

Seislog for PocketPC

Hovedprosjekt ved Høgskolen i Bergen, Avdeling for ingeniørutdanning,
Linje for data, studieretning datateknikk



Odd Johan Berland
Frode Fjeldstad

Bergen, HiB/Institutt for den faste jords fysikk

15.06.2001



Høgskolen i Bergen Avdeling for Ingeniørutdanning

TITTELSIDE FOR HOVEDPROSJEKT

<i>Rapportens tittel:</i> Seislog for PocketPC	<i>Dato:</i> 15.06.2001
	<i>Rapportnummer:</i> 4
<i>Forfatter(e):</i> Odd Johan Berland Frode Fjeldstad	<i>Antall sider u/vedlegg:</i> 55
	<i>Antall sider vedlegg:</i> 7
<i>Studieretning:</i> Datateknikk	<i>Antall disketter:</i> 1 CD
<i>Vegleder ved studieretning:</i> Lars Edgard Berg	<i>Gradering:</i> Ingen
<i>Merknader:</i>	

<i>Oppdragsgiver:</i> Insitiutt for den faste jords fysikk Allégt. 41, 5007 Bergen	<i>Oppdragsgivers referanse:</i>
<i>Oppdragsgivers kontaktperson:</i> José Åsheim Ojeda (jose.ojeda@ifjf.uib.no)	<i>Telefon:</i> 55 58 87 80
<i>Sammendrag:</i> Oppgaven tar for seg problemstillinger ved porting av Seislog for Windows til Windows CE 3.0 og PocketPC. Hensikten med PocketPC versjonen er å gjøre målesystemene mer mobil og anvendelige i felt. Mindre vekt og lengre batterilevetid enn bærbar PC er viktig i denne sammenheng. Seislog for Windows er et program for logging av seismologiske instrument. Måledata leses kontinuerlig fra serieport (RS-232), tidfestes og lagres til disk. GPS kan benyttes til tidsfesting av data. Programmet kan konfigureres til å oppdage og registrere skjelv og innkommende måledata kan vises grafisk under kjøring. Alle hendelser i programmet logges i en loggfil. Måledata kan vises og analyseres med programmet SeisAn. Seislog for Windows er skrevet i Visual C++ og benytter SDK. Viktige deler av oppgaven er tilpasning av grafisk grensesnitt, innføring av Unicode og drøfting av forskjeller mellom Win32 og Windows CE API.	

Stikkord:

Windows CE	PocketPC	Seislog
------------	----------	---------

Høgskolen i Bergen, Avdeling for ingeniørutdanning

Postadresse: Postboks 7030, 5020 BERGEN

Tlf. 55 58 75 00

Fax 55 58 77 90

Besøksadresse: Nygårdsgaten 112, Bergen

E-post: post@hib.no

Hjemmeside: <http://www.hib.no>

Forord

Prosjektet tok for seg portingen av Seislog for Windows9x/NT/2000 til PocketPC. Underveis oppdaget vi mange forskjeller mellom Windows CE 3.0 og Win32 API. Window CE 3.0 er kun et subsett av Win32 API. Likhetene var imidlertid så store, at selve portings delen gikk fort. Forprosjektet startet i januar 2001, og ble avsluttet i april. Implementasjonen av programmet varte fra april til juni 2001.

Mesteparten av forprosjektet gikk med til å lese den gamle Seislog-koden. Den var på nesten 1 Mb ren kode, men koden var likevel lett å forstå. Øyvind Natvik ved Instituttet for den Faste Jords Fysikk har laget Seislog for Windows9X/NT/2000, og hans arbeid har gjort vår oppgave vesentlig lettere. Vi vil derfor takke Øyvind for god og oversiktlig kode, og for hans veiledning underveis i prosjektet. Videre vil vi takke vår oppdragsgiver José Aasheim Ojeda, for hans hjelp og støtte i arbeidet. Vår veileder ved Høgskolen i Bergen var Lars E. Berg, vi takker også han for samarbeidet.

Innhold

1. Innledning	5
1.1 Oppdragsgiver	5
1.2 Problemstilling	5
2. Definisjon av oppgaven	6
2.1 Kravspesifikasjon	6
3. Analyse	10
3.1 Prinsipp for jordskjelvmåling	10
3.2 Seislog for Windows	12
3.3 Windows CE	25
3.4 PocketPC	29
4. Vurdering og valg av verktøy	32
5. Implementering	33
5.1 Generelt om forskjeller mellom Win32 og Windows CE API	33
5.2 Unicode	34
5.3 Grensesnittet	38
5.4 Andre endringer	46
6. Testing	48
6.1 Testing og feilsøking på PocketPC	48
6.2 Testing av Seislog for PocketPC	49
7. Oppsummering	51
7.1 Tilbakeblikk	51
7.2 Videreutvikling av Seislog	52
Referanser	55

1. Innledning

1.1 Oppdragsgiver

Institutt for den faste jords fysikk (IFJF) består av 40 ansatte fordelt på vitenskapelige, tekniske og administrative stillinger og 36 hovedfag- og mastergradstudenter. IFJF tilbyr undervisning i 32 enkeltemner fordelt på de tre studieretningene paleomagnetisme-geomagnetisme, petroleumsgEOFysikk og seismologi.

Den faste jords fysikk omfatter studiet av jordens oppbygning, sammensetning og utvikling ved hjelp av fysiske metoder. Kartlegging av jordskorpen og de ressurser som er knyttet til den er av sentral betydning. IFJF har ansvar for jordskjelvstasjonene i Bergen. Målinger foretas på flere forskjellige punkter i Bergensområdet. Instituttet har et internasjonalt forskningsmiljø med flere utenlandske studenter og stipendiater.

1.2 Problemstilling

Seislog er et system for logging av seismologiske data utviklet på IFJF. Systemet finnes for ulike operativsystemer, blant annet QNX og Windows. Måledata leses kontinuerlig fra seismologiske instrumenter via serieporten (RS-232 grensesnitt), tidsstemples og lagres til fil i et ringbuffer. Ved å konfigurere noen parametre er programmet også i stand til å analysere data og oppdage skjelv. Informasjon om skjelv skrives i egne filer uavhengig av ringbufferet. Data fra Seislog kan analyseres og vises grafisk i programmet SeisAn, som også er utviklet på IFJF.

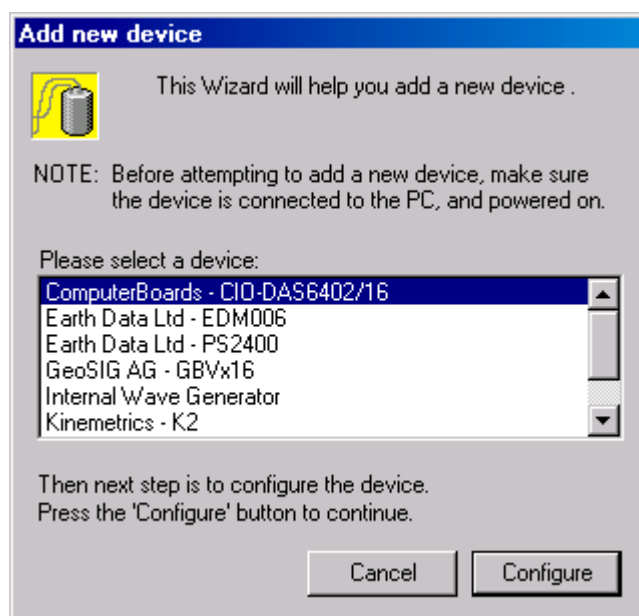
Oppgaven består i å porte *Seislog for Windows*, som er en av utgavene av Seislog, til Windows CE 3.0 for kjøring på PocketPC. Denne versjonen skal hete *Seislog for PocketPC*. Bruksområdet for Seislog for PocketPC vil blant annet være testing av bevegelsessensorene på målestasjonene. Tidligere har man benyttet bærbar PCer i dette arbeidet. Selv om disse er relativt små, er de unødvendig store i felt og har kort batterilevetid. Hensikten med å porte Seislog er å gjøre utstyret for logging mindre ressurskrevende og mer portabelt.

2. Definisjon av oppgaven

2.1 Kravspesifikasjon

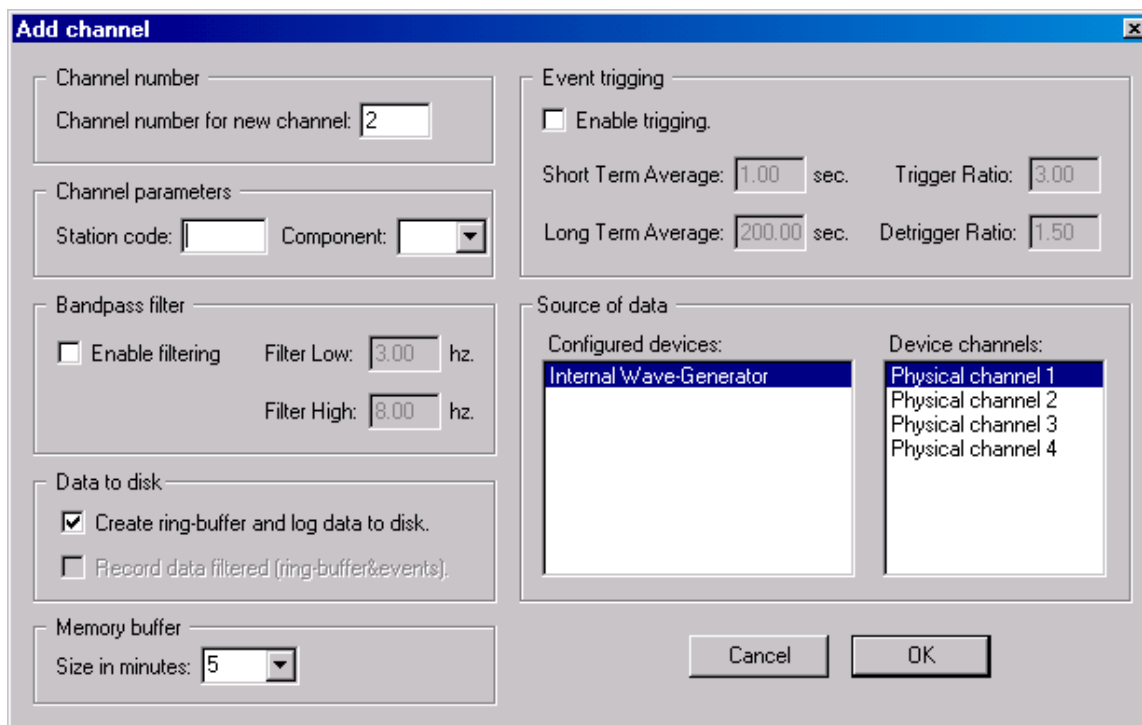
Vi ser her på de ulike aktivitetene i Seislog for Windows:

- **Starte og stoppe logging av data**
Logging startes og stoppes av bruker ved egne menyvalg
- **Administrere enheter (bevegelsessensorer)**
For at Seislog skal kunne kommunisere med instrumentene, må de konfigureres. Det er mulig å ha flere instrumenter konfigurert på forskjellige COM-porter.



Figur 1 Seislog støtter flere seismologiske instrumenter.

- **Administrere kanaler**
En bevegelsessensor kan måle bevegelser i flere retninger. For hver retning tilordnes en datakanal. I konfigurasjonen av denne kan en velge ulike parametre, blant annet om data fra kanalen skal lagres til disk og om kanalen skal trigge ved skjelv.

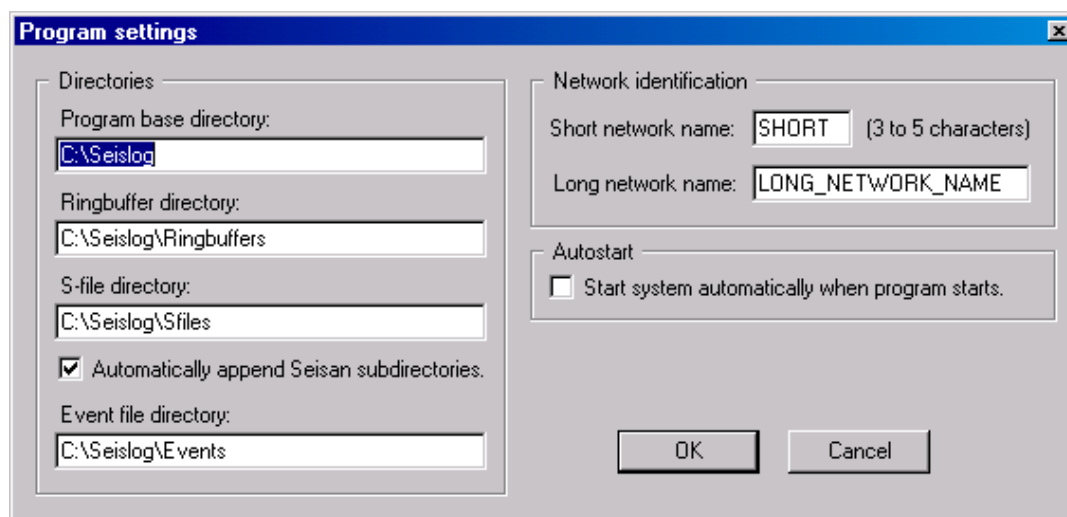


The 'Add channel' dialog box is used for configuring a new channel. It contains several sections: 'Channel number' with a text box for the channel number (set to 2); 'Channel parameters' with 'Station code' and 'Component' dropdowns; 'Bandpass filter' with 'Enable filtering' checkbox and 'Filter Low'/'Filter High' frequency inputs (set to 3.00 and 8.00 Hz); 'Data to disk' with 'Create ring-buffer and log data to disk' (checked) and 'Record data filtered (ring-buffer&events)' (unchecked) checkboxes; 'Memory buffer' with a 'Size in minutes' dropdown (set to 5); 'Event triggering' with 'Enable triggering' checkbox and 'Short Term Average'/'Long Term Average'/'Trigger Ratio'/'Detrigger Ratio' inputs; and 'Source of data' with 'Configured devices' (showing 'Internal Wave-Generator') and 'Device channels' (showing 'Physical channel 1' through 'Physical channel 4') lists. 'Cancel' and 'OK' buttons are at the bottom right.

Figur 2 Parametre for konfigurering av kanaler

- **Endre programinnstillinger**

Brukeren av Seislog må definere i hvilke kataloger på datamaskinen, data og systemkonfigurasjoner skal lagres.



The 'Program settings' dialog box is used for general program configuration. It contains two main sections: 'Directories' and 'Network identification'. The 'Directories' section includes text boxes for 'Program base directory' (C:\Seislog), 'Ringbuffer directory' (C:\Seislog\Ringbuffers), 'S-file directory' (C:\Seislog\Sfiles), and 'Event file directory' (C:\Seislog\Events), along with a checked 'Automatically append Seisan subdirectories' checkbox. The 'Network identification' section includes 'Short network name' (SHORT, 3 to 5 characters) and 'Long network name' (LONG_NETWORK_NAME) text boxes. An 'Autostart' section has an unchecked 'Start system automatically when program starts' checkbox. 'OK' and 'Cancel' buttons are at the bottom right.

