Avdelningen för Datavetenskap

Andreas Dahlberg och Martin Bengtsson

Utveckling av drivrutin till Intermec Easycoder 501 XP

Development of a driver for Intermec Easycoder 501 XP

Examensarbete 15 hp

Datavetenskap

Datum:	2010-06-09
Handledare:	Katarina Asplund
Examinator:	Martin Blom
Ev. löpnummer:	C2010:05

Karlstads universitet 651 88 Karlstad

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

Information@kau.se www.kau.se

Utveckling av drivrutin till Intermec Easycoder 501 XP

Andreas Dahlberg, Martin Bengtsson

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

Andreas Dahlberg

Martin Bengtsson

Godkänd, 2010-06-09

Handledare: Katarina Asplund

Examinator: Martin Blom

Sammanfattning

Detta examensarbete har utförts på uppdrag av företaget Prevas AB i Karlstad. Målet med arbetet var skapa en drivrutin till etikettskrivaren Intermec Easycoder 501 XP. Denna drivrutin skulle motsvara en drivrutin som Prevas AB hade utvecklat för olika etikettskrivare av märket Zebra. Detta arbete skulle då tillåta Prevas AB att erbjuda kunder med Intermec samma funktionalitet som de kunder som använder Zebra.

Den funktionalitet som drivrutinen till Intermec skulle ha var utskrift av textfält och de två typerna av streckkoder som används i Zebradrivrutinen. Drivrutinen behövde även kunna placera dessa objekt på rätt position på etiketten.

Drivrutinen som har utvecklats till Intermecen fungerar och har det mesta av funktionaliteten som Zebradrivrutinen har, med vissa mindre begränsningar.

Development of a driver for Intermec Easycoder 501 XP

Abstract

The work in this thesis has been done on behalf of Prevas AB in Karlstad. The goal with the project was to create a driver for the label printer Intermec Easycoder 501 XP. This driver should have the same functionality as a driver that Prevas AB had developed for different label printers of the brand Zebra. The driver would then allow Prevas AB to offer customers that use Intermec the same functionality as the customers that use Zebra.

The functionality that the driver for Intermec should have was printing of text fields and the two types of barcodes that the Zebradriver use. The Intermecdriver also needed to place the text fields and barcodes in the right position on the label.

The driver that has been developed for the Intermec works and has most of the functionality that the Zebradriver offers, with a few limitations.

Innehållsförteckning

Figurförteckning

1 Introduktion

Detta projekt har utförts hos Prevas, som är ett av de ledande företagen i Norden inom inbyggda system och industriell IT. Bakgrunden till arbetet var att Prevas hade utvecklat en drivrutin för att skriva ut etiketter med Zebraskrivare som hanterar streckkoder, textfält samt serialisering[1] av dessa. Prevas vill nu också ha en drivrutin med samma funktionalitet till etikettskrivare av märket Intermec. Prevas tyckte att uppgiften var intressant för att den skulle ge företaget ökad kunskap om utrustningen som deras produkter arbetar mot.

1.1 Syfte

Målet var att skapa en skrivardrivrutin för Intermec Easycoder 501 XP som gör samma sak som drivrutinen för Zebraskrivarna, det vill säga hantering av streckkoder, textfält samt serialisering av dessa. Eftersom Zebra och Intermec använder olika språk för att skriva ut så innebar det mycket översättande av Zebraskrivarnas inbyggda språk.

1.2 Resultat

Vårt mest betydelsefulla resultat är att vi lyckats skapa en fungerande drivrutin till Intermec Easycoder 501 XP med vissa midre begränsningar. Drivrutinen har funktioner för att skriva ut fälten där de ska skrivas ut och textfält samt de två typerna av streckkoder skrivs ut. Ett annat viktigt resultat är att vi har lyckats med att implementera serialisering av textfält och streckkoder.

1.3 Avgränsningar

Utvecklingsarbetet har endast inriktat sig mot etikettskrivaren Intermec Easycoder 501 XP. Drivrutinen klarar bara av att hantera textfält samt streckkodstyperna Code 39 och Code 128. Code 128 är därermot begränsad jämfört med Zebradrivrutinen därför att Intermecdrivrutinen inte klarar av Code 128:s delmängder. En annan avgränsning under projektets gång är att vi använder endast standardtypsnittet.

[1] Serialisering är när man ökar värdet på streckkoden mellan utskrifter.

1.4 Disposition av uppsatsen

I kapitel 2 så går vi igenom generellt om etiketter, Code 39 streckkoder, Code 128 streckkoder, serialisering, Intermec Easycoder 501 XP, Fingerprint, Zebra 90XiII och ZPL II. Nästa kapitel beskriver de olika delarna i systemet runt Zebradrivrutinen och hur systemet arbetar. Sedan så beskrivs lite kort de verktyg vi har använt under arbetets gång, hur vi har implementerat drivrutinen, problem som vi har löst, problem som vi inte har löst och möjliga vidareutvecklingar hos Intermecdrivrutinen i det fjärde kapitlet. Resultaten från vår implementation behandlas i kapitel 5 tillsammans med våra slutsatser.

2 Bakgrund

I detta kapitel introduceras till de två skrivarna som projektet är inriktat på samt programspråken som används för att instruera dem om hur etiketterna ska formateras och skrivas ut. Avsnitt 2.1 beskriver översiktligt vad etiketter är och går igenom serialisering. I nästa avsnitt beskrivs Code 128-streckkoden och avsnitt 2.3 beskriver Code 39-streckkoden. Efter det beskrivs Intermec Easycoder 501 XP och dess funktioner. Det femte avsnittet beskriver det BASIC-inspirerade språket Fingerprint och nästa beskriver Zebra 90XiII och dess funktioner. Det sista avsnittet beskriver programspråket ZPL II.

2.1 Etiketter och serialisering

Etiketter är självhäftande papperslappar med information. Informationen kan bestå av bilder, textfält och streckkoder. Drivrutinen som har utvecklats hanterar enbart textfält och streckkoder. Streckkoderna är av typen Code 128(se avsnitt 2.2) eller Code 39(se avsnitt 2.3). Informationen i textfälten och streckkoderna kan antingen vara statisk, det vill säga att den inte ändras, eller så kan den bestå av ett serienummer som skall ändras efter att ett angivet antal av etiketter med samma nummer har skrivits ut (Serialisering).

Ex: Ett textfält med serialiseringsdata initieras till värdet ”123”. Antalet kopior som skall skrivas ut innan serienumret ändras sätts till två och ökningen sätts till tre. Utskrift av exempelvis fem etiketter kommer att ge följande värden på textfältet: ”123”, ”123”, ”126”, ”126” och ”129”.

2.2 Code 128-streckkod

Code 128 [7] är en av de mest använda symbolkodningar [9] för streckkoder som finns idag. Den klarar av att koda alla de 128 tecken som ingår i ASCII-teckenkodningen [10].

En Code 128-streckkod består av följande 6 delar (se figur 2.1):

1. Tyst zon - blankt utrymme före första tecknet

2. Starttecken - ett tecken som anger vilken symbolkodning och teckengrupp som används
3. Data - de värden som streckkoden ska representera
4. Kontrolltecken - kontrollvärde beräknat på streckkodens startkod och data
5. Stopptecken - tecken som anger att detta är slutet på streckkoden
6. Tyst zon - blankt utrymme efter sista tecknet

Figur 2.1: Exempel på Code 128-streckkod

I Code 128 består varje tecken av tre mörka och tre ljusa streck, förutom stoppteckenet som har fyra mörka och tre ljusa streck. Varje tecken startar med ett mörkt streck och avslutas med ett ljust streck (med undantag för stoppteckenet som både avslutas och inleds med ett mörkt streck). Streckens bredd kan vara ett, två, tre eller fyra enheter. De sex streck som tillsammans utgör ett tecken har en sammanlagd bredd på elva enheter. De tre svarta streckens sammanlagda bredd måste vara ett jämnt antal enheter samtidigt som de vita strecken sammanlagda bredd skall vara udda.

För att kunna representera alla 128 ASCII-tecken och samtidigt få en kompakt representation av numeriska värden så har Code 128 tre olika teckensatser:

- **Teckensats 128A** - ASCII-tecken 0 till 95 (0-9, A-Z, kontrolltecken och symboler) samt specialtecken (t.ex. start- och stopptecken).
- **Teckensats 128B** - ASCII-tecken 32 till 127 (0-9, A-Z, a-z, symboler) samt specialtecken.

- **Teckensats 128C** - Numeriska värden kodas med så kallad dubbel densitet. Detta innebär att siffror kodas i par för att få en kompaktare representation av dem. Till exempel blir kodningen av 123456 bara tre tecken lång ((12),(34),(56)). Eftersom siffror kodas i par så krävs det att datan innehåller ett jämnt antal siffror för att teckensatsen 128C ska kunna användas.

