Utveckling av programvara för inrapportering av vägstatus och utrustning längs med vägar

Development of software for reporting of road condition and equipment along roads

Johan Björk

EXAMENSARBETE
Datateknik
2010
Nr: E3938D

HÖGSKOLAN Dalarna

EXAMENSARBETE, C-nivå Datateknik

Program	Reg nr	Omfattning
Digitalbrott och eSäkerhet, 180 hp	E3938D	15 hp
Namn Johan Björk	Datum 2010-05-21	
Handledare Hans Jones	Examinator Mark Dougherty	
Företag/Institution Logica Sweden AB (Borlänge)	Kontaktperson vid företaget/institutionen Sofia Hagström	
Titel Utveckling av programvara för inrapportering av vägstatus och utrustning längs med vägar		
Nyckelord Väghållare, inrapportering, väginformation, GPS, kartor, ljudinspelning, bildtagning, Windows Mobile, C++, MFC		

Sammanfattning

Väghållare, det vill säga myndigheter som t.ex. Trafikverket och kommuner med ansvar för vägarna, har databaser som innehåller information om status på och utrustning längs vägar. För att upprätthålla kvaliteten på denna information vill man ha en mobil tillämpning som ger möjlighet till uppföljning i personalens fordon. Logica har en utvecklingsplattform, MoveITS, som innehåller ett antal av de grundfunktioner som behövs för att lösa ovanstående behov, exempelvis GPS-positionering och presentation av lagrad information på karta. Systemet finns för Windows XP respektive Windows Mobile.

Syftet med examensarbetet var att utöka MoveITS med det tillägg som ger möjlighet att rapportera in de avvikelser som observeras. Tillägget ska hantera inspelning av ett meddelande på ett trafiksäkert sätt samt GPS-positionering av meddelandet. Visual Studio 2008 användes för programutveckling och programmeringsspråket var C++. Mobiltelefonen HTC Touch Diamond 2 med operativsystemet Windows Mobile 6.1 användes för testning.

Resultatet blev, istället för ett tillägg till MoveITS, en egen applikation. Denna lösning valdes på grund av att vissa delar av MoveITS behövde uppdateras för att fungera. Applikationen kan förutom inspelning av ett meddelande och angivande av GPS-positionering även hantera bildtagning. Funktioner från applikationen kan kopieras och inkluderas i andra system.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

DALARNA
University College

DEGREE PROJECT
Data Engineering

Programme	Reg number	Extent
Digital crime and eSecurity, 180 ECTS	E3938D	15 ECTS
Name of student	Year-Month-Day	
Johan Björk	2010-05-21	
Supervisor	Examiner	
Hans Jones	Mark Dougherty	
Company/Department	Supervisor at the Company/Department	
Logica Sweden AB (Borlänge)	Sofia Hagström	
Title		
Development of software for reporting of road condition and equipment along roads		
Keywords		
Road maintenance organizations, follow-up reporting, road information, GPS, maps, sound recording, picture taking, Windows Mobile, C++, MFC		

Summary
Road maintenance organizations, that is, public authority like Trafikverket and districts with the responsibility for roads, have databases that contain information regarding the condition off and equipment along roads. In order to maintain the quality of this information a mobile solution is needed that allows follow-up in the workers vehicles. Logica has a development platform, MoveITS, which contains a number of functions that are needed to deal with the above need, for example GPS-positioning and presentation of stored information on a map. The system is available on Windows XP and Windows Mobile.

The purpose of this thesis was to increase MoveITS with the supplement that allows the user to report deviations that is observed. The supplement will handle recording of a message in a safe way without risking traffic incidents as well as GPS-positioning of a message. Visual Studio 2008 was used for program development and the programming language was C++. The mobile phone HTC Touch Diamond 2 with the operating system Windows Mobile 6.1 was used for testing.

The result was, instead of a supplement to MoveITS, a stand-alone application. This solution was chosen because parts of MoveITS required updates to function. The application can handle recording of a message and

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

state GPS-positioning as well as picture taking. Functions from the application can be copied and included in other systems.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE

Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

Förord

Jag är tacksam mot alla som har gjort det här projektet möjligt.

Jag vill särskilt tacka:

- Ansvariga på Logica, min kontaktperson Sofia Hagström och personalansvariga Tina Englund.
  Daniel Hedén som hela tiden funnits till hands för teknisk handledning samt bistått med goda råd vid problemlösning.
  Övrig personal för intresse och hjälp.

- Min handledare på skolan Hans Jones för stöd, tips och värdefulla idéer.

Innehållsförteckning

1. Inledning

1.1 Bakgrund

Väghållare, det vill säga myndigheter som t.ex. Trafikverket och kommuner med ansvar för vägarna, har databaser som innehåller information om status på och utrustning längs vägar. För att upprätthålla kvaliteten på denna information vill man ha en mobil tillämpning som ger möjlighet till uppföljning i personalens fordon.

Fordonen ska positioneras via GPS och systemet ska visa aktuell status där man befinner sig. Det ska även vara möjligt att ange avvikelser från det verkliga värdet och var denna avvikelse finns. Hanteringen av informationen måste ske på ett trafiksäkert sätt. Detta kan utföras med röstuppspelning och röstinspelning.

Examensarbetet utfördes åt konsultföretaget Logica Sweden AB i Borlänge under deras ITS-avdelning. Logica är ett internationellt företag med cirka 39 000 medarbetare som levererar IT-lösningar i 36 länder. I Sverige finns cirka 5 200 anställda.

1.2 Syfte

Logica har en utvecklingsplattform, MoveITS, som innehåller ett antal av de grundfunktioner som behövdes för att lösa ovanstående behov, exempelvis GPS-positionering och presentation av lagrad information på karta. Systemet finns för Windows XP respektive Windows Mobile.

Syftet med examensarbetet var att utöka MoveITS med det tillägg som ger möjlighet att rapportera in de avvikelser som observeras. Detta är tänkt att göras genom t.ex. röstinmatning. Varje inspelat röstmeddelande ska också GPS-positioneras. Vid återkomst till kontoret kan alla röstmeddelanden spelas upp. Man ser då var inspelningen skett och kontroll och eventuell justering i databasen kan göras vid behov.
Även presentation av information från systemet kan behöva ske på ett trafiksäkert sätt genom t.ex. röstuppläsning.

1.3 Problemformulering

Arbetet för att uppnå målet med ett så långt som möjligt körbart inrapporteringsprogram kunde uppdelas i två problem. Dessa bestod av dels den inläsning och planering som krävdes för att lösa uppgiften, dels själva utvecklingen av applikationen med åtföljande funktionstestning.

1.4 Avgränsningar

Under arbetets gång framkom det att vissa uppdateringar av MoService och MoITR krävdes för att dessa skulle kunna användas i projektet. Dessa uppdateringar bedömdes vara tidskrävande, exempelvis konvertering av Mimer till SQLite. Ett beslut togs därför att inte använda MoITR. Se rubrik 3.3 och 3.4 för mer detaljerad beskrivning.

Presentation av väginformation i form av uppspelning av ljudmeddelande finns redan inbyggt i MoITR. Därför behövde denna funktion inte läggas in i tillägget.

Inrapporteringsapplikationen hanterar ingen kontakt med en server för överföring av de filer som skapas. Detta är en möjlig vidareutveckling av applikationen.

1.5 Metod

Arbetet genomfördes i huvudsak på Logicas kontor i Borlänge. Kompetensen var utspridd bland ett antal anställda som i olika mån fanns tillgängliga beroende på uppdragsläget. En bärbar dator och ett användarkonto ordnades. Utvecklingsmiljön för programvaran var Visual Studio 2008. Den hårdvara som behövdes stod Logica för. Den kom att innefatta en mobiltelefon med GPS och stöd för Windows Mobile-applikationer. Mobiltelefonen var en HTC Touch Diamond 2 med operativsystemet Windows Mobile 6.1.

Arbetet kom att inledas med studier av Windows Mobile-programmering med C++ och MFC. Undersökning av MoveITS och planering av det tillägg som skulle skapas kom därefter. Den största delen av arbetet utgjordes sedan av själva utvecklingen av applikationen då även dokumentation och olika kodexempel studerades. Rapportskrivning kom att till stor del ske parallellt. Även funktionstester av tillägget utfördes under utvecklingstiden. Slutligen kom redovisning av arbetet.

2. Utredning

2.1 Windows Mobile

2.1.1 Historia

Alla Windows Mobile-versioner upp till 6.5 använder sig av ett operativsystem som heter Windows CE [I1]. Microsoft utvecklade den första versionen av Windows CE redan 1996. Detta operativsystem var från början tänkt att köras på enheter med minimal minneskapacitet. Version 1.0 och 2.0 användes i plattformarna Handheld PC och Palm-sized PC. Dessa blev dock ingen större succé utan det var först vid millenniumskiftet när Pocket PC kom ut som popularitet ökade och operativsystemet började användas mer.

Pocket PC är en handdator med en tryckkänslig skärm på minst 240x240 pixlar. Den använder Windows CE-operativsystem. Den första versionen av Pocket PC kom år 2000. Denna version använde sig av operativsystemet Windows CE 3.0 som innebar en stor uppgradering från de tidigare versionerna. Stora delar av koden skrevs om ända ner på de lägsta nivåerna. Senare kom Pocket PC 2002 och Smartphone 2002. Dessa använde sig också av Windows CE 3.0. Skillnaden mellan en Windows Mobile Smartphone och en Pocket PC är att en Smartphone har telefonfunktioner men saknar tryckkänslig skärm och använder sig istället av knappar.

År 2003 skapades begreppet Windows Mobile, se [I2] och [I5]. Detta begrepp var tänkt att knyta samman alla olika typer av operativsystem som t.ex. Windows CE för Pocket PC och Windows CE för Smartphones. Windows Mobile är alltså olika varianter av operativsystemet Windows CE. Samtidigt kom Pocket PC 2003 och Smartphone 2003 ut. Till dessa kunde man använda olika versioner av Windows Mobile, som Windows Mobile 2003 för Pocket PC Premium Edition, Phone Edition, Professional Edition och Windows Mobile 2003 för Smartphones. Microsoft gav tillverkarna möjlighet att välja vilket operativsystem de ville använda i just sin enhet. Alla dessa baseras egentligen på Windows CE 4.x.

På grund av att hårdvaran hela tiden utvecklades och blev bättre var det möjligt att lägga in mer avancerad funktionalitet. Många av de program som vanligtvis finns på en dator, som t.ex. Microsoft Outlook, Internet Explorer, Word, Excel och Windows Media Player, lades in i operativsystemen fast i mindre versioner och med mycket mindre funktionalitet. Senare år 2004 kom även en uppgraderad version som kallades för Windows Mobile 3.0 SE. Med denna kom små förändringar som t.ex. mer support för andra skärmstorlekar.

