella värden i exempelenhetsprofil

Denna enhetsprofil innehåller tre grupper, varav de två första har säkerhetsnivå 0 och gruppen *CAN* har säkerhetsnivå 1. Till grupp ett (id 1), här kallad *Temperatur,* läggs två ärvärden, *kanaltemp* som är ett numeriskt värde och *extern sensor* som är en checkbox/booelsk typ (visas som ”Ja” eller ”Nej” eftersom detta är ett ärvärde). Till gruppen *Övrigt* (id 2) skapas ett ärvärde kallat *Brandvarnare inst*, även detta ett booleskt värde. Gruppen *CAN* (id 3) har inga ärvärden.

Till gruppen *Temperatur* skapas en inställning med namnet *Maxtemp* som är av typ 2, d.v.s. visas som ett numeriskt värde med + och – knappar. Här är det även inlagt en minimal temperatur på 0 grader och en maximal temperatur på 50 grader. Inställningen *Maxtemp* har även tilldelats enheten *grader*. Till gruppen *Övrigt* skapas en inställning av typ 3, vilket är en ”mode selector”. Denna lista tilldelas tre element kallade *modeA, modeB* och *modeC*. Gruppen *CAN* har två inställningar, *CANparam1* och *CANparam2*. *CANparam1* är en checkbox och *CANparam2* är en lista med två element benämnda *can_modeA* och *can_modeB*. Eftersom gruppen *CAN* har säkerhetsnivå 1 måste man ange rätt säkerhetskod vid uppstart av programmet för att den ska bli synlig.

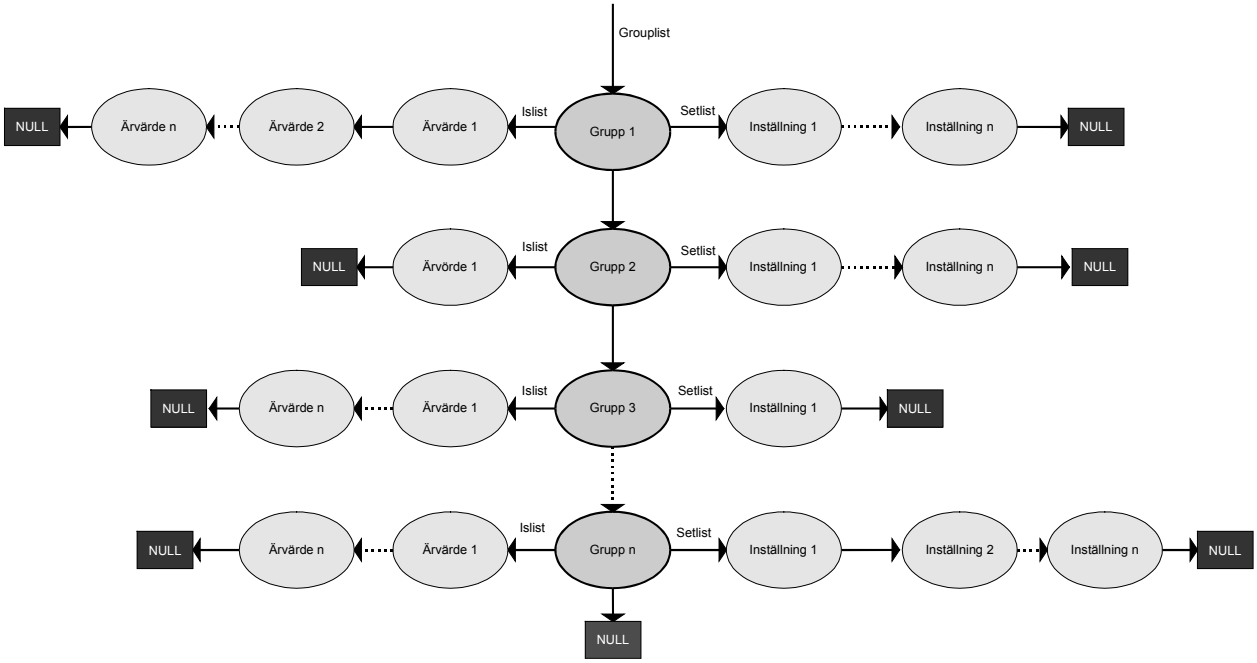
Slutligen ingår även två manuella värden i enhetsprofilen. Först en checkbox som symboliserar *Fungerande don* och slutligen ett numeriskt värde för *Uppmätt maxflöde.* Manuella värden tillhör som sagt ingen grupp och har heller ingen säkerhetsnivå.

6.4 Applikationen ILCAT

I det stora hela följer även ILCAT strukturen för en generell Palmapplikation vilken är beskriven i 5.4 med den stora skillnaden att det finns en mer komplicerad datastruktur i botten, fler formulär och därmed också fler händelsehanterare.

Applikationen arbetar med grupper som består av olika till gruppen relaterade ärvärden och inställningar. Dessutom finns manuella värden, men de saknar grupptillhörighet. Vilka grupper och tillhörande ärvärden och inställningar som finns tillgängliga definieras helt av en enhetsprofil. Eftersom det finns en hel del olika enhetsprofiler kommer innehållet i handdatorns minne att vara olika beroende på aktuell enhetsprofil, d.v.s. dynamiskt. Efter en del initierande kod exekveras `AppHandleEvent()`, vilken beroende på aktuellt formulär anropar motsvarande händelsehanterare. Eftersom det finns sex formulär i programmet finns det även sex händelsehanterare. I figur A3.2 visas ett flödesschema över vilka händelsehanterare som anropas beroende på aktuellt formulär.

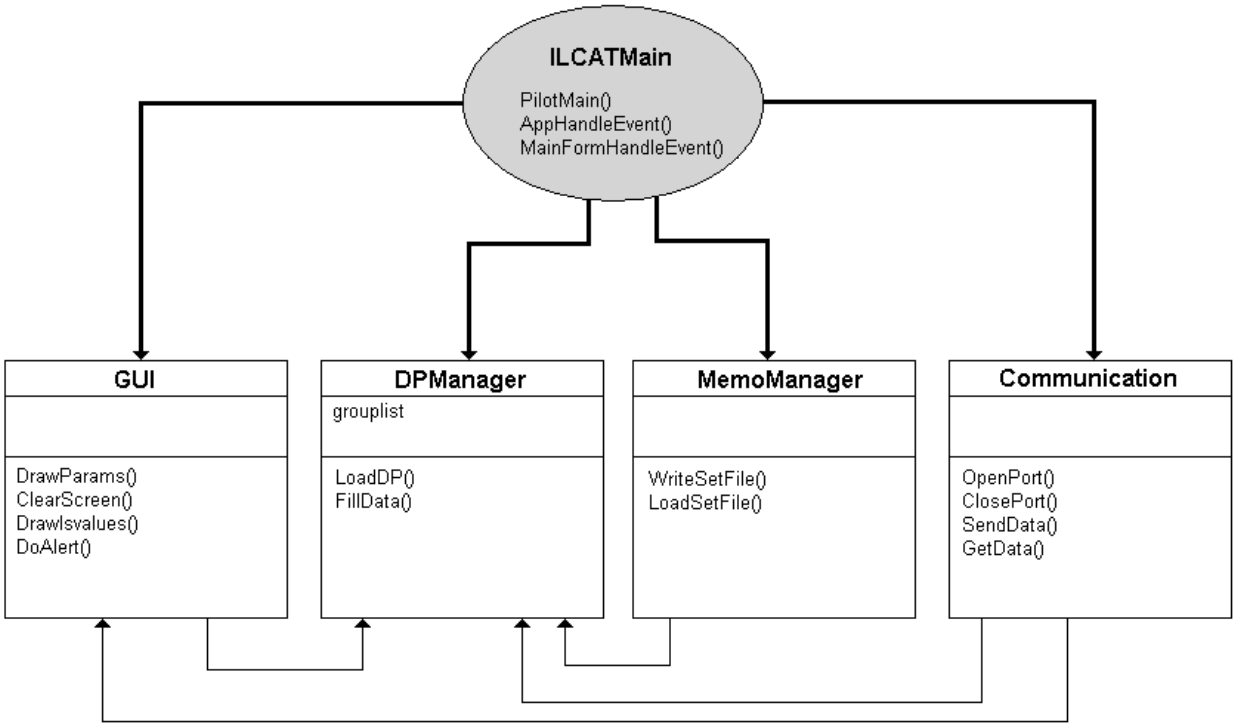
Generellt opererar programmet på en datastruktur bestående av sammankopplade länkade listor enligt figur 6.2 nedan.



Figur 6.2 - generell datastruktur som används i ILCAT

Grouplist är en pekare till listan med alla grupper. I varje element i *grouplist* finns ytterligare två pekare, *setlist* vilken pekar på en länkad lista med alla till gruppen relaterade inställningar och *islist* vilken pekar på en lista med alla till gruppen relaterade ärvärden. Utöver detta lagras manuella värden i en enkellänkad lista vilken ej visas i figur 6.2 ovan.

Programmet innehåller fyra klasser. En figur över de fyra klasserna samt huvudprogrammet visas i figur 6.3, där även ett urval av klassernas medlemsfunktioner och medlemsvariabler visas.



Figur 6.3 – Applikationens klasser och huvudprogram samt deras beroende.

- GUI - Hanterar det grafiska användargränssnittet
- DPManager - Hanterar enhetsprofiler
- MemoManager - Hanterar databasen MemoDB
- Communication - Hanterar kommunikationen

Eftersom programmet är händelsestyrt anropas klassernas medlemsfunktioner från `AppHandleEvent()` beroende på att en användare tryckt på olika grafiska alternativ. I figur A3.4 finns flödesschema över den viktigaste händelsehanteraren, `MainFormHandleEvent()`, dock är kommunikationen ej med här eftersom den visas i ett annat flödesschema, se avsnitt 6.5. De medlemsfunktioner som är av intresse att ge en utförligare förklaring till presenteras nedan.

- `bool DPManager::LoadDP(char* deviceprofile)`

Metod för att skapa den dynamiska datastrukturen enligt figur 6.2 ovan. `LoadDP()` tar en enhetsprofil som argument vilket internt representeras av en sex byte lång sträng. Metoden startar med att se om dess argument är giltigt, d.v.s. om en databas med motsvarande namn finns lagrad i handdatorns minne. Om så är fallet öppnas denna databas och data extraheras. `LoadDP()` anropar därför diverse hjälpfunktioner för att skapa och sätta in nya listelement i datastrukturen, `DPManager::InsertGroup()`, `DPManager::InsertIsvalue()` och `DPManager::InsertParam()`. Utöver grupper, ärvärden och inställningar skapas här även den separata länkade listan med manuella värden, metoden `DPManager::InsertManual()` används till detta. `LoadDP()` returnerar `true` om skapandet av datastrukturen lyckades, eller `false` om något blev fel. Fel kan t.ex. uppstå då metoden ej kunde hitta enhetsprofilen i minnet, eller att det inte fanns tillräckligt mycket ledigt minne för att skapa ett nytt listelement.

- `bool DPManager::FillData(char* data)`

`Filldata()` är en metod för att fylla datastrukturen, skapad av `LoadDP()`, med data som är hämtat från ett don. Metoden tar en sträng som argument som innehåller den nya informationen. Eftersom en datastruktur är skapad när denna metod anropas kan man enkelt traversera strukturen och beroende på typ plocka ut rätt antal byte från argumentet och konvertera dessa. `FillData()` anropar `int DPManager::StrToValue(char* str)` för detta ändamål. En metod som tar en sträng som argument, omvandlar denna till ett heltal enligt definitionen för numeriska värden vilken presenteras senare i detta avsnitt. `StrToValue()` returnerar det konverterade värdet. Om typen däremot är en sträng kopieras beroende på om det är en sträng för 40 tecken eller tio tecken, rätt antal byte från argumentet in i datastrukturen. Precis som `LoadDP()` returnerar `FillData()` `true` eller `false` beroende på om metoden lyckades fylla strukturen eller ej.