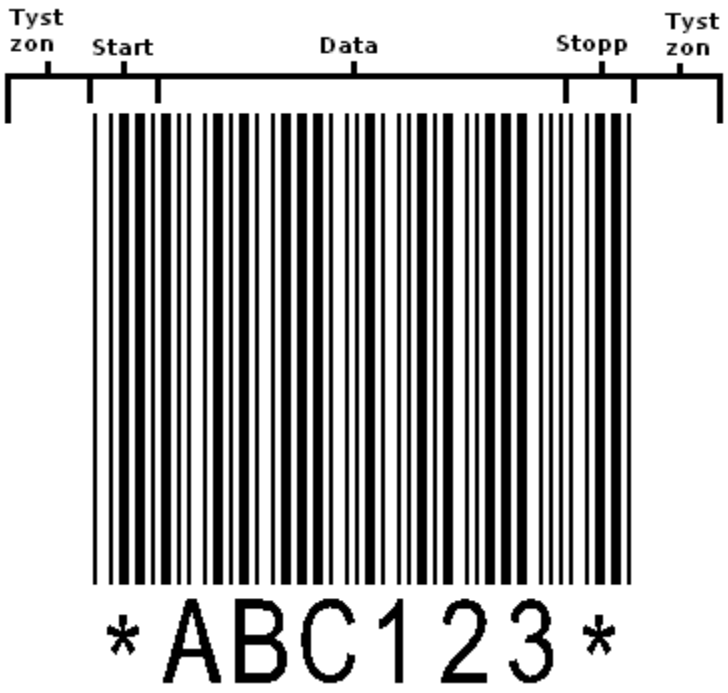
En streckkod kan innefatta kodningar från en eller flera av de olika teckensatserna. Byte mellan de olika teckensatserna utförs med hjälp av olika specialtecken.

För att kunna kontrollera att en kod har avlästs korrekt så innehåller alla Code 128-streckkoder ett kontrolltecken. Kontrolltecknets värde räknas ut genom att summera värdet av varje tecken i datafältet multiplicerat med sin position(tecknet längst till vänster har position 1). Till summan adderas också starttecknets värde. Därefter divideras summan med 103 och den rest som blir över blir kontrolltecknet.

2.3 Code 39-streckkod

Code 39 [8] är en äldre symbolkodning som fortfarande används flitigt och har stöd för kodning av 43 olika tecken. De tecken som kan kodas är A-Z, 0-9, -, ., $, /, +, % och mellanslag. Utöver dessa innehåller koduppsättningen ytterligare ett tecken som används för att beteckna start och stopp av en streckkod. Den brukar betecknas som ’*’ när den skrivs ut som text. En Code 39-streckod innehåller följande element (se figur 2.2):

1. Tyst zon - blankt utrymme före första tecknet
2. Starttecken - ett tecken som anger vilken symbolkodning som används
3. Data - de värden som streckkoden ska representera
4. Stopptecken - tecken som anger att detta är slutet på streckkoden
5. Tyst zon - blankt utrymme efter sista tecknet



Figur 2.2: Exempel på Code 39-streckkod

Streckkoderna är uppbyggda av svarta och vita streck som antingen kan vara smala eller breda. Förhållandet mellan de breda och smala streckens bredd kan väljas mellan 1:2 och 1:3. Ett kodat tecken representeras som nio streck, fem svarta och fyra vita, varav tre är breda och resten smala. Eftersom alla tecken startas och avslutas med ett svart streck så placeras ett vitt streck in som avskiljare mellan alla konstellationer av svarta och vita streck.

Code 39 innehåller inget kontrolltecken men det finnns en variant, Code 39 mod 43, som räknar ut en kontrollsiffra och lägger tilll den innan stopptecknet. Kontrolltecknets värde räknas ut genom att alla teckens värde (0-42), exklusive start- och stopptecken, adderas och därefter divideras med 43. Resten av divisionen blir kontrolltecknet. Det finns också en variant som stödjer kodning av ASCII-tabellens alla 128 tecken.

2.4 Intermec Easycoder 501 XP

Figur 2.3: Intermec Easycoder 501 XP skrivare

Intermec 501 XP [1] (se figur 2.3) är en etikettskrivare som är särskilt anpassad för att kunna skriva ut många etiketter snabbt. Den har en maximal utskriftshastighet på 300 mm/sekund och en utskriftsupplösning på 300 dpi. 501 XP stöder 15 olika typsnitt i grundutförandet. Man kan även ladda in fler typsnitt i skrivarens Flashminne eller genom att använda ett minneskort.

Skrivaren har en parallellport och en serieport. Fördelen med den parallella porten är att det kan användas direkt utan att några kommunikationsinställningar behöver göras. Den seriella porten kräver att vissa inställningar utförs innan det kan användas, men i gengäld kan man erhålla meddelande ifrån skrivaren om något fel uppstår.

2.5 Fingerprint 7.80

Fingerprint v7.80 [2] är det programmeringsspråk som används i Intermec 501 XP och även i andra Intermecskrivare. Fingerprint är ett BASIC-inspererat programmeringsspråk som gör

det möjligt att lätt skapa etikettformat och ställa in skrivaren, vilket gör att man kan skapa sina egna etiketter.

Till Fingerprint medföljer också Intermec Direct Protocol. Intermec Direct Protocol tillåter att man hämtar layoutinformation och data från värddatorn. Layoutinformationen kan då användas till att skapa etiketter med väldigt lite programmering. Intermec Direct Protocol har också felhantering samt en räknarfunktion.

Fingerprint har sju olika bildfilsöverförings-protokoll. De vanligaste är Intelhex, UBI00, UBI01, UBI02 och UBI03 [2]. Dessa protokoll kan hantera bitmap-bildformatet [11]. Det finns även ytterliggare ett protokoll, UBI10 [2], som är ett kombinerat protokoll/filformat för bildöverföring. Bitmapformat har två olika versioner som används av Fingerprint:

- Bitmap, bit representation: Bilden är kodad i 16-bitars ord. Bitarna i varje ord läses från minst signifikanta biten först(bit 0).
- Bitmap, Run Length Limited(RLL): I RLL så använder man sig av komprimering för stora bitmap-bilder.

Språket stöder olika teckensnitt samt olika tecken för olika språk. Varje Intermec-skrivare har också flera olika streckkodsgeneratorer.

2.5.1 Fingerprintinstruktioner

Instruktionerna i Fingerprint inleds med namnet på programinstruktionen som skall utföras och följs eventuellt av en kommaseparerad lista med parametrar.

Exempel på instruktioner:

- **FONT** - Används för att välja typsnitt, typsnittstorlek och eventuell lutning på bokstäverna.
- **PRTXT** - Anger den text som skall skrivas ut i ett textfält.
- **PRPOS** - Anger koordinaterna för startpositionen för till exempel en textrad, streckkod eller en bild.
- **BARTYPE** - Anger vilken typ av streckkod som skall användas.
- **BARHEIGHT** - Anger höjden på strecken i en streckkod i punkter.

- **PRBAR** - Anger indata till en streckkod.
- **REBOOT** - Startar om skrivaren.

Nedan följer ett exempel på Fingerprintkod som skriver ut en etikett med en kvadrat på:

1. **NEW**
2. **100 PRPOS 100,100**
3. **200 PRBOX 200,200,5**
4. **300 PRINTFEED**
5. **400 END**
6. **RUN**

1. Instruktionen NEW tömmer skrivarens arbetsminne så att ett nytt program kan skapas.
2. Det första talet är radnumrering. PRPOS sätter startpunkten för det som ska skrivas på etiketten till 100 punkter från origo i både x-led och y-led.
3. PRBOX skapar en låda som sätter höjden till 200 punkter, bredden till 200 punkter och tjockleken på linjerna till 5 punkter.
4. PRINTFEED gör så att den specificerade etiketten skrivs ut.
5. END avslutar programmet och stänger ned alla filer och enheter.
6. Instuktonen RUN gör så att programmet körs.

2.5.2 Intermec Direct Protocol

Intermec Direct Protocol [4] är en del av Fingerprint. Protokollet kan användas på två sätt:

- För att skapa en etiketts layout bestående av fält med både bestämd och föränderlig information. Värddatorn väljer sedan en layout och ger den föränderlig indata som en enkel sträng.
- För att ta emot indata och formateringsinstruktioner som en kontinuerlig sträng av data direkt från värddatorn.

Intermec Direct Protocol aktiveras och avaktiveras med instruktionen INPUT ON respektive INPUT OFF. Det finns även instruktioner som bara kan användas i Intermec Direct Protocol:

- **COUNT&** - används för att skapa en räknarvariabel.
- **FORMAT INPUT** - instruktion som anger separatorerna för LAYOUT RUN instruktionen.

- **LAYOUT END -** instruktion som stoppar skapandet av en layout och sparar den.
- **LAYOUT INPUT -** instruktion som startar skapandet av en layout-beskrivning.
- **LAYOUT RUN -** instruktion som kör en layout-beskrivning som redan finns. Om en PF kommer efter LAYOUT RUN så skrivs upplägget av etiketten ut.
- **ERROR** - med denna instruktion så kan man definera felmeddelande och startar felhanteringen.

2.6 Zebra 90XiII

Figur 2.4: Zebra 90XiII skrivare

Zebra 90XiII [6] (se figur 2.4) är en etikettskrivare. När den är kopplad till en värddator fungerar den som ett komplett system för att skriva ut etiketter, biljetter med mera. För att skapa en etikett måste man antingen definiera utseendet i ZPL II, eller så måste man använda ett mjukvaruprogram.

Zebra 90XiII stöder tio olika inbyggda typsnitt. Man kan även ladda in fler typsnitt genom att installera ett extra EPROM [12] med typsnitt. När det är installerat så kan man använda de typsnitt som finns på eprommet tillsammans med skrivarens inbyggda typsnitt. Man kan även använda PCMCIA-typsnitt kort [13] för att få tillgång till extra typsnitt.

Zebra 90XiII har en seriell port och en parallell port. När man använder den seriella porten så kan man göra olika inställningar. När den parallella porten är korrekt inställd så fungerar inte den seriella porten. Den parallella porten erhåller data från värddatorn, men kan inte skicka tillbaka skrivarens statusinformation.