År 2005 kom Windows Mobile 5.0 ut. Detta baserades på Windows CE 5.0. Tidigare versioner använde sig av RAM-minne för att lagra all användarinformation. Endast operativsystemet lagrades i ROM. Detta innebar att om enheten förlorade ström så försvann all data som användaren

hade sparat eftersom det låg i RAM. Den nya versionen av Windows Mobile använde sig istället av flashminne för att lagra användar-informationen och endast exekverande processer lagrades i RAM [I2]. Detta medförde att även om strömmen försvann så kom användarens redan sparade data att finnas kvar. Detta gjorde även att batteriet behövde laddas mindre eftersom flashminne inte kräver konstant strömtillförsel för att behålla data till skillnad från RAM. Förutom detta förbättrades även många av de Microsoft-program som redan fanns sedan tidigare versioner och nya program lades in. Även mer stöd för kommunikation gjordes som t.ex. förbättrade bluetooth-stöd och GPS.

Windows Mobile 6 lanserades år 2007 [I3]. Det finns tre olika versioner av operativsystemet. Den första heter Windows Mobile Standard Edition och är till för att användas på vanliga Smartphones utan tryckkänslig skärm. Den andra är Windows Mobile Professional Edition som används på Pocket PC:s med telefonfunktioner och den tredje versionen är Windows Mobile Classic som används på Pocket PC:s utan telefonfunktioner. Windows Mobile 6 baseras på operativsystemet Windows CE 5.0. Förbättringarna från Windows Mobile 5 är framför allt kommunikationen mellan applikationer och olika tjänster. Synkronisering är enklare för att t.ex. koppla enheten till datorn. Den har också ett enklare och snabbare användargränssnitt.

2008 kom Windows Mobile 6.1. Det är denna version av operativsystemet som kom att användas i det här examensarbetet. Förändringarna i 6.1 är mycket små jämfört med version 6. Senare år 2009 kom även Windows Mobile 6.5. Även denna version innehåller endast små förändringar från den tidigare, bland annat mer fingervänligt gränssnitt. Alla mobiltelefoner med operativsystemet Windows Mobile 6 eller senare kallas för Windows Phones. Tabell 1 nedan visar utvecklingen av Windows Mobile från år 2000 till nutid.

	Pocket PC 2000	Pocket PC 2002	Windows Mobile 2003	Windows Mobile 2003 SE	Windows Mobile 5.0	Windows Mobile 6	Windows Mobile 6.1	Windows Mobile 6.5
Pocket PC (Without Mobile Phone)	Pocket PC 2000	Pocket PC 2002	Windows Mobile 2003 for Pocket PC	N/A	Windows Mobile 5.0 for Pocket PC	Windows Mobile 6 Classic	Windows Mobile 6.1 Classic	N/A
Pocket PC (With Mobile Phone)	Pocket PC 2000 Phone Edition	Pocket PC 2002 Phone Edition	Windows Mobile 2003 for Pocket PC Phone Edition	Windows Mobile 2003 SE for Pocket PC Phone Edition	Windows Mobile 5.0 for Pocket PC Phone Edition	Windows Mobile 6 Professional	Windows Mobile 6.1 Professional	Windows Mobile 6.5 Professional
Smartphone (Without Touch Screen)	N/A	Smartphone 2002	Windows Mobile 2003 for Smart-phone	Windows Mobile 2003 SE for Smart-phone	Windows Mobile 5.0 for Smart-phone	Windows Mobile 6 Standard	Windows Mobile 6.1 Standard	Windows Mobile 6.5 Standard

Tabell 1 Windows CE- och Windows Mobile-utveckling [T1]

2.1.2 Framtiden

Just nu pågår utvecklingen av Windows Mobile 7 (Windows Phone 7 Series). Det ryktas om att telefoner med Windows Phone 7 kommer att lanseras under oktober-november 2010. Operativsystemet använder helt ny kod och kommer därför inte att vara kompatibelt med äldre telefoner som använder andra Windows Mobile-versioner, se [I4] och [I20]. Windows Phone 7 Series användargränssnitt bygger inte på Windows CE, istället har en blandning av Windows Media Center, Xbox Dashboard, Zune HD interface och urban signage använts. Det nya gränssnittet kallas för ”metro”. Applikationer som är utvecklade för äldre Windows Mobile-operativsystem kommer inte att fungera på Windows Phone 7.

Windows Phone 7 har stöd för kapacitiva skärmar vilket innebär att det räcker med endast beröring av skärmen för att ett tryck ska registreras. Tidigare har resistiva skärmar använts som kräver hårdare tryck för att de ska reagera.

2.2 Windows Mobile-programmering

Det finns vissa skillnader när det gäller utveckling av applikationer till Windows Mobile jämfört med vanlig utveckling av Windows-applikationer. En Windows Mobile-enhet har oftast betydligt mindre lagringsutrymme. Detta medför att många bibliotek som används för att programmera måste

förminskas så de tar så lite plats som möjligt men ändå innehåller de allra viktigaste funktionerna. Ytterligare en viktig detalj är att kompilera applikationerna så att de fungerar på just den destinationsmiljö som applikationen är tänkt att köras på. Beroende på vilket språk som används kan applikationen dessutom bli processorberoende, vilket innebär att den endast fungerar på en viss typ av processorarkitektur.

2.2.1 Hanterad och ohanterad kod

Det finns två viktiga typer av programkod som kan skapas när man programmerar för Windows Mobile. Dessa är hanterad och ohanterad kod, enligt s.26 i [L1].

Ohanterad kod är kod som exekverar direkt på processorn. Detta gör att ohanterad kod blir processorberoende. Om applikationen ska användas på flera olika processorer måste man därför kompilera applikationen i flera olika versioner. Språk som skapar ohanterad kod är C och C++. Ohanterad kod kan även kallas för maskinberoende kod eller plattformsberoende kod. En fördel med ohanterad kod är att det inte finns något mellanlager mellan exekveringen på processorn och koden. Detta gör att applikationer som använder ohanterad kod blir snabba.

Hanterad kod är programkod som på något sätt hanteras innan den exekverar på processorn, som t.ex. programkod som exekveras av en virtuell maskin. Mellanlagret gör att hanterad kod inte blir processorberoende utan koden kommer att kunna exekvera på flera olika processorer. Språk som skapar hanterad programkod är .NET-språk som t.ex. C#, men även C++ kan skapa hanterad kod vid vissa omständigheter, se tabell 2 nedan. Mellanlagret gör dock att exekveringen inte blir lika snabb som med ohanterad kod.

Att programmera i C och C++ kräver mer av programmeraren. Det krävs mer kod och mer saker att tänka på för att skapa olika funktionaliteter jämfört med att programmera i C#. Det mellanlager som finns mellan hanterad kod och processorn har blivit så effektivt att nackdelarna har minskat när det gäller programmering med hanterad kod. Även hårdvaran har förbättrats vilket gör att det inte är lika viktigt att applikationerna är så snabba som möjligt. Detta gör att applikationer skapade med C# och hanterad kod blir allt vanligare, vilket i sin tur kan medföra att nya applikationer som skapas för Windows Phone 7 kommer att skrivas i språk som skapar hanterad programkod.

	C	C++	C#
Kodtyp	ohanterad	ohanterad/ hanterad	hanterad

Tabell 2 Kodtyp som genereras av olika programmeringsspråk för Windows Mobile [T2]

2.3 Utrustning

2.3.1 Mobiltelefon

Den mobiltelefon som användes för det här projektet var en HTC Touch Diamond 2 (se bild 1). Telefonen har operativsystemet Windows Mobile 6.1 Professional installerat. Den har måtten 108mm hög, 53mm bred och 14 mm djup och väger 118 gram. Den använder sig av en 528 MHz Qualcomm MSM7201A-processor och har 288 MB RAM och 512 MB ROM inbyggt minne. Skärmupplösningen är 800x480.

Det finns även stöd för GPS vilket var ett krav för detta arbete. För utförligare specifikationer se [I10] och bilaga A.

Bild 1 HTC Touch Diamond 2 [B1]

2.3.2 Utvecklingsmiljö

Utvecklingen av applikationen utfördes på den bärbara datorn, en Thinkpad Tseries 6458-AY2, som tillhandahölls av Logica. På denna dator installerades Windows XP Professional med Service Pack 3. För att skriva koden för applikationen användes Visual Studio 2008 som även det installerades.

För att göra det möjligt att skapa applikationer för Windows Mobile i Visual Studio 2008 installerades Windows Mobile 6 Professional SDK (Software Development Kit). Detta innehåller bibliotek, emulatorer och mycket annat för att förenkla utveckling och testning av applikationer till Windows Mobile.

För testning av applikationer direkt på mobiltelefonen installerades och användes ActiveSync 4.5. Det är en programvara som används för att ansluta sig till en mobiltelefon med Windows Mobile. Med anslutningen går det att kommunicera med mobiltelefonen och bland annat föra över filer eller installera/ta bort program. ActiveSync fungerar på Windows XP eller tidigare Windows operativsystem. På Vista eller Windows 7 används Windows Mobile Device Center för att ansluta sig till mobiltelefonen.

010 Editor [I30], som är en hex-editor, användes för att undersöka wave-formatet vilket används för att spara ljuddata.

3. Lösning och resultat

3.1 MoveITS

MoveITS är en teknisk plattform som består av ett antal olika komponenter. Syftet med MoveITS koncept är att hantera befintliga problem i det dagliga användandet av geografisk data kopplat till databaser med väginformation.

Mycket tid behövdes för att studera MoveITS. Många av delarna var föråldrade och uppgraderingar krävdes. Därför lades mer arbete än beräknat ner för att försöka få igång dessa delar.

3.2 MoITR och GapiDraw

MoITR är en applikation som använder sig av MoveITS. MoITR är utvecklad för att visa väginformation på en karta. Det är tänkt att väghållare ska kunna använda denna applikation för att se vilken väginformation som ska finnas längs med vägarna, t.ex. hastighetsskyltar.

Grafiken i MoITR hanteras med GapiDraw. Det är ett grafikbibliotek som är specialutvecklat för speltillverkning med C++ för Windows Mobile, se s.364 i [L1]. Det kan även användas till utveckling av andra grafiska applikationer som t.ex. navigationssystem. GapiDraw har sitt ursprung från Viktoriainstitutet i Göteborg men lyftes över till företaget Develant Technologies AB år 2004.

GapiDraw utnyttjar GAPI (Game API). GAPI är ett programmerings-gränssnitt som kan användas till flera olika språk som t.ex. C, C++ och C# för Windows Mobile. GAPI är specialgjort för att vara så snabbt som möjligt men det har väldigt liten funktionalitet. GapiDraw har betydligt mer funktionalitet men är fortfarande mycket snabbt. Det har exempelvis stöd för att skriva text på skärmen vilket inte GAPI har.

Tidigare har GapiDraw version 3.5 beta använts i MoveITR, men nu finns GapiDraw version 4.2. Den nya versionen användes till det här projektet. En del förändringar från 3.5 beta till 4.2 medförde att MoITR tyvärr inte fungerade. De bestod av medlemsvariabler som inte hade stöd i 4.2, vissa funktioner som hade bytt namn och olika parametrar som hade bytts ut. Korrigeringar i MoITR gjordes därför för att få applikationen att fungera till GapiDraw 4.2. Mer om versionsändringar finns på [I7].