- `int Communication::SendData()`

Denna metod anropas då en användare vill skicka inställningar till ett don. Till skillnad från `FillData()` måste `SendData()` omvandla ett värde till motsvarande 2-byte representation (se nedan) vilket sker med metoden `char* Communication::ValueToStr(int value)`. `SendData()` traverserar åter igen datastrukturens inställningar och sätter ihop en lång sträng, en s.k. sendstring. Beroende på inställningens typ kopieras innehållet (typer av textkaraktär) eller så omvandlas det m.h.a. `ValueToStr()` och läggs till sendstring. Slutligen sätter `SendData()` samman ett SEND_DATA_PACKET (se nedan) och skickar detta över IR-länken. Metoden returnerar antalet byte som skickades över länken.

- `void GUI::DrawSettings()`

För att tända kontroller i samband med att användaren vill titta på inställningar anropas `DrawSettings()`. Metoden traverserar datastrukturen enligt figur 6.2 ovan och tänder kontroller av motsvarande typ. D.v.s. om ett element i en inställningslista har typ CHECKBOX kommer motsvarande checkbox att tändas. I fallet med en checkbox kommer också kontrollens namn (eng. label) att ändras beroende på namnet i listelementet. `DrawSettings()` utför liknande operationer för de andra datatyperna specificerade enligt tabell 6.2.

- `void GUI::ClearScreen()`

Metod för att rensa innehållet på skärmen. Alla kontroller som nu visas kommer att släckas.

När programmet avslutas anropas funktionen `AppStop()`, vilken utför följande operationer:

- Skriver ner värden på variabler av global karaktär i databasen preferences
- Om en enhetsprofil är laddad skapas en ny databas där värden på element i datastrukturen enligt figur 6.2 lagras
- Datastrukturen avallokeras
- Programmet avslutas

Vid uppstart anropas funktionen `AppStart()`, vilken utför följande operationer:

- Öppnar databasen preferences, återställer värden på globala variabler
- Om en enhetsprofil var aktiv då programmet avslutades senaste gången kommer detta att framgå av variabler i databasen preferences och databasen skapad i `AppStop()` öppnas och strukturen återställs

6.5 Kommunikationslänken Palm ↔ IDCC

Palm OS SDK tillhandahåller ett antal Managers genom vilka funktionsanrop sker. En Manager som är av intresse och som introducerades i Palm OS version 3.0 för att seriellt kunna kommunicera med omvärlden är Serial Manager. En tabell över de mest användbara funktionerna som Serial Manager API tillhandahåller visas i tabell 6.7.

Funktion	**Beskrivning**
`SrmOpen()`	Öppna en seriell port. Kan även öppna en infraröd port.
`SrmReceive()`	Ta emot data genom en av `SrmOpen()` öppnad port.
`SrmSend()`	Skicka data över en av `SrmOpen()` öppnad port.
`SrmControl()`	Sätt parametrar till en öppen port, som t.ex. hastighet, antal stopbitar eller paritet.

Tabell 6.7 – Vanligt förekommande funktioner hos Palm OS Serial Manager

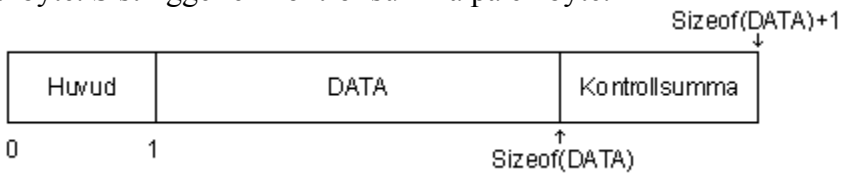
Vanligtvis används Serial Managern för kommunikation via den i Palmenheter inbyggda seriella porten. `SrmOpen()` kan även dirigeras att öppna en IR-port om en sådan finns tillgänglig på handdatorn i fråga. Genom Serial Managerns funktioner `SrmSend()` och `SrmReceive()` kan man således ”låtsas” att man kommunicerar via seriell port medan Serial Manager egentligen översätter det man skickar till IrDA-kompatibla ljuspulser. `SrmOpen()` kan med olika argument öppna de lager av IrDA-stacken som Palm stödjer. ILCAT är implementerad med hastigheten 19200 bitar per sekund.

För att få en tillförlitlig kommunikationslänk har fyra olika paket definierats, se tabell 6.8, tabellen ger även en kort beskrivning till de olika paketen.

Paket	**Beskrivning**
INFO_PACKET	Används för att initiera kommunikationen samt att skicka bekräftelser (eng. acknowledgements).
DP_PACKET	Skickas från donet till handdatorn för att tala om vilken enhetsprofil som är aktiv.
CURRENT_DATA_PACKET	Skickas från donet till handdatorn och innehåller dess inställningar och ärvärden.
SEND_DATA_PACKET	Skickas från handdatorn till donet och innehåller nya inställningar.

Tabell 6.8 – Paket som används i applikationen samt en kort beskrivning.

Generellt har alla paket ett utseende enligt figur 6.4. Först ett huvud (eng. header) som är en byte vilket innehåller information om t.ex. typ av paket. Därefter kommer data (information) i varierande antal byte. Sist ligger en kontrollsumma på en byte.



Figur 6.4 – Generellt utseende hos paket

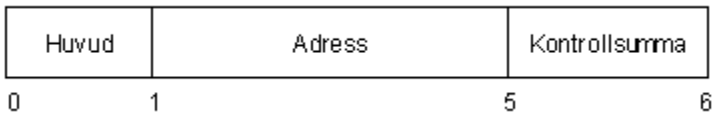
Kontrollsumman beräknas med hjälp av den logiska XOR-funktionen.

$$kontrollsumma(k) = paketbyte(k)\ XOR\ kontrollsumma(k-1), k\ heltal$$
$$kontrollsumma(1) = paketbyte(1)$$

där 0<k<n. n är längden på hela paketet.

De fyra paket som används i kommunikationen tillsammans med förklaring till de olika fälten samt eventuella värden presenteras nedan.

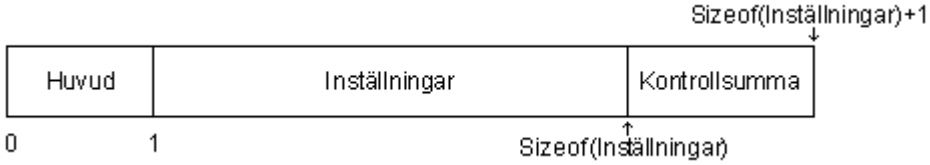
INFO_PACKET



Huvud - Innehållet i denna byten specificerar olika slags funktion, 1 byte.
- 0x52 - Hämta enhetsprofil, förbered för utgående data.
- 0x4E - Hämta enhetsprofil, förbered för inkommande data.
- 0x48 - Begär data.
- 0x4B - Överföringen lyckades.
- 0x45 - Fel i överföringen.

Adress - Adress som anger vilken enhet paketet gäller, 4 byte

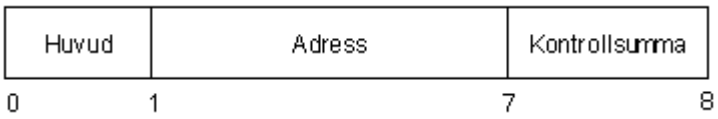
SEND_DATA_PACKET



Huvud - Typ av paket, 1 byte.
- 0x53 - Ny Data.

Inställningar - De aktuella inställningarna i handdatorn, sizeof(Inställningar) byte.

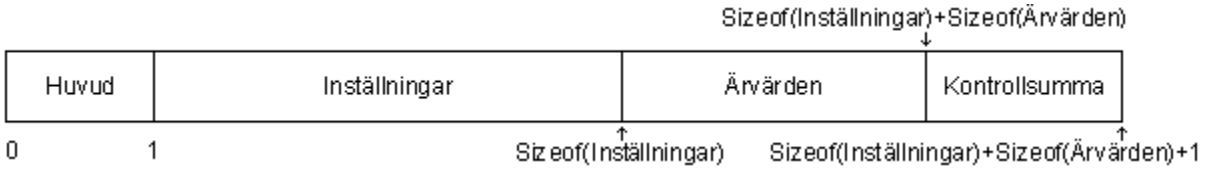
DP_PACKET



Huvud - Typ av paket, 1 byte.
0x44 - Enhetsprofil.

Enhetsprofil - Donets enhetsprofil, 6 byte.

CURRENT_DATA_PACKET

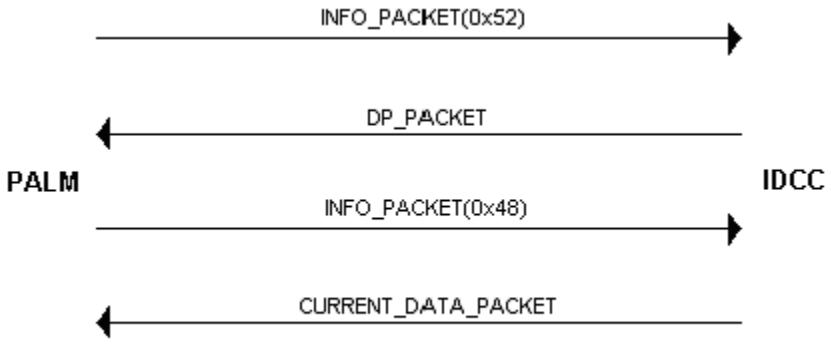


Huvud - Typ av paket, 1 byte.
0x43 - Aktuell data

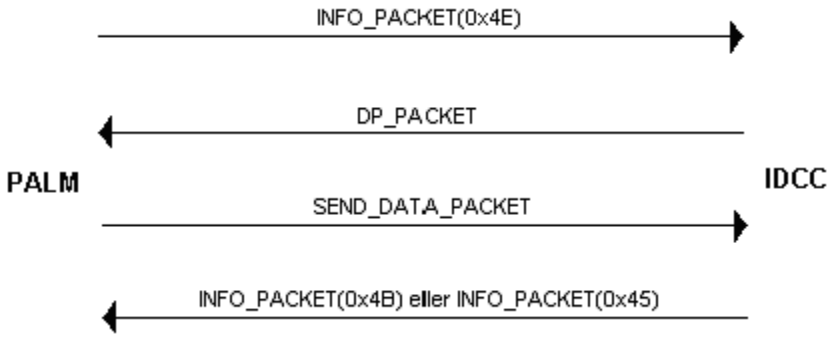
Ärvärden - Donets aktuella ärvärden, sizeof(Ärvärden) byte.

Inställningar - Donets aktuella inställningar, sizeof(Inställningar) byte.

En överskådlig bild av hur paket skickas när man hämtar data visas i figur 6.5 och när man skickar data i figur 6.6.



Figur 6.5 – Paketordning för operationen hämta data.



Figur 6.6 – Paketordning för operationen skicka data.

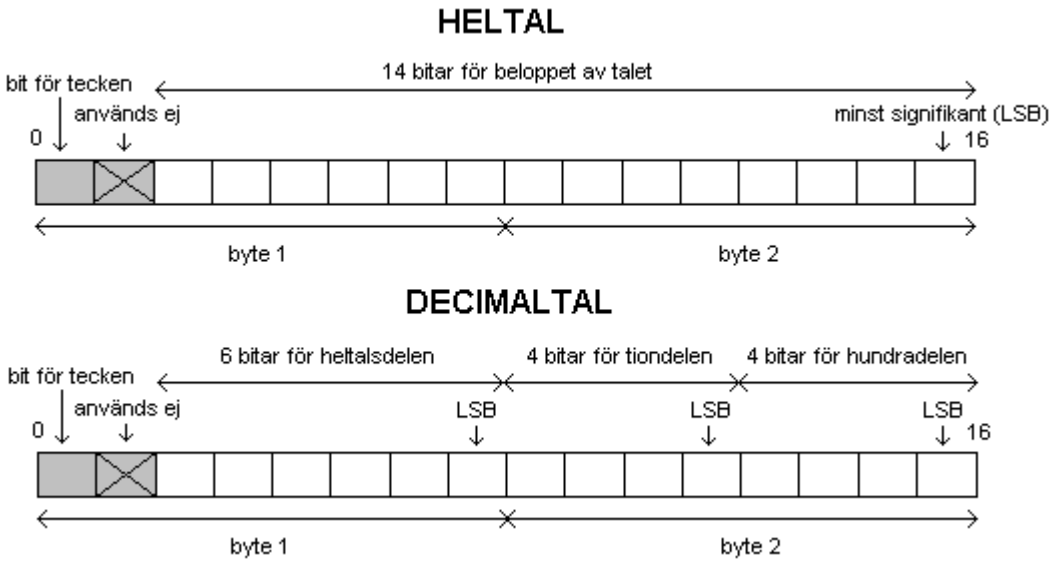
När en användare väljer att hämta data från ett don startar kommunikationen med att handdatorn skickar ett INFO_PACKET med header 0x52. Om kommunikationen lyckades får sändaren tillbaka ett DP_PACKET. Om detta paket är giltigt skickas ett nytt INFO_PACKET iväg, nu med header 0x48. Om även detta lyckades svarar donet med ett CURRENT_DATA_PACKET. När data hämtats och ett giltigt DP_PACKET och CURRENT_DATA_PACKET anlänt anropar ILCAT metoden `LoadDP()`, som tar fältet enhetsprofil från DP_PACKET som argument. Med ett giltigt paket menas en korrekt kontrollsumma och rätt typ av paket, d.v.s. rätt pakethuvud. `LoadDP()` kontrollerar om det finns en enhetsprofil med detta namn lagrat som en databas i handdatorn och skapar dessutom en datastruktur som motsvarar den aktuella enhetsprofilen, se beskrivning av `LoadDP()` ovan.