2.7 ZPL II

ZPL II [3] står för Zebra Programming Language II. Detta är ett högnivåspråk för etikettdefinition och skrivarkontroll. Originaldrivrutinen genererar ZPL II-kod som sparas i en textfil. Språket kan programmeras i skrivbara ASCII-tecken [10], vilket medför att det är lätt att skicka data och format genom nätverk.

ZPL II innehåller instruktioner för en mängd olika typsnitt och streckkoder. Det finns instruktioner för hur skrivfält positioneras på etiketten och huruvida etikettens innehåll ska vara horisontell eller vara roterad 90, 180 eller 270 grader. Bilder kan användas så länge de är binära eller i hexadecimalt format.

Några av de funktioner som ZPL II har är följande:

- Serialiserade etikettfält, där användaren väljer startvärde och ökning eller minskning av detta värde.
- Skalbara typsnitt.
- Programmerbar etikettsreplikerarräknare, kvantitetskontroll och skrivarpauser som medför att man kan dela in etiketter i användbara grupper.
- Enkel linjegrafik för att förhindra etikettförskrivning.
- Bitmap-bildgrafik med biblioteksfunktion (för att ha mer än en bild i minnet som kan användas).

2.7.1 ZPL II-instruktioner

Instruktioner i ZPL II består av ett prefixtecken följt av två bokstäver. Efter bokstavskombinationen kan det också följa olika parametrar beroende på om det behövs eller inte.

Det finns två olika sorters instruktioner i ZPL II: formatinstruktioner och kontrollinstruktioner. Formatinstruktioner används för att definera etikettens utseende. Bland annat så bestämmer de etikettens längd, typ av fält, fältstart och fältdata. Formatinstruktioner använder prefixet ^ före koden för en instruktion.

Exempel på formatinstruktioner:

- **^XA** - Markerar början på ett etikettformat.
- **^XZ** - Markerar slutet på ett etikettformat.
- **^FO** - Sätter koordinaterna för ett fälts utgångspunkt relativt etikettens startposition som angetts med ^LH-instruktionen. Instruktionen anges tillsammans med de önskade koordinaterna. Ex: ^FO10,20 sätter positionen för ett fälts övre vänsterhörn till x-koordinat 10 och y-koordinat 20.
- **^FD** - Definierar fältets dataträng som skall skrivas ut på etiketten. Strängen kan innehålla alla skrivbara tecken förutom de två som används som instruktionsprefix, alltså ^ och ~.

Kontrollinstruktioner får skrivaren att omedelbart utföra en handling, till exempel tömma skrivarens minne. De kan avbryta och exekveras före formatinstruktioner i skrivarens buffer. Kontrollinstruktioner använder prefixet ~ före koden för en instruktion.

Exempel på kontrollinstruktioner:

- **~JA** - Raderar alla formatinstruktioner som finns i skrivarens buffert och avbryter utskrift.
- **~PA** - Matar ut en tom etikett.
- **~PP** - Försätter skrivaren i pausläge. Om utskrift pågår sker det efter den aktuella etiketten är färdig.

ZPL II är en vidareutveckling av Standard ZPL. Huvudskälet till utvecklingen av ZPL II var att man önskade minska tiden mellan när en skrivare får etikettformat-data och när den första etiketten skrivs ut. Detta lyckades man med genom att ändra sättet på hur ZPL-skript skrivs.

Det finns två stora skillnader mellan ZPL II och Standard ZPL:

I Standard ZPL så väntar man på ^XZ (End Format)-instruktionen innan man bearbetar datafälten medan man i ZPL II bearbetar datafälten när de tas emot. Denna ändring gjordes för att det skulle ta mindre tid för utskriften att utföras. Det lades också till många nya instruktioner i ZPL II.

För att kunna använda ZPL II fullt ut måste en del kommandon komma före den första ^FD (Fältdata) instruktionen. Dessa instruktioner är ^JM, ^LH, ^LL, ^LR, ^LS, ^PM, ^PO, ^PR och ^PF. Detta gäller förstås bara om man använder dessa instruktioner.

- **^JM -** Ändrar antalet punkter per millimeter. Normalt antal punkter är 12 punkter/mm.
- **^LH -** Bestämmer startpunkten på etiketten.
- **^LL -** Bestämmer längden på etiketten. Används bara när det man skriver etiketten till inte redan är uppdelat.
- **^LR -** Inverterar alla färger, det som är svart blir vitt och det som är vitt blir svart.
- **^LS -** Kommando som gör att allt justeras till vänster.
- **^PM -** Skriver ut hela etiketten spegelvänd.
- **^PO -** Skriver ut etiketten upp och ned.
- **^PR -** Bestämmer utskrivningshastigheten.
- **^PF -** Gör så att skrivaren snabbt går vidare om botten på etiketten är tom.

Nedan följer ett enkelt exempel på ZPL II kod.

1: ^XA

2: ^LH30,30

3: ^FO20,10^AFN,56,30^FR^FDZEBRA^FS

4: ^FO20,80^B3N,Y,20,N,N^FDAAA001^FS

5: ^ISR:EXERPROG.GRF,N

6: ^XZ

7: ^XA^ILR:EXERPROG.GRF,N^XZ

1. ^XA startar etikettformatet.
2. Hempositionen sätts till 30 punkter till höger och 30 punkter ner från den övre kanten av etiketten.
3. -^FO sätter startpositionen för fältet relativt hempositionen.

   -^AF väljer teckensnitt F och sätter teckenstorleken till 56 punkters höjd och 30 punkters bredd.

   -^FD startar ett fält med data.

   -ZEBRA är datan i fältet.

   -^FS avslutar fält med data.
4. -^FO sätter startpositionen för fältet relativt hempositionen.

   -^B3N,Y,20,N,N väljer Code 39 streckkod.

   -^FD startar ett fält med data.

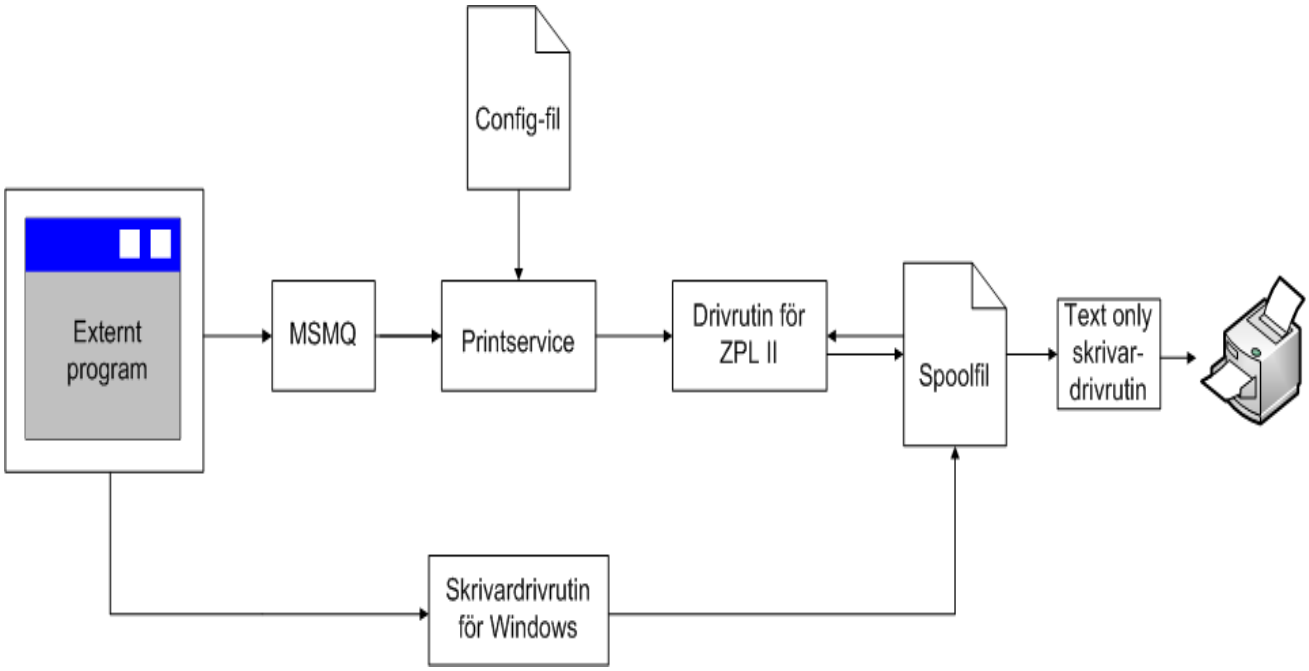
   -AAA001 är datan i fältet vilken tar formen av en viss streckkod.

   -FS avslutar fält med data.
5. ^IS sparar formatet som en grafisk bild som heter EXERPROG.GRF, men skriver inte ut efter att ha sparat.
6. ^XZ indikerar att det här är slutet på etikettformatet.
7. ^XA startar etikettformatet. ^ILR:EXERPROG.GRF,N laddar och skriver ut den grafiska bilden sparad som EXERPROG.GRF. ^XZ avslutar etikettformatet.

3 Existerande system för Zebra-skrivare

I detta kapitel ges en översikt av hur det nuvarande systemet med Zebradrivrutinen fungerar och är uppbyggt. Alla olika delar beskrivs i avsnitt 3.1. Delarna som beskivs är externt program, MSMQ, Windows skrivardrivrutin för Zebra, Printservice, printService.config, Zebradrivrutinen, Spoolfil och Text only-skrivardrivrutinen. I avsnitt 3.2 så går vi igenom hur Zebrasystemet fungerar från början till slut.