3.3 MoService

MoService är en service som innehåller olika tjänster som måste köras på den mobila enheten för att olika applikationer som t.ex. MoITR ska fungera. MoService hanterar bland annat GPS, kartinformation och kontakt med databasen.

Det var en hel del problem med att få igång MoService. MoService använder sig av Bzip2, se [I31] och rubrik 3.3.3, och Zlib, se [I32] och rubrik 3.3.4, vilka det saknades inkluderingsfiler till. För att få dessa att fungera laddades nya versioner av dem ner, kompilerades och inkluderades. De nya versionerna är specialversioner som är till för operativsystemet Windows CE.

MoService använder sig av Mimer för att hantera kommunikationen mellan databasen med väginformation och enheten. Mimer är numera licensierat och kan därför inte användas utan att en licens skaffas. Därför beslutades det att SQLite (se 3.3.1) skulle användas istället. SQLite är enklare att använda och det kräver mindre kod för att utföra olika funktioner jämfört med Mimer. En wrapper (se 3.3.2) till SQLite testades för att ytterligare förenkla användandet.

Arbetet med att konvertera Mimer till SQLite visade sig dock vara mycket krävande. Flera tusen rader kod behövde konverteras vilket den tekniske handledaren uppskattade till en veckas heltidsarbete för en erfaren C++-programmerare. Detta projekt innefattar inte den uppgiften som istället kommer att lösas inom Logica.

3.3.1 SQLite

SQLite [I33] är ett bibliotek som kan inkluderas och användas i applikationer. Med hjälp av SQLite kan man bland annat hantera en databas och kontakter med andra databaser. SQLite använder filer som lagras på hårddisken för att hantera databasen och data i databasen. SQLite-biblioteket kan bli mycket litet beroende på vilka inställningar man väljer. Det kan till och med vara under 180 kb. Detta gör att det är praktiskt att använda på enheter med lite minnesutrymme som t.ex. mobiltelefoner. Idag är SQLite ett av de vanligaste sätten att hantera databaser på små enheter. Att använda SQLite är helt gratis och källkoden är öppen för allmänheten.

3.3.2 Wrapper

Att använda en wrapper är ett sätt att förenkla användandet av t.ex. ett bibliotek. Det är ett lager utanpå vilket fungerar som ett nytt gränssnitt mot biblioteket och oftast gör användandet mycket enklare. En wrapper kan även användas för att hantera kod mellan olika plattformar, den fungerar då som en brygga mellan två olika språk.

3.3.3 Bzip2

Bzip2 är till för att hantera komprimering av data så att det kräver mindre utrymme [I31]. Bzip2 utvecklades 1996 av den brittiske kompilatorutvecklaren Julian Seward. Det är idag mycket vanligt och används i många olika applikationer. Det är gratis att använda och källkoden är öppen för allmänheten.

3.3.4 Zlib

Zlib är till för att komprimera och packa data [I32]. Zlib skapades av fransmannen Jean-loup Gailly (komprimering) och amerikanen Mark Adler (dekomprimering). Precis som Bzip2 är det helt gratis och källkoden finns öppen för allmänheten.

3.4 Beslut att inte använda MoITR och MoService

På grund av att det innebar mycket arbete för att få igång MoITR och MoService togs beslutet att inte använda dem i just det här projektet. Istället skapades olika klasser och funktioner som enkelt kan inkluderas och integreras i andra system. Dessa klasser och funktioner kommer t.ex. att kunna integreras i MoITR när det fungerar. Beslut togs även om att en applikation som använder sig av de här funktionerna skulle skapas. Denna applikation ska förutom att kunna spela in ljud även kunna ta en bild samt spara GPS-positionen där detta görs.

Presentation av information från systemet finns redan inbyggt i MoITR. Därför behövde denna funktion inte skapas. Detta hade heller inte varit möjligt i den fristående applikationen på grund av att denna inte har tillgång till den trafik- och väginformation som MoITR har.

3.5 Inspelning av ljud

Applikationen ska kunna hantera inspelning av ett ljudmeddelande. För att möjliggöra detta gjordes olika tester och undersökningar.

Till att börja med användes C# för att spela in ett meddelande. Till C# och Windows Mobile finns det ett bibliotek som heter SDF (Smart Device Framework). SDF innehåller olika funktioner som gör hantering av bland annat ljudinspelning mycket enklare. SDF Community Edition är gratis att använda. Det var mycket enkelt att skapa en applikation som spelade in ett meddelande med SDF, endast några rader kod krävdes. På grund av att applikationen använder C++ gick det dock inte att direkt använda SDF. Därför gjordes undersökningar för att ta reda på om det gick att ge applikationen stöd för C# -bibliotek.

För vanliga Windows-applikationer i C++ går det att lägga till ett alternativ som ger stöd för CLR (Common Language Runtime). Om alternativet är aktivt går det att inkludera C# -bibliotek i ett C++ -projekt, men tyvärr finns inte detta alternativ till Windows Mobile-projekt och därför gick det inte att använda SDF till den här programlösningen.

Undersökning gjordes av ljudinspelning i C++. Det är betydligt svårare att spela in ljud i C++ jämfört med C#. Det finns olika funktioner som kan hantera kommunikationen mellan ljudenheten (ljudkortet) och applikationen. Dessa funktioner har namnen WaveInxxxx och WaveOutxxxx där xxxx kan vara olika namn som t.ex. Open. WaveIn används för att spela in ljud medan WaveOut används för att spela upp ljud. Den första WaveIn-funktionen som måste anropas är waveInOpen. Denna ser till att kontakten med ljudenheten initieras. Vid anropet skickas en WAVEFORMATEX-struktur med [I29], vilken innehåller inställningar på ljudets format. Även andra parametrar skickas med som exempelvis en pekare till en så kallad callback-funktion som anropas från ljudenheten vid vissa händelser.

Data som spelas in av ljudenheten lagras i buffrar. Buffrarna förbereds först med funktionen waveInPrepareHeader som tar emot olika inställningar och en referens till bufferten. Med hjälp av waveInAddBuffer-funktionen läggs sedan en buffert till som ljudenheten kan använda. För att ljudinspelningen ska gå så smidigt som möjligt bör flera buffrar användas. Om endast en buffert används kan data gå förlorad eftersom ljudenheten inte alltid kommer att ha tillgång till en ledig buffert att spara data till.

Vid vissa händelser kan ljudenheten skicka meddelanden till applikationen. Dessa meddelanden tas emot av den funktion som angavs i waveInOpen-anropet. Om meddelandet som skickas är av typen WM_DATA betyder det att en buffert är full. Data från bufferten tas då emot av funktionen och bör hanteras på något sätt där, t.ex. spara data till fil. När sedan applikationen är färdig med bufferten skickas den åter till ljudenheten med waveInAddBuffer.

För att starta inspelningen görs ett anrop med waveInStart och för att stoppa den anropas waveInStop. Om ingen ny inspelning ska göras måste buffrarna rensas och frigöras. Detta görs med funktionen waveInUnprepareHeader. Efter att detta är gjort kan ett anrop med waveInClose göras för att avsluta kontakten med ljudenheten. Bild 2 nedan visar hur WaveIn-funktionerna används. Se [I28] för mer information om inspelning och uppspelning av ljud med wave-formatet.

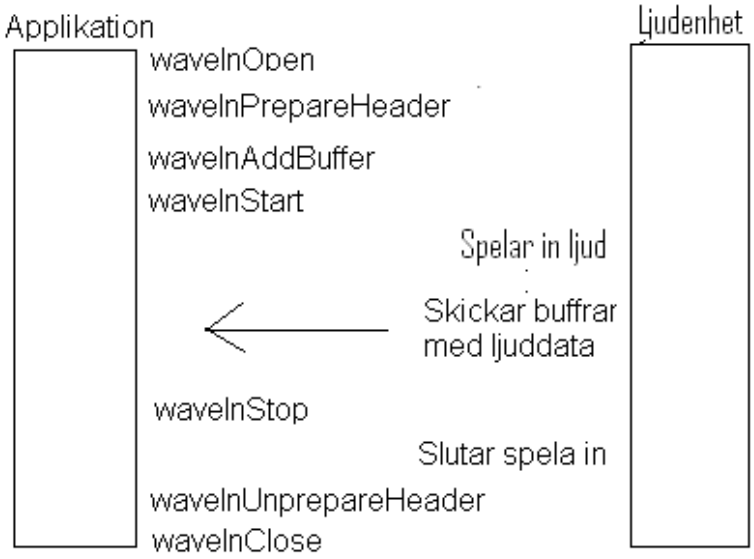

Bild 2 Hur WaveIn-funktioner används för att spela in ljud [B2]

3.6 Spara ljuddata till fil

Att spara ljuddata till en fil är inte trivialt. För Windows-applikationer finns det flera olika bibliotek som kan användas för just det ändamålet. Exempelvis går det att använda mmio (memory mapped input/output) - funktioner som är gjorda för att förenkla skapandet av bland annat filer med RIFF-format (mer information om RIFF längre fram). Det finns även en klass som heter CWaveFile som sköter det mesta själv när man skriver till wave-filer. Men tyvärr har ingen av dessa stöd i Windows Mobile.

Det ljudfilsformat som användes i det här projektet var wave-formatet. Det användes på grund av att det har en enkel struktur och därför är enkelt att använda. Det stöds dessutom av de flesta musikspelare.

Beroende på att det inte finns färdiga funktioner att använda för att skriva till en wave-fil skapades egna funktioner för just detta. För att kunna skapa egna funktioner gjordes undersökning av hur en wave-fil är uppbyggd. En wave-fil använder RIFF (Resource Interchange File Format), se [I14]. Detta är ett format som har skapats för att hantera multimediafiler. RIFF byggs upp av chunkar. Först i chunken kommer ett chunk-ID, detta är 4 byte stort. Därefter kommer 4 byte som innehåller storleken på datadelen i chunken och till sist kommer datadelen.

En wave-fil består av en chunk som kallas för RIFF, i denna anges filformatet (wave i detta fall) och filens storlek. Efter RIFF-chunken kommer en fmt-chunk och en data-chunk. I fmt-chunken finns information om ljudformatet och i data-chunken finns ljuddata. Strukturen av en wave-fils uppbyggnad syns i bild 3.

"RIFF"	Chunk ID
Size of rest of file	Filstorlek
"WAVE"	Chunk data (wave fil)
"fmt "	Chunk ID
WAVEFORMATEX Size	Chunk datastorlek
WAVEFORMATEX	Chunk data (ljudformatet)
"data"	Chunk ID
Data Size	Chunk datastorlek
PCM Data	Chunk data (ljudet)

Bild 3 Wave-strukturen [B3]

Vid utvecklingen av de funktioner som skapar wave-filen användes 010 Editor för att se att allt blev rätt. Detta är en hex-editor där det går att se hex-koden av en fil. Till 010 Editor användes även en .wav template som gjorde det mycket enkelt att se wave-filens struktur.