Figur 3 Generelle programinnstillinger i Seislog

- **Konfigurere triggeret**

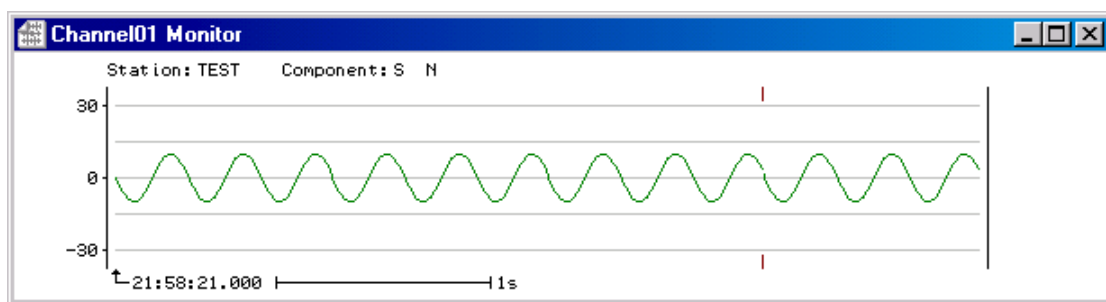
Her defineres de parametre som bestemmer om en hendelse skal registreres. To av disse parametrene er Short Time Average (STA) og Long Time Average (LTA). Når forholdet mellom STA og LTA blir stort nok utløses en hendelse, blir en "event-fil", blir skrevet til disk.

- **Simulere hendelser (skjelv)**

Hendelser kan simuleres. Manuelle simulerte hendelser kan brukes i sammenheng med testing, og samme informasjon skrives som ved virkelig hendelse.

- **Vis logging av seismologiske data grafisk på skjerm**

I et vindu i Seislog kan man grafisk observere de seismologiske målingene direkte. På den måte kan feil ved enheten raskt oppdages.



Figur 4 Måledata fra én kanal

- **Logg av meldinger fra programmet**

Under kjøring av programmet, sendes det en rekke meldinger om ulike hendelser som inntreffer. Disse blir skrevet til en logg fil, og vises på skjerm i et vindu. Eksempel på en melding kan være "system started", når logging av data starter.

- **Activity Monitor**

Activity Monitor er et vindu som viser ulike typer statusinformasjon til brukeren. Dato og tidspunkt, kanalnummer og -navn, skriving til disk, at det kommer inndata fra enhet osv. All denne informasjonen lagres for hver kanal. Figuren under viser aktivitets monitoren i Seislog.

Activity Monitor											
Ch	Station	Comp.	Date and Time	In/Disk/Trig/Snc	Network Trig. 1-5	Sample	STA and LTA	Offset (DC)	RB File	Filter (Low / High)	Device Binding
01	TEST	A N	12.06.2001 22:04:28.000			0	0.4	0.5	R00001	3.00 8.00	Internal Wave-Generator

Figur 5 Statusinformasjon for Seislog vises i Activity Monitor

Oppgaven går ut på å lage en versjon av ”Seislog for Windows”, som skal kjøres på Windows CE 3.0 til bruk på PocketPC. Seislog for PocketPC skal inneholde den samme funksjonene som Seislog for Windows har.

Det finnes flere forskjellige typer bevegelsessensorer som kan tilkobles Seislog. Det er derfor et mål at så mange som mulig av disse enhetene også kan tilkobles PocketPC versjonen av Seislog. Disse enhetene må testes over en lengre tidsperiode, for å sikre stabilitet.

Det grafiske brukergrensesnittet må lages på nytt på PocketPC versjonen av Seislog, men må likevel vært lett gjenkjennelig for tidligere brukere av Seislog for Windows 9X.

3. Analyse

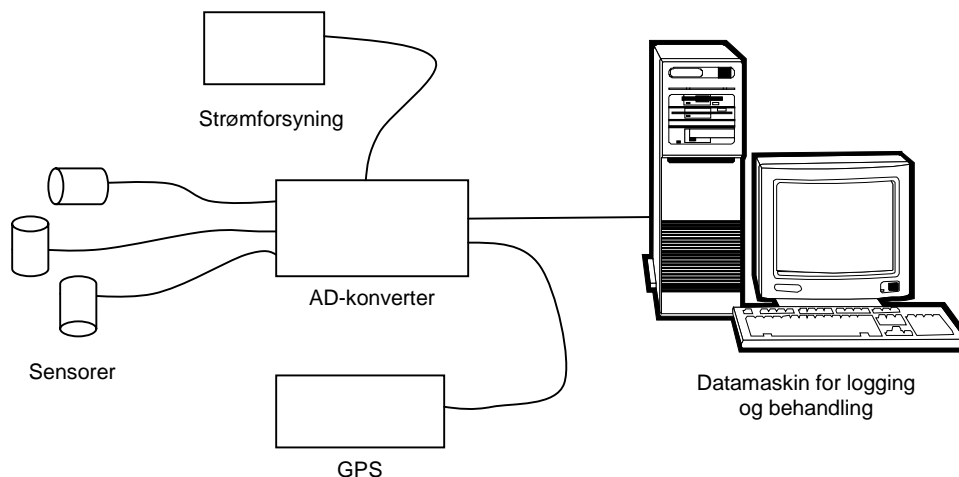
I denne delen tar vi for oss prinsipper for jordskjelvmåling og titter inn under panseret på Seislog for Windows. Vi ser deretter på konseptet PocketPC og operativsystemet Windows CE.

3.1 Prinsipper for jordskjelvmåling

Jordskjelvbølger blir oppdaget og registrert ved hjelp av seismometre eller seismografer, basert på meget enkle prinsipper. På en roterende trommel tegnes bevegelsene av en penn. Pennen er montert på et tungt lodd, som henger i en fjær eller en pendel festet på en ramme. Når bakken og rammen påvirkes av seismiske vibrasjoner, vil rammen settes i bevegelse. Loddet vil henge rolig på grunn av tregheten, og en relativ bevegelse oppstår mellom pennen og den roterende trommelen

En seismograf har vanligvis minst tre sensorer. To sensorer måler bevegelser i nord-sør og øst-vest retning, og en sensor måler vertikale bevegelser. Ved at de komponentretningen måles hver for seg, og ved å ha flere målestasjoner plassert på ulike steder på jorden, er det mulig å bestemme det nøyaktige brennpunktet til et hvert jordskjelv.

Moderne instrumenter er fremdeles basert på de første seismometrene. Hovedforskjellen idag er at loddet er magnetisk, og opphengt i en elektrisk spole som kan bevege seg fritt i én komponentretning. Relative bevegelser mellom magneten og spolen, induserer elektrisk strøm. De analoge signalene kan digitaliseres, og sendes til en datamaskin. I datamaskinen kan måledata behandles av et program, og lagres i digital form på harddisker eller magnetbånd. Alle målesignaler må tidsfestes, og det er viktig at ulike målesignaler har synkronisert klokke. For å sikre at data på ulike målestasjoner tidsfestes korrekt, benyttes tidsinformasjon fra satellitter, tatt imot gjennom GPS. Denne er enten koblet til AD-konverteren, eller direkte til datamaskinen. Dersom GPS er koblet til AD-konverter, mottar datamaskinen ferdig tidsstemplede data. Hvis ikke, er det opp til loggeprogrammet å synkronisere data mot en GPS.



Figur 6 Prinsippskisse for moderne seismometer

En prinsippskisse for et moderne seismometer er vist i figur 6. Skissen er basert på Earth Data PS2400 fra EarthData Ltd. Dette instrumentet har tre sensorer (én for hver komponentretning) og benytter en 24-bits AD-konverter med punktprøvingsrate opp til 250Hz. GPS kobles til gjennom en egen port på digitaliseringsenheten. Måledata sendes kontinuerlig til datamaskin gjennom serielt grensesnitt.

Sensorer, AD-konverter og GPS kan også lukkes inn i én enkelt enhet, som i GeoSig 316 fra GeoSys AG. Et bilde av dette instrumentet er vist i figur 7.



Figur 7 GeoSig 316 fra GeoSys AG.

3.2 Seislog for Windows

Seislog er delt opp i moduler, som hver har ansvar for spesifikke oppgaver i programmet. Med en modul menes en implementasjonsfil (*.cpp) og eventuell tilhørende header fil. Vi gir her i hovedsak en *overordnet* beskrivelse av hver modul sitt ansvar og oppgave. Seislog består av i alt over 20 moduler som til sammen utgjør nesten 1MB kildekode, og det vil før for langt å gi en mer fullstendig beskrivelse av hver modul i Seislog. I programmet finnes det prinsipper/løsninger som benyttes i flere moduler, som gjør at enkelte modulene har noen fellestrekk. Vi legger vekt på å få fram prinsippene og viser noe kode for å få frem enkelte av disse.

Mange av modulene i Seislog har egne tråder, og mange av trådene har egen meldingsløkke. Synkronisering mellom trådene utgjør en viktig del av programmet. Seislog er i høy grad hendelsesdrevet og mye av synkroniseringen i programmet skjer med events. En vanlig måte i Seislog å sende melding til en annen tråd, er å lage et event objekt, sende meldingen og så vente på at tråden som ble kalt har behandlet denne meldingen og frigjør event objektet. Men Seislog har også noen egne løsninger for synkronisering og meldingsutveksling. Hver av disse ser vi på i noen spesifikke moduler, men det er viktig å merke seg at de benyttes i flere av modulene.

Behandlingen av hvordan meldinger sendes i programmet er noe overfladisk av følgende årsaker: Alle API-kall for trådsynkronisering brukt i Seislog finnes også på PocketPC og måten meldinger er brukt i programmet gjør koden relativt lett å forstå. Vi har derfor ikke funnet det hensiktsmessig å dvele for mye ved disse detaljene.

MsgList

I Seislog er det ofte mer enn én tråd som er interessert i en gitt melding. Eksempel på en slik melding er TM_BUFFERUPDATED, som sendes fra lesertrådene for seismometrene når innbuffer er fylt opp. Både Channel, Monitor, Main Monitor og Monitor-modulene er interessert i denne meldingen. Funksjonen PostThreadMessage kan bare sende melding til én tråd, spesifisert i som ett av parametrene i kallet. I noen tilfeller er tråder heller ikke interessert i alle meldinger. Et abonnement-system for meldinger er laget for sende meldinger til spesifiserte tråder.

Systemet er implementert som en klasse MsgList, som har en kjedet liste med noder av typen MSGNODE som en av sine medlemmer. MSGNODE representerer en gitt melding som en gitt tråd eller vindusprosedyre abonnerer på.

```
struct MSGNODE {
    UINT    uiMessage;    // Meldingen som abonneres på
    DWORD   idThread;     // Tråd ID for abonnerende tråd eller
    HWND    hWnd;         // evt vindushandle hvis vindusprosedyre
    MSGNODE *next_node;   // Peker til neste node
};
```

Klassen MsgList har en peker til en slik liste som en av sine medlemmer, og definerer noen operasjoner på nodene:

```
class MsgList {
private:
    DWORD NumNodes;
    MSGNODE *PtrFirstNode, *PtrLastNode;
    VOID DestroyList(VOID);
public:
    MsgList::MsgList(void);
    MsgList::~~MsgList(void);
    BOOL Subscribe(SUBSCRIBE*);
    VOID UnSubscribe(SUBSCRIBE*);
    VOID Notify(UINT, WPARAM, LPARAM);
};
```

Funksjonen Subscribe starter et abonnement på en melding for en tråd eller vindusprosedyre, mens UnSubscribe stopper et abonnement. Parameter til disse funksjonene er en peker til en struct av typen SUBSCRIBE, som i hovedsak er lik MSGNODE strukturen, men mangler nodepekeren:

```
struct SUBSCRIBE {
    UINT uiMessage; // Meldingen som abonneres på
    DWORD idThread; // Hvis abonnenten er tråd,
    HWND hWnd;      // Hvis abonnenten er vindusprosedyre
};
```

En tråd som sender en eller flere melding som flere andre tråder er interessert i å motta, instansierer et objekt av typen MsgList når tråden opprettes. I meldingsløkken for tråden, må den behandle meldingene TM_SUBSCRIBE, og TM_UNSUBSCRIBE. Disse meldingen sendes fra tråder som ønsker å abonnere på en meldingen og en peker til en SUBSCRIBE struct må følge med meldingen. Tråden registrerer meldingen i databasen ved å kalle Subscribe. Når abonnerende tråder skal sendes en melding, kalles Notify med denne meldingen, og med de aktuelle parametrene. Notify løper gjennom listen og sender melding til alle tråder eller vinduer som er registrert med denne meldingen. Et utdrag fra meldingsløkken til en device-manager-tråd, er vist under:

```
// NotifyList er definert som:
// MsgList NotifyList;

// En tråd eller et vindu ønsker å abonnere på en melding

case TM_SUBSCRIBE:

// Med meldingen TM_SUBSCRIBE kommer også en peker
// til en MSG_PARAM som lParam i meldingen.
// Pekeren til SUBSCRIBE structen er vPointer-
// medlemmet i MSG_PARAM structen

if (ID.vPointer)
    if (!NotifyList.Subscribe((SUBSCRIBE*)ID.vPointer))
    {
        OD.bStatus=FALSE;
        OD.wStatus=3;
    }
    break;
```

```
// Avslutt abonnement på en melding for en tråd
case TM_UNSUBSCRIBE:
    if (ID.vPointer)
        NotifyList.UnSubscribe((SUBSCRIBE*)ID.vPointer);
    break;

// TM_BUFFERUPDATED sendes fra seriell-lese tråden
// når nye data er lest inn, og sendes her videre
// til alle interesserte tråder

case TM_BUFFERUPDATED:
    NotifyList.Notify((UINT)TM_BUFFERUPDATED,
                      (WPARAM)CurrentThreadID,
                      (LPARAM)BlockNr);

    break;
```

Process

Process er laget for å holde orden på kjørende tråder og avhengigheter mellom disse. Modulen består av én tråd, ProcKeeperThrd og en del hjelpefunksjoner. Process har en database over trådene, representert som en tabell med elementer av typen PROCESS_INFO. PROCESS_INFO er en struktur som er definert slik:

```
struct PROCESS_INFO {
    CHAR ProcType;                // Process type
    CHAR ProcName[65];            // Unique name of process.
    DWORD ThreadID;               // ThreadID of process.
    BYTE NumDependencies;         // Number of other processes
                                // this process depend on.
    DWORD Dependencies[DEPEND_TAB_SIZE]; // ThreadID of dependency
                                // processes.
};
```

ProcType er et tall som kan være en av følgende:

```
0 = No process
1 = DeviceManager
2 = Ringbuffer
3 = Monitor
4 = MainMonitor
5 = Channel,
6 = WriteEvent
7 = EventDetection
8 = SecondTicker.
```

ProcKeeperThrd har en egen meldingsløkke, og kommunikasjon med tråden skjer gjennom et sett av meldinger. For enkelte meldinger som sendes til ProcKeeperThrd, er det behov for å sende med mer informasjon enn hva som er plass i wParam og lParam i PostThreadMessage. Respons på enkelte meldinger krever også at informasjon sendes fra ProcKeeperThrd. En struct MSG_PARAM er laget for å overføre ekstra informasjon ved meldinger.

```
struct MSG_PARAM {
    BOOL    bStatus;
    WORD    wStatus;
    DWORD   wStatus;
    TCHAR   Status[256];
    LPVOID  vPointer;
    DWORD   idThread;
};
```

ProcKeeperThrd har to variable av denne typen; ID og OD. ID er meldingsparametre for innkommende melding og OD er for utgående melding. Tråder som sender melding til ProcKeeperThrd for å motta informasjon, lar lParam peke til en utfylt MSG_PARAM struct. På samme måte kan en motta informasjon ved å la lParam peke til en tom MSG_PARAM struct. Når ProcKeeperThrd mottar meldingen i form av en MSG struct, kopieres data som wParam peker til, over i ID. Utdata kopieres til inn i OD, og lParam settes til å peke til denne structen.