3.1 Systembeskrivning



Figur 3.1: Systembild av Zebradrivrutinen

Figur 3.1 visar de olika delarna i systemet vilka beskrivs nedan.

3.1.1 Externt program

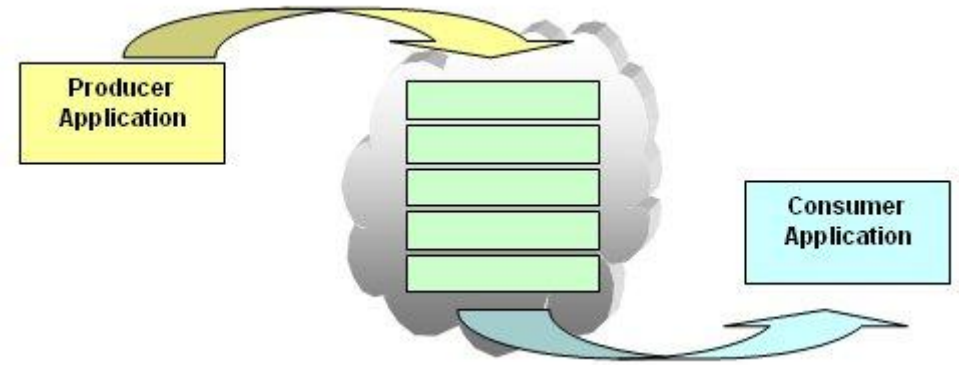
Detta är ett program som skapar ett MSMQ-meddelande (se avsnitt 3.1.3) och skickar det till en kö. Det skickar också iväg data till Windows skrivardrivrutin för Zebra.

3.1.2 Windows skrivardrivrutin för Zebra

Windows skrivardrivrutin skapar ZPL II-kod och skriver den till spoolfilen. Denna drivrutin hanterar allt det statiska på etiketterna, sådant som med andra ord inte ändras mellan utskrifter.

3.1.3 MSMQ

MSMQ [5] (Microsoft Message Queuing) är ett meddelandeprotokoll som används för att skicka meddelande mellan applikationer på ett säkert sätt. Mottagna meddelande sparas och skickas vidare tills de nått sin destination. Det betyder att meddelande kommer fram även om dess destination inte är tillgänglig vid tidpunkten då meddelandet först skickas (se figur 3.2). Stöd för MSMQ-protokollet finns i alla senare Windows-versioner men är inte installerat som standard. Meddelandet som skickas innehåller XML-formaterad information om de text- och streckkodsfält som skall skrivas ut samt information om den skrivare som skall användas. Meddelandet skickas till en kö som sedan vidarebefodrar det till applikationen Printservice.



Figur 3.2: Hur MSMQ´s kösystem fungerar

MSMQ-meddelandena är skrivna i XML. Strukturen på MSMQ-meddelanden ser ut på följande sätt. Först så har man en del som heter HEADERINFORMATION. Den innehåller bland annat:

- Namnet på spoolfilen (se avsnitt 3.1.7) som ska användas.
- Användarnamn.
- Sökväg till spoolfilen.
- Skrivarmodellen.
- Vad skrivaren vill ha utskriften i för format
- Namnet på datorn.
- Namnet på drivrutinen.

- Idnummer för jobbet.
- Namnet för Windowsdrivrutinen.
- Statusen för utskriften.
- Sökvägen till platsen där spoolfilens olika iterationer arkiveras.
- Tiden för att gå till nästa arbete i skrivarkön.

Headerinformationen följs av FUNCTIONOBJECT. Den innehåller bland annat:

- Justering av utskriften till antingen 0, 90, 180 eller 270 grader.
- Streckkodens höjd i mikrometer.
- Teckensnittets höjd i mikrometer.
- Kontrollsiffra.
- Namnet på teckensnittet.
- MODE används endast av streckkoden Code 128.
- DATA bestämmer vad som ska stå i textdelen av objektet.
- Vilken typ av objekt som ska skrivas ut.
- SERIAL_REPLICATES säger hur många kopior av varje serienummer man ska ha.
- Teckensnittets bredd i mikrometer.
- PRINT_INTERPRETATION_ABOVE bestämmer om serienummret ska stå under eller ovanför streckkoden om streckkod finns.
- Både en x- och en y-koordinat som bestämmer var på etiketten den ska skrivas.

3.1.4 Printservice

Printservice är ett program utvecklat av Prevas och vars uppgift är att ta emot XML-formaterade meddelande som innehåller information som ska skrivas ut. När programmet startas läses XML-filen printServie.config. Den innehåller information om de befintliga drivrutinerna samt information om de skrivare som stöds. De drivrutiner som finns deklarerade laddas in och verifierar att de angivna inställningarna är korrekta. Eventuella fel skrivs ut i Printservice-programmet.

Programmet är sedan inaktivt i väntan på att en något ska skrivas ut. När så sker skickas information till programmet via utskriftskön. Programmet avgör med hjälp av den mottagna informationen vilken drivrutin som ska anropas och vidarebefodrar utskriftsinformationen till en drivrutin. Om den angivna drivrutinen inte finns deklarerad i printService.config eller om utskriftsinformation på något sätt är felaktig och genererar ett fel när drivrutinen bearbetar den, så visas detta felmeddelande i den högra textrutan i Printserviceapplikationen.

3.1.5 printService.config

Detta är en XML-fil som innehåller data om olika skrivare och typsnitt som kan användas med drivrutinerna. printService.config är också den fil som innehåller information om de drivrutiner som finns. Filen laddas in av Printservice.

3.1.6 Zebradrivrutinen

Zebradrivrutinen är en .dll-fil [14] som laddas in av Printservice. Dess uppgift är att ta emot en lista av hashtabeller och utifrån dessa generera utskriftskod i det språk som skrivaren förstår, exempelvis Fingerprint eller ZPL II.

För att kunna interagera med Printservice så måste drivrutinen implementera gränssnittet IDriver som innehåller tre metoder:

- **getName()** – Denna metod returnerar det angivna namnet på drivrutinen. Printservice använder den här metoden för att avgöra vilken drivrutin som ska anropas. Returvärdet från getName() jämförs med det angivna värdet av XML-elementet DriverName i meddelandet som har skapats i MSMQ Studio.

- **setSettings()** - Anropas när Printservice startas och tar emot de inställningar som finns definierade i xxx.xml som parameter. Metoden verifierar att alla inställningar är

korrekt angivna. Om något fel upptäcks så genereras ett exception som fångas upp av Printservice.

- **executeDriver()** - Anropas av Printservice när ett utskriftsmeddelande har mottagits där DriverName-elementet matchar drivrutinens namn. Metoden tar emot information om vilken modell skrivaren är av samt en lista av de fält som skall skrivas ut på etiketten. Därefter genereras den kod som behövs för att skriva ut fälten i det språk som skrivaren förstår. Den befintliga spoolfilen läses in och slås samman med den genererade koden på ett lämpligt sätt innan alltihopa skrivs tillbaka till filen. Eventuella fel genererar ett exception som fångas upp av Printservice.

3.1.7 Spoolfil

En spoolfil är en fil där data lagras temporärt i väntan på fortsatt bearbetning. I vårt fall innehåller spoolfilen den utskriftsinformation som har genererats av Windows skrivardrivrutin för Zebra. Filen kompletteras sedan med ytterligare information av den drivrutin som anropas av Printservice. När spoolfilen är komplett skickas den vidare till skrivaren för utskrift. Ett exempel på en spoolfil finns i bilaga A.

3.1.8 Text only skrivardrivrutin

Text-only-drivrutinen tar emot och skickar vidare ren text till den enhet som finns ansluten på den angivna porten. Drivrutinen finns inkluderad i Windows och är utvecklad för att fungera med alla sorts skrivare.

3.2 Hur Zebrasystemet fungerar

I det externa programmet så bestämmer man utseendet på etiketterna och hur många etiketter man ska skriva ut. Sedan anropar applikationsprogrammet Windows skrivardrivrutin för Zebraskrivare. Denna skriver då ZPL II-kod till spoolfilen beroende på vad applikations-programmet skickade till Windows skrivardrivrutin.

Därefter så anropas setSettings-funktionen av Printservice. I denna funktion kontrollerar man att all information är korrekt i XML-filen printService.config. Funktionen går igenom och kontrollerar så att all information om skrivarmodellerna och typsnitten är korrekta. Efter detta skickar det externa programmet ett MSMQ-meddelande till Printservice-applikationen. Då anropas getName-funktionen som kontrollerar vilken drivrutin som ska användas.

Sedan så tar Printservice-applikationen emot meddelandet och gör om det till en hashtabell [15] som skickas till ZPL II-drivrutinen genom att använda executeDriver-funktionen.

Drivrutinen är skriven i Visual Basic-kod (se avsnitt 4.1.2) där man får in en hashtabell för hur en etikett ska se ut och vilka inställningar drivrutinen ska köra med. Hashtabellen skickas till Zebradrivrutinen genom executeDriver-funktionen.

Detta fungerar på följande sätt:

1: Först så körs executeDriver-funktionen. Där börjar man med att anropa funktionen getHeadInfo(). Denna funktion fyller headInfo med det som finns i headerInformation, det vill säga modellnamn, sökväg till spoolfil med mera.

2: Efter detta så använder programmet findModel-funktionen för att kontrollera så att modellen finns och fyller MODEL med det som finns i headInfo.