Filen myWav.wav som har öppnats i 010 Editor syns i bild 4. En wave-template har körts på filen. Templaten gör att det går att se filens struktur (nedre halvan av bilden), där det går att se de tre chunkarna. I RIFF-chunken syns exempelvis filens storlek som är 11069 byte samt filtypen WAVE.

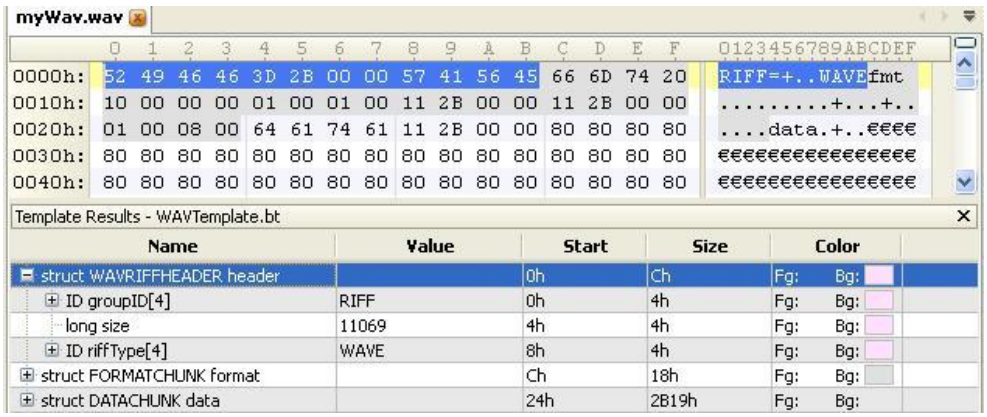

Bild 4 010 Editor med en wave-fil öppen och en wave-template aktiv. [B4]

3.7 Klassen JBSound

För att förenkla användandet av ljudinspelning och sparandet av ljuddata till en wave-fil skapades klassen JBSound (se programkod i bilaga B). Genom att använda denna klass är det mycket enkelt att spela in ljud med C++ och

Windows Mobile. Klassen döljer mycket för användaren och erbjuder endast funktioner för att starta inspelningen, stoppa den och ändra ljudformat.

Start()-funktionen används för att starta inspelningen. Den tar sökvägen och namnet på var ljudfilen ska sparas som enda argument. Funktionen skapar en tråd och startar den. Tråden sköter sedan inspelningen och ser till att inspelningen inte låser andra funktioner i mobiltelefonen som t.ex. användargränssnittet. För att stoppa inspelningen räcker det att använda funktionen Stop(). Om användaren av klassen vill ändra ljudformat kan detta göras genom funktionen setFormat() som tar fyra argument. Dessa är antalet kanaler (oftast en eller två), antalet samplingar per sekund, antalet bitar per sampling samt formatet på wave-ljudet som t.ex. kan vara WAVE_FORMAT_PCM. Blockriktning och byte i genomsnitt per sekund beräknas automatiskt inuti funktionen.

Hantering av buffrar och uppbyggnaden av wave-filen sker helt i klassen. Användaren behöver inte bry sig om detta. Om användaren däremot vill ändra antalet buffrar eller hanteringen av wave-filen måste detta utföras genom att gå in i källkoden för klassen och göra förändringar. Det finns ingen funktion för att hantera detta. Klassen JBSound kan enkelt integreras i andra system genom att kopiera JBSound.h och JBSound.cpp.

3.8 Kamera med GPS-taggade bilder

Väghållare kan även behöva ta foton för att dokumentera de fel som påträffas längs med vägarna, exempelvis sprickor och hål som ska åtgärdas. En funktion som kan ta bilder och spara dem med GPS-position kan därför möta detta behov.

Det finns redan stöd för C++-programmering för att använda de inbyggda kamerafunktionerna i Windows Mobile. Funktionen SHCameraCapture kan anropas för att starta den inbyggda kamerahanteringen [I22]. Denna funktion tar emot en parameter som är en referens till en SHCAMERACAPTURE-struktur vilken innehåller inställningar som kameran behöver. Det kan gälla att t.ex. ta en stillbild eller filma, var bilden/videon ska sparas och namnet samt upplösningen på bilden/videon.

Genom att använda SHCameraCapture-funktionen och SHCAMERACAPTURE-strukturen skapades en ny funktion, CameraCapture (se programkod i bilaga C), som endast tar emot filnamnet och på så sätt förenklas användandet ytterligare. Denna funktion kan enkelt kopieras och integreras i andra system.

3.9 GPS-hantering

De ljudfiler och bildfiler som sparas av applikationen ska sparas tillsammans med GPS-position. Därför gjordes undersökningar av hur kontakten med GPS-enheten i mobiltelefonen kan hanteras.

En GPS skickar data i ett standardformat som heter NMEA (National Marine Electronics Association), för mer information om NMEA se [I23]. NMEA-data kan vara svår att tyda och det krävs en hel del programmering för att få ut den information som behövs som t.ex. latitud och longitud. Istället för att försöka tyda NMEA-data direkt kan GPS Intermedia Driver Functions användas [I24]. GPS Intermedia Driver förenklar användandet av en GPS. För att hantera GPS Intermedia Driver måste en länk till biblioteket Gpsapi.lib skapas och header-filen gpsapi.h inkluderas i C++-applikationen.

Till att börja med så öppnas en kontakt med GPS-enheten. Detta görs genom att använda GPSOpenDevice. Därefter är det bara att använda GPSGetPosition för att hämta GPS-data. Det som returneras av GPSGetPosition är en GPS_POSITION-struktur. Denna innehåller data om bland annat den nuvarande positionen. När användandet av GPS-enheten är klart anropas GPSCloseDevice för att avsluta kontakten med GPS:en. Det är mycket viktigt att anropa GPSCloseDevice så tidigt som möjligt eftersom GPS:en konsumerar stora mängder energi.

3.10 Klassen JBGPS

Klassen JBGPS (se programkod i bilaga D) skapades för att det ska vara enkelt att använda en GPS. Genom att kopiera filerna JBGPS.h och JBGPS.cpp kan klassen enkelt inkluderas i andra system.

För att använda JBGPS måste ett objekt av klassen skapas. Med detta objekt kan sedan funktionerna Start() och Stop() anropas vilka startar och stoppar GPS:en. Efter att Start() har anropats startar en ny tråd, denna innehåller en loop som loopar tills Stop() anropas. I loopen hämtas en GPS_POSITION-struktur som sparas till current_pos som är en publik variabel. Loopen innehåller även ett anrop till Sleep som ser till att tråden sover i några sekunder innan nästa GPS_POSITION hämtas.

Om den GPS_POSITION-struktur som returneras inte innehåller giltig data betyder det oftast att något är fel, t.ex. att GPS-enheten inte kan kalkylera sin position. För att kontrollera detta skapades GPS_status som är en publik variabel vilken innehåller statusen på GPS-positionen. Genom att använda de två publika variablerna current_pos och GPS_status kan användaren av klassen enkelt få GPS-status och GPS-position.

Inledningsvis blev det problem med att få tag på en GPS-position. Detta problem uppstod på grund av att GPS-enheten öppnades och stängdes inne i loopen som finns i tråden. Detta medförde att för varje position som

hämtades så öppnades enheten och därefter hämtades positionen. På den korta tiden efter att kontakten öppnades hann inte GPS-enheten kalkylera GPS-positionen och därför returnerades en tom GPS_POSITION-struktur. För att lösa detta problem flyttades öppningen och stängningen av GPS-enheten ut utanför loopen. Detta gör att så fort Start() anropas kommer enheten att öppnas och sedan vara öppen ända tills det att Stop() anropas.

En nackdel med den här lösningen är att GPS-enheten kräver mycket batterienergi medan kontakten är öppen, till skillnad mot den tidigare metoden när kontakten öppnades och stängdes inne i loopen och mindre mängd energi användes. Detta får ändå anses acceptabelt på grund av att GPS-enheten nu ger giltiga positioner.

3.11 Applikationen EXapp

Applikationen EXapp blev den slutliga lösningen på projektet och skapades för att visa hur de tidigare skapade klasserna och funktionerna kan utnyttjas. Applikationen innehåller funktioner för att spela in ett meddelande, ta en bild samt spara resultatet till en fil med datum, tid och GPS-position som namn.

EXapp använder C++-biblioteket MFC (Microsoft Foundation Class). MFC används framför allt för att det innehåller funktioner som betydligt underlättar skapandet av ett grafiskt gränssnitt. Det innehåller även många andra funktioner som gör användandet av C++ mycket enklare. För mer information om MFC se [I26] och s.313 i [L1].

EXapp består av ett fönster med fyra knappar (se bild 5). Fönstret innehåller även två textfält där GPS-status och filstatus fylls i vid vissa omständigheter. Knapparna gjordes stora vilket gör att de enkelt kan tryckas och att användaren kan hantera applikationen på ett trafiksäkert sätt.

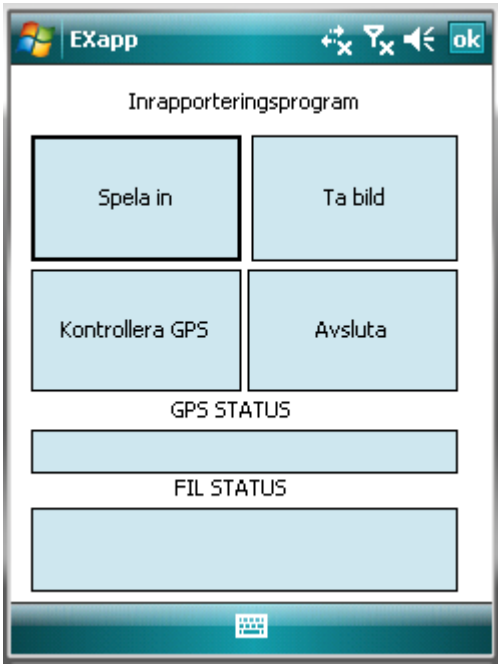

Bild 5 Applikationen EXapp [B5]

Om användaren trycker på Spela in kommer inspelningen av ett ljudmeddelande att starta. Texten på knappen ändras därmed till Sluta spela in och användaren kan trycka på denna igen för att avsluta inspelningen. Efter att Sluta spela in har trycks in kommer textfältet GPS STATUS att innehålla GPS-statusen. Den kan vara ”Ingen GPS data” eller ”GPS aktiv”. Textfältet FIL STATUS kommer att innehålla namnet på den fil som ljuddata har sparats till. Om ingen giltig GPS-position finns kommer filnamnet att endast innehålla datum och tid. Trycker användaren på Ta bild kommer kamerahanteringen att aktiveras och med denna kan användaren ta en bild för att därefter återgå till applikationen. Textfälten GPS STATUS och FIL STATUS kommer då att vara ifyllda med GPS-status och filstatus. Bild 6 visar hur det kan se ut efter att användaren har spelat in ett meddelande.