När väl datastrukturen är skapad anropar applikationen metoden `FillData()` vilken tar fälten inställningar och ärvärden från CURRENT_DATA_PACKET som argument. Eftersom informationen i ett detta paket ligger lagrat med avseende på enhetsprofilen och att handdatorn har samma enhetsprofil är det enkelt att traversera datastrukturen och beroende på typ plocka ut rätt antal byte från paketet och konvertera dem till motsvarande värden, se beskrivning av `FillData()` ovan.

När en användare skickar över ny data från handdatorn till ett don skickas först ett INFO_PACKET med kommandot 0x4E. Donet svarar med ett DP_PACKET. På detta sätt kan man enkelt kontrollera att användaren inte försöker skicka data mot fel typ av don, d.v.s. mot fel enhetsprofil. Om denna kontroll lyckas skickar ILCAT den nya datan i ett SEND_DATA_PACKET. När donet har tagit emot och hanterat detta skickas ett INFO_PACKET tillbaka till applikationen för att se om kommunikationen fungerade och att rätt data levererades. Beroende på om överföringen lyckades, om det blev timeout eller om kontrollsumman ej stämmer står detta i första byten i INFO_PACKET, d.v.s. ett huvud som är 0x4B eller 0x45. Eftersom donet och applikationen beräknar kontrollsummor med samma algoritm kan båda sidorna av kommunikationslänken enkelt verifiera korrektheten i ett pakets datainnehåll.

Om man tittar tillbaka på tabell 6.2 ser man att det finns en storlek i byte utskrivna efter varje typ där numeriska värden (typ 0,1,2,3,4 och 9) skickas som två byte (16 bitar) och typer som är av textkaraktär (typ 5,6,7 och 8) skickas som en byte/tecken vilka kodas enligt ASCII-standard (American Standard Code for Information Interchange). Numeriska värden kan tolkas antingen

som rena heltal eller som decimaltal. Se figur 6.6 nedan för beskrivning av de olika bitarnas betydelse för numeriska värden.



Figur 6.6 – Bitrepresentation hos de två byte för numeriska värden

Första biten avgör tecknet hos det numeriska värdet både för heltal och för decimaltal. Typerna 0, 1, 2, 3 och 4 har en bitrepresentation enlig heltal i figur 6.6, d.v.s. att de sex minst signifikanta bitarna i den mest signifikanta byten tillsammans med hela den minst signifikanta byten anger beloppet av det numeriska värdet. Typen 9 har en tolkning enligt decimaltal i figur 6.6, d.v.s. att de sex minst signifikanta bitarna i den mest signifikanta byten anger heltalsdelen. De fyra mest signifikanta bitarna i den minst signifikanta byten tiondelen och slutligen de fyra minst signifikanta bitarna i den minst signifikanta byten hundradelen.

I nuläget används ej de fyra byte för adress vilka ingår i ett INFO_PACKET. Detta är tänkt för den framtida expansionen att kunna hämta och skicka data från och till två eller flera enheter vars infraröda mottagare är placerade inom räckhåll för handdatorn.

Ett översiktligt flödesschema över kommunikationen hos handdatorn visas i figur A3.3.

6.6 Konverteringsprogram för enhetsprofiler

För att konvertera enhetsprofiler från syntax i appendix 4 till ett av Palm läsbart format har programmet DPtoPDB utvecklats. Programmet är kommandoradsbaserat och har följande syntax.

```
DPtoPDB <infil> <utfil> <namn> <separatortecken>
```

I tabell 6.8 visas en mer genomgående beskrivning till de olika argumenten.

Argument	Beskrivning
infil	Namn på fil med indata i form av en enhetsprofil specificerad enligt appendix 4
utfil	Namn på den av programmet genererade pdb-filen, vilken är läsbar av Palm OS. Det är denna fil som ska överföras till en handdator via hotsync.
namn	Namn på enhetsprofilen, det är detta namn man anger för att öppna databasen i Palm OS. Jämför med strukturen över en PDBHeader i kapitel 5.3.
separatortecken	Det tecken som har använts i infil för att separera olika argument enligt appendix 4.

Tabell 6.8 - argument till programmet DPtoPDB

DPtoPDB skapar vid start fyra huvuden till länkade listor, ett för grupper, ett för ärvärden, ett för inställningar och ett för manuella värden. Vidare har fyra nedan angivna strukturer definierats för att lagra information från enhetsprofilen.

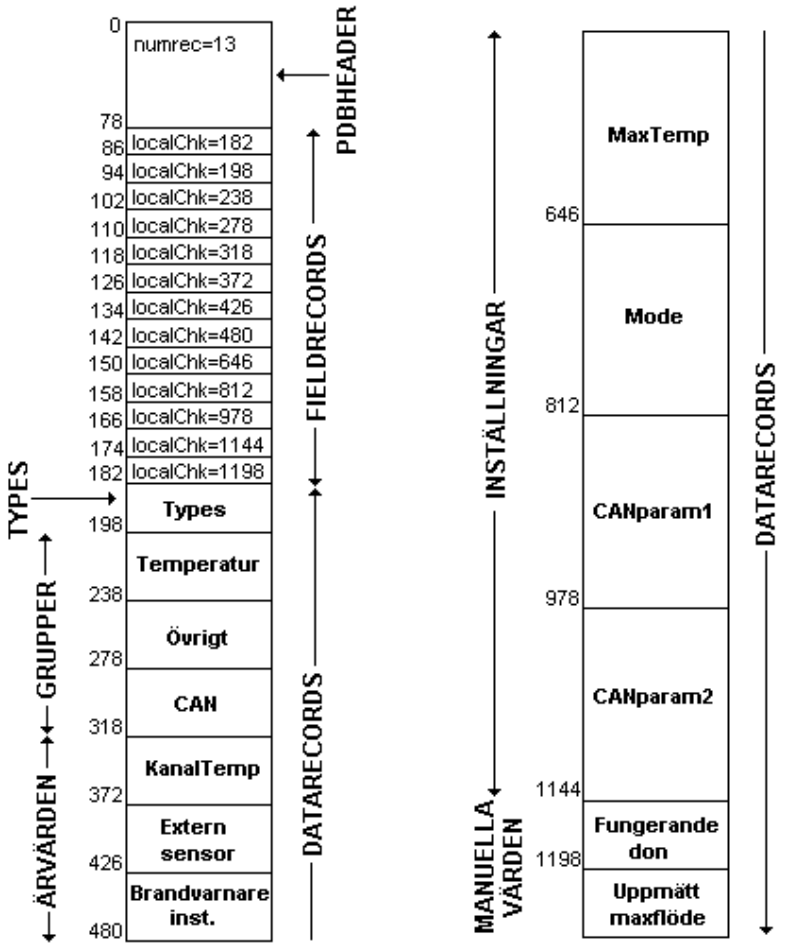
```
struct group
{
              char name[32];
              int id;
              int sec level;
};

struct isvalue
{
              char name[32];
              char unit[10];
              int gruppid;
              int sec level;
              int type;
};

struct setting
{
              char name[32];
              char unit[10];
              char listitems[5][10];
              int gruppid;
              int type;
              int sec level;
              int max;
              int min;
              int nolistitems;
};

struct manual value
{
              char name[32];
              char unit[10];
              int type;
              int max;
              int min;
};
```

DPtoPDB läser rad för rad i textfilen där enhetsprofilen är lagrad, plockar ut de separerade värdena och beroende på vad raden definierar skapas nya listelement av respektive typ (se strukturer ovan). Dessutom skapas en fjärde struktur, `struct types`, för att hålla reda på antalet grupper, ärvärden, inställningar och manuella värden. Detta underlättar extrahering i applikationen på handdatorn. Informationen i dessa fyra länkade listor kommer att utgöra datarecords då utfilen tolkas som en record-databas, se figur 5.2.

```
struct types
{
            int number_of_groups;
            int number_of_isvalues;
            int number_of_settings;
            int number_of_manuals;
};
```

Programmet skapar även strukturen PDBHeader, se kapitel 5.3, och beroende på infilens innehåll ett antal FieldHeader-strukturer. DPtoPDB räknar ut och sätter rätt localchunk i dessa fälthuvuden och fyller även i strykturen PDBHeader med nödvändig information. Slutligen skrivs PDBHeader, alla FieldHeader-strukturer samt informationen från de fyra länkade listorna ner till utfilen.

Precis som i figur 5.2 följer record-databasen för enhetsprofiler den generella mallen, men med lite andra datarecords. I figur 6.7 visas hur databasen som DPtoPDB skapar kan se ut för den exempelenhetsprofil som användes i avsnitt 6.3.



Figur 6.7 – Utseendet hos den av DPtoPDB skapde record-databasen

6.7 Injusteringsprotokoll

Om man har valt att synkronisera handdatorn mot Microsoft Outlook kommer informationen från applikationen Memopad att synkroniseras till anteckningar i Outlook. Programmet Ips (injusteringsprotokollskaparen) har utvecklats för att skapa injusteringsprotokoll. Programmet genererar ett antal kommaseparerade filer beroende på antalet olika enhetsprofiler i indatafilen.

Programmet är kommandoradsbaserat och har följande syntax.

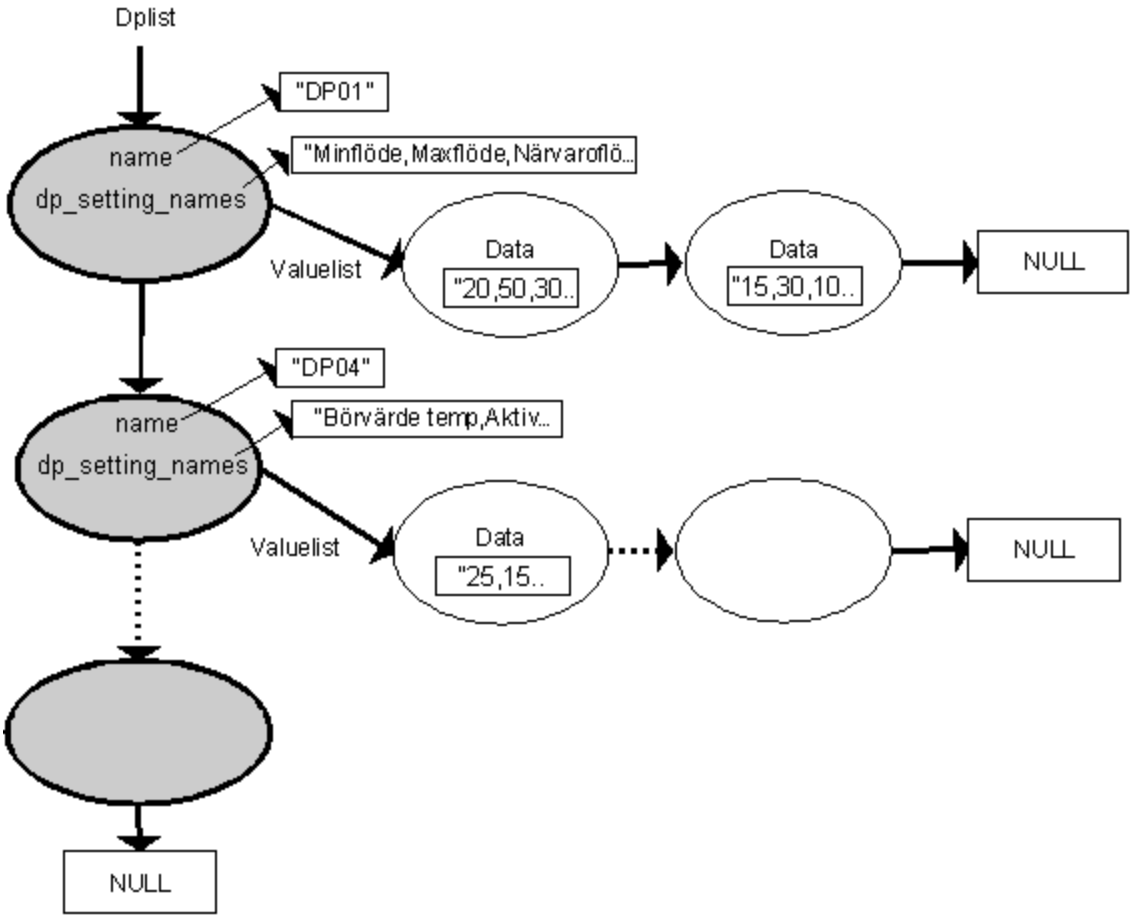
```
ips <infil>
```

`Infil` är den fil som erhålls då man från Microsoft Outlook markerar anteckningar och sparar dessa som en textfil. Denna operation i Outlook kommer spara alla markerade anteckningar som en fil.