3: Sedan läses spoolfilen in och delas upp vid ^PQ med funktionen splitSpoolFile. Funktionens returvärde sparas till variabeln strSpoolCmdEnd. Innehållet i filen som står innan ^PQ skrivs tillbaka till filen.

4: Sedan går programmet igenom de fält som skickats in i hashtabellen och bifogar dessa fält till spoolfilen. Om det är ett fält som innehåller serienummer så sparas antalet upprepade utskrifter som skall ske innan serienumret räknas upp i headInfo.SERIAL_REPLICATES. Om flera fält innehåller serienummer så sparas antalet upprepningar från det fält som ska upprepas flest gånger.

5: Därefter anropas setSerialReplicates. Denna funktion gör så att det skrivs ut ett antal etiketter innan serienumret ökas. Om det finns tio etiketter med olika serienummer och dessa ska skriva ut tre av varje serienummer så blir det totala antalet etiketter 30. I setSerialReplicates så skickas det som sparades i strSpoolCmdEnd in tillsammans med headInfo.SERIAL_REPLICATES. Det är headInfo.SERIAL_REPLICATES som bestämmer hur många av varje serienummer som skall skrivas ut. Dessa två inskickade parametrar kombineras och sparas till strSpoolCmdEnd.

6: Innehållet i strSpoolCmdEnd bifogas sedan till spoolfilen.

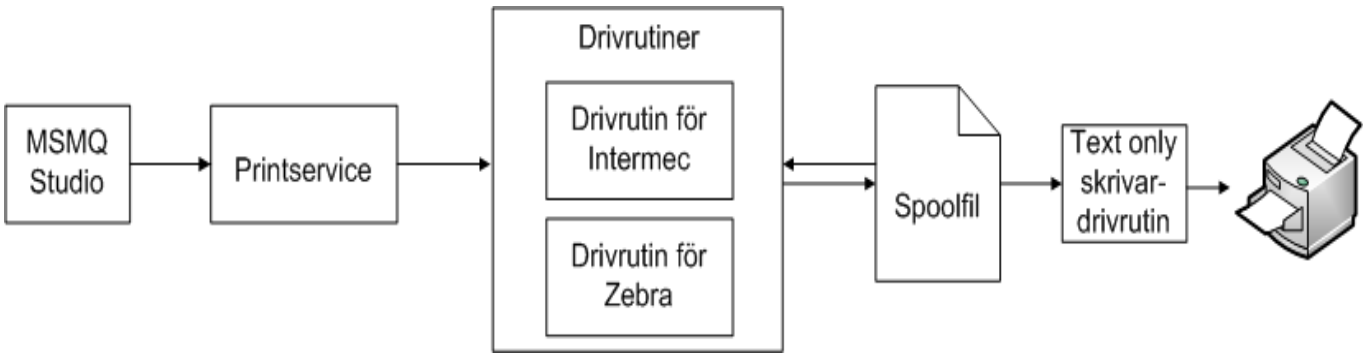
Spoolfilen skickas därefter till en Text only skrivardrivrutin. Text only tar emot en text-fil som den skickar vidare till skrivaren som då skriver ut etiketter efter filens innehåll.

4 Implementation

Det här kapitlet kommer beskriva utvecklingsmiljön (se avsnitt 4.1) som har använts under arbetets gång och hur Intermecdrivrutinen (se avsnitt 4.2) har implementerats. De olika områdena som vi har modifierat för att Intermecdrivrutinen ska ge samma funktionalitet som Zebradrivrutinen är bland annat textfält, streckkoder, serialisering, typsnitt och position. Vi går också igenom de olika problem vi har haft under projektets gång (se avsnitt 4.3). Kapitlet avslutas med en sammanfattning (se avsnitt 4.5).

4.1 Utvecklingsmiljö

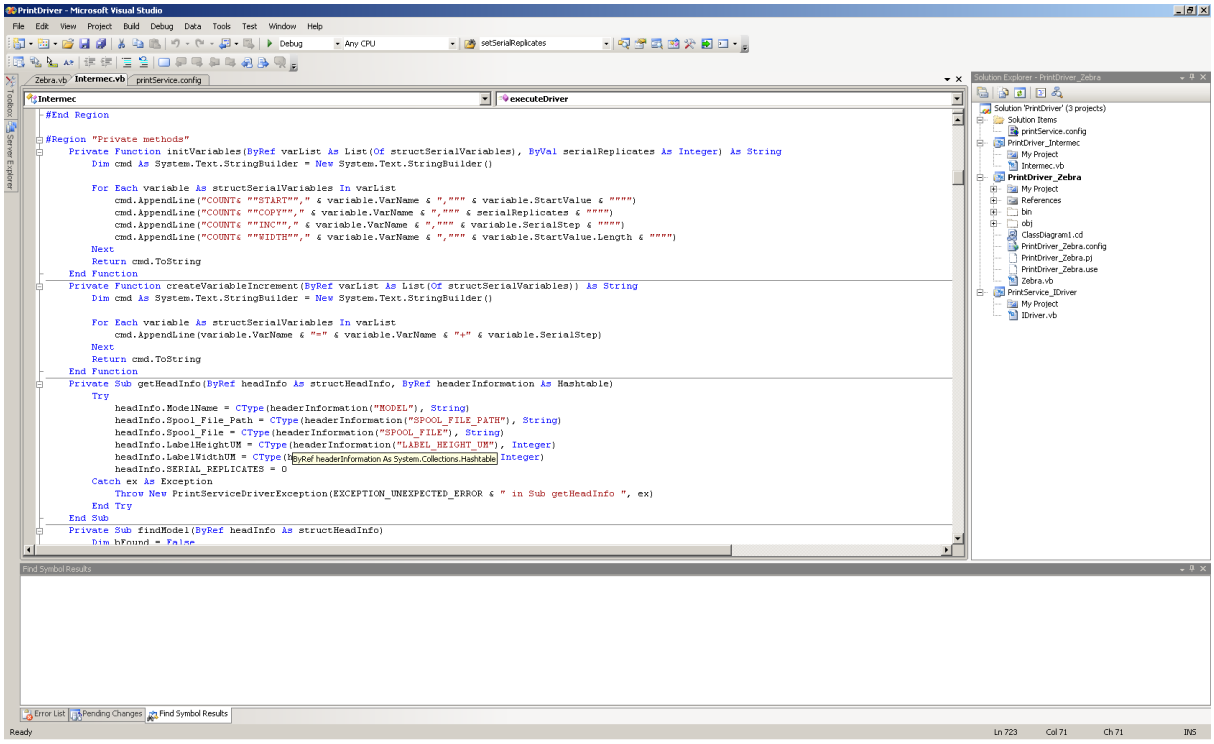
I det här avsnittet beskrivs de verktyg som har använts under implementationen av drivrutinen.



Figur 4.1: Denna bild beskriver hur vår drivrutin arbetar

Utvecklingsmiljön (se figur 4.1) består av MSMQ Studio, Printservice, spoolfil, Text only skrivardrivrutin och en Intermec Easycoder 501 XP (se avsnitt 2.5). MSMQ Studio diskuteras i avsnitt 4.1.3. Printservice är samma som för Zebran (se avsnitt 3.1.4). Text only skrivardrivrutinen är också samma (se avsnitt 3.1.8).

4.1.1 Microsoft Visual Studio 2008

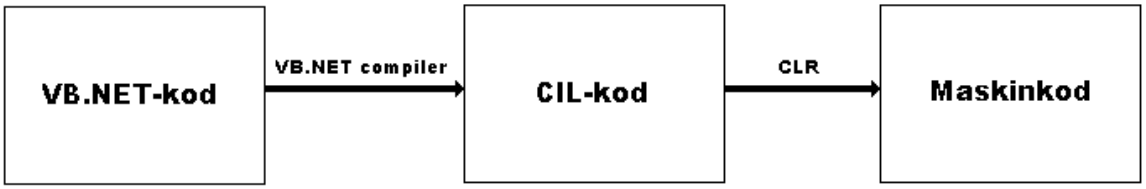


Figur 4.2: Bild av Visual Studios arbetsmiljö.

Visual Studio 2008 (se figur 4.2) är en utvecklingsmiljö från Microsoft som bland annat innehåller källkodseditor, kompilator och debugger. Det finns också inbyggda verktyg för att hantera utseendet på grafiska program. De programmeringsspråk som stöds är C/C++, C# och Visual Basic.NET. Stöd för andra språk kan installeras separat. Texteditorn som används för att skriva källkod med använder sig av Intellisense. Detta är en form av autokomplettering vilket underlättar när man skriver kod. Med den inbyggda debuggern kan man stega igenom programkoden rad för rad vilket underlättar testning och felsökning av det program man utvecklar.