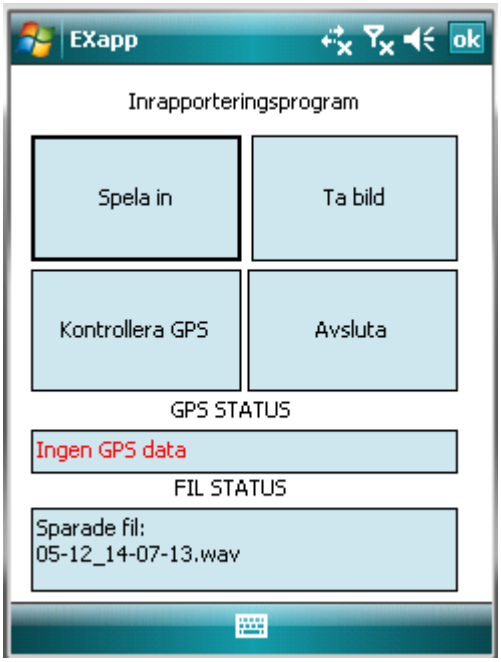

Bild 6 Hur EXapp kan se ut efter att ett ljudmeddelande har spelats in [B6]

För att användaren ska ha möjlighet att kontrollera om det finns en giltig GPS-position lades en knapp med just den funktionaliteten in. När användaren trycker på Kontrollera GPS kommer GPS-statusfältet att fyllas i. Den sista knappen avslutar applikationen.

Det är mycket troligt att användaren av applikationen kommer att spela in ett meddelande och ta en bild på samma plats. Om förflyttning en kort bit då sker mellan inspelning och bildtagning kan GPS-positionen i filnamnen bli olika trots att bilden och inspelningen hör till samma inrapportering. För att det ska bli lättare att se vilka filer som hör ihop lades en kontroll av GPS-position in. Denna kontroll kontrollerar den senast gjorda inspelnings- eller bildtagningspositionen med den nuvarande positionen. Om positionsändringen är mycket liten kommer den gamla positionen att användas istället för den nuvarande och på detta sätt får de båda filerna samma position i filnamnet.

EXapp använder sig av JBSound, CameraCapture och JBGPS för att hantera ljudinspelning, bildtagning och GPS-positionering. Ett klassdiagram visas i bild 7 nedan. I bilden syns klassen CEXappDlg. Det är denna klass som hanterar det grafiska gränssnittet samt vad som inträffar vid vissa händelser som t.ex. knapptryckningar. Detta är den viktigaste klassen i EXapp och den innehåller ett objekt av klassen JBGPS och ett annat av klassen JBSound. CEXappDlg innehåller även funktionen CameraCapture. De fyra ”OnBnClicked”-funktionerna är de funktioner som anropas vid respektive knapptryckning. Endast de funktioner som tidigare har förklarats har tagits med i klassdiagrammet. Övriga mindre viktiga funktioner har utelämnats.

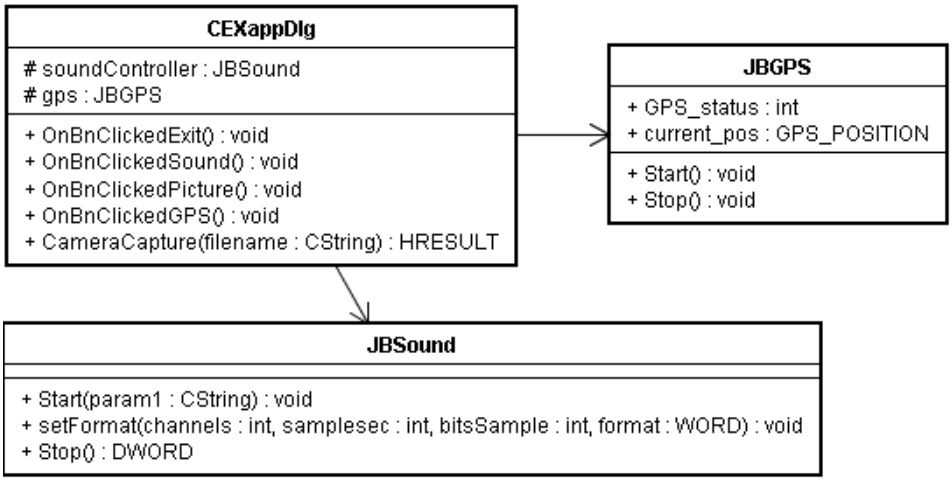

Bild 7 Klassdiagram av en del av applikationen EXapp [B7]

3.12 Funktionstest av EXapp

För att kontrollera funktionaliteten på EXapp utfördes olika tester. Syftet med dessa tester var att kunna utföra korrigeringar om något inte fungerade som det var tänkt.

Inledande tester bestod av att kontrollera när GPS-enheten ger giltiga positioner och det visade sig då att detta inte skedde vid vissa omständigheter. En av dessa omständigheter kunde t.ex. vara position inne i vissa byggnader. Det kunde även bero på hur många satelliter som var i rätt position. I Logicas kontorslokaler i Borlänge fungerade GPS:en oftast på den övre våningen men inte på den nedre. Även beroende på GPS-enhetens kvalitet kan det vara olika svårt att få en GPS-position. Att det ibland var vissa problem med att få en position i den mobiltelefon som användes i det här projektet behöver inte betyda att det uppstår samma problem för andra enheter. Tester visade även att det kan ta flera minuter efter start av GPS-enheten innan denna får en giltig position.

Även en testkörning av applikationen genomfördes. En biltur från Logicas kontor i Borlänge till Hälsinggården i Falun gjordes där meddelanden spelades in och bilder togs. Resultatet blev 10 olika filer. Varje fil hade en GPS-position i namnet. Nedan syns namnen på de 10 filerna.

1. 05-10_12-16-24Lat60,47311Lng15,45597.wav
2. 05-10_12-18-58Lat60,48332Lng15,45502.jpg
3. 05-10_12-21-01Lat60,49696Lng15,51012.jpg
4. 05-10_12-23-35Lat60,52570Lng15,55304.wav
5. 05-10_12-26-17Lat60,55735Lng15,58220.jpg
6. 05-10_12-29-57Lat60,59761Lng15,62164.jpg
7. 05-10_12-33-16Lat60,60507Lng15,63973.jpg
8. 05-10_12-35-32Lat60,59796Lng15,67496.jpg
9. 05-10_12-37-54Lat60,59024Lng15,69672.jpg
10. 05-10_12-39-17Lat60,58760Lng15,70168.wav

För att få en bättre bild av om positioneringarna blev riktiga lades koordinaterna in på en karta. Programmet Google Earth användes för att visa karta och för funktionen att markera koordinater. Till att börja med konverterades koordinaterna från decimalminuter till grader, minuter och sekunder, en konverterare finns på [I27]. De lades sedan in som positioner på en karta i Google Earth vilket visas i bild 8 nedan.

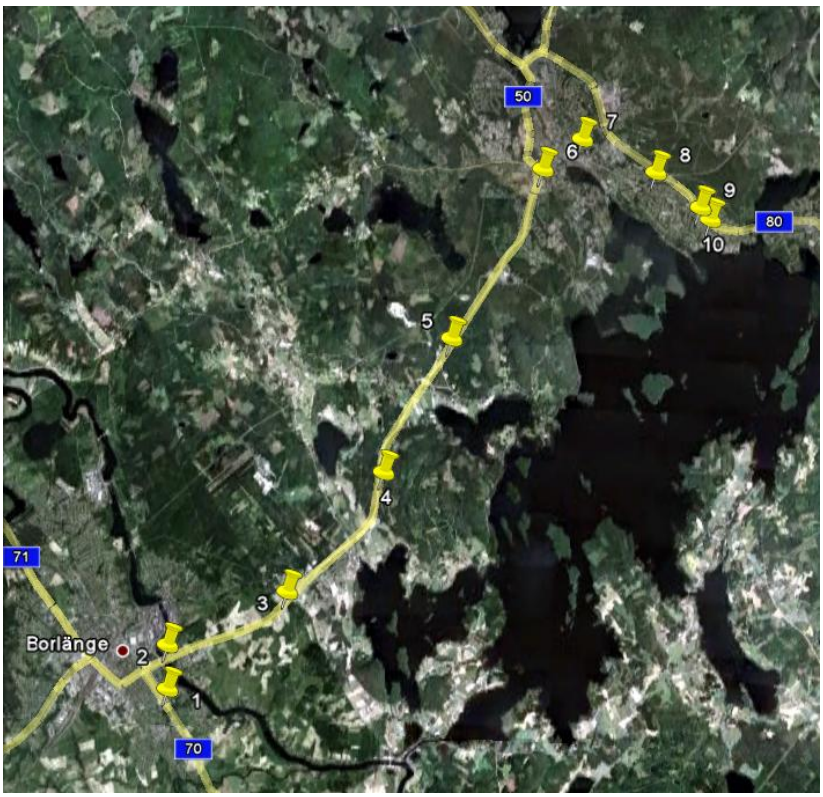

Bild 8 Google Earth med koordinater från testet [B8]

Testet var lyckat, positioneringarna blev mycket bra. I bilden visas alla koordinater men inte hur precisa de är. Det går dock att zooma in i Google Earth för att se exakt var positionerna är satta och dessa visade sig vara mycket nära de ställen där meddelandena spelades in och bilderna togs.

3.13 Alternativ lösning till väghållares behov

Som alternativ till att utnyttja MoveITS finns det även andra system som går att använda i syfte att skapa inrapporteringsmöjligheter med hjälp av karta. Bland annat har Google skapat enkla funktioner för att visa kartmaterial. Med hjälp av Google Maps går det även att skapa egna kartor och använda dessa för att markera olika punkter [I16]. Denna funktion kallar Google för MyMaps. Där finns inga möjligheter att få fram väginformation som t.ex. hastighet men det är mycket enkelt att markera en plats. Det skulle därför vara möjligt för väghållare att med hjälp av Google Maps markera platser

längs med vägen för att registrera olika fel. På grund av att informationen och kartorna sparas på en server behöver då ingen överföring göras från mobilen vid återkomst till kontoret utan det går att direkt surfa till kartan genom en webbläsare.

Google Maps fungerar på de flesta telefoner med javastöd, bland annat Windows Mobile, Android och Symbian mobiltelefoner. Det går att få kartor både som satellitbilder och vanliga kartor (se bild 9). Google har redan utvecklat en applikation till Androidtelefoner som kan hantera kartor som är skapade i MyMaps [I21].