Programmet skapar vid start en länkad lista, `dplist`, vars noder innehåller tre olika dataelement.

- Sträng för enhetsprofilens namn, `name`
- Sträng för lagring av namn på enhetsprofilens inställningar (t.ex. Maxflöde, Minflöde), `dp_setting_names`. Dessa namn kommer lagras kommaseparerade i strängen.
- Pekare till en länkad lista, `valuelist`, vars dataelement är en sträng, `data`, där inställningarnas värden skrivs ner i kommaseparerat format.

I figur 6.8 visas hur denna datastruktur ser ut.



Figur 6.8 – Datastrukturen i applikationen Ips.

Eftersom en inställningsfil skapad av ILCAT har ett fördefinierat utseende, se figur 7.11, kan Ips enkelt extrahera information om inställningars namn och värden . Ips läser rad för rad ur den av Outlook skapade textfilen och sätter in nya element i datastrukturen enligt figur 6.8. Om en inställning påträffas som motsvarar en ny enhetsprofil, d.v.s. den finns inte i listan, skapas ett nytt listelement i `dplist` och namnet på enhetsprofilen lagras nodens variabel `name`. Dessutom skapas även ett nytt element i nodens `valuelist` och inställningarnas värden skrivs in i `valuelist->data`. Om en inställning påträffas som motsvarar en enhetsprofil som redan finns i strukturen kommer endast ett nytt element i nodens `valuelist` att skapas och inställningarnas värden lagras.

När `infil` är slut kommer programmet att traversera `dplist` och skapa filer med filnamn enligt värdet i nodernas variabel `name`, d.v.s. `dplist->name`. I dessa filer skrivs först nodens sträng `dp_setting_names` och därefter traverseras `dplist->valuelist` och alla `dplist->valuelist->data` skrivs ner. Resultatet är ett antal filer beroende på antalet olika enhetsprofiler i indatafilen.

7. Resultat

7.1 Användargränssnitt

Ett program för att titta på ärvärden och inställningar samt möjlighet att skicka inställningar till ett tilluftsdon har implementerats på en Palm handdator. Programmet är indelat i två lägen, ett för ärvärden och ett för inställningar. I varje läge presenteras ett antal knappar som är aktuella för det rådande läget. Förutom ärvärden och inställningar finns även manuella värden vilka kan fyllas i då användaren väljer att spara en inställningsfil (se nedan). När applikationen startas presenteras ett formulär där användaren kan fylla i användarinställningar i form av namn, företag, telefonnummer samt en säkerhetskod, se figur 7.1.

Efter detta inledande formulär visas i figur 7.2 och 7.3 det generella utseendet hos ILCAT. Om en grupp innehåller mer information än vad som får plats på skärmen visas en rullningslist i skärmens högra kant .

Figur 7.1 – Användare

Figur 7.2 – Ärvärden

Figur 7.3 – Inställningar

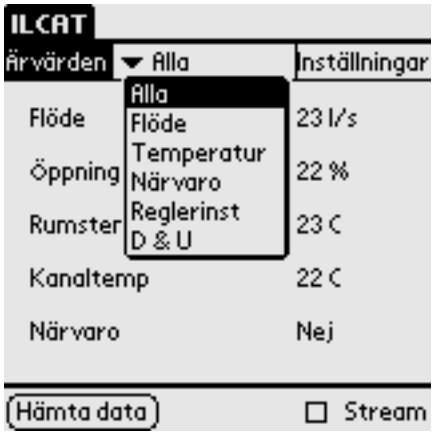
För att få en indelning på ärvärden och inställningar är dessa grupperade i grupper. Som exempel kan nämnas att man har en grupp som heter ”Temperatur” där alla ärvärden och inställningar som har med temperatur att göra läggs in. Möjligheten att titta på allt samtidigt finns också genom valet ”Alla” i listan. Figur 7.4 visar listan med grupper som är placerad mitt emellan knapparna ärvärden och inställningar.

Som ses i figur 7.2 och 7.3 finns i de två lägena ett antal knappar. Funktionen hos dessa presenteras i tabell 7.1. Inställningsfiler är i applikationen kallade ”setfiler” p.g.a. grafisk platsbrist.

Knapp	**Funktion**
Hämta data	Hämta data från ett don.
Skicka	Skicka iväg nuvarande inställningar.
Ladda setfil	Ladda en tidigare sparad inställningsfil.
Stream	Kontinuerlig hämtning av data.

Tabell 7.1 - funktion för knappar i grafiskt användaregränssnitt

I figur 7.6 visas det formulär som presenteras när en användare väljer att ladda en inställningsfil. Här presenteras alla de inställningsfiler som finns lagrade i handdatorn.



Figur 7.4 – Möjlighet att välja mellan olika grupper

Figur 7.5 – Formulär för att fylla i manuella värden

Figur 7.6 – Formulär för att ladda inställningsfiler

Programmet har ett menysystem där förutom samma funktioner som finns via knapparna i de två lägena även finns ett par extra funktioner. Figur 7.7 illustrerar de olika valen i menyn.

Figur 7.7 – Menysystemet hos ILCAT

De extra menyfunktionerna är *Spara setfil*, *Hjälp*, *Om* och *Användare*. *Hjälp* startar Clipper och presenterar en PQA-fil som innehåller en kort introduktion till programmet samt ett mindre felsökningsschema. Valet *Användare* visar återigen formuläret med användarinformation, se figur 7.1, vilket ger användaren en möjlighet att ändra dessa inställningar under exekvering. *Spara setfil* tar användaren till ett nytt formulär där det finns möjlighet att fylla i manuella värden samt filnamn, se figur 7.5. Mer om inställningsfiler i 7.5.

Applikationen innehåller ett antal extra funktioner. För att öka eller minska ett numeriskt värde kan man trycka på plus- eller minusknapparna, se t.ex. figur 7.5, eller bara trycka på namnet på den inställningen man vill ändra, detta indikeras med en liten pil till vänster om inställningen. När pilen syns kan användaren använda tangenterna scrollup och scrolldown, vilka är två av de sex

standardtangenterna för att öka respektive minska inställningens värde. En markerad inställning visas i figur 7.8, här kan man också se rullningslisten.

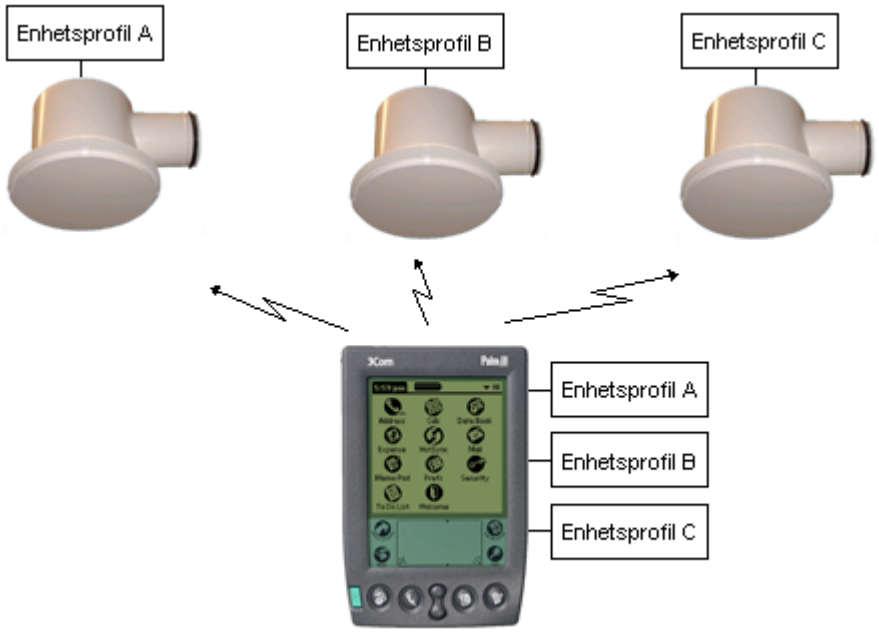
Programmet arbetar även med strängar, vilka p.g.a. sin storlek kan vara svåra att presentera. Om man trycker på en inställning som motsvarar en sträng presenteras ett nytt formulär med bättre editeringsmöjligheter, se figur 7.9.

Figur 7.8 – Markerad inställning för enklare ändring.

Figur 7.9 – Textinmatning för längre strängar.

7.2 Enhetsprofiler

Olika don har olika mängd kringutrustning, t.ex. kanske ett don inte har en brandvarnare installerad, då vill man inte att grafiska komponenter för en brandvarnare ritas ut. Det som således visas då man väljer att titta på t.ex. ärvärden för en grupp ska vara helt generiskt och måste byggas upp dynamiskt under applikationens exekvering beroende på vilken typ av don man kommunicerar med. Att göra ett statiskt användargränssnitt och kompilera olika versioner för olika typer av don är en lösning som inte är speciellt smidig. Istället har ett system med s.k. enhetsprofiler utvecklats. Varje don kommer att definiera de ärvärden och inställningar som är intressanta att kommunicera och presentera på handdatorn i enlighet med en enhetsprofil, vilken finns lagrad i donets mjukvara. Handdatorn kommer således att innehålla ett flertal profiler för att kunna kommunicera med olika don. Beroende på enhetsprofilen i donet kan rätt grafiskt användargränssnitt visas. Figur 7.10 visar hur systemet med enhetsprofiler fungerar.



Figur 7.10 – Systemet med enhetsprofiler.

En enhetsprofil innehåller information om vilka ärvärden och inställningar som är intressanta för varje specifik enhet. I enhetsprofilen finns t.ex. också information om typ av värde, om värdet har några maximala eller minimala gränser samt om det är någon speciell säkerhetsnivå till detta värde. Ärvärden och inställningar kan grupperas för enklare navigering.

Förutom grupper, ärvärden och inställningar har man via enhetsprofilen möjlighet att ange manuella värden, vilka måste anges när en användare väljer att spara en inställning, se t.ex. figur 7.5. Manuella värden ger programmet en ytterligare möjlighet att spara undan information som varken är ärvärden eller inställningar, d.v.s. som inte har direkt anknytning till regulatorn. Som exempel kan nämnas att ett manuellt värde kan vara en checkbox för ”Fungerande don” eller en numerisk komponent för ”Uppmätt maxflöde”.

Enhetsprofiler är ett sätt att lösa den dynamiska delen av användargränssnittet. Dessa profiler skapas i en vanlig texteditor enligt en speciell syntax, se appendix 4. Textfilerna konverteras till ett av Palm läsbart format (record-databaser), se kapitel 6.6 för mer information angående konverteringen, och laddas slutligen in i handdatorn.