Noen meldinger som ProcKeeperThrd behandler er:

TM_REGISTER_PROCESS	Sendes fra alle tråder når de starter, utenom ProcKeeperThrd, main og systemlog. Kallende prosess registreres i databasen som en kjørende prosess. Alle prosesser som registreres, må gis et unikt navn. Dette navnet gis i cStatus-medlemmet i MSG_PARAM strukturen.
TM_GET_NUM_REGISTERED	Legger antall registrerte prosesser i OD.dwStatus
TM_UNREGISTER_PROCESS	Denne meldingen sendes fra alle prosesser (tråder) når de avslutter. Aktuell prosess fjernes fra databasen
TM_GET_PROCESS_BY_NAME	Kan brukes til å finne ut om visse prosesser kjører eller ikke. Hvis funnet vil prosessens tråd-ID bli returnert til kallende prosess. ID.cStatus inneholder navnet på prosessen hvis tråd-ID skal finnes.

For de fleste av meldingen over finnes det hjelpefunksjoner som letter kommunikasjon med prosessdatabasen. Funksjonene sender den meldingen til ProcKeeperThrd, venter på at tråden svarer, og returnerer med returverdi. Synkronisering skjer med events.

Her er et eksempel med funksjonen IsSystemRunning som returnerer TRUE dersom det finnes registrerte prosesser. Dette eksempelet er representativt for hvordan de fleste av de 11 hjelpefunksjonene kommuniserer med ProcKeeperThrd.

```

1  BOOL IsSystemRunning(VOID)
2  {
3      HANDLE hEvent;
4      MSG_PARAM MsgParam;
5      extern DWORD ProcKeeperThreadId;
6      hEvent=CreateEvent(NULL, FALSE, FALSE, NULL);
7      PostThreadMessage(ProcKeeperThreadId,
8                          TM_GET_NUM_REGISTERED,
9                          (WPARAM)hEvent, (LPARAM)&MsgParam);
10     WaitForSingleObject(hEvent, INFINITE);
11     CloseHandle(hEvent);
12     return (MsgParam.dwStatus>0);
13 }
```

I linje 6 opprettes en Auto-reset event. I linje 7 sendes meldingen `TM_GET_NUM_REGISTERED` til `ProcKeeperThread` for å hente antall registrerte prosesser. De to siste parametrene er de mest interessante. `wParam` er handle til event objektet som `ProcKeeperThread` vil frigjøre når den har behandlet meldingen, og `lParam` er peker til en `MSG_PARAM` struktur som vi vil ha utdata i. I `ProcKeeperThrd` skjer følgende:

```
...
MSG msg
MSG_PARAM OD, ID;
...
while (GetMessage(&msg, (HWND)NULL, 0, 0)) //Meldingsloop for tråden
{
    ...
    if (msg.lParam) // lParam er peker til MSG_PARAM
                    // struct'en fra IsSystemRunning
    ID=((MSG_PARAM*)msg.lParam); // Inkomende meldingsparam lagres i
                                // ID (MSG_PARAM struct)

    // Siden TM_GET_NUM_REGISTERED er en GET-melding brukes
    // ikke ID videre i dette eksempelet.
    // Initialiser return-meldingsparametre
    // i OD strukturen til default-verdier

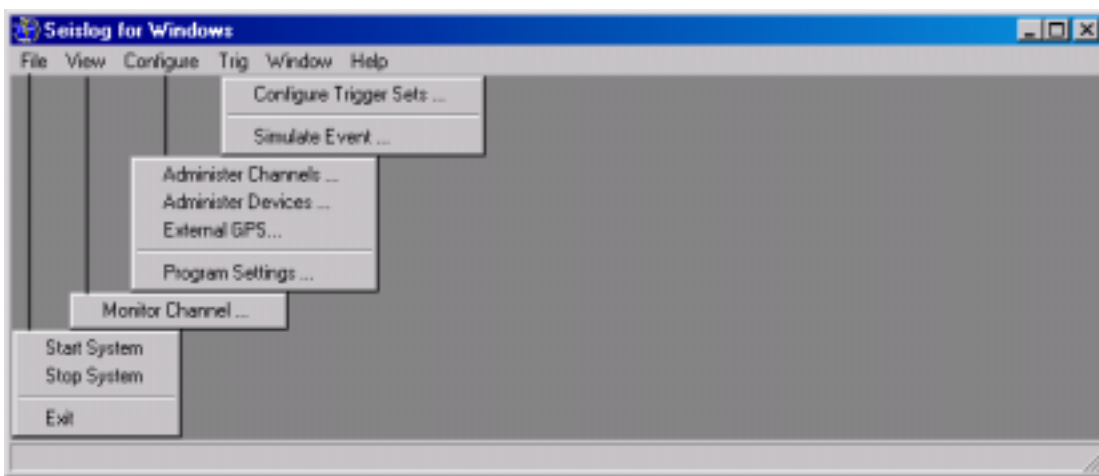
    OD.bStatus=TRUE; // Verdi skal returneres
    OD.idThread=CurrentThreadID; //ID for ProcKeeperThrd

    switch (msg.message) { //Behandle meldinger
        ...
        case TM_GET_NUM_REGISTERED:
            //Lagrer utdata i OD-objektet
            OD.dwStatus=(DWORD)NumRegProcesses;
            break;
        ...

    //Dersom det er utdata
    if (((MSG_PARAM*)msg.lParam)!=NULL && msg.wParam!=NULL)
    {
        *(MSG_PARAM*)msg.lParam=OD; //Sett utdatapeker
        SetEvent((HANDLE)msg.wParam);
    }
}
```


Seislog hovedmodul

- Inneholder en WinMain og en rekke globale variable og definisjoner av strukturer som benyttes i programmet.
- Sørger for at bare én instans av Seislog kjører om gangen. Dette gjøres ved å bruke en named Mutex, som opprettes ved oppstart av programmet og frigjøres ved avslutning.
- Leser og skriver registerverdier i Windows Registry. Registeret benyttes til å lagre programinstillinger som katalogstier og navn på målestasjon (network names). En dialogboks åpnes for å be om nye data dersom parametre mangler eller er feil.
- Registrerer vindusklasser for Hovedvindu, hendelseslogg, monitor-vinduer.
- Modulen inneholder vindusprosedyren for hovedvinduet og meldingsløkke for denne. Vindusprosedyren for hovedvinduet vil gjennom sin levetid i hovedsak behandle menyvalgene i Seislog.



Figur 8 Hovedvinduet i Seislog og de viktigste menyvalgene . Hendelsesloggen er ikke vist.

Men denne vindusprosedyren behandler også en viktig egendefinert melding, WM_STARTUP, som blir sendt idet programmet starter opp. Når denne meldingen mottas, skjer blant annet følgende:

- Vinduet for hendelseslogg opprettes og posisjoneres. Dette vinduet er et MDI child vindu.
- GPS tidsynkroniseringstråd startes (i clocksync modulen)
- Process-keeper tråd startes (i process-modulen)
- Data for konfigurerte enheter leses fra disk (devices modulen)
- Data for konfigurerte kanaler leses fra disk (channels modulen)
- Systemet starter logging hvis automatisk oppstart er valgt

Ticker

Ticker består av én tråd, TickerThrdProc, og sender meldingen TM_TICKER til alle registrerte tråder hvert sekund. Abbonement på meldingen håndteres med MsgList, og tråden håndterer kun meldingen TM_SUBSCRIBE, TM_UNSUBSCRIBE og TM_EXIT. Tråden startes fra modulen StartStop, men benyttes bare av modulen Trigger.

StartStop

Start og stopp av systemet håndteres med to tråder i denne modulen; StartSystemThrdProc og StopSystemThrdProc. Disse trådene opprettes av hhv funksjonene StartSystem og StopSystem, som igjen er kalt fra meldingsløkken for hovedvinduet som respons menyvalgene File | Start og File | Stop. StartSystemThrdProc utfører følgende før den terminierer (har ingen meldingsløkke)

- Gråer ut programmenyen
- Initialiserer ProcKeeper – tråden
- Starter en TickerThrdProc
- Registrerer alle enheter i Devices tabellen (global)
- Registrerer alle kanaler i Channels tabellen (global)
- Registrerer triggersets
- Laster alle registrerte kanaler vha LoadDevices (starter vider tråder for de spesifikke enhetsinstanser)
- Laster alle registrerte kanaler: kanaltrådene startes
- Laster triggers
- Starter kanaler
- Starter devices
- Starter triggers
- Starter Main Monitor vinduet ved å starte tråden MainMonThrdProc (MainMon – modulen)

StopSystemThrdProc tar systemet ned for landing, ved å stoppe alt som ble startet. Enkelte tråder sendes kun meldingen TM_EXIT; andre tråder medfører noe mer omfattende stopp: Siden PostThreadMessage ikke venter til meldingen er behandlet, kan en ikke være sikker på at alle tråder er stoppet. Etter at tråden for hovedvinduet (ikke vindusprosedyren) er sendt TM_EXIT, sendes den meldingen TM_DUMMY (en egendefinert melding som ikke behandles) kontinuerlig med 100 ms intervall. Så lenge PostThreadMessage returnerer sann, er tråden i stand til å håndtere meldinger, og er derfor ikke ferdig stoppet:

```
PostThreadMessage(MsgParam.dwStatus, TM_EXIT, 0, 0);  
while (PostThreadMessage(MsgParam.dwStatus, TM_DUMMY, 0, 0)) Sleep(100);
```

Enhetsdrivere

De ulike seismometrene representerer og sender måledata i forskjellig format. Ikke alle tidsstempler data, og antall fysiske kanaler kan variere mellom enhetene. Enhetsdriverne abstraherer bort forskjellene mellom enhetene, og presenterer data for resten av programmet i et felles format. Enhetsdriverne består av tre hoveddeler:

- Tråd for lesing fra serieport
- Tråd for å administrere enheten
- Dialogboksprosedyre for konfigurering av enheten

Når lesetråden startes, settes serieporten opp med riktige egenskaper for aktuell enhet. Deretter går den inn i en løkke der den leser fra serieport, legger innleste data et buffer og informerer administrasjon om status for lesingen gjennom meldinger. Administrasjonstråden kontrollerer lesingen, og kan avslutte lesningen. I så fall terminerer tråden. Tråden som administrerer enheten har egen meldingsløkke. Noen av meldingene som behandles er:

TM_START	Start lesing fra serieporten ved å opprette lesertråden
TM_STOP	Stopp lesing fra serieporten
TM_CONFIGURE	Åpne dialog for konfigurering av enheten. Meldingen er sendt fra funksjon i modulene Devices.
TM_SHOW_CONFIG	Viser hvilken konfigurasjon enheten er satt opp med.
TM_DEVICEREADER_NODATAINPUT	Sendes fra leser-tråden når data ikke lenger mottas
TM_DEVICEREADER_DATAINPUTOK	Sendes fra leser-tråden når data igjen mottas fra serieport etter avbrudd i lesing.
TM_BUFFERUPDATED	Når lesertråden har lest data i bufferet, sender den denne meldingen. Meldingen sendes så viderer til alle tråder som abonnerer den.

Enhetene er representert i programmet representert med strukturer av typen `DEVICE_CONFIG`. Medlemmene i strukturene avhengig av den enkelte enhet. Her er strukturen for GeoSys x16:

```
struct DEVICE_CONFIG {
    WORD DeviceID;           // MUST be '200'.
    BYTE NumChannels;        // Can be 1 or 3.
    TCHAR PortName[6];       // COM1, COM2 ...
    TCHAR CommSpeed[7];      // Communication speed.
    TCHAR Model[8];          // Can be 'GBV-116' or 'GBV-316'.
    BYTE SampleRate;         // Can be 25, 50, 100 or 200.
    DOUBLE TimeCorrection;    // Time correction (seconds).
    BOOL UsePCClock;         // If TRUE then time tagging is based on
                           // the PC clock.
    BOOL SyncPCClock;        // If TRUE then device driver will
                           // synchronize the system clock.
};
```

Konfigurasjonen av enheter lagres persistent på disk etter konfigurering av enhet. Ved oppstart av driveren leses konfigurering til minne.

Devices

Programmet har støtte for flere seismometre, og hver tilkoblet enhet kalles en enhetsinstans. Devices har ansvar for å administrere enhetsinstanser. I dette ligger å:

- Starte konfigurering av enhetsinstanser (selve konfigureringen av enheten gjøres i enhetsdriveren for den aktuelle enheten)
- Slette enhetsinstanser
- Starte driverne for de respektive enheter.
- Stoppe driverne.

For hver av oppgavene finnes det funksjoner og dialogboksprosedyrer. Modulen StartStop benytter funksjonen i denne modulen for å starte og stoppe systemet. Modulen har ingen tråder.

Channel

For hver kanal finnes det en instans av en tråd ChannelThrd som finnes i modulen Channel. Denne sørger for at data blir filtrert, lagt i ringbuffer i minne og skrevet til ringbuffer på disk. Modulen tar seg også av administrering av kanaler, dvs legge inn, modifisere og slette kanaler. Konfigurasjoner legges i en struktur av typen CHANNEL_CNF_FILE, som skrives til disk:

```
struct CHANNEL_CNF_FILE    // Structure describing the contents of the
'channel.cnf' file.
{
    TCHAR StationCode[5+1]; //Station code.
    TCHAR Component[4+1];   //Component.
    BOOL ApplyFilter;       //TRUE if bandpass filtering is requested.
    FLOAT FilterLow;        //Bandpass-filter parameter (Low cutoff freq.).
    FLOAT FilterHigh;       //Bandpass-filter parameter (High cutoff freq.).
    BOOL UseTriggering;     //TRUE if channel is going to trig events.
    FLOAT ShortTermAverage; //Trigger parameter;
    FLOAT LongTermAverage;  //Trigger parameter;
    FLOAT TriggerRatio;     //Trigger parameter;
    FLOAT DetriggerRatio;   //Trigger parameter;
    BOOL LogToDisk;         //TRUE if channel. should log data to disk.
    BOOL LogFiltered;       //TRUE if filtered data is to be logged to disk
    TCHAR DeviceName[65];   //Name of the configured device the ch. uses.
    BYTE DeviceChannel;     // The physical channel used on the device.
    BYTE BuffSizeInMinutes; // Size of memory-buffer measured in minutes.
};
```

Ved oppstart av tråden leses konfigurasjon til minne.

Ringbuffer

For hver kanal kan det settes opp et ringbuffer. Ringbufferet består av flere filer og har plass til data for et bestemt tidsrom. Når dette er fylt opp, skrives de eldste data over. Ringbufferet konfigureres med to parametre: Antall dager totalt og antall minutter måledata i hver fil. Ut fra disse parametrene beregnes antall filer.

Modulen Ringbuffer består av:

- En tråd for administrering av ringbufferet
- En funksjon som skriver ringbufferet til fil
- En funksjon som genererer filene til et nytt ringbuffer
- Dialogboksprosedyre for konfigurering av ringbuffer
- Hjelpefunksjoner.