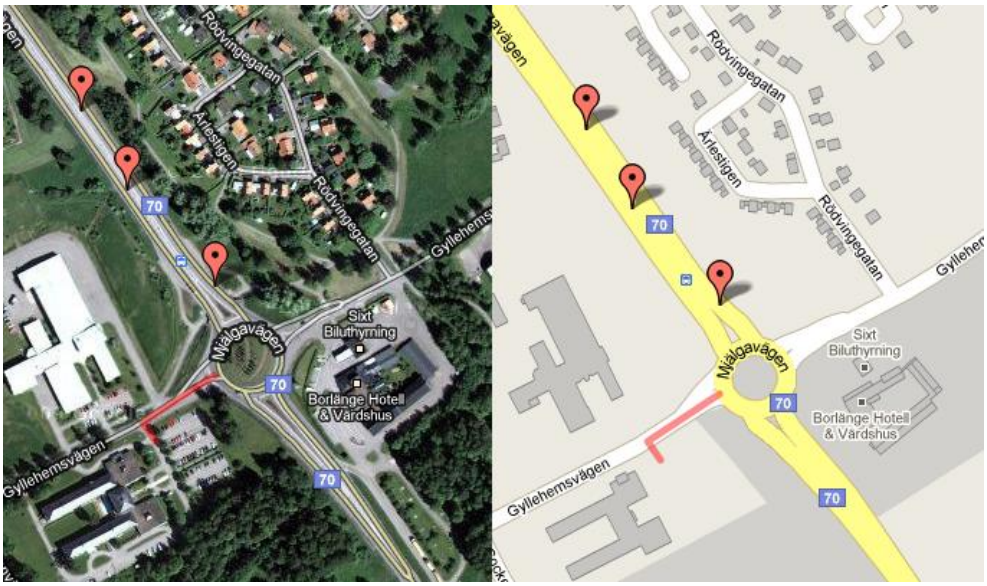

Bild 9 Google maps satellitbild till vänster och vanlig Google maps-karta till höger [B9]

Vissa nackdelar finns dock med att använda Googles egna kartfunktioner. Man har bland annat ingen kontroll över vilka funktioner som finns och vad de ska göra. Exempelvis behövs ett trafiksäkert sätt att mata in punkter och spela in meddelanden.

Google har skapat Google Maps API [I18], som ger användaren möjligheten att hantera sina egna funktioner. Det går att lägga in egna Google Maps-kartor på en hemsida med javascript. Google Maps API erbjuder även olika verktyg för att manipulera kartor. På detta sätt kan en applikation som passar väghållares ändamål skapas. Om Google Maps API planeras att använda i ett kommersiellt syfta måste dock en licens till Google Maps API-premier köpas.

4. Slutsatser

4.1 Resultat och problem

Syftet med examensarbetet var att använda MoveITS och MoITR för att utveckla inrapporteringstillägget. Resultatet blev istället en helt egen applikation, EXapp, som inte använder sig av MoveITS och MoITR. Däremot har applikationen samma funktioner som det tillägg som planerades, plus funktion för bilddokumentation. Anledningen till att MoITR inte användes var att vissa förändringar behövdes för att det skulle fungera. Dessa förändringar innebar alltför mycket arbete för att rymmas inom detta projekt men kommer eventuellt att utföras av någon Logicaanställd i framtiden.

Klasserna för ljudinspelning, GPS samt funktionen för bildtagning skapades så att de enkelt kan kopieras och inkluderas i andra system. På så sätt går det att utan problem flytta över funktionerna till MoITR om detta önskas vid ett senare tillfälle. Presentation av information från systemet finns redan inbyggt i MoITR. Därför behövde denna funktion inte skapas. Detta hade heller inte varit möjligt i EXapp på grund av att denna inte har tillgång till den trafik- och väginformation som MoITR har.

En del problem framkom vid inspelning av ljud. Det finns ingen färdig funktion som hanterar sparandet av ljuddata till fil för Windows Mobile. För att lösa detta skapades därför egna funktioner. Det krävdes stor noggrannhet och omfattande kontroll för att få sparandet till fil att fungera, men när det väl var klart fungerade det bra och klassen JBSound kunde skapas.

Det uppstod även problem med att få giltiga GPS-positioner från GPS-enheten. Detta problem berodde på att kontakten med GPS-enheten öppnades och stängdes för varje position som hämtades. GPS-enheten hinner då inte kalkylera en giltig position efter att kontakten öppnas till det att applikationen frågar efter en position. Detta orsakade att GPS-enheten alltid returnerade ogiltiga positioner. Problemet löstes genom att låta kontakten med GPS-enheten pågå under hela den tid GPS:en är startad.

Hanteringen av bildtagning var mycket enkel. Inga större problem uppstod vid skapandet av funktionen CameraCapture som hanterar fotograferingen.

Mycket tid för inlärning och studerande av MFC bidrog till att skapandet av EXapp gick relativt snabbt och på grund av att klasserna JBGPS och JBSound samt funktionen CameraCapture var tillverkade tidigare var det enkelt att inkludera dessa i applikationen. Målet med projektet var ett så långt som möjligt funktionsdugligt inrapporteringsprogram och genom skapandet av EXapp-applikationen hoppas jag att mitt arbete kommer att kunna vara till nytta.

4.2 Vidareutveckling

I framtiden kommer applikationer som är skapade för Windows Mobile 6.5 eller äldre att bli allt mindre användbara. Detta beror på att Windows Mobile 7 inte kommer att ha stöd för applikationer som är skapade till tidigare versioner av Windows Mobile. Det kan ta lång tid innan Windows Mobile 6.5 och äldre versioner försvinner helt, men vid vidareutveckling kan det vara fördelaktigt om en version av Windows Mobile 7 eller något annat nyare operativsystem skapas. På så sätt behöver kunderna inte köpa gamla produkter för att få applikationen att fungera.

Ytterligare funktionalitet som kan läggas in i applikationen kan vara automatisk överföring av sparade ljudfiler och bildfiler till en server. Detta skulle medföra att användaren aldrig behöver överföra filerna på egen hand.

Även verifiering av ljudmeddelanden och bilder kan vara en värdefull funktion, vilket skulle innebära att användaren inte behöver gå in i filutforskaren för att kontrollera ljud- och bildfiler. Verifieringen måste dock genomföras på ett trafiksäkert sätt.

4.3 Reflektioner

Arbetet med det här projektet var mycket intressant och lärorikt. Speciellt när det gäller programmering i C++ medförde det betydligt ökade kunskaper. Bland annat Windows Mobile-programmering och MFC var helt nya områden för mig och en utmaning att sätta sig in i. Det var också en värdefull erfarenhet att få jobba med en så pass stor uppgift ute i ”verkligheten”.

Det var synd att resultatet inte blev ett tillägg till MoITR. Det hade varit spännande att se MoITR under körning men eftersom uppdatering krävdes var detta inte möjligt.

Tidsmässigt fungerade det mesta mycket bra. Undersökningen av MoITR och olika försök för att få igång det var det enda som tog längre tid än planerat. På Logica’s kontor i Borlänge hade jag tillgång till all utrustning och information som jag behövde.

5. Ordlista

.NET:	Ramverk som innehåller ett stort bibliotek. Det ger användaren möjlighet att skapa Windows-applikationer med olika programmeringsspråk som har .NET-stöd.
Android:	Operativsystem och mjukvaruplattform för mobiltelefoner.
Bluetooth:	Standard som har tagits fram för trådlös kommunikation mellan olika enheter.
C++:	Programmeringsspråk
Flashminne:	Minne som det går att lagra data på. Det är inte flyktigt vilket betyder att data inte går förlorad även om strömmen försvinner.
Funktion:	Del av ett datorprogram som kan anropas för att utföra en viss uppgift.
Google Earth:	Program som visar geografiinformation på ett virtuellt jordklot.
GPS:	Global Positioning System, satellitsystem som är till för positionering.
Javascript:	Skriptspråk som används framför allt med webbläsare.
Klass:	Programkod som samlar en mängd relaterade attribut och funktioner.
Kompilator:	Datorprogram som översätter programkod från ett högnivåspråk till ett lågnivåspråk, oftast till maskinkod.
MFC:	Microsoft Foundation Classes är ett C++-bibliotek.
Programbibliotek:	Samling funktioner som kan inkluderas och användas vid utveckling av program.
RAM:	Random Access Memory. Ett minne som oftast används som arbetsminne för att lagra temporära processer, är flyktigt vilket innebär att data försvinner när strömmen försvinner.

Referens:	Hänvisning till en variabel eller ett annat objekt i ett program.
ROM:	Read Only Memory. Liknande RAM-minne men det går endast att läsa data från det och inte skriva till det.
Parameter:	Variabeltyp som skickas med i ett funktionsanrop.
Symbian:	Operativsystem avsett främst för handdatorer och mobiltelefoner.
Tråd:	Exekveringsenhet inom en process i en dator.
WAVE:	Filformat som är skapat av Microsoft och IBM, används för att lagra ljuddata.

6. Referensförteckning

6.1 Litteraturreferenser

[L1] Niittula Tommy (2005), *Windows Mobile-programmering.* Studentlitteratur, Lund. ISBN: 9144036183

6.2 Internetreferens

[I1] Information om Windows CE.
< http://seditaville.com/academic/summary/The History of Microsoft Mobile OS(Ver 2).doc > 2010-04-26

[I2] Historien om Windows Mobile.
< http://www.brighthub.com/computing/windows-platform/articles/1295.aspx > 2010-04-26

[I3] Information om Windows Mobile 6.
< http://www.brighthub.com/computing/windows-platform/articles/1296.aspx > 2010-04-26

[I4] Information om Windows Mobile 7.
< http://www.tekniken.nu/nyheter_och_trender/windows_mobile_7_series > 2010-04-26

[I5] Historien och information om Windows Mobile.
< http://www.notebooks.com/2010/04/12/a-brief-history-of-windows-mobile/ > 2010-04-26

[I6] Hemsida med information om GapiDraw.
< http://www.gapidraw.com/index.php > 2010-04-26

[I7] GapiDraw, version ändringsinformation.
< http://www.gapidraw.com/docs/gapidraw/releasehistory.htm > 2010-04-26

[I8] Information om hur man kan använda Smart Device Framwork. Kan kräva registrering.
< http://www.devx.com/wireless/Article/36134 > 2010-04-26

[I9] Information om hur wave-filer är uppbyggda.
< http://msdn.microsoft.com/en-us/library/aa446573.aspx > 2010-04-26

[I10] HTC Touch Diamond 2, specifikation.
< http://www.htc.com/www/product/touchdiamond2/specification.html > 2010-04-26

[I11] Exempel på hur man kan spela in ljud med C++ för Windowsapplikationer.
< http://www.codeproject.com/KB/winsdk/SoundRecord.aspx > 2010-04-26

[I12] MSDN forumpost om mmio.
< http://social.msdn.microsoft.com/Forums/en-US/vssmartdevicesnative/thread/4f482803-6c81-4b75-9f1b-5d08834be9c9> 2010-04-26

[I13] Information om strukturen av uppbyggnaden av wave-filer och annat.
< http://www.codeguru.com/cpp/g-m/multimedia/audio/article.php/c4739 > 2010-04-26

[I14] Information om RIFF-strukturen.
< http://sharkysoft.com/archive/lava/docs/javadocs/lava/riff/wave/doc-files/riffwave-content.htm > 2010-04-26

[I15] Filhantering för Pocket PC.
< http://msdn.microsoft.com/en-us/library/ms838343.aspx > 2010-04-26

[I16] Komma igång med Google Maps.
<http://maps.google.se/support/bin/static.py?page=guide.cs&guide=21670&topic=21676&answer=144347 > 2010-04-28

[I17] Film som visar hur Google Maps används till mobiltelefoner.
< http://www.youtube.com/watch?v=wWNa5b9lMRY > 2010-04-28