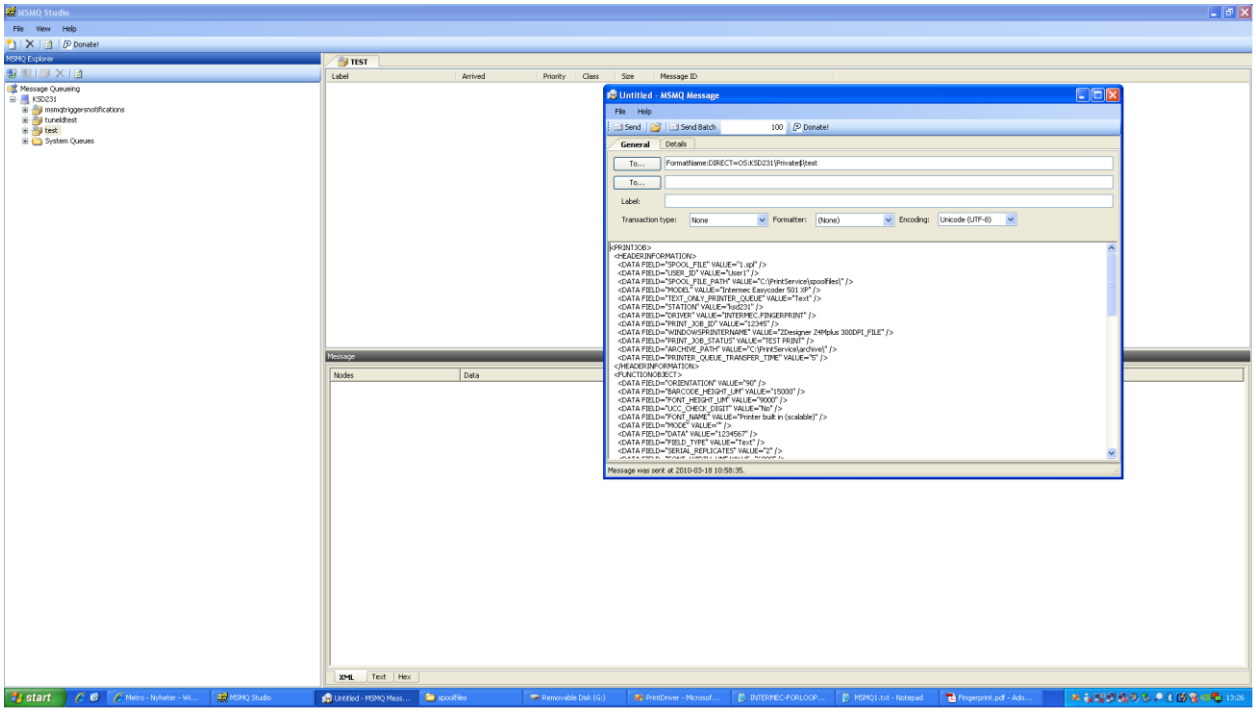
4.1.2 Visual Basic.NET

Visual Basic.NET är ett objektorienterat språk som utvecklats av Microsoft. Språket är en vidareutveckling av Microsofts Visual Basic []. Program skrivna i Visual Basic.NET kompileras till CIL-kod (se figur 4.3), som är ett objektorienterat assemblerspråk. CIL är processor- och plattformsoberoende och exekveras av .NET:s virtuella maskin CLR. Vid exekvering konverteras CIL-koden till maskinkod av en Just-In-Time-kompilator och exekveras sedan av processorn.



Figur 4.3: Bild som visar hur Visual Basic-kod blir till Maskinkod

4.1.3 MSMQ Studio



Figur 4.4: Bild av MSMQ Studio och en öppen MSMQ-meddelandefönster.

För att enkelt skapa och skicka meddelande till köer användes programmet MSMQ Studio (se figur 4.4). När detta program användes så körde vi via en färdig kö som heter test. Med denna kö så skapar man meddelande och fyller dessa med information i XML-format.

4.2 Implementation av drivrutin till Intermec Easycoder 501 XP

I detta avsnitt beskrivs implementationen av drivrutinen för Intermec Easycoder 501 XP. Först ges en generell beskrivning av implementationen. Därefter beskrivs vilka modifikationer som har gjorts för att skapa textfält och hur typsnittshanteringen har implementerats. Även implementationen av serialiseringen av streckkoder och textfält gås

igenom. Andra områden som vi var tvungna att ändra i berörde postioneringen av de olika fälten och modifikationer för att få streckkoder utskrivna . Skapandet av spoolfilen har också modifierats.

4.2.1 Översikt

Större delen av Visual Basic-koden i drivrutinen kopierades från Zebradrivrutinen, vilken består av 1385 rader kod. Överallt där det stod ZPL eller Zebra byttes det till Fingerprint och Intermec. För det mesta så var det variabelnamn som ändrades. Nästa steg var att göra så att Intermecdrivrutinen genererade Fingerprint-kod som motsvarade Zebradrivrutinens ZPL II-kod så bra som möjligt. Nedan följer ett exempel på kod i Zebradrivrutinen och motsvarande kod i Intermecdrivrutinen.

Kod i Zebradrivrutinen:

```
Dim strCode39 As String = "^B3" & strO & "," & strMod43 & ",," &
strPI & strPIA
```

Kod i Intermecdrivrutinen:

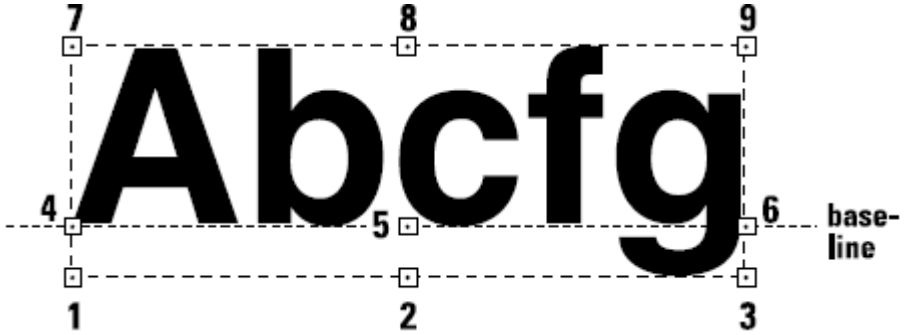
```
strCode39.AppendLine("DIR " & strO)
        If strMod43 = "Y" Then
            strCode39.AppendLine("BARTYPE ""CODE39C""")
        Else
            strCode39.AppendLine("BARTYPE ""CODE39""")
        End If
        If strPI = "Y" Then
            strCode39.AppendLine("BARFONT ON")
        Else
            strCode39.AppendLine("BARFONT OFF")
        End If
```

Dessa två kodavsnitt gör ungefär samma sak. Först bestämmer de justeringen på streckkoden. De skapar en Code 39-streckkod som antingen är vanlig Code 39 eller Code 39C som använder delmängden C för Code 39. De bestämmer också om det bara ska finnas en streckkod eller om det ska finnas text som tillhör streckkoden. Zebrakoden kan göra en sak som Intermeckoden inte kan, nämligen att bestämma om serienumret som tillhör streckkoden ska skrivas ut ovanför eller under streckkoden. I Intermec så finns det inget inbyggt sätt för att få serienummret ovanför streckkoden.

I printService.config har det lagts till några taggar med information för Intermec Easycoder 501 XP samt information om vilka olika typsnitt den ska kunna använda.

4.2.2 Modifikationer hos textfält

Vi fick leta reda på ett antal instruktioner i Fingerprint för att textfälten skulle se likadana ut som i Zebradrivrutinen. Dessa instruktioner är DIR, ALIGN, FONT, PRPOS och PRTXT. DIR bestämmer justeringen hos textfältet. I Zebradrivrutinen är startpositionen som standard punkt 4 (se Figur 4.5) men den tar och räknar om startpunkten till en punkt som motsvarar punkt 7 i Intermec genom att lägga till typsnittshöjden. I Intermec så är standardpunkten 1(se Figur 4.5).. Detta ändras på med ALIGN 7 som sätter startpositionen till punkt 7(se Figur 4.4).



Figur 4.5: Bild av hur ALIGN positionerar startpunkten hos ett textfält

Sedan sätts typsnitt och typsnittsstorlek med FONT-instruktionen(Se avsnitt 4.2.3).PRPOS tar de omräknade värdena för x- och y-koordinaterna och bestämmer var fältet ska ligga på etiketten. Koordinaterna är i mikrometer när de kommer till drivrutinen och den måste räkna om dessa till skrivarens punkter. När textfältet inte ska serialiseras skriver kommandot PRTEXT bara ut datavärdet.

4.2.3 Typsnittsmodifikationer

För textutskrifter användes bara det inbyggda Swiss 721 BT typsnittet, som också är standardtypsnittet i Intermec Easycoder 501 XP. I Zebradrivrutinen används fler typsnitt, men vi vet inte vilka de är ekvivalenta med i Intermecen. Intermecskrivare och Zebraskrivare hanterar storleken av typsnittet på olika sätt. Zebraskrivare tar samma sorts punkter som bestämmer positionen hos fält för att ange höjden och bredden hos typsnittet. I Intermecskrivare däremot så användes en speciell typsnittspunkt som är 0,352 mm, vilket

medförde att den funktion som räknar ut typsnittshöjden behövde modifieras. Höjden för typsnittet är avståndet från toppen av den högsta bokstaven till botten på den lägsta bokstaven. För att ta reda på bredden i Intermecen så vill FONT att bredden ska anges i procent jämfört med höjden. För att få fram detta värde så divideras bredden med höjden. Resultatet multipliceras sedan med 100.

4.2.4 Serialisering av streckkoder och textfält

Om ett fält ska serialiseras så finns det angivet i informationen som har skickats från Printservice. I ZPL II finns det en instruktion som utför serialiseringen automatiskt, vilket inte finns i Fingerprint. Serialiseringen har lösts genom att använda COUNT&-instruktionen. Detta medförde att Intermec Direct Protocol måste vara aktiverat (se avsnitt 2.5.2). Vi började implementera serialiseringen genom att, när vi skapar ett textfält eller ett streckkodsfält, först kontrollera om etiketterna ska serialiseras. Om serialisering används så skapar programmet CNT0$ som är variabelnamnet på en räknare i Fingerprint. Räknaren ökar med ett för varje serialiserat fält som ska finnas i etiketten.