Trigger

For å oppdage og registrere skjelv kan det settes opp triggerset på kanalene. Modulen trigger inneholder alt som trengs for å sette opp, endre triggerset og slette triggerset og oppdage skjelv. Data analyseres i en egen tråd, EventDetectionThrd. Måledata som inneholder skjelvet skrives til disk ved hjelp av WriteEvent. EventDetectionThrd behandler blant annet følgende meldinger .

TM_START	Start å detektere skjelv
TM_STOP	Stopp å detektere skjelv
TM_BUFFERUPDATED	Denne sendes fra enhetsdrivere når buffer er oppdatert, og EventDetectionThrd analyserer data når denne mottas.

WriteEvent

Når Trigger skal skrive data for et skjelv starter den tråden WriteEventThrdProci WriteEvent. På denne måten kan Trigger fortsette å detektere nye skjelv, samtidig som data skrives om forrige skjelv. Data skrives gjennom UnformPCFile, på samme måte som ved ringbuffer. WriteEvent inneholder også en del funksjoner for å skrive filene.

UnformPCFile

Seislog skriver datafiler i et eget "Seisan waveform" filformat. Dette er basert på "unformatted file" i Fortran. Dette betyr at filen inneholder ekstra tegn mellom hver blokk. Eksakt hvordan dette gjøres, er avhengig av kompilator. Programmet for å analysere data, Seisan, kan lese filer generert av programmer laget i ulike versjoner av Fortran. Seislog for Windows benytter étt av disse formatene, og UnformPCFile sørger for formatteringen. UnformPCFile er klasse, og deklartert på følgende måte:

```

#define UNICODE

class UnformPcFile {

public:
    UnformPcFile(WORD MaxBlockSize);
    ~UnformPcFile();
    BOOL Open(LPCTSTR);
    BOOL Write(DWORD, BYTE, BYTE*, DWORD*);
    BOOL WriteLeftovers(DWORD*);
    BOOL Seek(LONG);
    VOID Close();
    DWORD GetFilePointer();
    VOID Flush();

private:
    BOOL FileIsOpened;
    HANDLE FileHandle;
    DWORD FilePointer;
    BYTE NumLeftovers;
    BYTE LeftOverBytes[128];
    PBYTE pLocalBuffer;
};

```

// 1

Filer skrives blokkvis og innsetting av spesialtegn mellom blokker skjer transparent. Data som ikke opptar en hel blokk, legges i et buffer, LeftOverbytes. Før filen lukkes, skrives disse med WriteLeftoverbytes. UnformPCFile benytter en klasse FifoBuffer som implementerer en standard First-In –First-Out købuffer. En modul som skal skrive data i ”unformatted file”, instansierer et objekt av klassen UnformPCFile for hver fil. Både Ringbuffer og WriteEvent benytter UnformPCFile.

Monitor

Denne modulen har ansvaret for visning av måledata for en kanal i et vindu. For å sikre at vindusprosedyren MonitorWndProc alltid mottar meldinger så lenge vinduet er åpent, er meldingsløkken lagt i en egen tråd; MonitorThrd. MonitorThrd startes fra dialogboksprosedyren for ”Monitor Channels” dialogen.

MonitorThrd kjører så lenge Monitor vinduet er åpent og inneholder meldingsløkken for dette. Monitor vinduet er basert på SeislogMonitorClass og har MonitorWndProc som vindusprosedyre. Vinduet overleveres en peker til en struktur av typen MON_WND_DATA, som er utfyllt i tråden MonitorThrd før vinduet opprettes. Denne strukturen er definert slik:

```

struct MON_WND_DATA {
    DWORD ChThreadId;
    TCHAR Station[5+1];           // Navn på stasjon
    TCHAR Component[4+1];        // Måleretning
    VOID *pChBuffer;             // Peker til data-buffer
    WORD ClntHeight;
    WORD ClntWidth;
    WORD ClientAreaMinWidth;
    POINT MinTrackSize;
    WORD GraphWidthInPixels;
    WORD GraphWidthInBlocks;
};

```

```

WORD NextBlockToDraw;
LONG MaxVertScale;
DOUBLE VertScaleFactor;
LONG AutoMaxScale;
LONG SampleMax;
LONG SampleMin;
HFONT hFont;
HBITMAP hBitMap;
DEVMNDR_INTERF_STRUCT DS; // Informasjon om dataformat
SYSTEMTIME GraphTime;
BOOL ReadyToPaint;
BOOL bInitialCalc;
DOUBLE SampleAverage;
WORD SampleCount;
DOUBLE DeviceOffset;
MON_WND_CNF WndConfig;
HDC MemDC;
INT YPosLast;
};

```

Merk at étt av medlemmene er en peker til databuffer. MON_WND_CNF inneholder vinduskonfigurasjonene og benyttes for å skrive denne til disk:

```

struct MON_WND_CNF {
    BOOL AutoScaling;
    LONG UserScale;
    BOOL AutoOpen;
    WINDOWPLACEMENT WndPlacement; position.
};

```

De viktigste meldingen som håndteres i vindusprosedyren er WM_PAINT og TM_BUFFERUPDATED. Når WM_PAINT mottas, tegnes hele vinduet om igjen mens ved TM_BUFFERUPDATED tegnes bare den delen av vinduet som skal er endret. TM_BUFFERUPDATED kommer så ofte nye data leses inn, vanligvis hvert sekund. Monitor-vinduet tegner alltid data for hele slike perioder, og minst en periode om gangen. Det tegnes alltid over det som ble tegnet først, på en ”sirkulær” basis.

Library

Generelle hjelpefunksjoner som kan brukes i flere moduler, er samlet i et eget bibliotek.

- Behandling av filer og kataloger (søking, oppretting, lesing, skriving)
- Konvertering mellom og manipulering av datatyper.
- Tegning av bitmaps på skjerm
- Styring av menyer (GUI)
- Generering av tilfeldige tall
- Tegning av bitmaps

Dlgprocs

Prosedyrene for enkelte dialoger er samlet i DlgProcs. Dette gjelder:

- Creating Ringbuffer
- Configure External GPS
- Program Setting
- Monitor Channels

ClockSync

Dersom seismometerne ikke er utstyrt med GPS, kan data tidsstemples fra systemklokken i datamaskinen. Systemklokken kan så korrigeres gjennom en GPS koblet til en COM-port. ClockSync modulen utfører den siste delen, og består av to tråder; en GPS manager tråd, og en GPS reader tråd. GPS manager tråden starter samtidig Seislog, og går hele programmets levetid. Tråden starter opp reader tråden, og overvåker denne. Andre tråder kan gjennom manager tråden abonnere på meldinger som forteller om status for readertråden. GPS reader tråden leser kontinuerlig strenger fra COM-porten, og tolker den for å finne klokkeslett. Dersom avviket mellom lest tid og GPS tid overskrider en fastsatt grense (0.1 sek), settes systemtiden til GPS tid. Ved statusendringer informeres manager tråden. Status kan endres dersom GPS ikke lenger får kontakt med noen satellitter, eller dersom datamaskinen misser kontakt med GPS. Parametre for konfigurering av GPS settes i en dialogen "Configure External GPS". Dialogboksprosedyren denne dialogen finnes i modulen DlgProc.

3.3 Windows CE

Introduksjon til Windows CE

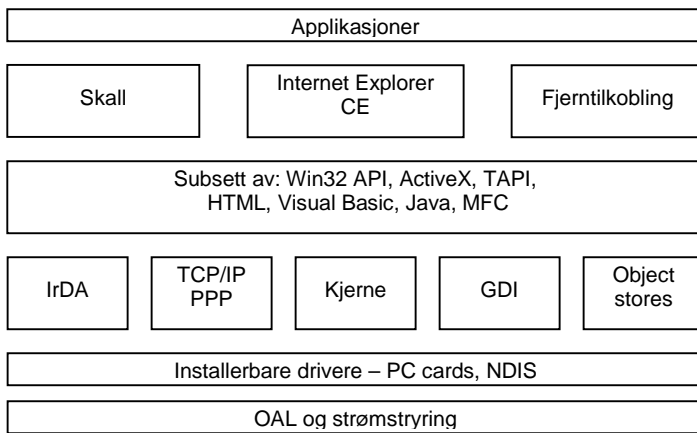
Windows CE ble lansert i første versjon i 1996, og er utviklet av Microsoft Corporation. I likhet med Windows NT og 2000, er det et 32-bits fleroppgave, flertrådet operativsystem, men det finnes noen viktige forskjeller. Den viktigste forskjellen er kanskje maskinvaren Windows CE er ment kjørt for, og at operativsystemet ligger i en ROM brikke. Operativsystemet er derfor også lite i forholdet til de andre utgavene av Windows. Årsakene til arkiturendringen er at Windows CE skal passe i enheter som tidligere ikke har benyttet programvare fra Microsoft.

For som med det meste Microsoft (og et hvert selskap med respekt for seg selv) foretar seg, handler Windows CE også om å erobre markedsandeler. I de siste årene har kravet til direkte tilgang til informasjon uansett hvor en befinner seg, økt betraktelig, særlig i forretningslivet. En rekke nye små fleksible og bærbare ”databanker” i alle slags varianter har oppstått, for å hjelpe til med å samle, håndtere og sende informasjon. I dette markedet ønsker Microsoft å kunne tilby Windows CE, som et felles operativsystem for alle enhetene. PocketPC er bare én slik enhet, og det finnes flere selskaper som produserer denne enheten, med forskjellig maskinvare. Vi skal senere se hvordan arkitekturen i Windows CE er laget for å tillate forskjellige maskinvareplattformer.

Windows CE har vokst frem fra flere separate prosjekter ved Microsoft, som startet tidlig på 1990-tallet. Microsoft har eksperimentert med flere tilnærminger til maskinvare og programvare, inkludert et forsøk på å redusere kjernen i Windows NT til en håndterlig størrelse, slik at den kunne plasseres i ROM. Også et objektorientert konsept ble forsøkt, men dette gikk for mye på tvers av Win32 API. Microsoft slo seg til slutt ned på en fullstendig nyutviklet kjerne som skulle være kompatibel med Win32 API.

Arkitektur

Windows CE er utviklet med Windows NT som grunnlag. I bunnen av operativsystemhierarkiet ligger maskinvare, som for eksempel kan være en PocketPC bygget rundt en MIPS, ARM eller SH3 prosesser. Over dette finnes OEM Abstraction Layer (OAL), som isolerer maskinvarespesifikke forskjeller i systemet fra den standard CE kjernen; akkurat som Hardware Abstraction Layer (HAL) separerer maskinvare og kjernen i Windows NT. OAL er også ansvarlig for skjermvisning, inn- og utenheter, kommunikasjon og strømstyring.



Figur 9 Den modulære oppbygningen av Windows CE minner mye om Windows NT

Kjernen i WindowsCE, CoreDLL, er ansvarlig for tråd og prosess "scheduling", program lasting av DLLer, det meste av minnehåndtering og avbrudd- og unntakshåndtering. Enkelte valgfrie komponenter inkluderes eller utelates, avhengig av bruksområde.

Over kjernen finnes det andre moduler som hver utfører spesifikke oppgaver, som filsystemhåndtering, skjermvisning og enhetsstyring. USER og GDI funksjonen som finnes i Windows 98 og NT er i CE

kombinert til en modul kalt GWES (Graphics, Windowing, and Events Subsystem). Her finnes funksjonalitet nødvendig for opprette og håndtere vinduer, dialogbokser, kontroller og ressurser som ikoner og menyer. Det finnes også funksjoner for å tegne tekst og grafikk, motta input fra bruker. Filsystem håndteres av en modul FILESYS, og DEVICE modulen, håndterer ikke helt utventet enheter. Enhver av disse modulene kan utelates, men dersom de er tilgjengelig, lastes de etter kjernen inn i et separat minneområde.

I tillegg til disse fundamentale modulene, kan det legges til et bredt utvalg av andre moduler som utfører forskjellige tilleggsoppgaver, som kommunikasjon og nettverkstilknytning, PC Card støtte, kryptering, OLE og COM. Hva som er tilgjengelig på en spesifikk enhet utover basis, er opp til produsenten av enheten.

Fleroppgavekjøring

Windows CE er et fleroppgave, flertrådet operativsystem. Flere prosesser kan kjøre samtidig og prosesser kan ha flere tråder. Operativsystem-kjernens "scheduler" sørger for at hver tråd får tilstrekkelig prosessortid. Størrelsen på tidsluken kan endres av utstyrsprodusent for optimalisering av ytelse. Windows CE benytter maskinvare for å holde prosessers minneområde fra hverandre.

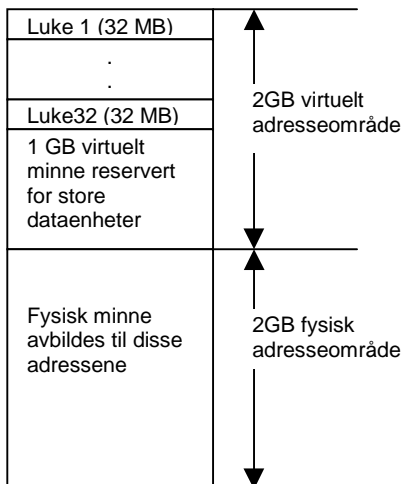
Maksimalt antall samtidige prosesser i Windows CE 3.0 er 32, men hver prosess kan starte en ny tråd så lenge det finnes ledig systemminne. Minst fire, men mer typisk sju eller åtte sysemprosesser startes når operativsystemet idet starter opp. Prioritet settes på tråder, og ikke på prosesser. Tråden med høyest prioritet kjører først, uansett hvilken prosess den kjører i. Tråder med samme prioritet, kjøres på "round-robin" basis. En prosess kan lage en "barneprosess", som kan arve objekt handles og andre tilgangsrettigheter fra foreldreprosessen.

Meldinger mellom tråder og operativsystemet eller tråder, håndteres av deres komponenter for håndtering av meldingskø i GWES. Operativsystemet stopper midlertidig tråder som ikke utfører noe og har tom meldingskø, for å spare CPU-sykler.

Minne og filhåndtering

I likhet med Windows 98 og NT, har CE et 32-bits flatt minneområde, med en 4GB teoretisk begrensning på fysisk og virtuelt minne. I Windows CE er fysisk minne delt opp i to deler, kalt "program memory" og "object store".

"Program Memory" er analogt til RAM i en desktop PC, og gir lagring for kjørende prosesser og deres tråder, og arbeidsminne for applikasjoner som kjører direkte fra ROM. Programmer som ligger i ROM, inkludert operativsystemet, trenger ikke å kopieres til "program memory" før det kjøres. Alle programmer kan dynamisk allokere minne for strenger, strukturer og andre data i "Program Memory".



Figur 10 Minneområder
i Windows CE

Windows CE bruker virtuelt minne for å håndtere og allokere programminne. Applikasjoner forespør om en blokk virtuelt minne og Windows CE avbilder det virtuelle minne til fysisk minne. Påfølgende minneallokeringer gir ikke nødvendigvis påfølgende fysiske minnelokasjoner.