[I18] Information om Google Maps API.
< http://code.google.com/intl/sv-SE/apis/maps/index.html > 2010-04-28

[I19] Information om Google Maps API-premier.
< http://code.google.com/intl/sv-SE/apis/maps/documentation/premier/guide.html > 2010-04-28

[I20] Information om Windows Phone 7 Series.
< http://anandtech.com/show/2969/windows-phone-7-series-the-anandtech-guide > 2010-04-29

[I21] Information om MyMaps applikation till Andriod-telefoner.
< http://googlemobile.blogspot.com/2008/12/your-maps-in-your-hands-for-holidays.html > 2010-04-29

[I22] SHCameraCapture-funktionen.
< http://msdn.microsoft.com/en-us/library/Aa454996 > 2010-04-29

[I23] NMEA-dokumentation.
< http://www.gpsinformation.org/dale/nmea.htm > 2010-05-07

[I24] GPS Intermedia Driver-dokumentation.
< http://msdn.microsoft.com/en-us/library/bb202086.aspx > 2010-05-07

[I25] Hur man använder GPS Intermedia Driver-funktioner.
< http://social.msdn.microsoft.com/forums/en-US/vssmartdevicesnative/thread/bc697ecc-7362-4fc8-8577-73cef0f92d82 > 2010-05-07

[I26] MFC-dokumentation
< http://msdn.microsoft.com/en-us/library/d06h2x6e%28v=VS.80%29.aspx> 2010-05-07

[I27] Koordinatkonverterare
< http://www.fcc.gov/mb/audio/bickel/DDDMMSS-decimal.html > 2010-05-10

[I28] Information om waveform audio-funktioner.
< http://msdn.microsoft.com/en-us/library/aa909811%28v=MSDN.10%29.aspx > 2010-05-12

[I29] Information om WAVEFORMATEX-strukturen.
< http://msdn.microsoft.com/en-us/library/aa908934%28v=MSDN.10%29.aspx > 2010-05-12

[I30] Information om 010 Editor < http://www.sweetscape.com/010editor/ > 2010-05-21

[I31] Information om Bzip2. < http://www.bzip.org/ > 2010-05-21

[I32] Information om Zlib. < http://www.zlib.net/ > 2010-05-21

[I33] Information om SQLite. < http://www.sqlite.org/ > 2010-05-21

6.3 Bildreferenser

[B1] Bilden är hämtad från prisjakts hemsida.
< http://www.prisjakt.nu/produkt.php?p=402308 > 2010-04-26

[B2] Bilden är skapad av Johan Björk.

[B3] Bilden är hämtad från MSDN-dokumentation med vissa redigeringar av Johan Björk.
< http://msdn.microsoft.com/en-us/library/aa446573.aspx > 2010-05-12

[B4] Bilden är en skärmdump av 010 Editor med en .wav fil öppen och en wave-template.

[B5] Skärmdump av applikationen EXapp.

[B6] Skärmdump av applikationen EXapp.

[B7] Klassdiagram av en del av applikationen EXapp skapad av Johan Björk.

[B8] Skärmdump av Google Earth med tio positioner placerade.

[B9] Skärmdump av en testkarta som har skapats på
< http://maps.google.com/ > 2010-04-28

6.4 Tabellreferenser

[T1] Tabellen är hämtad från
< http://en.wikipedia.org/wiki/Windows_Mobile > 2010-05-12

[T2] Tabellen är skapad av Johan Björk.

Bilagor

Bilaga A, HTC Touch Diamond 2, specifikation

Specifikationen är hämtad från HTC:s hemsida 2010-05-16.

Processor	Qualcomm® MSM7200A™, 528 MHz
Operating System	Windows Mobile® 6.1 Professional
Memory	ROM: 512 MB RAM: 288 MB
Dimensions	107.85 X 53.1 X 13.7 mm (4.25 X 2.09 X 0.54 inches)
Weight	117.5 grams (4.15 ounces) with battery
Display	3.2-inch TFT-LCD touch-sensitive screen with 480 X 800 WVGA resolution
Network	HSDPA/WCDMA: • Europe/Asia: 900/2100 MHz • Up to 2 Mbps up-link and 7.2 Mbps down-link speeds Quad-band GSM/GPRS/EDGE: • Europe/Asia: 850/900/1800/1900 MHz (Band frequency, HSUPA availability, and data speed are operator dependent.)
Device Control	TouchFLO™ 3D Zoom bar
GPS	Internal GPS antenna
Connectivity	Bluetooth® 2.0 with Enhanced Data Rate and A2DP for wireless stereo headsets Wi-Fi®: IEEE 802.11 b/g HTC ExtUSB™ (11-pin mini-USB 2.0 and audio jack in one)
Camera	Main camera: 5.0 megapixel color camera with auto focus Second camera: VGA CMOS color camera
Audio supported formats	AAC, AAC+, eAAC+, AMR-NB, AMR-WB, QCP, MP3, WMA, WAV, MIDI, M4A
Video supported formats	WMV, ASF, MP4, 3GP, 3G2, M4V, AVI

Battery	Rechargeable Lithium-ion battery Capacity: 1100 mAh Talk time: • Up to 300 minutes for WCDMA • Up to 340 minutes for GSM Standby time: • Up to 500 hours for WCDMA • Up to 360 hours for GSM Video call time: Up to 150 minutes (The above are subject to network and phone usage.)
Expansion Slot	microSD™ memory card (SD 2.0 compatible)
AC Adapter	Voltage range/frequency: 100 ~ 240V AC, 50/60 Hz DC output: 5V and 1A
Special Features	FM Radio, G-Sensor

Bilaga B, Programkod för klassen JBSound

JBSound.h

```
// JBSound.h : header file for JBSound class
// Author: Johan Björk
// Date: 2010-05-06

#pragma once

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' before including this
file for PCH"
#endif

#include "resourceppc.h"
#include <MMSystem.h>
#define MAX_BUFFERS   3

class JBSound
{
private:

		VOID OpenDevice();
		VOID CloseDevice();
		VOID PrepareBuffers();
		VOID UnPrepareBuffers();
		UINT FillDevices();
		VOID prepareWaveFile(CString filename,
WAVEFORMATEX m_stWFEX);

		BOOL m_bRun;
		HWAVEIN m_hWaveIn;
		WAVEFORMATEX m_stWFEX;
		WAVEHDR m_stWHDR[MAX_BUFFERS];
		HANDLE m_hThread;
		CString m_csErrorText;
		CFile myFile;
		int totalSize;
		int totalDataSize;
		CString filename;

// Construction
public:
		JBSound(); // standard constructor
		~JBSound(); // destructor

public:
		VOID StartRecording();
		VOID ProcessHeader(WAVEHDR * pHdr);
		VOID JBSound::SetFormat(int channels, int
sampleSec, int bitsSample, WORD format);
		VOID JBSound::Start(CString name);
		DWORD Stop();
};
```