Det som ska serialiseras är de sista siffrorna i serienumret. Detta har lösts genom att använda reguljära uttryck. Alla tecken i serienumret läses från höger till vänster tills ett tecken som inte är en siffra hittas. Uppdelningen av datavärdet sparas för senare användning. Siffrorna i variabelnamnen sparas till en lista tillsammans med startvärdet som är sifferdelen av uppdelningen, hur många etiketter som ska vara likadana innan uppräkning och hur mycket den ska öka med när den ökar. Ett exempel på detta kunde vara serienummret FA11111G2222. Numret delas i två delar, FA11111G och 2222 och båda delarna sparas undan för senare användning.

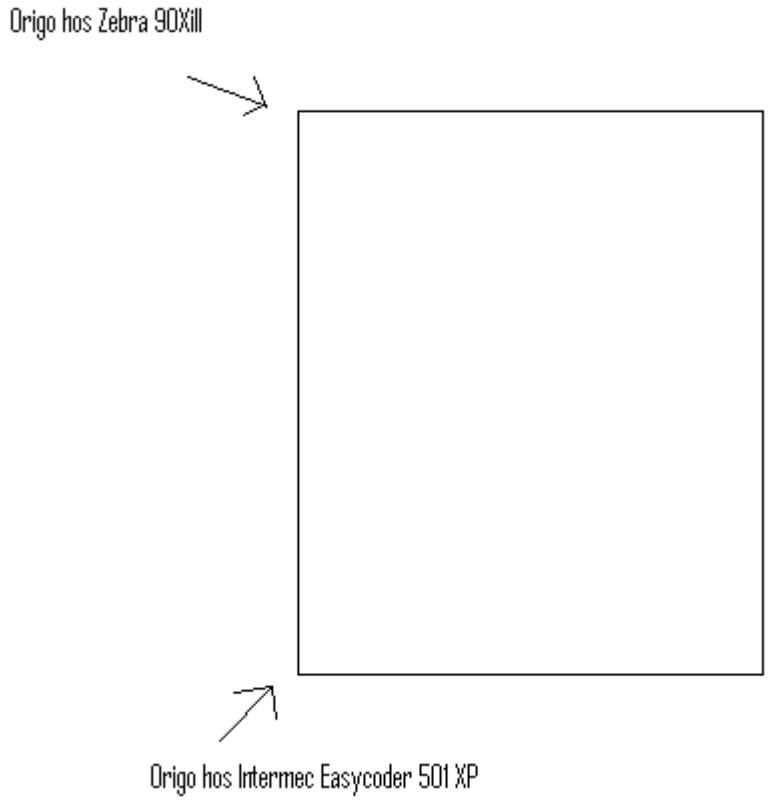
Beroende på om det är ett textfält eller en streckkod som ska serialiseras så skapas antingen en PRTXT- eller en PRBAR-instruktion. I bägge fallen sätts den statiska strängen ihop med delen som ska serialiseras och resultatet skrivs ut. Sedan skapas de olika räknarna. För varje variabel i den undansparade listan med startvärde osv. skapar vi en räknare. Ett exempel på hur räknaren instantieras kan se ut på följande sätt:

```
COUNT& ”START”, 0, ”123”
COUNT& ”COPY”, 0, ”3”
COUNT& ”INC”, 0, ”2”
```

COUNT& ”WIDTH”, 0, ”3”

Här har CNT0$ skapats med startvärdet 123 och den ska skriva ut tre kopior av en etikett. När den ökar serienummret så ska den öka med två. Streckkodens serienummer får heller inte vara mindre än tre tecken bred. Alla instantieringar av räknare måste göras innan de används och detta löstes genom att sätta in instantieringarna före de olika fälten i spoolfilen.

4.2.5 Modifikationer som krävdes för att få ett likvärdigt positionssystem



Figur 4.6: Här ser vi hur Intermecen och Zebran hanterar var origo är på etiketterna

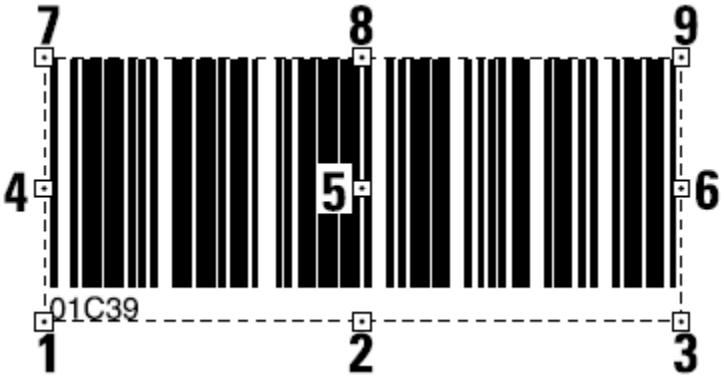
Precis som figur 4.6 visar är startpunkten hos Zebra 90 XiII i det övre vänstra hörnet på etiketterna medan startpunkten hos Intermec Easycoder 501 XP är i det nedre vänstra hörnet. För att göra så att Intermec Easycoder 501 XP har samma origo som Zebra 90XiII var det nödvändigt att lägga till etikettens höjd i MSMQ-meddelandet. Detta används sedan för att räkna fram en ny position genom att ta etikettens höjd minus y-koordianten. Detta ger en position som har det övre vänstra hörnet som origo.

4.2.6 Modifikationer hos streckkoder

Drivrutinen har stöd för två olika sorters streckkoder, "Code39" och "Code128". För att välja typ av streckkod används kommandot "BARTYPE X", där x är namnet på streckkodstypen. För att välja kod 39 så används "CODE39" eller "CODE39C", där den senare inkluderar en genererad kontrollsiffra på slutet.

Kod 128 innehåller tre olika delmängder med teckenuppsättningar. Vi använder kommandot "BARTYPE CODE128", vilket innebär att den bäst passande delmängden väljs automatiskt.

Koordinaterna där streckkodens övre vänstra hörn skall placeras finns angivna i mikrometer och räknas om till skrivarpunkter. För att tala om för skrivaren att det är streckkodens övre vänstra hörn som skall placeras vid den angivna punkten används kommandot "ALIGN 7"(Se bild 4.7). Därefter sätts positionen med kommandot "PRPOS X,Y". Streckkodens orientering sätts med kommandot "DIR X", där X är en siffra mellan ett och fyra som anger hur streckkoden skall roteras.



Figur 4.7: Bild av hur ALIGN positionerar startpunkten hos ett streckkodsfält

Om streckkodens textrepresentation skall skrivas ut under streckkoden så sätts det valda typsnittet och dess storlek med kommandot "BARFONT". Om textrepresentation inte skall skrivas ut så anges detta med "BARFONT OFF".

Höjden på streckkoden bestäms med kommandot "BARHEIGHT X", där X är den önskade höjden angiven i punkter. Storleksförhållandet mellan streckkodens smala och breda linjer ställs in med "BARRATIO X,Y", där X anger de breda linjernas tjocklek relativt de smala och Y anger de smala linjernas tjocklek relativt de breda. Exempelvis så betyder "BARRATIO 3,1" att de breda linjerna är tre gånger tjockare än de smala. Efter att linjernas inbördes storlek har satts kan man bestämma den faktiska storleken i punkter med kommandot "BARMAG

X", där X anger förstoringen. Om instruktionerna "BARRATIO 3,1" och "BARMAG 2" anges så innebär det att linjerna kommer att vara 6 respektive 2 punkter breda.

4.2.7 Modifikationer vid spoolfilsskapande

Den utskriftsinformation som genereras av drivrutinen måste infogas i spoolfilen på ett sätt så att den genererade informationen skrivs ut korrekt tillsammans med spoolfilens tidigare innehåll. I den existerande drivrutinen för Zebra görs detta enkelt genom att den genererade fältinformationen infogas innan utskriftskommandot ^PQ.

Här nedan visas ett exempel på hur en spoolfil kan se ut efter att genererad data från zebra-drivrutinen har infogats. Den fetstilta texten markerar den information som infogats av drivrutinen.

```
  ^XA~TA000~JSN^LT0^MNW^MTD^PON^PMN^LH0,0^JMA^PR4,4^MD0^JU
S
  ^LRN^CI0^XZ
  ~DG000.GRF,64000,100,
  ,::::::::::::::::::::::::::::::::::... ^XA
  ^MMT
  ^LL0623
  ^LS0
  ^FT0,640^XG000.GRF,1,1^FS
  ^A0R,72,48^FT8,160^SN123,1,Y^FS
  ^PQ4,0,1,Y^XZ
  ^XA^ID000.GRF^FS^XZ
```

I exemplet har instruktioner infogats som skriver ut ett textfält vars värde ska ökas med 1 för varje utskriven etikett. Dess startvärde är ”123”. Raden innehåller också instruktioner för att välja typsnitt och teckenstorlek samt för att välja fältets position på etiketten.

Insättningen av utskriftsinformationen som har genererats för utskrift på Intermec-skrivare blir mer komplex på grund av att räknare måste skapas för att kunna räkna upp värdet på de fält som skall serialiseras. Koden för att skapa och initiera räknarvariablerna måste infogas före etikettdefinitionen där de används. Dessutom måste de två instruktioner som positionerar och skriver ut den bild som windows-drivrutinen har skapat flyttas in i etikettsdefinitionen. det löses på följande sätt:

1. Spoolfilens innehåll läses in och de rader som positionerar och skriver ut den bild som har genererats av windows-drivrutinen letas upp. Dessa rader börjar med kommandona PP och PM.