Windows CE har totalt 4GB virtuelt adresseområde. Dette adresseområdet er splittet i to deler, der 2GB er reservert for maskinvaretilgang for operativsystemet, og 2GB som virtuelt delt adresseområde for applikasjoner. I dette delte adresseområde, reserverer Windows CE 33 "luker" (eng: slots) á 32MB for å kjøre prosesser. Hver prosess har en 32 MB luke for prosesskode, lastede DLLer, stabel for tråder og heap, og luke 0 holder alltid inneværende prosess. Resten av det delte virtuelle

minnet brukes av operativsystemet til blant annet "Memory Mapped Files" og store virtuelle allokeringer. Merk at Windows CE kan kjøre maksimalt 32 prosesser samtidig.

Virtuelt minne håndteres i sider ("pages") som har enten 1KB eller 4KB størrelse, avhengig av CPU. Windows CE markerer hver side som enten ledig, reservert eller opptatt.

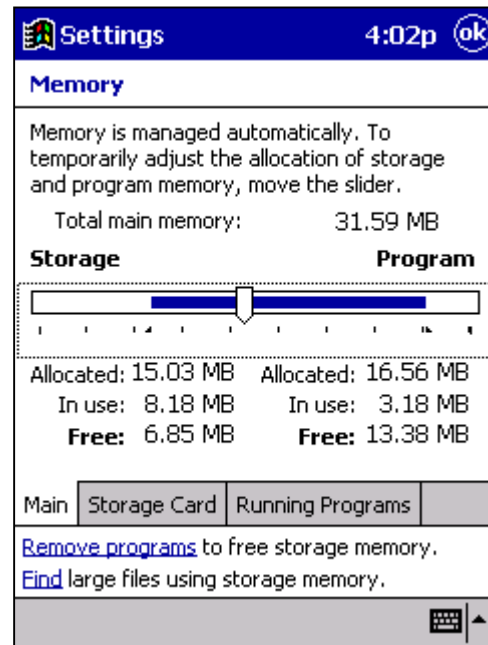
- En ledig side kan allokere til enhver applikasjon til enhver tid.
- En reservert side er holdt av for en applikasjon, men er ennå ikke avbildet til fysisk minne
- En opptatt side er allokert til fysisk minne og er i bruk av en applikasjon.

Windows CE allokere sider langs 64kB regioner.

”Object Store” har samme oppgave som en harddisk på en desktop PC. Her lagres programmer og datafiler når maskinen er slått av. Et backup batteri ivaretar innholdet dersom nivået i hovedbatteriet blir for lavt. Konseptuelt består object store av tre typer persistent lagring: filsystem, databaser og systemregisteret.

Databasemodellen i WindowsCE har flat struktur (såkalt flatfil database) og er optimalisert for liten og effektiv lagring. Dataoperasjoner er transaksjonsorienterte for å unngå tap av data; ved strømsvikt under endring i database, kan systemet tilbakespore til den siste korrekte tilstanden.

Filsystem og databaser trenger ikke nødvendigvis bare eksistere i object store, de kan også finnes i ROM, eller på eksterne enheter som Compact Flash kort. Data opprettes og hentes avhengig av lagringstype. Operativsystemet gir sømløs integrasjon mellom ROM baserte applikasjoner og RAM baserte data.



Figur 11 Minneinstillinger på PocketPC

Skillelinjen i systemminnet mellom ”object storage” og ”program memory” styres av Windows CE, men kan midlertidig endres av bruker. I Windows CE 3.0 er object store begrenset til totalt 256 MB, med 32MB maksimal størrelse for enkeltfiler. Maksimalt antall objekter (filer, databaser osv) er 4 millioner.

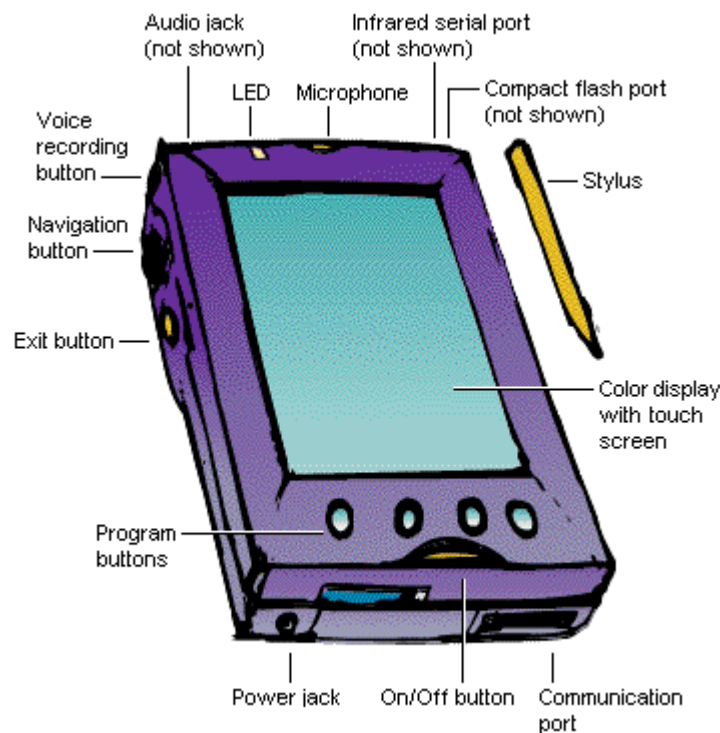
Kommunikasjon

Det finnes tre former for kommunikasjon i Windows CE:

- Seriell tilkobling, infrarød link med eller uten IrDA og modemer. Serielle enheter gis et COM port nummer og dataoverføring behandles som om systemet skriver til filer.
- **Nettverkstilkobling** gjennom f.eks. CompactFlash kort, som benytter Internett-standarder som HTTP, TCP/IP, FTP, og ICMP; Windows Sockets; NDIS, PPP og SLIP; eller RAS. Protokollstabelen starter på topp med WinInet for internett browsing, og WNet for tilgang til delte filer og utskrift på andre maskiner. WinInet er imidlertid synkron og støtter bare HTTP og FTP, mens WNet bare støtter Microsofts nettverkstilkobling.
- **TAPI**, et subsett av ”Telephony API” fra Win32.

3.4 PocketPC

Den neste illustrasjonen viser de vanligste maskinvarekomponentene en finner på PocketPC (figuren er hentet fra MSDN april 2001).



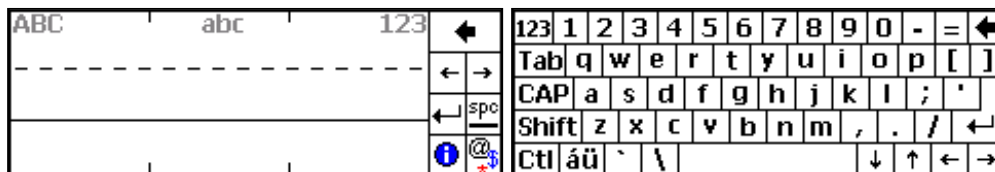
Figur 12 PocketPC

- **Trykkfølsom skjerm**

Skjermen er en LCD skjerm som er trykkfølsom. LCD skjermen er orientert som et portrett, med en oppløsning på 240 x 320 pixler. Denne oppløsningen og størrelsen gir brukeren et klart og brukervennlig grafisk grensesnitt. Et trykk med pennen på LCD skjermen sender en vindus melding WM_LBUTTONDOWN, den samme som finnes på desktop PCer når man trykker på venstre museknapp. Tilsvarende kan man holde pennen på LCD skjermen i 2 sek. da vil man fremtvinge en WM_RBUTTONDOWN melding, for høyre museknapp. Det er også mulighet for å dra elementer over skjermen med pennen. Det er en del begrensninger ved bruk av en penn i forhold til en mus. Oppdateringsraten for LCD skjermen er på minst 100 ganger per sekund.

- **Tastatur og penn**

En PocketPC har ikke et standard, fysisk tastatur. Man benytter i stedet en penn og et "input panel" for å taste inn tegn. Et "input panel" er et vanlig vindu på den trykkfølsomme skjermen. Det finnes to måter å skrive inn tegn på. Man kan få opp et tastatur lignende panel, der man trykker på bokstavene med pennen. En annen mulighet er en tegngjenkjenner. Da skriver man bokstavene med pennen, og PocketPCen gjenkjenner tegnet og skriver det på skjermen. Pennen er laget med en liten spiss som er ganske nøyaktig, uten å skade skjermen.



Figur 13 Vindu for tekstgjenkjenning (til venstre) og skjermtastatur (til høyre)

▪ Knapper

En PocketPC har noen få knapper. Disse kan trykkes på, holdes nede, eller brukes i kombinasjon med andre kontroller. Denne tabellen viser en oversikt over standard hvilke knapper en standard PocketPC er utrustet med:

Navigasjons kontroll	Beskrivelse
Av / på	Slå PocketPC av og på
Exit	Ikke støtte for
Utfør knapp	Tilsvarende ENTER knapp
Lyd opptak knapp	Aktiverer lyd opptaks program
Program knapp(er)	Starter et program (kan defineres)
Opp	Pil opp
Ned	Pil ned

▪ Audio input

Ikke alle PocketPCer støtter lydopptak, men for de som har lydopptak er det inkludert en innebygd mikrofon. Maskinvaren støtter 16-bits sampling på 8 kHz.

▪ Audio output

Det finnes en liten innebygd høyttaler som kan spille lyder i forbindelse med forskjellige aktiviteter i programmer (åpning, klikking, lukking og spille av lydfiler som .mp3 og .wav. Det finnes også et uttak for høretelefoner, eksterne høytalere osv.

▪ Utskrift

Det er ikke støtte for utskrift på PocketPC. Dette må utføres på en desktop PC.

▪ Strøm

Som for alle mobile enheter, er varigheten på batteriene viktig. En PocketPC har batterilevetid på opp til 15 timer på. I tillegg har den også et backup batteri, for å unngå tap av data om hovedbatteriet tømmes helt og ved uttak av hovedbatteri.

▪ Prosessor

Det finnes flere forskjellige prosessorer for PocketPC støtter Noen av disse er MIPS, SH3 og StrongARM. Et program må kompiles for hver av de ulike prosessorene

- **Minne**

Standard versjon av PocketPC enheter blir levert med minst 8MB ROM, og 8 MB RAM. En profesjonell versjon vil ha minst 12 MB ROM, og 8 MB RAM. Operativsystemet ligger i ROM. For å spare plass er mange komponenter i operativsystemet komprimert. Ved behov dekomprimerer operativsystemet komponenten, og legger den i den delen av minnet som benyttes til kjøring av programmer.

- **Innebygget serieport**

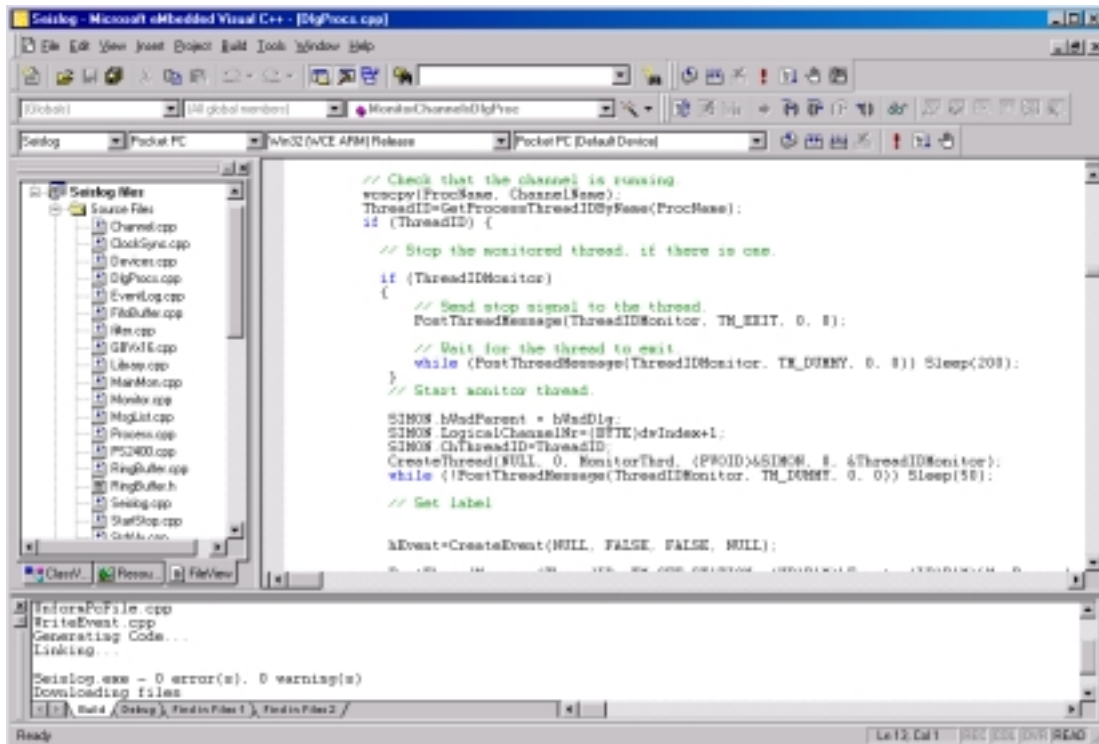
En PocketPC har en innebygd seriell port med 16550 UART. Baudraten kan variere mellom 9600 til 115200. Den samme porten benyttes også til USB.

- **Infrarød kommunikasjon seriell port**

En PocketPC har en seriell port, som overfører data via infrarødt lys. Den støtter standarden for Infrared Data Association (IrDA). PocketPC enheter kan kommunisere med andre PocketPCer, desktop PCer, eller andre Windows CE enheter.

4. Vurdering og valg av verktøy

Seislog for Windows er skrevet i C++ i Microsoft Visual Studio. Til utvikling av PocketPC versjon benyttet vi Microsoft eMbedded Visual C++, en del av pakken Microsoft eMbedded Visual Tools 3.0. eMbedded Visual Tools er i skrivende stund eneste tilgjengelige verktøy for utvikling av programmer for Windows CE 3.0. Programpakken kan lastes ned gratis på Internett.



Figur 14 Microsoft eMbedded Visual C++ 3.0

PocketPCer produseres i dag med tre forskjellige prosessortyper; ARM, MIPS og SH3, og eMbedded Visual Tools kan kompilere programmer for alle disse prosessortypene. Verktøyet har også store likhetstrekk med Microsoft Visual Studio, men benytter et subsett av MSDN til hjelpesystem. Innebygget i eMbedded Visual Tools finnes hjelpeprogrammene Remote Spy++, Remote Registry Editor, Remote Heap Walker, Remote Process Viewer, Remote Zoomin og Remote File Viewer.

5. Implementering

5.1 Generelt om forskjeller mellom Win32 og Windows CE API

Selv om Windows CE API er basert på Win32 API, er det noen viktige forskjeller mellom Windows CE API og Win32 API. Vi gir her en oversikt over de viktigste forskjellene, og går senere i detalj på noen av emnene:

- Bare Unicode strenger støttes i Windows CE API.
- Windows CE er et subsett av Win32 API. Noen Win32 funksjoner er ikke støttet, og ingen 16-bit Windows funksjoner er støttet. Funksjoner som ikke er støttet, må erstattes med alternative funksjoner eller en må benytte ”omveier”.
- Noen elementer i Win32 API som er støttet, har redusert funksjonalitet eller har fått annen betydning. Dette gjelder for eksempel WaitForMultipleObjects, som returnerer så fort ett av objektene går til signalisert tilstand. Den reduserte støtten vises også enkelte steder i parametre som ignoreres. For eksempel ignoreres peker til sikkerhetsattributter i CreateEvent.
- Windows CE API inneholder elementer spesifikt for dette APIet. De fleste av disse er i tilknytning til grensesnittet (skjerm og input panel) og styring av maskinvare.
- Noen Win32 funksjoner er erstattet med ekvivalente for Windows CE. For eksempel er verktøylinje og meny kombinert til en del kalt Command Bar, som har et eget API.
- Windows CE støtter de fleste strukturene i Win32, men enkelte medlemmer i strukturene er ikke i bruk. Enkelte medlemmer støtter heller ikke alle valgmuligheter.
- Noen meldinger, deriblant WM_* -meldinger er ikke støttet i WindowsCE. Noen ikke-støttede meldinger er modifisert for Windows CE, slik at de har annen betydning enn i Win32. Windows CE har også enkelte spesifikke meldinger som WM_HIBERNATE.
- Windows CE ingen C++ exceptions.