Enligt syntaxen för enhetsprofiler har man när man skapar en profil möjlighet att för grupper, ärvärden och inställningar ange en säkerhetsnivå (ett tal mellan 0 till 4). Denna möjlighet gör att man kan skapa hela grupper eller kanske endast ett ärvärde som bara är synligt för de användare som uppfyller rätt säkerhetsnivå. När man startar programmet har man möjlighet att ange en säkerhetsnivå, se figur 7.1. Detta ger en extra dimension till programmet då man kan ha data för mer avancerade inställningar men som kanske inte är nödvändiga för vardagliga injusteringar. Dessutom skapar man möjligheten att ha med variabler av felsökningskaraktär, t.ex. någon intern variabel i regulatorn som ofta brukar orsaka problem. Det sistnämnda kan spara tid när ett don inte fungerar som det ska och måste felsökas.

För att på ett enkelt sätt kunna konvertera enhetsprofiler enligt syntax i appendix 4 till ett av Palm OS läsbart format (pdb) har programmet DPtoPDB skrivits. Konverteringsprogrammet är skrivet för att konvertera filer från ett operativsystem som kör på Intel eller Intel-kompatibel processor, t.ex. Windows.

7.3 Kommunikationslänken Palm ↔ IDCC

Enligt IrDA-specifikationen ska en enhet som kommunicerar via IrDA SIR, vilket både Palm och MCP2120 gör, garantera korrekt data inom ett avstånd på 1m och en vinkel på ±15°. Givetvis är det svårt att göra noggranna vinkelbedömningar och avståndsbedömningar för hand, men överlag uppfyller ILCAT specifikationen mer än väl. Räckvidden är oftast minst det dubbla mot vad IrDA specificerar. I tabell 7.1 presenteras resultatet av kommunikationslänkens prestanda för olika vinklar och avstånd.

Vinkel (°)	Avstånd (m)	Resultat	Anmärkning
0	0.5	Ok	
	1	Ok	
	1.5	Ok	
	2	Ok	
	~3+	ej ok	timeout
30	0.5	Ok	
	1	Ok	
	1.5	Ok	
	2	Ok	
	~2.5+	ej ok	tveksam
50+	0.5	ej ok	-
	-	-	

Tabell 7.1 – Resultat, kommunikationslänken Palm ↔ IDCC

Ljus reflekteras mot ytor och beroende på egenskaper i det reflekterande materialet kommer en andel energi att gå förlorad. Dessutom är s.k. diffus spridning vanlig vid ojämna material. Inga tester har gjorts för att simulera vad detta skulle betyda, men för dagligt bruk hinner reflektioner i t.ex. bordsytor, golv, underplåten till donet mm. avta innan de åter når sändaren, och väl där ha en för låg energi för att fångas upp av den infraröda mottagaren. Om detta mot all förmodan skulle inträffa kommer paketens kontrollsumma ej att överrensstämma med det man förväntar sig och ett felmeddelande kommer presenteras för användare. IrDA har själva gjort ett djupgående test som behandlar yttre störningar i form av solljus, elektromagnetiska fält, gasfyllda lysarmaturer och andra externa störningar, för information om detta test refereras till [4].

Som mjukvaran i IDCC är implementerad i skrivande stund fungerar ej kommunikationen då motorn arbetar.

Äldre versioner av Palm OS än 3.5 innehåller också en Serial Manager, men eftersom en ny version av denna med ett antal extra funktioner släpptes i samband med version 3.5 är det denna som används vid implementeringen. Programmet fungerar därför ej på äldre Palm handdatorer.

7.4 Inställningsfiler och injusteringsprotokoll

För att få en överblick av vad man har gjort är det lämpligt att kunna spara en inställning, framförallt om man använder ILCAT till att injustera ett flertal don i en större byggnad. Vidare är det av intresse att kunna ladda in en sparad inställning. Man skulle kunna tänka sig att man skapar inställningsfiler som innehåller en typinställning för ett speciellt don. Om man definierade en egen struktur och skrev data i en databas för detta ändamål skulle man även behöva en annan applikation för att titta på dessa filer.

Istället används den i Palm redan inbyggda Memopad, ett litet program för att skriva anteckningar (eng. memo) och som finns förinstallerat som standard på alla Palm handdatorer. För att lagra anteckningarna har Memopad en egen databas, vilken man även kan öppna och läsa från eller skriva till från andra program. När en användare väljer att spara en inställningsfil sparas aktuella inställningar och de manuella värdena som ett memo i Memopads databas. Förutom inställningar och manuella värden sparas även namn på användaren, klockslag och datum samt vilken enhetsprofil det rör sig om. På detta sätt kan man enkelt gå in i Memopad och läsa vad man gjorde för inställningar. Inställningsfilerna går således att ladda in igen. I figur 7.11 visas hur en inställningsfil kan se ut i Memopad.

Figur 7.11 – En inställningsfil i applikationen Memopad

För att utföra synkronisering med en stationär dator finns enligt 5.1 ett flertal program tillgängliga, s.k. Conduits. Om man väljer att synkronisera mot Microsoft Outlook kommer t.ex. alla memos (från Memopad) som finns i handdatorn att överföras som anteckningar i Outlook, detta har vi tagit vara på för att skapa ett injusteringsprotokoll. För detta ändamål har programmet Ips (Injusteringsprotokollskaparen) utvecklats. Genom att markera de inställningsfiler (anteckningar) man är intresserad av i Outlook, spara dessa som en textfil och till slut köra Ips genereras kommaseparerade filer. De flesta kalkylprogram (t.ex. Excel) har stöd för kommaseparerade filer och på detta sätt kan man på ett enkelt sätt få upp ett protokoll över vad man gjort för inställningar. Detta underlättar avsevärt om man vill ha en mer överskådlig bild om man har gjort inställningar på t.ex. 200 don.

8. Analys

8.1 Användargränssnitt

Palm OS tillhandahåller funktioner för att dynamiskt skapa grafiska kontroller (checkbox, field, list o.s.v.). Ett alternativ var att använda dessa istället för de fem rader med statiska kontroller som finns i nuvarande läge. Det hade i så fall fungerat så att det tillsammans med varje element som skapas i den generella datastrukturen även hade skapats en grafisk kontroll beroende på elementets typ. Detta hade gett programmet ett helt dynamiskt utförande. På detta sätt skulle man t.ex. kunna ange var på skärmen man vill positionera en speciell kontroll.

Möjligheten att skapa kontroller dynamiskt introducerades i Palm OS 3.5 men var där inte speciellt stabila utan innehöll en hel del buggar. Funktionerna fungerar stabilt först i Palm OS 4.0. Eftersom ILCAT ska fungera på Palm OS 3.5 som tidigaste version kunde denna teknik ej tillämpas. Istället introducerades de statiska kontrollerna som tänds och släcks beroende på vad som ska visas. Vilket sätt man än väljer att använda är resultatet transparent för användaren. Skillnaden skulle vara att man med dynamiskt skapade kontroller ej begränsats till fem kontroller per sida.

8.2 Enhetsprofiler

En applikation för att generera databaser av de enhetsprofiler man skapar enligt syntaxen i appendix 4 är nödvändig. Det finns en hel del mjukvara som kan konvertera diverse filer till det av Palm OS läsbara pdb-formatet. Dessa program är ofta av speciell karaktär för bestämda ändamål. Visserligen finns det gratis mjukvara som konverterar en textfil eller en kommaseparerad fil till en Palm databas men om man använder dessa måste man lösa problemet med att extrahera data på handdatorn. Av denna anledning har ett eget konverteringsprogram (DPtoPDB) utvecklats. Tekniken med typkonvertering tillämpas för att skapa ett enkelt extraheringsförfarande på Palmsidan.

Programmet DPtoPDB konverterar en textfil skapad på ett system kör på Intel-kompatibel processor. Anledningen till att detta är värt att nämna är att olika kompilatorer omvandlar processorns representation av numeriska värden som är större än en byte (eng. multi-byte datatypes) på olika sätt. Intel och Intel-kompatibla processorer tolkar numeriska värden i vad som kallas för små endianer. Palm OS använder som bekant processorer från Motorola vilket tolkar data i stora endianer. Som exempel kan nämnas att en `unsigned short int` (16 bitar) som tilldelas värdet 0x07 tolkas som0x0070 i små endianer medan det i stora endianer tolkas som 0x0700. Eftersom detta är två helt olika tal krävs det en omvandlig av numeriska värden i konverteringsprogrammet för att få rätt värden väl inne i handdatorn.

Systemet med enhetsprofiler innehåller dock ett antal begränsningar, t.ex. kan man bara ha en datatyp av typ MODE per enhetsprofil. Generellt gäller också att endast fem val är tillgängliga per datatypen LIST, däremot kan man till skillnad från typen MODE ha flera kontroller av typen LIST per enhetsprofil. För att kommunicera med IDCC är dock detta ingen begränsning, t.ex. kan nämnas att regulatorn i sitt maximala utförande kan operera i tre olika lägen.

8.3 Applikationen ILCAT

En viktig funktionalitet är att program som en gång startats ska återkomma till det tillstånd de avslutades i nästa gång det startas. För detta ändamål finns databasen preferences vilken är beskriven ovan och denna räcker långt. Men eftersom ILCAT under exekvering innehåller en dynamiskt datastruktur är det svårt att deklarera variabler i strukturen för preferences vilka alltid ska motsvara datastrukturen. Man skulle tillsammans med pekare och genom att lagra information om hur många grupper, ärvärden och inställningar man har tillsammans med aktuell enhetsprofil komma runt detta men jag har istället valt att skapa en ny databas för att lagra datastrukturens värden. Denna teknik har visat sig fungera bra och användaren återvänder till samma tillstånd som programmet avslutades i. Detta innebär givetvis en nackdel eftersom ärvärden är momentana värden och blir snabbt inaktuella, speciellt i snabba reglersystem. ILCAT innehåller för närvarande ingen kontroll om hur länge sedan det var en uppsättning ärvärden var aktuella.

De två byte som används för att representera numeriska värden har ett område från -16384 till 16384 för heltal. Om det numeriska värdet tolkas som ett decimalt tal är området -31.99 till 31.99. Decimal tolkning infördes för att ge möjlighet till bättre precision vid injustering, t.ex. kan temperatur vara en inställning som ska ha tiondel. För närvarande används en 10-bitars AD-omvandlare i donet varför de till belopp större talen ej används.

Att utnyttja alla bitar för att representera data är dock ingen smart lösning med tanke på framtida utveckling. Det kan mycket väl hända att man upptäcker att det skulle behövas en extra bit för att kännetecknа en ny egenskap, vilket skulle kunna exemplifieras med bitar för skriv- och läsrättigheter. Eftersom inga större datamänger överförs vid kommunikation med IDCC skulle en numerisk datatyp däremot ha kunnat representeras av ett större antal byte utan märkbara prestandaförluster. En helt annan tanke är att alltid använda en byte för att skicka ett numeriskt värde, vilket kan tolkas som 255 steg, och tillsammans med enhetsprofilerna definiera en mall för hur varje inställningen eller ärvärde ska tolkas, exempelvis i form av en offset och en multiplikator. På detta sätt skulle man kunna representera väldigt stora tal, men vara begränsad till de 255 steg som en byte erbjuder.

8.4 Kommunikationslänken Palm ↔ IDCC

Systemet klarar avstånd som är längre än vad IrDA specificerar. Troligtvis beror den längre räckvidden på att man genom att montera olika stora resistorer tillsammans med TFDS4500 kan öka känsligheten och styrkan. Dock får man vara noga med att inte överstiga gränsen för vad som kan skada ögonen. Men man kan placera sig betydligt närmare den gränsen i IDCC jämfört med motsvarande konfiguration i handdatorn, eftersom den ej går att ändra. Ökar man känsligheten ökar således också komponentens känslighet för brus vilket måste tas i beaktning vid val av resistor. TFDS4500 sitter monterad en bit från hålen i bottenplattan till IDCC. Detta ger en mindre rymdvinkel för sändning och mottagning vilket inverkar negativt på maximala vinklarna i resultatet. En tänkbar lösning är att montera ut kretsen med en sockel för att öka denna vinkel.