Nedan visas ett exempel på hur en spoolfil som genererats av windowsdrivrutinen kan se ut:

```
INPUT ON
IMAGE LOAD "SSGFX000.PCX",15812,""
ÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿ…
PP69,3438:AN7
PM"SSGFX000.PCX"
LAYOUT RUN ""
PF 4
PRINT KEY OFF
KILL "tmp:SSGFX000.PCX"
```

Markeringen visar raderna med PP- och PM-kommandot.

2. Kod för deklaration och initiering av räknarvariabler samt kod som deklarerar start av ettikettsdefinition infogas innan raden med PP-kommandot. Nedan visas spoolfilens innehåll med de infogade raderna markerade.

```
INPUT ON
IMAGE LOAD "SSGFX000.PCX",15812,""
ÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿ…
FORMAT INPUT "#","@","&"
COUNT& "START",0,"123"
COUNT& "COPY",0,"1"
COUNT& "INC",0,"1"
COUNT& "WIDTH",0,"3"
LAYOUT INPUT "tmp:LABEL1"
PP69,3438:AN7
PM"SSGFX000.PCX"
LAYOUT RUN ""
PF 4
PRINT KEY OFF
KILL "tmp:SSGFX000.PCX"
```

I exemplet skapas en räknare med referensnummer noll, vilket innebär att man vid utskrift kan referera till räknaren genom variabelnamnet ”CNT0$” för att infoga dess värde. Räknarens startvärde sätts till ”123” och ökningen per utskriven etikett sätts till ett.

3. Koden som genererats för de text- och streckkodsfält som skall skrivas ut infogas på raden efter PM-kommandot. Slutligen infogas instruktioner för att avsluta etikettsdefinitionen och sedan läsa in den från minnet. I exemplet nedan infogas kod för att skriva ut ett textfält som innehåller räknaren som skapades tidigare.

```
INPUT ON
IMAGE LOAD "SSGFX000.PCX",15812,""
ÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿÛÿÿÿ…
FORMAT INPUT "#","@","&"
COUNT& "START",0,"123"
COUNT& "COPY",0,"1"
COUNT& "INC",0,"1"
COUNT& "WIDTH",0,"3"
LAYOUT INPUT "tmp:LABEL1"
PP69,3438:AN7
PM"SSGFX000.PCX"
DIR 1
ALIGN 7
FONT "Swiss 721 BT",43,0,40
PRPOS 360,1441
PRTXT CNT0$
LAYOUT END
LAYOUT RUN "tmp:LABEL1"
PF 4
PRINT KEY OFF
KILL "tmp:SSGFX000.PCX"
```

Exemplet visar hur den slutgiltiga spoolfilen ser ut efter att all information från vår drivrutin har infogats. När spoolfilens innehåll skickas till skrivaren kommer fyra etiketter skrivas ut med ett textfält vars värde ökar med ett för varje etikett.

4.3 Problem

I detta avsnitt beskriver vi de problem vi har haft och hur vi har löst dessa.

- Ett problem vi hade var att startpositionen för Zebran och Intermecen inte var desamma (se avsnitt 4.2.5). Vi löste detta genom att lägga till höjden av etiketten i MSMQ-meddelandet.
- Vi hade också problem med typsnittsstorleken. Intermecen använde en annan punkt för att hantera typsnitt än för att bestämma position (se avsnitt 4.2.3). Detta medförde att storleken på texten blev enormt mycket större än den annars skulle ha vara. Detta löste vi genom att ta typsnittshöjden i mikrometer och dela den med 352. På detta sätt fick vi korrekt typsnittsstorlek.
- Ett annat problem vi hade var att få rätt position på de olika fälten. Metoden för att lösa detta problem var att vi skrev ut något på Zebraskrivaren och sedan skrev vi ut

samma sak på Intermecen. Beroende på vad den skrev ut så gjorde vi en ändring i placeringen av fälten.

- En sak som vi inte har lyckats implementera är att få serienumret att skrivas ut ovanför streckkoden. Detta är en funktion som Zebran klarar av. I Zebran så bestäms detta genom att antingen sätta en parameter till Y eller N. Y bestämmer att texten ska stå ovanför streckkoden. Intermecen har inte någon sådan lätt lösning. Den möjliga lösningen vi hade var att när man skapade streckkoden så skulle man skapa ett textfält ovanför streckkoden. Tidsbrist har gjort att vi inte har kunnat implementera detta.
- Delmängderna till CODE 128 har inte fåtts att fungera. Vi har prövat Code-kommandon, till exempel: PRBAR ”«C1111«B1”. Detta kommando ska ge en streckkod där de fyra första siffrorna använder delmängd C när den skrivs och den sista siffran ska använda delmängd B. Det som sker när detta försöks skriva ut är att en etikett matas ut men inget skrivs ut. Lampan på skrivaren som indikerar att något är fel lyser också rött. Vi försökte också köra med CODE128A, CODE128B och CODE128C men samma sak händer. Så av någon anledning fungerar inte delmängderna till Code 128-streckkoden.

4.4 Sammanfattning

I detta kapitel har vi gått igenom hur vi implementerat drivrutinen för Intermec Easycoder 501 XP. Vi har också gått igenom och beskrivit några av de verktyg vi har använt under detta projekt. Avslutningvis beskriver de problem vi har haft under projektet.

5 Resultat och Slutsats

Meningen med detta projekt var att Intermecdrivrutinen skulle ha samma funktionalitet som Zebradrivrutinen. Det mesta av funktionaliteten har vi implementerat men det är fortfarande några funktioner som saknas.

Följande funktioner har vi implementerat: vi har gjort så att positionssystemet är ekvivalent med det som används i Zebran och vi har sett till att fälten hamnar på rätt koordinater när de skrivs ut. En annan sak vi har lyckats implementera är att typsnittstorleken stämmer med det som kommer i MSMQ-meddelandet. Drivrutinen skriver ut textfält med textdata och skriver ut streckkoder av typen Code 39 och Code 128 med eller utan serienumret. Vi har också implementerat serialisering för textfält och streckkoder.

Följande funktioner saknas: vi har inte fått delmängder till Code 128 att fungera och vi har inte fått serienumret att skrivas ut ovanför streckkoden.

Följande punkter kan behövas utvecklas i senare versioner av drivrutinen och kringliggande program:

- Det skulle behövas implementeras fler typsnitt eftersom vi använder standardtypsnittet hela tiden.
- Det skulle behövas läggas till fler skrivare i printService.config-filen.
- Man behöver fixa så att delmängderna till Code 128 streckkoden fungerar och är implementerad på samma sätt som till Zebradrivrutinen.
- Man behöver fixa så att serienummret kan skrivas ut ovanför streckkoden.
- Det skulle behövas göras så att applikationsprogrammet skickar med höjden på etiketten när etiketter ska skrivas ut med Intermec.

Vi tyckte att detta var ett mycket intressant och lärorikt projekt. Det intressanta var att detta projekt var så annorlunda jämfört med allt vi har gjort tidigare och att vi arbetade direkt mot skrivarna. Det som vi har lärt oss under projektets gång var hur skrivarna fungerar samt hur

etikettdefinitioner skrivs i Fingerprint och ZPL II. Vi har lärt oss under arbetetsgång att Fingerprint och ZPL II har många skilnader som vi var tvungna att tänka på när vi arbetade. Till exempel så är Fingerprint ett mycket mer lättläst språk om man jämför med ZPl II. En annan skillnad som vi upptäckte under arbetetsgång var att ZPL II är ett mer kompakt språk när man jämför med Fingerprint. ZPL II använder oftast en instruktion och dess parametrar där Fingerprint använder flera instruktioner för att få samma funktionalitet.

6 Referenser

[1] Intermec Printer AB, (2002), User's Guide EasyCoder 501 XP Bar Code Label Printer

[2] Intermec Technologies Corporation, (2004), Programmer's Reference Manual Intermec Fingerprint v7.80

[3] Zebra Technologies Corporation, (1998), ZPL II Programming Guide

[4] Intermec Technologies Corporation, (2005), Programmer's Reference Manual Intermec Direct Protocol v8.60

[5] Microsoft Corporation, (2010), Message Queuing (MSMQ)[www], Hämtad från http://msdn.microsoft.com/en-us/library/ms711472(VS.85).aspx, (2010-05-04)

[6] Zebra Technologies Corporation, (1997), XiII User's Guide

[7] Wikipedia, Code 128, http://en.wikipedia.org/wiki/Code_128, (2010-05-19)

[8] Wikipedia, Code 39, http://en.wikipedia.org/wiki/Code_39, (2010-05-20)

[9] Wikipedia, Character encoding, http://en.wikipedia.org/wiki/Character_encoding, (2010-06-15)

[10] Wikipedia, ASCII, http://en.wikipedia.org/wiki/ASCII, (2010-06-15)

[11] Wikipedia, Bitmap, http://en.wikipedia.org/wiki/Bitmap, (2010-06-15)

[12] Wikipedia, EPROM, http://en.wikipedia.org/wiki/Eprom, (2010-06-15)

[13] Wikipedia, PCMCIA, http://en.wikipedia.org/wiki/PCMCIA_card, (2010-06-15)

[14] Wikipedia, .dll, http://en.wikipedia.org/wiki/Dynamic-link_library, (2010-06-15)

[15] Wikipedia, hashtabell, http://en.wikipedia.org/wiki/Hashtable, (2010-06-15)

[16] Wikipedia, Visual Basic, http://en.wikipedia.org/wiki/Visual_basic, (2010-06-15)