For å lette portingen av programmer til Windows CE, er noen funksjoner som ikke er støttet definert i makroer ved hjelp av andre funksjoner som støttes. Dette gjelder for eksempel funksjonen ZeroMemory (definert i winnt.h for PocketPC).

```
#define ZeroMemory(Destination,Length) memset((Destination),0,(Length))
```

5.2 Unicode

En viktig del av porting av programmer til Windows CE innebærer å gjøre programmet klar for Unicode, dersom dette ikke allerede er gjort. Denne delen av portingen av programmer fra Windows 9X/NT/2000 til CE, kan, og definitivt også bør gjøres i desktop PC versjonen, dersom denne versjonen ikke allerede er Unicode basert. Fordelen med dette er klar: Ved å videreutvikle direkte i desktop versjonen, i samme utviklingsverktøy og kompilator, fortsetter en med noe som allerede fungerer, og de eneste feilene en må konsentrere seg om er forbundet med Unicode. Feilsøking blir lettere og hele programmet kan testes i sin helhet før resten av portingen tar til. Med dette vil typisk en stor del av feilene være eliminert, før en engang har startet med å kompilere for Windows CE og PocketPC.

Introduksjon til Unicode

Unicode er en standard grunnlagt av Apple og Xerox i 1988. I 1991 ble det opprettet et konsortium for å utvikle Unicode. Dette består i dag av blant annet Apple, Compaq, Hewlett-Packard, IBM og Microsoft .

Unicode standarden definerer koder for representasjon av tegn i de fleste skriftspråk som finnes i verden i dag. Tegnene omfatter blant annet latinsk, gresk, arabisk, japanske, kinesiske og norske. I tillegg finnes matematisk og tekniske symboler symboler, piler, diakrater (for eksempel '~') osv. I alt har Unicode standarden koder for nær 39 000 tegn fra alle verdens alfabeter og symboler per dags dato.

Alle tegn i Unicode strenger er 16 bits verdier (opptar 2 bytes), og det finnes derfor mulighet for totalt 65 536 ulike kodeverdier. 18 000 kodeverdier er reservert for fremtidig bruk, mens 6 400 verdier kan benyttes av maskin- og programvareprodusenter for egne tegn og symboler. De første 127 kodeverdien i Unicode er ekvivalent med ASCII tegnsettet, og de påfølgende opp til 255 er Latin 1 tegnsettet.

Windows CE og Unicode

Windows CE er designet for maskiner med begrenset minne og lagringskapasitet. Samtidig skal det være et internasjonalt operativsystem, til bruk over hele verden. For å møte begge disse kravene, besluttet Microsoft å kún støtte Unicode i Windows CE API. Dersom ANSI skulle støttes i tillegg, ville dette blant annet øke størrelsen på operativsystemet. En bør merke seg at manglende støtte for ANSI strenger bare gjelder API-kall; C Run-time Library for Windows CE, støtter både Unicode og ANSI strenger.

Win32 funksjonsprototype og C run time library

Win32 funksjonsprototyper finnes i en generisk ANSI og Unicode versjon. Dokumentasjon gir generiske funksjonsprototyper som kan kompiles for å produsere enten Unicode eller ANSI prototyper. For eksempel er alle prototypene som er vist i følgende kode for funksjonen SetWindowText:

Generiske prototyper:

```
BOOL SetWindowText(HWND hwnd, LPCTSTR lpText );
```

Header filen gir det generiske funksjonsnavnet implementert som en makro:

```
#ifndef UNICODE
#define SetWindowText SetWindowTextW
#else
#define SetWindowText SetWindowTextA
#endif // !UNICODE
```

Preprossoren ekspanderer makroen til enten ANSI eller Unicode funksjonsnavn, avhengig av om UNICODE er definert eller ikke. Bokstaven A (ANSI) eller W (wide) er lagt til på slutten av det generiske funksjonsnavnen. Headerfilen gir så ANSI og Unicode funksjonsprototypene som i følgende eksempel

ANSI prototyper:

```
BOOL WINAPI SetWindowTextA(HWND hwnd, LPCSTR lpText);
```

Unicode prototyper:

```
BOOL WINAPI SetWindowTextW(HWND hwnd, LPCWSTR lpText);
```

I Windows CE er UNICODE (og _UNICODE) alltid definert, slik at alle makroer ekspanderer til Unicode versjonen. Dersom en kaller ANSI versjonen direkte i kode, vil den kompilere, men kan ikke linkes.

C Run-time Library for Windows CE har egne funksjoner for behandling av Unicode tekststrenger. Forskjellen i navnet på de fleste funksjonene, er at "str" i ANSI versjonen er byttet ut med "wcs", for "wide character string" i Unicode versjonen. Et Unicode tegn er representert som en signed short, men dette er vanligvis skjult gjennom en makro, for eksempel TCHAR. Strenger må omslutes med en makro for å gjøres om til unicode. Også disse finnes også i generiske varianter.

Seislog og Unicode

Alle tekststrenger i Seislog for PocketPC er Unicode stenger av typen TCHAR, med unntak av strenger som skriver til filer med filformat der ANSI tegn kreves. Dette gjelder Ringbuffer, WriteEvent og UniformPCFile (benytter Seisan filformatet). Alle strengene og tegn som skrives til filer i disse modulene er ANSI typer og behandles med ANSI strengbehandlingsfunksjoner. Strenger som skal skrives og som er representert som Unicode strenger, konverteres først til ANSI. Eventlog skriver også loggmeldinger som ANSI strenger.

Konfigurasjonsfiler i Seislog for PocketPC benytter Unicode representasjon for tekststrenger og er derfor ikke kompatible med konfigurasjonsfilene i original Seislog for Windows. Unicode-endringer er ikke kommentert i kildekoden dersom det bare er rene erstatninger. Her en tabell over erstatninger av strengbehandlingsfunksjoner:

ANSI	UNICODE	Kommentar
_stricmp	_wcsicmp	Sammenligning av strenger på små bokstaver
atoi	_wtoi	Konverterer en streng til et integer
itoa	_itow	Konverterer et integer til en streng
sprintf	swprintf	Formattering av strenger
strcat	wscat	Konkatinerer av strenger
strchr	wcschr	Plukker ut et tegn i en streng
strcmp	wscmp	Sammenligning av strenger
strcpy	wscpy	Kopierer tegn fra en streng til en annen
strlen	wcslen	Lengden av en streng
strncpy	wcsncpy	Kopierer tegn fra en streng til en annen
sscanf	swscanf	Leser formatterte data fra en streng
strncmp	wcsncmp	Sammenligner to strenger
strncat	wcsncat	Konkatinerer av strenger
itoa	_ltow	Konverterer langt heltall til streng
atol	_wtol	Konverterer tegn til langt heltall

Datatypen char er bare erstattet med TCHAR om den benyttes til Unicode tegn eller tekststreng. Enkelte steder i programmer er variabler av typen char benyttet for tall, altså i betydningen byte. Disse er fremdeles char. ANSI strenger og tegn i programmet benytter typen CHAR. Alle Unicode strenger i programmet er omsluttet med TEXT makroen. Verken UNICODE eller _UNICODE er definert, da TCHAR uansett oversettes til signed short ved kompilering for Windows CE i eMbedded visual C++. Merk at UNICODE og _UNICODE må defineres ved eventuell kompilering av kode til Unicode versjon for Windows 9X.

I programmer som benytter ANSI strenger vil det typisk finnes kode som implisitt tar for gitt at et tegn opptar én byte. Dette er ofte i forbindelse med allokering av minne for tekststrenger, eller skriving av tekststrenger til disk, som i følgende utdrag kode:

```
// Write message to the log file.
wscat(LogMsg, TEXT("\r\n"));
WriteFile(hLogFile,
    LogMsg,
    (DWORD)(wcslength+2),
    &BytesWritttten,
    NULL);
```

Koden over er ikke riktig; bare halve strengen skrives til disk. Antall bytes som skal skrives er strenglengden multiplisert med antall bytes pr tegn. Løsningen er derfor å multiplisere med sizeof(TCHAR):

```
...
        (DWORD)(wcslength+2)*sizeof(TCHAR),
...
```

Et annet problem som kan oppstå i forbindelse med filskriving, er når programmet skal skrive i et filformat som inneholder ANSI strenger. I slike tilfeller må eventuelle UNICODE strenger først konverteres til ANSI. Dersom strenger ikke brukes i API, kan det være like greit å la de være char-strenger fra begynnelsen av for å unngå unødig konvertering.

I Seislog har datafiler en header som inneholder strenger og strenger må skrives som ANSI strenger for å være kompatibelt med andre versjoner av Seislog. Dersom disse skrives til disk som Unicode-strenger blir filformatet endret, noe som ikke er ønskelig. Strenger som ikke brukes i API-kall, kan en representere som char-strenger, og behandle med ANSI strengfunksjoner fra C Run-time Library. Strenger som også brukes i API kan representeres som Unicode-strenger, og konverteres til ANSI når de skrives til disk. Den konverterte strengen lagres midlertidig i en buffer. Eksempel på dette vises i koden under, som er utdrag fra modulen WriteEvent. I eksempelet skrives filheader for en event fil. Alle elementer i filheaderen samles i en array 'line' av typen BYTE, For å navigere byte-strengen benyttes en variabel '_LinePtr' av typen peker til char.

```
// Sett inn event ID. Denne benyttes ikke i API og
// er derfor representert som char.

_LinePtr=(PCHAR)(line+22);
if (SI.ManualTrig) {
    *_LinePtr='M';
} else {
    // Insert trigger set number.
    sprintf(_str31, "%ld", SI.TrgSetNumber); // ANSI funksjon
    *_LinePtr=_str31[0];
}

// NetworkCode er en global strengvariabel definert som
// også benyttes i API kall, og er definert som
// TCHAR NetworkCode[6];
// Network code må konverteres til ANSI før kopiering

CHAR _NetworkCode[6]; // Oppretter buffer
wcstombs(_NetworkCode, NetworkCode, 6); // Konverterer
_LinePtr=(PCHAR)(line+45);
CopyMemory((PVOID)_LinePtr, (PVOID)_NetworkCode, 3);

...

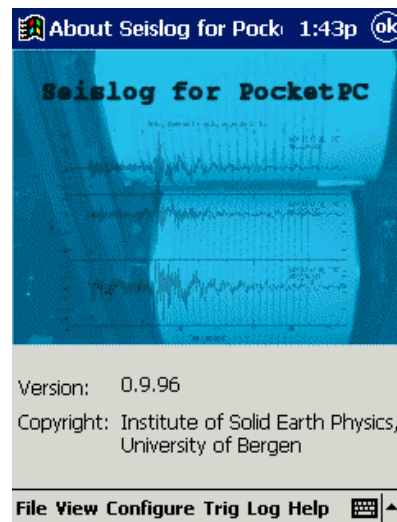
WriteFile(hFile, (PVOID)line, 82, &NumberOfBytesWritten, NULL);
```

5.3 Grensesnittet

Det er laget en fullstendig ny ressursfil for programmet. De aller fleste ressurser har samme navn. I seislog er det flere kontroller i ulike dialoger som benytter samme ressurs-ID. Flere benytter for eksempel ID_EDIT1. Dette resulterte i problemer i eMbedded Visual C++, selv om det burde vært uproblematisk. Problemet ble løst ved at de berørte kontroller ble gitt en ny (og bedre) ressurs-ID, for eksempel ID_SAMPLE_RATE.

Fullskjem-dialoger

Med en skjerm på 320x200 piksler, er det avgjørende at en kan unytte plassen i dialoger maksimalt. Med fullskjem-dialoger benyttes hele skjermen og navigasjonslinjen øverst i dialogen har tittel på dialogen. Til høyre på denne, er det også mulighet for å plassere OK-knappen. Funksjonen **SHInitDialog** benyttes til å sette disse utvidede egenskapene for dialogen, og i tillegg gir den mulighet til å styre input panel til en viss grad. Egenskapene fylles ut i en struktur av typen **SHINITDLGINFO**, som gis som parameter til **SHInitDialog** i **WM_INITDIALOG**. Trykk på OK-knappen sender **WM_COMMAND** –melding med **IDOK** som parameter. Koden for "OK", "Lukk" eller lignende knapper må derfor plasseres der denne meldingen fanges opp. Følgende kode viser en "About" dialog. Selve dialogen er vist i figur 15.



Figur 15: En fullskjem-dialog. Merk at OK-knappen ikke finnes inne i selve dialogen, men i navigasjonslinjen.

```

BOOL CALLBACK AboutDlgProc(HWND hDlg, UINT message,
                           WPARAM wParam, LPARAM lParam)
{
    SHINITDLGINFO shidi;
    extern TCHAR ProgVer[];
    switch (message) {
        case WM_INITDIALOG:
            shidi.dwMask = SHIDIM_FLAGS;
            shidi.dwFlags = SHIDIF_DONEBUTTON |
                           SHIDIF_SIPDOWN |
                           SHIDIF_SIZEDLGFULLSCREEN;
            shidi.hDlg = hDlg;
            SHInitDialog(&shidi);
            // Initialize the static text giving the program version.
            SetDlgItemText(hDlg, IDC_STATIC2, ProgVer);
            return TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK) {
                // Close the dialog box
                EndDialog(hDlg, LOWORD(wParam));
                return TRUE;
            }
            break;
    }
    return FALSE;
}

```

OK-knappen oppnås ved å sette flagget SHIDIF_DONEBUTTON, og SHIDIF_SIPDOWN setter input panelet til senket tilstand. Merk at selv om dialogen bli fullskjerm, vises ikke “Title” egenskapen til dialogen i navigation bar. For å eventuelt sette denne, brukes funksjonen **SHSetNavBarText**.