JBSound.cpp

```
/* A sound recording class. Use Start() and Stop() to start
and stop the recording.
The sound is automaticly saved to the filename and path
provided as a parameter to Start().
You can change the sound format by calling SetFormat
Author: Johan Björk
Date: 2010-05-06
*/

#include "stdafx.h"
#include "JBSound.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// JBSound construction
JBSound::JBSound()
{
            m_bRun=FALSE;
            m_hThread=NULL;
            m_hWaveIn=NULL;
            filename = L"";
            ZeroMemory(&m_stWFEX,sizeof(WAVEFORMATEX));
            ZeroMemory(m_stWHDR,MAX_BUFFERS*sizeof(WAVEHDR));

            // Set default sound format values
            m_stWFEX.nChannels = 1;
            m_stWFEX.nSamplesPerSec=11025;
            m_stWFEX.wBitsPerSample = 8;
            m_stWFEX.wFormatTag=WAVE_FORMAT_PCM;
            m_stWFEX.nBlockAlign=m_stWFEX.nChannels*m_stWFEX.w
BitsPerSample/8;
            m_stWFEX.nAvgBytesPerSec=m_stWFEX.nSamplesPerSec*m
_stWFEX.nBlockAlign;
}

// JBSound Destruction
JBSound::~JBSound()
{
}

// Thread function that starts the recording
DWORD WINAPI ThFunc(LPVOID pDt)
{
            JBSound *pOb=(JBSound*)pDt;
            pOb->StartRecording();
            return 0;
}

// Function to start record, takes the name and path where
the wave file should be saved
// for example: Start(L"\\Application Data\\Blip4.wav")
VOID JBSound::Start(CString name)
{
            filename = name;
            ZeroMemory(m_stWHDR,MAX_BUFFERS*sizeof(WAVEHDR));
            m_bRun=TRUE;
            m_hThread=CreateThread(NULL,0,ThFunc,this,0,NULL);
```

```
}

// Function to stop record
DWORD JBSound::Stop()
{
            m_bRun=FALSE;
            while(m_hThread)
            {
                        Sleep(100);
            }
            return 0;
}

// this function recieves messages from the sound device and
checks them
void CALLBACK waveInProc(HWAVEIN hwi,UINT uMsg,DWORD
dwInstance,DWORD dwParam1,DWORD dwParam2)
{
            WAVEHDR *pHdr=NULL;
            switch(uMsg)
            {
                        case WIM_CLOSE:
                                    break;

                        case WIM_DATA:
                                    {
                                                JBSound
*pDlg=(JBSound*)dwInstance;
                                                pDlg-
>ProcessHeader((WAVEHDR *)dwParam1);
                                    }
                                    break;

                        case WIM_OPEN:
                                    break;

                        default:
                                    break;
            }
}

// Writes the buffer data to the wave file
VOID JBSound::ProcessHeader(WAVEHDR * pHdr)
{
            MMRESULT mRes=0;

            TRACE(L"%d",pHdr->dwUser); // TRACE is only used
in debugg version
            if(WHDR_DONE==(WHDR_DONE &pHdr->dwFlags))
            {
                        // Write the buffer data to the file
                        myFile.Write(pHdr->lpData,pHdr-
>dwBytesRecorded);

                        // Add the buffer to the sound device
again

            mRes=waveInAddBuffer(m_hWaveIn,pHdr,sizeof(WAVEHDR
));
                        if(mRes!=0)
```

```
                        // add the amount of bytes written to
the file size and data size variables
                        totalSize = totalSize + pHdr-
>dwBytesRecorded;
                        totalDataSize = totalDataSize + pHdr-
>dwBytesRecorded;
            }
}

// Starts the connection with the sound device and prepares
the wave file
VOID JBSound::OpenDevice()
{
            MMRESULT mRes=0;

            mRes=waveInOpen(&m_hWaveIn,WAVE_MAPPER,&m_stWFEX,(
DWORD_PTR)waveInProc,(DWORD_PTR)this,CALLBACK_FUNCTION);
            if(mRes!=MMSYSERR_NOERROR)
            {
                        if (mRes == MMSYSERR_ALLOCATED)

            AfxMessageBox(L"MMSYSERR_ALLOCATED");
                        if (mRes == MMSYSERR_BADDEVICEID)

            AfxMessageBox(L"MMSYSERR_BADDEVICEID");
                        if (mRes == MMSYSERR_NODRIVER)

            AfxMessageBox(L"MMSYSERR_NODRIVER");
                        if (mRes == MMSYSERR_ALLOCATED)

            AfxMessageBox(L"MMSYSERR_NOMEM");
                        if (mRes == WAVERR_BADFORMAT)

            AfxMessageBox(L"WAVERR_BADFORMAT");
                        throw m_csErrorText;
            }
            prepareWaveFile(filename, m_stWFEX);
}

// Opens or creates the wave file and writes all the header
and chunk information
VOID JBSound::prepareWaveFile(CString name, WAVEFORMATEX
m_stWFEX)
{
            myFile.Open(name, CFile::modeCreate |
CFile::modeWrite);
            char outBuf[5] = "RIFF";
            myFile.Write(outBuf, 4);
            int fileSize = 50000; // placeholder, will be
overwritten after recording is done
            myFile.Write(&fileSize, 4);
            char outWave[5]= "WAVE";
            myFile.Write(outWave, 4);
            char outFmt[5] = "fmt ";
            myFile.Write(outFmt, 4);
            int fmtSize = sizeof(&m_stWFEX) * 4;
            myFile.Write(&fmtSize, 4);
            myFile.Write(&m_stWFEX, sizeof(&m_stWFEX) * 4);
            char outData[5] = "data";
            myFile.Write(outData, 4);
```

```
            int dataSize = 45000; // placeholder, will be
overwritten after recording is done
            myFile.Write(&dataSize, 4);

            // Add number if bytes written to the total
filesize variable and set data size to 0
            totalSize = 4 + 4 + 4 + 4 + 4 + sizeof(&m_stWFEX)
* 4 + 4 + 4;
            totalDataSize = 0;
}

// Closes the connection with the sound device and writes the
final data size of the file to the file header
VOID JBSound::CloseDevice()
{
            MMRESULT mRes=0;

            if(m_hWaveIn)
            {
                        UnPrepareBuffers();
                        mRes=waveInClose(m_hWaveIn);

                        myFile.Seek(4,FILE_BEGIN); // go to 4
bytes into the file (this is where the total filesize is
stored)
                        myFile.Write(&totalSize,4);
                        int dataSizePos = 4 + 4 + 4 + 4 + 4 +
sizeof(&m_stWFEX) * 4 + 4;
                        myFile.Seek(dataSizePos,FILE_BEGIN); //
go to where the size of the sound data is stored
                        myFile.Write(&totalDataSize,4);
                        myFile.Close();
            }
            m_hWaveIn=NULL;
}

// starts the recording
VOID JBSound::StartRecording()
{
            MMRESULT mRes;

            try
            {
                        OpenDevice();
                        PrepareBuffers();
                        mRes=waveInStart(m_hWaveIn);
                        if(mRes!=0)
                        {
                                    throw m_csErrorText;
                        }
                        while(m_bRun)
                        {
                                    Sleep(100);
                        }
            }
            catch(PCHAR pErrorMsg)
            {
                        AfxMessageBox((LPCTSTR)pErrorMsg);
            }
            CloseDevice();
            CloseHandle(m_hThread);
```

```
            m_hThread=NULL;
            //SetStatus(L"Recording stopped...");
}

// Prepares the buffers and adds them to the sound device
queue
// The number of buffers can be changed by changing the
MAX_BUFFERS value in JBSound.h
VOID JBSound::PrepareBuffers()
{
            MMRESULT mRes=0;
            int nT1=0;

            for(nT1=0;nT1<MAX_BUFFERS;++nT1)
            {

            m_stWHDR[nT1].lpData=(LPSTR)HeapAlloc(GetProcessHe
ap(),8,m_stWFEX.nAvgBytesPerSec);

            m_stWHDR[nT1].dwBufferLength=m_stWFEX.nAvgBytesPer
Sec;
                        m_stWHDR[nT1].dwUser=nT1;

            mRes=waveInPrepareHeader(m_hWaveIn,&m_stWHDR[nT1],
sizeof(WAVEHDR));
                        if(mRes!=0)
                        {
                                    throw m_csErrorText;
                        }

            mRes=waveInAddBuffer(m_hWaveIn,&m_stWHDR[nT1],size
of(WAVEHDR));
                        if(mRes!=0)
                        {
                                    throw m_csErrorText;
                        }
            }
}

// removes the buffers from the sound device
VOID JBSound::UnPrepareBuffers()
{
            MMRESULT mRes=0;
            int nT1=0;

            if(m_hWaveIn)
            {
                        mRes=waveInStop(m_hWaveIn);
                        for(nT1=0;nT1<MAX_BUFFERS;++nT1)
                        {
                                    if(m_stWHDR[nT1].lpData)
                                    {

            mRes=waveInUnprepareHeader(m_hWaveIn,&m_stWHDR[nT1
],sizeof(WAVEHDR));

            HeapFree(GetProcessHeap(),0,m_stWHDR[nT1].lpData);

            ZeroMemory(&m_stWHDR[nT1],sizeof(WAVEHDR));
                                    }
                        }
```

```
            }
}

// Function to change the sound format. Must be done before
calling Start() if you want to change the format
// WORD format can be set to a standard microsoft waveform
audio type
// for example:
/*
WAVE_FORMAT_PCM
WAVE_FORMAT_IEEE_FLOAT
WAVE_FORMAT_EXTENSIBLE
WAVE_FORMAT_ADPCM
WAVE_FORMAT_XMA2
WAVE_FORMAT_WMAUDIO2
WAVE_FORMAT_WMAUDIO3
*/
VOID JBSound::SetFormat(int channels, int sampleSec, int
bitsSample, WORD format)
{
            m_stWFEX.nChannels = channels;
            m_stWFEX.nSamplesPerSec=sampleSec;
            m_stWFEX.wBitsPerSample = bitsSample;
            m_stWFEX.wFormatTag=format;
            m_stWFEX.nBlockAlign=m_stWFEX.nChannels*m_stWFEX.w
BitsPerSample/8;
            m_stWFEX.nAvgBytesPerSec=m_stWFEX.nSamplesPerSec*m
_stWFEX.nBlockAlign;
}
```

Bilaga C, Programkod för funktionen CameraCapture

```
// function to start the camera and take a picture
HRESULT CEXappDlg::CameraCapture(CString filename)//LPTSTR
filename)
{
    HRESULT         hResult;
    SHCAMERACAPTURE shcc;

    // Set the SHCAMERACAPTURE structure.
    ZeroMemory(&shcc, sizeof(shcc));
    shcc.cbSize             = sizeof(shcc);
    shcc.hwndOwner          = NULL;//hwndOwner;
    shcc.pszInitialDir      = TEXT("\\Application Data");
    shcc.pszDefaultFileName = filename;//TEXT("test.jpg");
    shcc.pszTitle           = TEXT("Kamera");
          shcc.StillQuality                =
CAMERACAPTURE_STILLQUALITY_NORMAL;
    //shcc.VideoTypes         =
CAMERACAPTURE_VIDEOTYPE_MESSAGING;
          shcc.VideoTypes                                =
CAMERACAPTURE_VIDEOTYPE_ALL;
    shcc.nResolutionWidth   = 0;
    shcc.nResolutionHeight  = 0;
    //shcc.nVideoTimeLimit    = 15;
    //shcc.Mode               =
CAMERACAPTURE_MODE_VIDEOWITHAUDIO;
          shcc.Mode                                      =
CAMERACAPTURE_MODE_STILL;

    // Display the Camera Capture dialog.
    hResult = SHCameraCapture(&shcc);

    // The next statements will execute only after the user
takes
    // a picture or video, or closes the Camera Capture
dialog.
          /*
    if (S_OK == hResult)
    {

    }*/

    return hResult;
}
```

Bilaga D, Programkod för klassen JBGPS

JBGPS.h

```
// JBSound.h : header file for the class JBGPS
// Author: Johan Björk
// Date: 2010-05-06

#pragma once
#include <gpsapi.h>

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' before including this
file for PCH"
#endif

class JBGPS
{
private:
		HANDLE m_hThread;
		bool m_bRun;

public:
		GPS_POSITION current_pos;
		int GPS_status;
		JBGPS();   // standard constructor
		~JBGPS();  // destructor

public:
		VOID JBGPS::Start();
		VOID JBGPS::Stop();
		VOID JBGPS::StartGPS();
};
```

JBGPS.cpp

```
/* A GPS class. Opens a connection to the GPS device and gets
the location every two seconds.
The position is saved in the public variable current_pos.
If no position was returned from the gps device GPS_status is
set to 0, if a position was returned it is set to 1
Author: Johan Björk
Date: 2010-05-06
*/

#include "stdafx.h"
#include "JBGPS.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Construction
JBGPS::JBGPS()
{
		m_bRun=FALSE;
		m_hThread=NULL;
		GPS_status=0;
}
```

```
//Destruction
JBGPS::~JBGPS()
{
            m_bRun = false;
}

// This function starts the gps thread
DWORD WINAPI ThreadFunc(LPVOID pDt)
{
            JBGPS *pOb=(JBGPS*)pDt;
            pOb->StartGPS();
            return 0;
}

// public function to start the gps, it creates the thread
VOID JBGPS::Start()
{
            m_bRun=TRUE;
            m_hThread=CreateThread(NULL,0,ThreadFunc,this,0,NU
LL);
}

// public function to stop the gps
VOID JBGPS::Stop()
{
            m_bRun=FALSE;
            while(m_hThread)
            {
                        Sleep(100);
            }
}

// This function opens the connection to the gps device, and
gets the location every two seconds.
// The location is saved in current_pos, if no location was
aquired GPS_status gets the value 0
// if a location was returned GPS_status is set to 1.
VOID JBGPS::StartGPS()
{
            MMRESULT mRes;
            HANDLE GPSopen;
            GPS_POSITION gps_pos;

            ZeroMemory(&gps_pos,sizeof(gps_pos));
            gps_pos.dwVersion = GPS_VERSION_1;
            gps_pos.dwSize = sizeof(gps_pos);

            // Opens the connection to the gps device
            GPSopen = GPSOpenDevice(NULL,NULL,NULL, 0 );

            try
            {
                        while (m_bRun)
                        {
                                    // ask the gps device for a
location
                                    mRes =
GPSGetPosition(GPSopen,&gps_pos,6000,0);
```

```
                                    if (mRes == ERROR_SUCCESS){
                                                current_pos =
gps_pos;
                                                if
(gps_pos.FixType != 0)
                                                {
            GPS_status = 1;
                                                }
                                                else if
(gps_pos.FixType == 0)
                                                {
            GPS_status = 0;
                                                }
                                    }
                                    else {
                                                DWORD error =
GetLastError();
                                    }
                                    Sleep(2000);
                        }
                        // close the connection to the gps
device
                        mRes = GPSCloseDevice(GPSopen);
            }
            catch(PCHAR pErrorMsg)
            {
                        AfxMessageBox((LPCTSTR)pErrorMsg);
            }

            m_hThread = NULL;
}
```