En klar nackdel med att kommunicera i det lägsta lagret av IrDA-stacken är att ingen felkontroll erhålls, något som finns implementerat i högre lager. Istället har jag, för att minimera priset, valt att själv implementera en felkontroll i samband med pakethantering.

Vidare kan man fråga sig hur det blir om två don är placerade riktigt nära varandra. Nu är det emellertid så att för att få en tillfredsställande luftcirkulation kommer två IDCC aldrig att placeras på mindre än två meters avstånd från varandra. Systemet är däremot redan förberett för kommunikation där enheter sitter nära varandra och man endast vill kommunicera med ett av dem. Paketdefinitionen i kapitel 6 ger möjlighet att tillsammans med INFO_PACKET ange en adress vilken kan användas för detta ändamål. I det befintliga systemet används inte möjligheten med adress utan är endast tillagt för framtida vidareutveckling.

Ett annat problem som teoretiskt kan uppstå, men som jag aldrig råkat ut för, är då man efter att ha skickat data från handdatorn till ett don tar emot ett INFO_PACKAGE som av någon anledning har fel kontrollsumma. D.v.s. att ILCAT säger till användaren att överföringen av nya inställningar misslyckades, medan den i själva verket inte gjorde det. Det finns lösningar till detta men av anledningen ovan har jag valt att låta den rådande implementeringen bestå. T.ex. skulle man kunna tänka sig att skicka samma information mer än en gång, en teknik som visserligen inte tar en märkbar tid eftersom överföringen sällan handlar om mer än 500 byte. Denna lösning bedömdes vara osmidig.

Det går ej att kommunicera med donet då motorn arbetar. Detta är en direkt följd av att programmet arbetar med interrupts och stänger således av processorns möjlighet att lyssna efter dessa då den arbetar med motorn. Att PIC16F har en 2-nivåers FIFO ger heller inte direkt utrymme för lagring för senare bruk.

8.5 Inställningsfiler och injusteringsprotokoll

Skapandet av inställningsfiler och injusteringsprotokoll förlitar sig på den inbyggda applikationen Memopad och synkronisering mot Outlook. Denna teknik är inte alls bra dels med tanke på att alla användare kanske inte har Microsoft Outlook, och dels att detta program kräver en windowsmiljö, vilket ej kan förutsättas. Möjligheten att en användare kan titta på en inställningsfil direkt i Memopad och se vad det är för slags inställning är givetvis smidig men det introducerar samtidigt en risk då programmet bygger på att texten i detta memo är rätt formaterad. Om en användare väljer att ändra texten i Memopad utan att veta om denna formatering kommer inställningsfilen ej att kunna laddas.

9. Avslutning

Primärt handlar detta examensarbete om att utveckla en intelligent fjärrkontroll till tilluftsdonet IDCC med hjälp av en Palm handdator. Genom införandet av enhetsprofiler kan ILCAT användas som fjärrkontroll mot ett flertal andra enheter. Systemet med enhetsprofiler erbjuder alltså ett mycket dynamiskt system som kan kommunicera med ett stort antal enheter.

LindinVent AB planerar p.g.a. de relativt små kostnader rent hårdvarumässigt som en infraröd länk kostar att i framtiden inkludera detta system i samtliga sina produkter där det kan vara av intresse att ha en enkel fjärrkontroll i form av en applikation på en Palm handdator. Eftersom systemet med enhetsprofiler gör detta möjligt och att allt programmeringsarbete redan är klart är det enda som behövs ett program på den nya enheten med stöd för infraröd kommunikation via protokollet och pakethanteringen som examensarbetet specificerar.

För att en enhet ska fungera tillsammans med ILCAT krävs följande:

- Enheten ska ha en uppsättning ärvärden vilka är av betydelse att visualisera
- Enheten ska ha en uppsättning inställningar vilka genom sin ändring kommer påverka systemet
- Möjlighet till infraröd kommunikation via IrDA SIR
- Mjukvaran i enheten ska kunna hantera enhetsprofiler och pakethantering specificerade enligt detta examensarbete

9.1 Andra tekniker, andra plattformar

En stor uppgradering hade varit att använda radio för kommunikationen istället, t.ex. Bluetooth. Men med tanke på de relativt IR stora kostnaderna är det orimligt om en större del av priset en kund betalar för ett tilluftsdon består i en fungerande Bluetooth-länk. Ett flertal produkter på marknaden innehåller möjlighet till kommunikation via IR och då oftast IrDA. Exempelvis skulle man kunna tänka sig att implementera en variant av ILCAT på en mobiltelefon. Dagens modeller har stöd för Java. Den oftast lilla displayen på en mobiltelefon skapar dock ingen vidare möjlighet till ett intuitivt gränssnitt. Visserligen är upplösningen den samma om inte bättre men displayen är fortfarande liten.

Vidare kan man tänka sig att en applikation för operativsystem som används på vanliga bärbara datorer implementeras, t.ex. Windows eller någon variant av Linux. Eftersom bärbara datorer oftast innehåller en infraröd port tillsammans med den betydligt större displayen har man möjlighet att skapa ett mycket intuitivt gränssnitt, däremot är en bärbar dator en klumpig lösning att ta med sig ut i fält. PocketPC är en annan variant av handdator, även dessa har stöd för infraröd kommunikation via IrDA vilket gör att de har samma förutsättningar för ett program liknande ILCAT som Palm har. PocketPC är dock aningen dyrare än vad en Palm är. Vem vet vad framtiden erbjuder, men troligtvis kommer vi få billigare hårdvara för Bluetooth och handdatorer som har stöd för det, vilket inbjuder till en uppgradering till radiokommunikation.

9.2 Framtida utveckling

I skrivande stund är ILCAT ett fungerande system för injustering och övervakning av ärvärden hos LindinVents tilluftsdon IDCC. Givetvis finns det utrymme för framtida vidareutveckling. En funktion som är av intresse men som ligger utanför ramen för detta examensarbete är möjligheten att kunna ladda över ny regulatormjukvara till donet via handdatorn. Inga ytterliggare undersökningar har gjorts om vad som krävs för att göra detta möjligt, men det hade troligtvis varit extremt kostnadseffektivt att ha möjligheten att via t.ex. en hemsida ta hem den senaste mjukvaran för donet och uppgradera med hjälp av handdatorn. För detta ändamål krävs mer mjukvara i donet, en s.k. bootloader, vilket inte är installerat för tillfället. Dessutom kanske det hade varit intressant att kunna ansluta handdatorn via kabel, fortfarande seriellt gränssnitt, för denna överföring då den kan ta lite längre tid. Eftersom IR kräver fri sikt och en vinkel inom specifikationerna kan det dessutom vara svårt att uppfylla dessa villkor vid en överföring utan kabel. Kontakt för anslutning via kabel är redan på plats i IDCC, och hos handdatorn är det inte speciellt mycket som behöver ändras.

För att få en mer tillförlitlig kommunikation som t.ex. fungerar även då motorn arbetar krävs troligtvis att programmet i donet opererar med fler exekveringspunkter, d.v.s i realtid. Att köra programmet under ett RTOS hade troligtvis även underlättat programmeringsarbetet. IDCC kommer inom en snar framtid att uppgraderas till en kraftigare processor, PIC18F. Huruvida det är möjligt att köra ett realtidsoperativsystem på denna har ej undersökt närmare.

Konverteringsprogrammet DPtoPDB är som tidigare nämnts ett kommandoradsbaserat program vilket inte är speciellt användarvänligt. Visserligen ligger ansvaret att skapa enhetsprofiler hos personal hos LindinVent AB men med dagens möjligheter att skapa grafiska program, t.ex. under Windows, hade ett bättre grafiskt gränssnitt givetvis varit på sin plats. Detta program skulle hantera administration och skapande av enhetsprofiler.

Datatyperna som enhetsprofilerna stödjer har definierats tillsammans med handledare på LindinVent AB utefter behovet i samband med IDCC. Det hade det varit bra att följa en mall för vad andra system använder för datatyper för att öka kompabiliteten, t.ex. LON och CAN är intressant. Samma sak gäller med de ovan definierade paketen vilka inte heller följer någon slags standard. För att öka kompatibiliteten även här skulle en mer generell definition av paketen ha behövts. Tonvikten har dock legat på att få ett fungerande system varvid egna datatyper och paket definierades.

Jag har tidigare nämnt Conduits, det program som kan installeras och köras i samband med Hotsync. En Conduit är inte skriven för ILCAT utan temporärt har vi istället löst problemet att få över information om sina inställningar till en stationär dator via det tidigare beskrivna systemet med Memopad och att synkronisera mot Microsoft Outlook. En Conduit tillsammans med en applikation på den stationära datorn som sköter presentationen, synkroniseringen och den eventuella omvandlingen till kalkylblad av alla injusteringsfiler hade därför varit önskvärd. Faktum är att denna applikation skulle kunna integreras med det program som sköter skapandet av enhetsprofiler. På detta sätt hade man fått en komplett applikation på en stationär dator för att:

- Skapa enhetsprofiler
- Synkronisera enhetsprofiler via Hotsync
- Skapa injusteringsfiler
- Synkronisera injusteringsfiler via Hotsync
- Skapa injusteringsprotokoll utifrån ett antal injusteringsfiler

IDCC kommer troligtvis inom en snar framtid förses med ett extra externt minne med vars hjälp mjukvaran i donet kan lagra logdata. Programmet kan t.ex. lagra värden på rumstemperatur och flöde var femte minut. En möjlighet i ILCAT att ta hem logdata och presentera detta i en kurva med lämpliga enheter och varierbara skalor en tänkbar framtida utveckling.

Ett system för att skapa undergrupper hos enhetsprofiler är ännu en framtida utvecklingsmöjlighet vilket skulle öka möjligheterna och skapa ett mer flexibelt system. D.v.s. ett system för att skapa sammanlänkade profiler och möjlighet att skapa beroende mellan två eller flera profiler. Vidare skulle en jämförelse mellan enhetsprofilerna hos CAN och ILCAT behövas för att se om enhetsprofilerna hos ILCAT kan integreras med de hos CAN, vilket skulle eliminera de enhetsprofiler som ILCAT nu bygger på helt.

10. Referenser

[1] Expertbilaga till allergiutredningens betänkande SOU 1989:77

[2] Thorsell Klimatkonsult AB, *Jämförelse mellan CAV- och VAV-system för luftbehandling i kontorsbyggnader*. http://www.lindinvent.se/PRODUKTER-VAV/CAV_VAV10.pdf, (2002-12-15)

[3] Tanenbaum, A. *Structured Computer Organization*, Prentice-Hall International 1999, ISBN 0-13-020435-8

[4] Hewlett-Packard m.fl., *Infrared Data Association Serial Infrared Physical Layer Specification version 1.4*, 2001. http://www.irda.org/standards/pubs/IrPHY_1p4.pdf, (2002-12-15)

[5] Peterson L och Davie B, *Computer Networks – a systems approach, second edition*, Morgan Kaufmann Publishers, 2000, ISBN 1-55860-577-0

[6] Williams T, m.fl. *Infrared* Data *Association Serial Infrared Link Access Protocol (IrLAP)*, 1996 http://www.irda.org/standards/pubs/IrData.zip, (2002-12-15)

[7] Seaborne A, m.fl. *Infrared Data Association Link Management Protocol*,1996 http://www.irda.org/standards/pubs/IrData.zip, (2002-12-15)

[8] Williams S, m.fl. *Infrared Data Association 'Tiny TP': A Flow-Control Mechanism for use with IrLMP*, 1996 http://www.irda.org/standards/pubs/Tinytp11.PDF, (2002-12-15)

[9] Megowan P, m.fl. *IrDA Object Exchange Protocol - OBEX,* 1999 http://www.irda.org/standards/pubs/OBEX1p2_Plus.zip, (2002-12-15)

[10] Suvak D, m.fl. *Infrared Data Association 'IrCOMM': Serial and Parallel Port Emulation over IR (Wire Replacement)*, 1995 http://www.irda.org/standards/pubs/ircomm10.pdf, (2002-12-15)

[11] Axtman D, m.fl. *Infrared Data Association LAN Access Extensions for Link Management Protocol, IrLAN*, 1997 http://www.irda.org/standards/pubs/IrLAN.PDF, (2002-12-15)