Command Bars

En “Command bar” er et kontroll-vindu som kan inneholde knapper, bitmaps, kombo-bokser og menyer. Windows-CE baserte applikasjoner benytter command-bars i stedet for separate menyer og verktøylinjer for å unytte plassen på skjermen best mulig. Funksjonen CommandBar_Create benyttes til å opprette en Command Bar. Denne returnerer en handle til Command Bar vinduet, som benyttes ved senere referanser, for eksempel til å sette inn en meny. Command Bars er ”barnevinduer” og ett av parametrene til funksjonen er handle til foreldrevinduet . Merk at størrelsen på Command Bar ikke endres automatisk når foreldrevinduet endrer størrelse, og må derfor håndteres spesielt. Her er et utdrag kode fra en vindusprosedyre som viser litt bruk av en command bar:

```
case WM_CREATE:
    // Opprett en Command Bar
    hwndCB = CommandBar_Create (hInst, hwnd, 1);

    // Sett inn en meny, IDM_MAIN_MENU
    CommandBar_InsertMenubar (hwndCB, hInst, IDM_MAIN_MENU, 0);

    break;

case WM_SIZE:
    // Informer command bar om at vinduet har endret størrelse
    SendMessage(hwndCB, TB_AUTOSIZE, 0L, 0L);
    break;

case WM_DESTROY:
    // Ødelegg command bar
    CommandBar_Destroy(hwndCB);
```

Menu Bars

En menu bar har flere likhetstrekk med en command bar. Den er i hovedsak en meny, men har også mulighet for å ha en knapper. Eventuelle knapper er også menypunkter, men vises som små bitmaps istedet for tekst. Både menyen og eventuelle bitmaps lages som egne ressurser. Menu bars er alltid på bunnen av skjermen, og inneholder knappen for styring av Input Panel, når denne skal vises. I de fleste PocketPC applikasjoner vil nok en menu bar være mer hensiktsmessig enn en command bar.

Menu bars opprettes med funksjonen SHCreateMenuBar. Eneste parameter til funksjonen er en struktur av typen SHMENUBARINFO. I denne settes alle egenskapene, som handle til foreldervindu, ressurs-ID for menyen og noen flagg. I utgangspunktet vil en menu bar kún vise knappen for styring av input panel, men denne kan slås av ved å sette et flagg.



Figur 16: En menu bar befinner seg på bunnen av skjermen og inneholder også knappen styring av Input Panel.

Store dialoger

Den begrensede skjermstørrelse har resultert i at fem store dialoger er splittet i egenskapsark:

- Program Settings
- Add Channel
- Modify Channel
- Add/Modify Triggerset (samme dialog brukes til begge formål)
- Device Configuration av Internal Wave-Generator

Hvert egenskapsark har sin dialogressurs og vindusprosedyre. Navnet på dialogressursene er det opprinnelige navnet på dialogen med ”_N” lagt til på slutten, der *N* er nummeret på egenskapsarket. Nummerering starter med 1 fra venstre. Dialogen IDD_ADD_CHANNEL er delt opp i IDD_ADD_CHANNEL_1, IDD_ADD_CHANNEL_2, IDD_ADD_CHANNEL_3 og IDD_ADD_CHANNEL_4.

Navnet på dialogboksprosedyrene er det opprinnelige navnet med ”N” lagt til på slutten der *N* har samme betydning som over. Dialogboksprosedyrene for den oppdelte ProgramSettingsDlgProc har derfor fått navnene ProgramSettingsDlgProc1, ProgramSettingsDlgProc2 osv.

For hvert egenskapsark må en fylle ut en struktur av typen PROPSHEETPAGE. Denne har blant annet peker til vindusprosedyre og dialogressurs som medlemmer. Windows setter opp egenskapsarkene i en tab kontroll i en dialog (kalt Property Sheet). Egenskapene for denne fylles ut i en PROPSHEETHEADER struktur, og en av medlemmene til denne er en peker til en tabell av PROPSHEETPAGE. Dialogen opprettes med funksjonen PropertySheet som har en peker til



Figur 17: Property Sheet for programinstillinger

PROPSHEETHEADER strukturen som parameter. Verken dialogen eller tab-kontrollen er derfor programmert eksplisitt.

Det er imidlertid laget en callback funksjon for hvert Property Sheet som kalles idet dette åpnes. Denne sørger for at dialogen blir vist i fullskjerm, og for Program Settings dialogen kalles denne ProgramSettingsPropSheetProc. For å åpne dialogen med egenskapsarkene, er det også laget en funksjon. Navnet har logisk sammenheng med navnet på egenskapsarkene; for dialogen Program Settings kalles denne ProgramSettingsDialogBox.

```

BOOL CALLBACK ProgramSettingsPropSheetProc(HWND, UINT,LPARAM lParam);
BOOL CALLBACK ProgramSettingsDlgProc1(HWND, UINT, WPARAM , LPARAM );
BOOL CALLBACK ProgramSettingsDlgProc2(HWND, UINT, WPARAM , LPARAM );
BOOL CALLBACK ProgramSettingsDlgProc3(HWND, UINT, WPARAM , LPARAM );
BOOL ProgramSettingsDialogBox(HWND);

```

```

BOOL ProgramSettingsDialogBox(HWND hWnd)
{
    //[PocketPC]
    PROPSHEETHEADER psh;    // The entire Property Sheet
    PROPSHEETPAGE    psp[3]; // The property pages
    INT i;

    // Init page structures with generic information.
    memset (&psp, 0, sizeof (psp));    // Zero out all unused values.
    for (i = 0; i < 3; i++) {
        psp[i].dwSize = sizeof (PROPSHEETPAGE);
        psp[i].dwFlags = PSP_DEFAULT | PSP_PREMATURE;
        psp[i].hInstance = hInst;    // Instance handle where the
                                     // dialog templates are located

        // Now do the page specific stuff.
        // Name of dialog resource for page 1
        psp[0].pszTemplate = MAKEINTRESOURCE(IDD_PROGRAM_SETTINGS_1);
        // Pointer to dialog proc
        psp[0].pfnDlgProc = ProgramSettingsDlgProc1 for page 1;
        psp[1].pszTemplate = MAKEINTRESOURCE(IDD_PROGRAM_SETTINGS_2);
        psp[1].pfnDlgProc = ProgramSettingsDlgProc2;
        psp[2].pszTemplate = MAKEINTRESOURCE(IDD_PROGRAM_SETTINGS_3);
        psp[2].pfnDlgProc = ProgramSettingsDlgProc3;

        // Init property sheet header structure.
        psh.dwSize = sizeof (PROPSHEETHEADER);
        psh.dwFlags = PSH_PROPSHEETPAGE | PSH_USECALLBACK;
        psh.hwndParent = hWnd;    // Handle of the owner window
        psh.hInstance = hInst;    // Instance handle of the application
        psh.pszCaption = TEXT ("Program Settings");
        psh.nPages = 3;    // Number of pages
        psh.nStartPage = 0;    // Index of page to be shown first
        psh.psp = psp;    // Pointer to page structures
        psh.pfnCallback = ProgramSettingsPropSheetProc;

        // Create property sheet. This returns when the user dismisses the sheet
        // by tapping OK or the Close button.

        PropertySheet (&psh);
        return 0;
    }
}

```

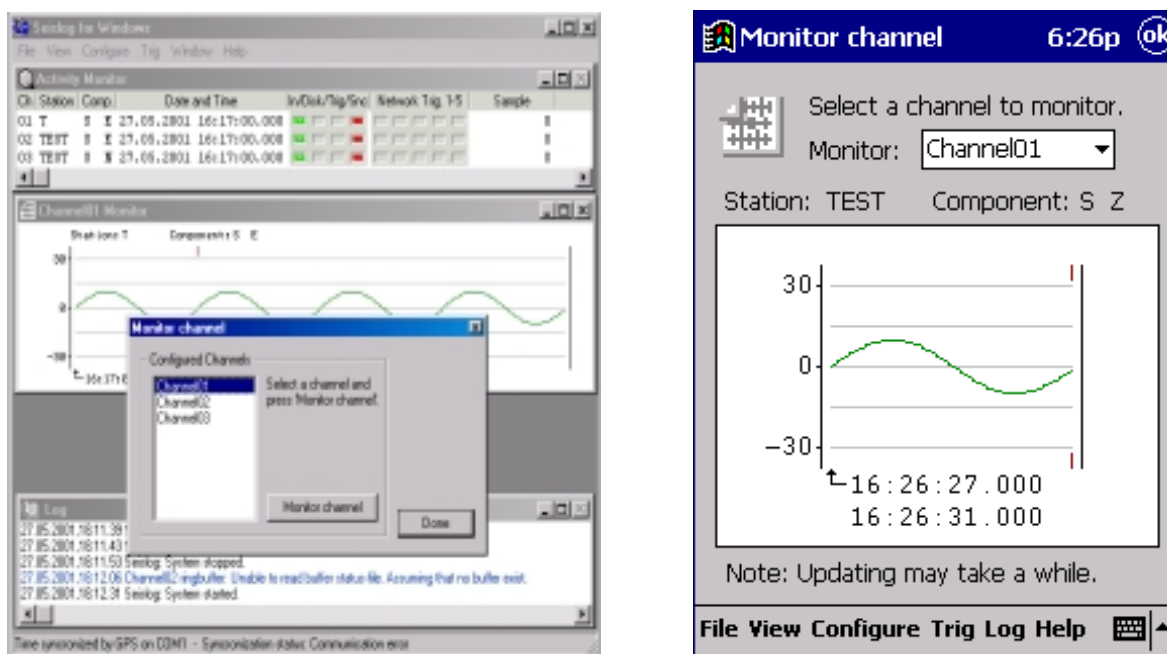
Vinduer

En av utfordringene i design av grensesnittet i Seislog for PocketPC, har vært å tilpasse MDI-arkitekturen med utallige vinduer åpnet samtidig, til en skjerm der til og med tittellinjen i vinduer må gi etter for plassbehovet.

De fleste PocketPC programmer viser som regel bare ett vindu om gangen. Flere vinduer blir fort uryddig. PocketWord, hvis storebror også er en MDI applikasjon, operer også med kun ett dokument om gangen. MDI frame vinduet i Seislog finnes fremdeles på PocketPC, men mangler MDI funksjonaliteten og er i stedet et helt vanlig vindu. Window menyen er tatt vekk og ingen MDI meldinger behandles.

For hver kanal i Seislog kan en vise et plot av målingene i et vindu. Hensikten med visuelt bilde av måledataene, er å å raskt kunne se om loggingen går riktig for seg. Ut fra visning av kun numeriske måleverdier er dette svært vanskelig.

Seislog for Windows gir bruker mulighet til å åpne flere vinduer samtidig, og vinduenes bredde kan varieres, avhengig av hvor mange sekunder måledata en ønsker å se. I en dialog, ”Monitor channel”, kan en velge kanaler som en ønsker å overvåke. Disse åpnes så i hovedvinduet i Seislog. I koden i Windows versjonen av Seislog, benyttes CreateMDIWindow. I PocketPC versjonen benyttes CreateWindow.



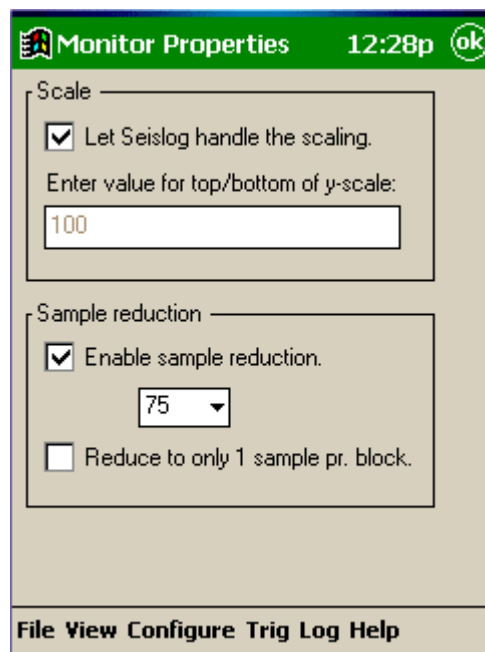
Figur 18 Grafisk visning av måledata i Seislog for Windows og Seislog for PocketPC. Skjermstør gjør at måledata fra kun én kanal kan vises om gangen på PocketPC.

På en PocketPC er det knapt nok plass til å se ett slikt vindu om gangen. Flere vinduer vil overlappe hverandre, og da vil også hensikten med å åpne flere vinduer falle bort. I PocketPC versjonen har vi derfor begrenset antall samtidig Monitor vinduer til étt. Dette vinduet vises i dialogen ”Monitor Channels”, der listen over kanaler er byttet ut med en combo-boks, og knappen for overvåkning er tatt bort. Ved valg i combo-boksen, vises graf for den valgte kanalen i et child-vindu. Koden for dialogen Monitor Channel er fullstendig

omskreve og koden for child vinduet er modifisert kode for Monitor vinduet. Monitor vinduet består i hovedsak av en tråd (MonitorThrd) og en vindusprosedyre. Child vinduet blir opprettet i tråden, og ikke i dialogen "Monitor channel". Handle til foreldervinduet for må derfor gis til MonitorThrd. Dette skjer gjennom et ekstra medlem i strukturen som det sendes peker til ved oppstart av tråden:

```
struct STARTINFO_MONITOR {
    BYTE LogicalChannelNr;        // Logical channel nr of channel to monitor.
    DWORD ChThreadID;            // ThreadID of the channel thread.
#ifdef WINCE
    HWND hParent;                // Handle to the parent window (the dialog)
#endif
};
```

Systemmenyen på Monitor vinduet i Seislog for Windows inneholder et valg for skalering. Dette gjøres i en egen dialog, og gjør det mulig å få vist flere sekunder med data på skjermen samtidig. i Seislog for Windows CE åpnes denne dialogen ved trykk på en knapp i dialogen "Monitor Channel"



Figur 19 Dialog for skalering av viste måledata

De fleste endringene i vindusprosedyren for Monitor, er i forbindelse med grafisk visning og plasseringer av elementer i vinduet. Funksjonen CreateFont finnes ikke for Windows CE API, den ble erstattet med en "struct" kalt LOGFONT, og en funksjon kalt CreateFontIndirect. For å skrive ut tekst er TextOut erstattet med DrawText. Denne funksjonen krever et rektangel for plassering av tekst. Kantene i dette settes for hver streng som skrives ut.

Funksjonene MoveTo og LineTo benyttes i Windows versjonen av Seislog til å tegne grafen. Disse finnes ikke i Windows CE API, men vi har implementert de ved hjelp av funksjonen PolyLine. Både koordinater og funksjonsnavn kunne derfor beholdes. Koden for MoveTo og LineTo er lagt i modulen Monitor. MoveTo og LineTo bygger på prinsippet om tilstandsgrafikk og trenger derfor en variabel for å holde orden på inneværende posisjon. Vi merker oss at det bare er én slik variabel og er ikke bundet til device context, Koden er derfor *ikke* brukbar dersom en hadde tillat flere Monitor vinduer samtidig i programmet.

```
POINT points[2] = {{0,0},{0,0}};

BOOL LineTo(
    HDC hdc, // device context handle
    int _x,  // x-coordinate of ending point
    int _y   // y-coordinate of ending point
)
{
    points[1].x = _x;
    points[1].y = _y;
    Polyline(hdc, points, 2);
    points[0].x = points[1].x;
    points[0].y = points[1].y;
    return TRUE;
}

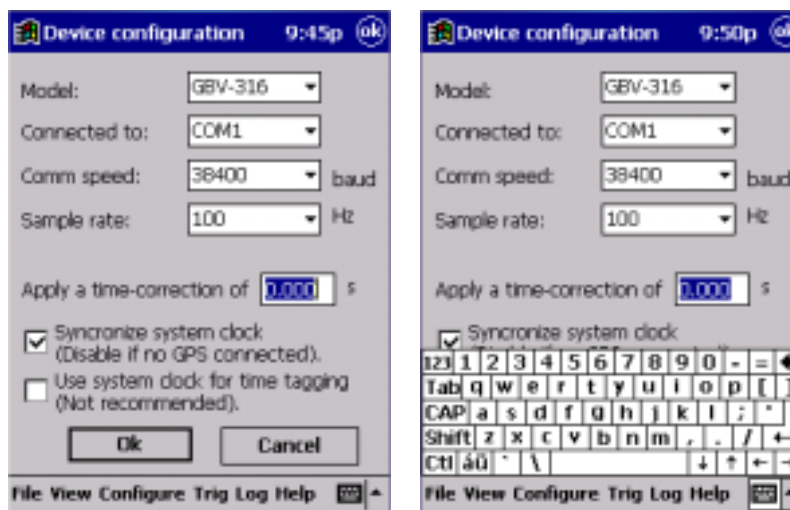
BOOL MoveTo(
    HDC hdc, // NOT used
    int _x,  // x-coordinate of ending point
    int _y   // y-coordinate of ending point
)
{
    points[0].x = _x;
    points[0].y = _y;
    return TRUE;
}
```

De samme endringene med tanke på skriving av tekst, gjelder også Activity Monitor. SetTextAlign er mye brukt i Activity Monitor. Denne finnes ikke på Windows CE, men samme effekt er oppnådd ved å sette plasseringen av teksten i rektangelet i funksjonen DrawText. Enkelte elementer i dette vinduet har også byttet plass og Network Trig er tatt bort. Dette er gjort fordi vinduet på PocketPC bare er 240 pixler bredt og det viktigste elementene bør vises uten å måtte scrolle. Elementene som er synlig er kanalnummer stasjon- og komponentkode og lys for inndata, disk, trigger og synkronisering .