[12] Chock R, m.fl. *IrDA Control Specification (Formerly IrBus) IrDA CIR (Control IR) Standard,* 1998 http://www.irda.org/standards/pubs/Irda_ControlV1p0E.zip, (2002-12-15)

[13] Suvak D, Wang L. *Infrared Data Association Minimal IrDA Protocol Implementation (IrDA Lite),* 1996 http://www.irda.org/standards/pubs/litever10.pdf, (2002-12-15)

[14] Temic, Telefunken, *TOIM3000/3232 - Infrared IrDA Integrated Interface Circuits*, 1996. http://www.elementy.pl/katalog/element.pdf/908992.pdf, (2002-12-15)

[15] Microchip, *MCP2120 – Infrared Encoder/Decoder*, DS21618A, 2001. http://www.microchip.com/download/lit/pline/analog/interfce/infrared/21618a.pdf, (2002-12-15)

[16] Vishay, *2.7 V to 5.5 V Serial Infrared Transceiver Module Family (SIR, 115.2 kbit/s),* Dokumentnummer 82514, 2001. http://www.vishay.com/document/82514/82514.pdf, (2002-12-15)

[17] Microchip, *PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers,* DS30292C, 2001 http://www.microchip.com/download/lit/pline/picmicro/families/16f6xx/40242a.pdf , (2002-12-15)
[18] Thomas Lindborg, *Produktbeskrivning aktivt tilluftsdon – IDCC*, 2002, Dokument IDCC08. http://www.lindinvent.se/PRODUKTER-VAV/IDCC08.pdf (2002-12-15)
[19] Palm hemsida, *Products, Services and Company information*, 2002 http://www.palm.com, (2002-12-15)
[20] Kadak hemsida, *Kadak AMX RTOS*, 2002, http://www.kadak.com, (2002-12-15)
[21] Kernigan B och Ritchie D. *The C Programming Language*, Prentice-Hall International, 1989, ISBN 91-970294-45
[22] Stroustrup B. *The C++ Programming Language –Third edition*, Adison Wesley, 1997, ISBN 0-201-8894-4
[23] PalmSource, *Palm OS Programmer's API Reference*, 2002 http://www.Palm OS.com/dev/support/docs/Palm OS/ReferenceTOC.html, (2002-12-15)
[24] Metrowerk hemsida, *Metrowerks Home*, http://www.metrowerks.com, (2002-12-15)
[25] Delorie hemsida, *DJGPP*, 2002, http://www.delorie.com/djgpp, (2002-12-15)

10.1 Figurer

[F1] Vishay TFDS4500, hämtad från [16]
[F2] IDCC fristående, hämtad från http://www.lindinvent.se (2002-12-15)
[F3] IDCC takmonterad, hämtad från http://www.lindinvent.se (2002-12-15)

Appendix 1 - Palm OS events

Här presenteras ett par av de I Palm OS vanligaste händelserna.

Händelse	**Beskrivning/Förekomst**
appStopEvent	Programmet avslutas. Denna händelse genereras av systemet då användaren stänger av handdatorn eller väljer att byta till en annan applikation. Vanligtvis anropas funktionen `AppStopEvent()` för att avallokera eventuella dynamiska variabler vid denna händelse.
ctlEnterEvent	Förekommer då ett penDownEvent genereras inuti en kontroll.
ctlSelectEvent	Förekommer då ett penUpEvent genereras inuti en kontroll. Exempelvis att användaren har tryckt på en knapp.
ctlRepeatEvent	Liknande ctlSelectEvent, men för en repeterande knapp istället.
frmOpenEvent	Ett formulär har öppnats. Tillsammans med denna händelse är det lämpligt att göra initierande kod för detta formulär, t.ex. initiera listor och fält. Denna händelse är ej systemgenererad utan lämnas till systemet efter det att ett program anropar t.ex. `FrmGotoForm()`.
frmLoadEvent	Ett formulär har laddats. Även detta är en händelse genererad av funktionsanrop i programmet.
frmCloseEvent	Ett formulär har stängts. Om dynamisk kod har allokerats t.ex. i samband med frmOpenEvent är det lämpligt att avallokera det vid denna händelse.
keyDownEvent	Förekommer då användaren trycker ner någon av de sex tangenterna, t.ex. scrollup eller scrolldown.
penDownEvent	Pennan trycks ner någonstans på skärmen. Denna händelse tillsammans med penUpEvent är nödvändiga för att ett flertal andra händelser ska kunna genereras, t.ex. ctlSelectEvent och lstSelectEvent.
penMoveEvent	Pennan har flyttats sedan föregående penDownEvent.
penUpEvent	Pennan lyfts från skärmen.
popSelectEvent	Användaren har valt ett alternativ från en lista.
sclEnterEvent	Förekommer då ett penDownEvent genereras i en rullningslist.
sclExitEvent	Förekommer då ett penUpEvent genereras i en rullningslist.
fldEnterEvent	Förekommer då ett penDownEvent genereras i ett textfält.
fldSelectEvent	Förekommer då ett penUpEvent genereras i ett textfält.
frmUpdateEvent	Genereras då programmet flyttar tillbaka från ett formulär till ett tidigare laddat för att uppdatera delar av skärmen.
lstSelectEvent	penUpEvent i en lista. Denna händelse markerar att användaren valt ett alternativ från listan.
lstEnterEvent	penDownEvent I en lista.
lstExitEvent	Genereras då pennan lyfts utanför en lista.
menuOpenEvent	Förekommer då en ny meny har initialiserats.
fldChangeEvent	Förekommer då ett teckenfält har ändrat innehåll.

Appendix 2 – Palm OS Managers

Här presenteras ett par av de vanligt förekommande Palm Managers.

Manager	Beskrivning/Innehåll	Exempelfunktioner
Alarm Manager	Hanterar alarm. Funktioner för att bl.a. ställa alarm.	`AlmGetAlarm()` `AlmSetAlarm()`

Data & Resource Manager	Funktioner för att hantera Palm databaser, det system som används för längre lagring. Exempelvis kan nämnas att här finns funktioner för att skapa, förstöra, skriva, läsa, sortera osv.	`DmCreateDatabase()`
		`DmDeleteRecord()`
		`DmOpenDatabase()`

Error Manager	Innehåller de reserverade orden i samband med try-catch block samt funktioner för att kasta undantag. Funktionerna här används för att skapa program som är mer toleranta mot fel.	`ErrCatcth`
		`ErrTry`
		`ErrThrow()`

Feature Manager	Innehåller funktioner för att undersöka egenskaper hos handdatorn, t.ex. vilken version av operativsystem som körs eller om det finns extra enheter kopplade till handdatorn, som exempelvis ett extra minneskort.	`FtrGet()` `FtrSet()`

File Streaming Manager	Sköter allt som har med filer att göra, funktionerna här har ofta en direkt motsvarighet till filfunktionerna i C, t.ex `fopen(), fread(), fwrite()` och `fclose()`	`FileClose()`
		`FileDelete()`
		`FileOpen()`
		`FileWrite()`

Float Manager	Funktioner för emulering av flyttal. Processorn i de flesta Palm handdatorer har ingen flyttalsdel.	`FlpGetExponent()`
		`FlpGetSign()`
		`FlpSetPositive()`

Graffiti Manager	Innehåller bildbehandlingsfunktioner för att tolka indata via pennan.	`GrfFilterPoints()`
		`GrfGetGlyphMapping()`
		`GrfMatch()`

Memory Manager	Hanterar åtkomst av det interna minnet. Funktionerna här används för att dynamiskt skapa minne, flytta minne, ändra storlek på en allokering, frigöra minne mm.	`MemMove()`
		`MemPtrFree()`
		`MemPtrNew()`
		`MemPtrResize()`

Sound Manager	Innehåller funktioner för att hantera den inbyggda högtalaren i Palm handdatorer, vilken kan spela enkla ljud.	`SndGetDefaultVolume()` `SndPlaySystemSound()`

String Manager	Kan jämföras med string.h i C vilken definierar funktionerna `strcpy(), strcat(), strcmp()` mm.	`StrCat()`
		`StrCopy()`
		`StrCompare()`

		`StrPrintF()`

System Event Manager	Innehåller funktioner för att komma åt systemets meddelandekö och på så sätt undersöka vilka händelser som finns lagrade där.	`EvtGetEvent()`
		`EvtGetPen()`
		`EvtSetNullEventTick()`
		`EvtWakeup()`

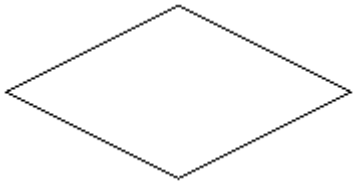
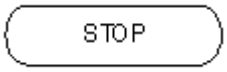
System Manager	Systemspecifika funktioner. T.ex. för att byta applikation, mjukvaru-reset, hitta bibliotek eller hur mycket batteritid som återstår.	`SysBatteryInfo()`
		`SysLibFind()`
		`SysUIAppSwitch()`

Text Manager	Mer komplexa stränghanteringsfunktioner, t.ex. att byta ut enstaka bokstäver i en sträng.	`TxtCharIsAlpha()`
		`TxtFindString()`
		`TxtReplaceStr()`

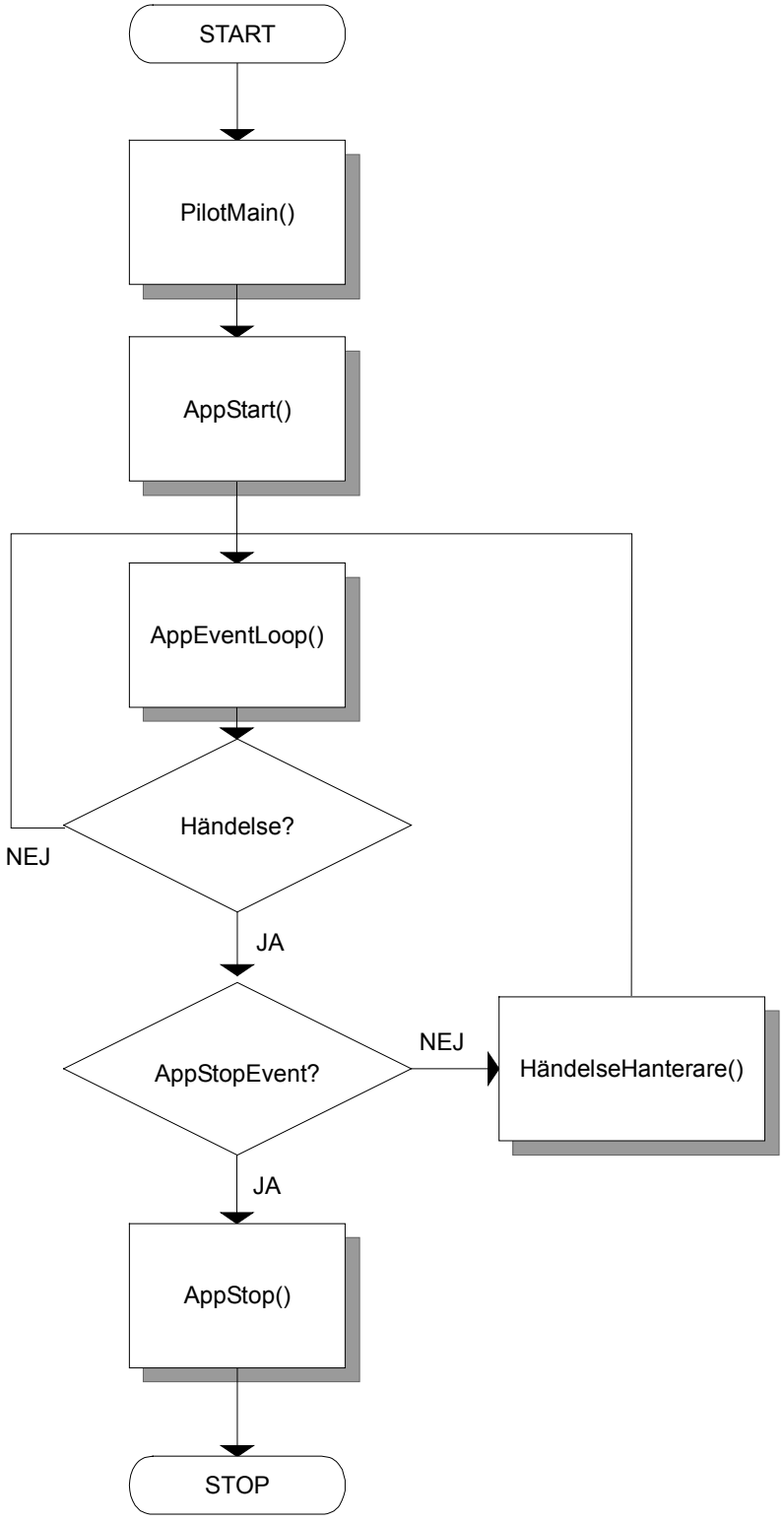
Time Manager	Funktioner för att komma åt datum och tid	`DateAdjust()`
		`DayOfWeek()`
		`TimGetSeconds()`
		`TimSetSeconds()`

Appendix 3 – Flödesscheman

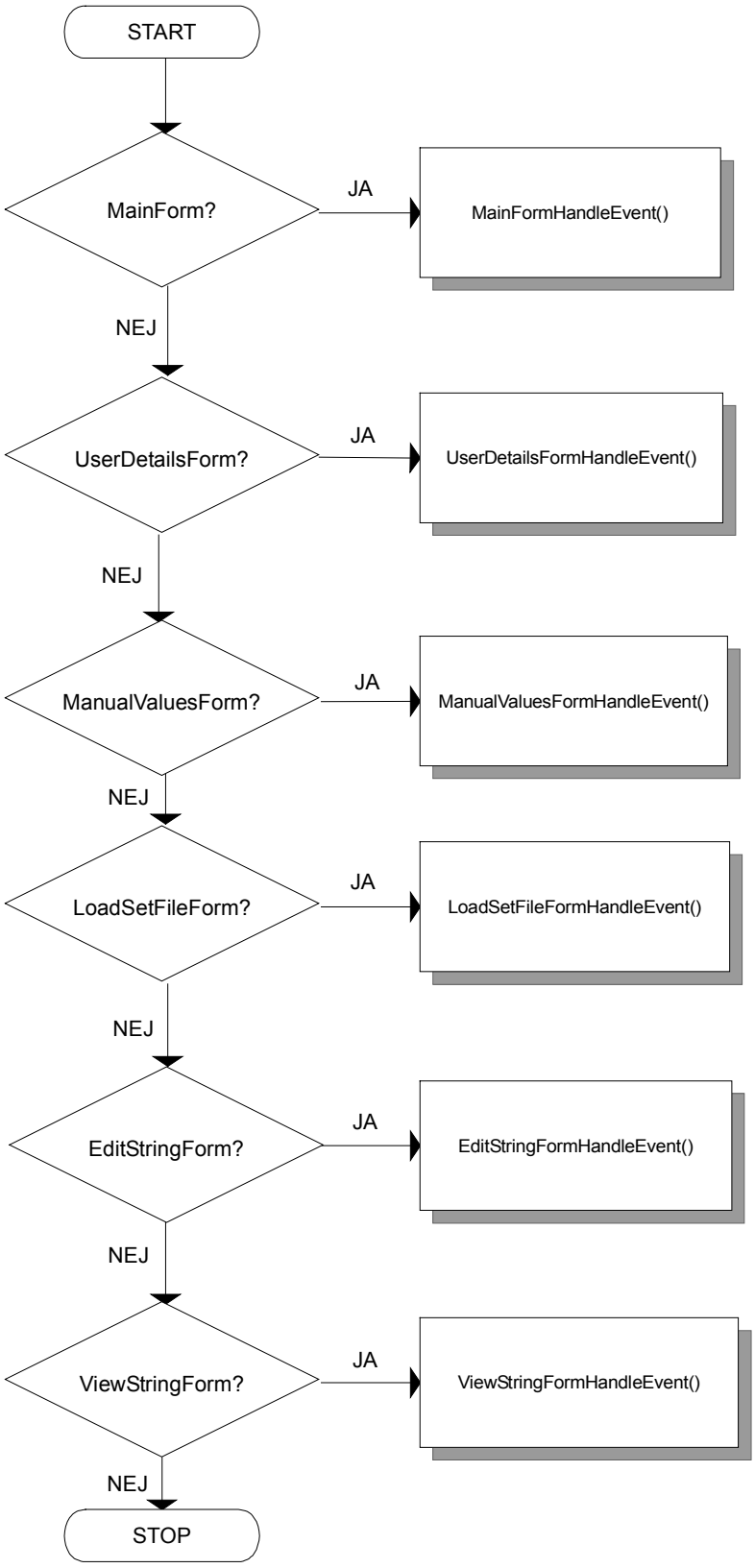
Följande flödesscheman bygger på symbolerna definierade i tabell A3.1 nedan.

Symbol	Förklaring
	Startpunkt för flödesschemat.
	Val, svarar mot en if-sats i programmeringssammanhang.
	Operation, del av intressanta instruktioner som utförs utan att anropa en subrutin. Förklaras med psuedokod, t.ex. ”Traversera listan och initiera A och B”.
	Subrutin, exempelvis LoadDP().
	Slutpunkt för flödesschemat.

Tabell A3.1 – Symboler vilka används i följande flödesschema.

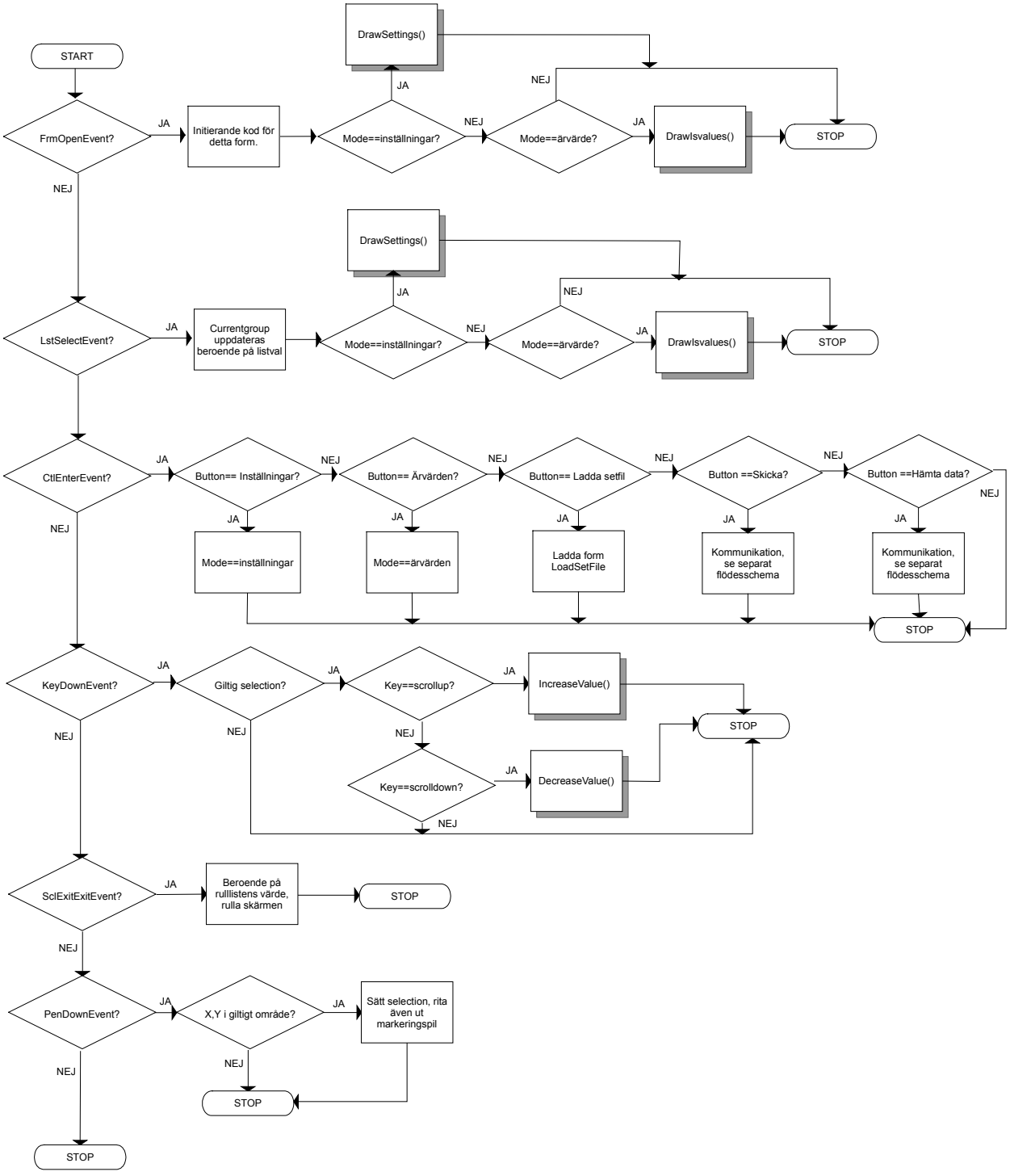


Figur A3.1 – Generell struktur hos en Palmapplikation



Figur A3.2 – Flödesschema för AppHandleEvent()

Figur A3.3 – Flödesschema för operationerna skicka och hämta data



Figur A3.4 – Flödesschema för händelsehanteraren till MainForm

Apendix 4 – Syntax för enhetsprofiler

Ny grupp

Syntax: G,namn,id,securitylevel

namn – sträng.
id - tal.
securitylevel - tal, 0-5.

Beskrivning: Ny grupp men namnet ”namn” och ID-nummer id. Securitylevel anger om gruppen är dold för andra utom användare med rätt säkerhetskod. 0 är lägsta säkerhetsnivå och 4 högsta. Securitylevel 5 motsvarar en grupp som aldrig visas. Anledningen till att man måste ange ett id är för att länka ihop ärvärden och inställningar med en speciell grupp.

Exempel: `G,Temperatur,1,0,`

Nytt ärvärde

Syntax: A,namn,gruppid,typ,enhet, securitylevel

namn - sträng.
gruppid - tal.
typ - tal.
enhet - sträng.
securitylevel - tal, 0-5 (se grupper för mer information).

Beskrivning: Skapar ett nytt ärvärde av angiven typ (se tabell 6.2). GruppID kopplar detta ärvärde till en tidigare skapad grupp. Enhet är den sträng som kommer stå efter ärvärdet i fråga, för närvarande finns ingen lösning på tecknet ’°’. Dessutom kan nämnas att enheten spelar ingen roll om typen är 0, d.v.s. checkbox. Precis som med grupper kan man även här ange om ett ärvärde ska ha en säkerhetsnivå.

Exempel: `A,kanaltemp,1,1,grader,0,`

Ny inställning

Syntax: I,namn,gruppid,typ,min,max,securitylevel,enhet,nolistel,listelement1, listelement2, listelement3,listelement4,listelement5

namn – sträng.
gruppid - tal.
typ - tal.
min - tal.
max - tal.

securitylevel - tal 0-5 (se grupper för mer information).
enhet - sträng (se ärvärden för mer information).
nolistel - antal listelement, således också antal enhetsprofiler.
listelement1 - sträng.
listelement2 - sträng.
listelement3 - sträng.
listelement4 - sträng.
listelement5 - sträng.

Beskrivning: Skapar en ny inställning av angiven typ. Precis som med ärvärden kopplar gruppid samman denna inställning med en grupp. Nolistel anger hur många listelement som ska visas i en dropdown-lista om typ är 3 eller 4. Max och min anger ramen för hur stor en inställningen kan bli. Försöker man öka eller minska utöver dessa gränser kommer ett felmeddelande att visas. Precis som med ärvärden kan man även här ange om en inställning ska ha en säkerhetsnivå.

Exempel: `I,maxflöde,2,2,0,50,0,l/s,0,`

Nytt manuellt värde

Syntax: M,namn,typ,min,max,enhet

namn - sträng
typ - tal
min - tal
max - tal
enhet - sträng

Beskrivning: Skapar ett nytt manuellt värde av angiven typ. Min och max anger övre och undre gränser då typ är 2. Enhet är det som kommer stå efter det manuella värdet för typ 2.

Exempel: `M,Uppmätt maxflöde,2,10,30,l/s,`