Både Activity Monitor, Eventlog og Monitor er blitt child vindu uten tittlellinje istedet for MDI vindu. Vinduet har fast plassering og størrelsen kan ikke endres.

Kontrol av input panel

Som vi har sett gir flere funksjoner anledning til å styre enkelte egenskaper til input panel. I enkelte dialoger kan det være nyttig å heve og senke input panel avhengig av hvilken type kontroll som har fokus. Edit kontroller krever at Input Panel er hevet, mens det er unødvendig ved avkryssningsbokser. Design av dialoger krever at edit kontroller plasseres over området Input Panel dekker i hevet tilstand. Avkryssings-bokser kan derimot plasseres i dette området for å utnytte plassen bedre, men da er Input Panel et direkte hinder.



Figur 20: For å unytte den begrensede plassen i dialoger, kan enkelte kontroller plasseres i området input panel dekker i hevet tilstand.

Styring av input panel kan selvsagt overlates fullstendig til bruker, men det finnes en løsning som forenkler dette. Ved å inkludere kontrollressursen WC_SIPPREF i ressursskriptet for dialogen, hjelper denne med å styre tilstanden til input panel. Panelet blir da i utgangspunktet senket, men heves dersom input fokus er i én av følgende kontroller

- Edit
- CAPEDIT
- Combo bokser
- Date time picker kontroller

Panelet senkes igjen når ingen av disse kontrollene har fokus. Følgene utdrag fra ressursskriptet viser hvordan kontrollen kan brukes:

```
CONTROL " ", -1, WC_SIPPREF, NOT WS_VISIBLE, -10, -10, 5, 5
```

Kontrollen har ingen meldinger assosiert med seg; det er kun kontrollens tilstedeværelse i ressursskriptet i dialogen som gir funksjonaliteten. CAPEDIT er en Edit-kontroll særegen for Windows CE, som automatisk gjør første bokstav i edit-feltet stor. Dersom det er mest hensiktsmessig å ha input panel hevet hele tiden, uansett type kontroll, kan dette gjøres med funksjonen **SHInputDialog**. Denne kan for eksempel kalles under behandling av WM_INITDIALOG, og vil da sørge for at Input Panel er hevet så lenge dialogen er aktiv.

5.4 Andre endringer

Vi gir her en oversikt over andre viktige endringer:

- På grunn av begrenset lagringsplass, konfigureres ringbufferet ut fra timer og ikke dager.
- I Windows CE settes prioritet direkte på tråder og ikke på prosesser. `SetPriorityClass` finnes ikke, og `SetThreadPriority` benyttes til å sette prioritet på tråder
- Meldinger som angår strømstyring, f.eks `WM_POWERBROADCAST`, `PBT_APMQUERYSPEND` og `PBT_APMBATTERYLOW` benyttes ikke på Windows CE. En PocketPC vil slå av etter en fastsatt tid og en teller sørger for dette. Denne telleren kan imidlertid nullstilles. Dette er ikke gjort i Seislog for PocketPC. Behandling av en ny melding spesifikt for Windows CE, `WM_HIBERNATE`, er lagt til i vindusprosedyren for hovedvinduet i Windows CE.
- Funksjonene `CheckDlgButton ()` og `IsDlgButtonChecked()` mangler i WindowsCE og vi har derfor implementert disse. Koden ligger i `Library`.
- System commands systemmeny-meldinger behandles ikke. Dette er imidlertid løst på andre måter, for eksempel med knapper.
- Seislog for Windows lagrer posisjon og størrelse for hvert vinduer i konfigurasjonsfiler. Til dette benyttes strukturen `WNDPLACEMENT`, som ikke er tilgjengelig i Windows CE. Siden alle vinduer i Seislog for PocketPC har fast størrelse og plassering, har vi ikke laget noen erstatning for dette.
- Win32 fil I/O er støttet på Windows CE og gjør det mulig, med noen unntak, å aksessere Object Store akkurat som en disk på Win32 filsystem. Den viktigste forskjellen er hvordan filer refereres til. Windows CE har ikke inneværende katalog ("current directory"). Alle stier tolkes derfor som absolutte referanser med i rotkatalogen som er `'\'`. Katalogen `".\dir"` tolkes som `".\dir"`. Alle funksjoner som baserer seg på konseptet med "current directory" er tatt bort fra API-et, men enkelte funksjoner har fått noen annen oppførel. Siden både `current ('.')` og `parent ('..')` er tatt bort, vil disse heller aldri bli funnet med funksjonen `FindFirstFile` og `FindNextFile`. Dette kan føre til logiske feil dersom programmet baserer seg på at disse finnes. Katalogen `'\seislog\devices'`. Når konfigurasjonen skal leses inn, søkes det etter slike underkataloger med filspesifikasjon `'*. *'`. Under Windows for Desktop PCer vil de to første treffene for et slikt søk være `'.'` og `'..'`. De to første treffene blir derfor hoppet over på desktop PC versjonen, mens de på PocketPC vil gi de to første katalogene for enheter. Denne spesifikke endringen gjelder funksjonen `RegisterDevices`. I likhet med funksjoner for aksessering av filsystemet, er det lite endringer i seriell kommunikasjon. `BuildCommDCB`, som er en mye brukt funksjon for å automatisk fylle ut en Device Control Block (DCB) er ikke med i Windows CE API, men dette løses enkelt ved å i stedet fylle ut de nødvendige egenskapen i DCB-strukturen manuelt. En annen endring er at alle porter brukt i `CreateFile` må etterfølges av kolon, f.eks `"COM1:"`. Dette skapte for øvrig en del hodebry for oss da vi lenge forsøkte å åpne port `"COM1"...`

- **Eventlog**

Eventlog vinduet viser et gitt antall av de siste meldingene. Antallet er satt med konstanten MSG_LIST_SIZE i Eventlog.cpp. I PocketPC versjon er dette antallet redusert for å avlast tegning

6. Testing

6.1 Testing og feilsøking på PocketPC

Før et program portes til PocketPC er det viktig at det er godt testet i desktop versjon, slik at en ikke drar med feil. Testing av programmer på PocketPC blir fort mer krevende enn på en vanlig desktop PC, og det er viktig at en har et så godt utgangspunkt som mulig.

I verktøyet eMbedded Visual Tools finnes det mulighet for å kjøre PocketPC programmer i en "Desktop PocketPC emulator". Denne emulatorens kan kjøres i Windows NT eller 2000 på en x86 prosessor. Programmer som skal kjøres i denne emulatorens kompiles for x86 maskinkode og kjøres som egne prosesser i Windows, akkurat som andre windows-programmer. Dette gir anledning til å bruke test- og feilsøkingsverktøy som finnes på desktop PCer.

Emulatorens har imidlertid enkelte begrensinger, og ikke alle problemer lar seg oppdage gjennom kjøring i emulatorens. Et eksempel er problemer knyttet til "byte alignment". Data er "aligned" dersom datastørrelse modulo minnedresse er 0. En x86 prosessor vil håndtere "unaligned" data ved å foreta flere minneakssereringer, mens en MIPS prosessor vil feile og gi hardware exception. Desverre finnes det også mindre eksotisk forskjeller mellom de ulike PocketPC varianter. Ulike produsenter av PocketPCer har også sine varianter av Windows CE API, som må tas hensyn til. Programmer for PocketPC må derfor testes på alle prosessorutgavene som programmet er ment kjørt på.

eMbedded Visual Tools gir mulighet for fjern-feilsøking av programmer mens de kjører på PocketPC. Minimumskravet er at en har en USB eller seriell-tilknytning til PC som kjører eMbedded Visual Tools. Debug-versjon av programmet overføres og startes opp gjennom eMbedded Visual Tools og gjennom dette verktøyet kan en så feilsøke med bruddpunkter, anvisning i kildekode under kjøring og andre fasiliteter.

Fjern-feilsøking er adskilling raskere dersom en har nettverktilknytning til PocketPC, i tillegg til USB eller annen seriell tilknytning. For feilsøking av seriell eller USB-port, kan en benytte infrarød tilknytning for å frigjøre denne porten.

6.2 Testing av Seislog for PocketPC

I dette kapittelet nevner vi bare større tester på slutten av prosjektet. De ulike delene av programmet er testet nøye underveis, særlig der ny kode er skrevet.

Langtidstesting for stabilitet

Testingen ble utført i følgende tisdrom

- **Fredag 01.06.2001 kl 15:00 til tirsdag 05.06.2001 kl 1000.**
- **Fredag 08.06.2001 kl 15:30 til mandag 11.06.2001 kl 1000.**

En Casio Cassiopeia E-125 PockePC med 32MB minne og 150MHz MIPS prosessor ble konfigurert som følger:

- Ekstern strømforsyning (ikke batteridrift)
- Lavest mulig lysstyrke på skjerm
- Ingen nettverkskort eller andre innstikkskort
- Alle funksjoner for automatisk avslåing av PocketPC ble slått av
- Av bruker programmer var det kun Seislog som kjørte

Seislog for PocketPC ble satt opp med følgende konfigurasjon

- Enhet: Nanometrics RD6 koblet til COM1 med 9600 baud hastighet og 50Hz sample rate.
- Kanaler: 6 kanaler med ringbuffer og triggering. Alle forhåndsvalgt verdier ble brukt (ringbuffer var på 2 timer).
- Triggerset: Étt triggerset med kanaler.

Data for ringbuffer tok ca 6 MB. Det ble generert ca 40 event-filer, som tilsvarer omkring 1MB. Testene avdekket ingen ustabiliteter, men flere mindre feil ble avdekket og korrigert. En av feilene var feil i dataformat, slik at måledata ikke var leselig i SeisAn.

Testing av batterilevetid

En Casio Cassiopeia E-125 PockePC med 32MB minne og 150MHz MIPS prosessor ble konfigurert som følger:

- Fult oppladet batteri
- Ekstern strømforsyning ble koblet ut
- Lavest mulig lysstyrke på skjerm
- Ingen nettverkskort eller andre innstikkskort
- Alle funksjoner for automatisk avslåing av PocketPC ble slått av
- Av bruker programmer var det kun Seislog som kjørte

Seislog for PocketPC ble satt opp med følgende konfigurasjon

- Enhet: Nanometrics RD6 koblet til COM1 med 9600 baud hastighet og 50Hz sample rate.
- Kanaler: 6 kanaler med ringbuffer og triggering. Alle forhåndsvalgt verdier ble brukt.
- Triggerset: Étt triggerset med kanaler.

Systemet startet logging kl 14.06.2001 kl 09:46. Maskinen ble så overlatt til seg selv med Seislog hovedvindu i forgrunnen. Ingen form for input av bruker ble gitt i testperioden og romtemperaturen var 21° C. Mellom kl 16:58 og 17:28 slo maskinen seg av på grunn av lavt batterinivå. Dette vil si at batterilevetiden er litt over 7 timer under ideelle forhold med den oppgitte konfigurasjonen. Batterilevetiden reduseres ved økt input fra bruker. Hvor mye dette utgjør er ikke testet.

Testing av kommunikasjonsfeil

Seislog ble konfigurert med enheten GeoSys 316 med tre kanaler, og systemet ble startet opp. Under kjøring ble så følgende former for kommunikasjonsfeil og hendelser testet i oppsatt rekkefølge:

Hendelse	Resultat
Seriekabel ble tatt ut av pocketPC	Main Monitor viste røde lamp, meldingen "Not receiving data" ble skrevet i hendelseslogg og systemet stoppet å logge.
Kabel ble satt inn igjen	Main Monitor lyste grønn, meldingen "Receiving data" ble skrevet i hendelseslogg og systemet startet umiddelbart å logge.
Seismometer ble slått av	Samme resultat som ved uttak av seriakabel
Seismometer ble slått på	Samme resultat som ved innsetting av seriekabel.

Seislog for PocketPC er altså i stand til å starte logging igjen etter brudd i kommunikasjon med instrumentene.

Testing av brå avslåing av PocketPC

Når en PocketPC slås av, "fryses" maskinen i denne tilstanden. Hvis en PocketPCen slås av under logging vil den klare å starte logging igjen automatisk ved oppstart. Det vil imidlertid oppstå tidshull i måledata.

7. Oppsummering

7.1 Tilbakeblikk

Implementasjonen på PocketC har tatt relativt kort tid, men det har vært mye å sette seg inn i før en har kunnet gå i gang med denne jobben. Gjennomgang av 1MB kildekoden har vært det mest tidkrevende. Planleggingen av prosjektet har også vært vanskelig, selv om vi har kommet i havn. Dette har flere årsaker: Vi hadde begrensede kunnskaper om emnene på forhånd og det var vanskelig å få oversikt over omfanget av oppgaven og hva som faktisk måtte gjøres. Mange ganger har vi sett at det vi antok skulle ta langt tid, har gått bemerkelsesverdig fort og smertefritt. Derimot er det flere uforutsette problemer som har tatt mye tid.

En viktig del av oppgaven har vært porting til Unicode. Den store mengden strengbehandlingsfunksjoner (over 1300) og strengvariable (over 300) har gjort bruk av søkerstatt helt nødvendig. Automatisering av denne oppgaven er ikke helt problemfritt, da en også feilaktig kan erstatte noe som ikke skal erstattes. I slike sammenhenger er det fort gjort at en lager seg merarbeid.

Tilpasning av det grafiske grensesnittet har også vært en sentral del av oppgaven. Mye av ”business”-koden fra Windows-versjonen av Seislog, har vi derimot kunne anvende uten andre modifiseringer enn Unicode-porting. Desverre har vi flere ganger sett at ikke alt fungerer som det skal, selv om kode fra Windows versjon kompilerer på Windows CE platformen. Dette har er særlig frustrerende i tilfeller der feilene skyldes at API-kall på Windows CE har samme navn som i Win32 API men har annen semantikk.

Porting har skjedd i to steg, først Unicode porting av Windows versjon, og deretter porting til Windows CE. I begge stegene har testing underveis vært vanskelig; tette koblinger og avhengigheter mellom modulene i programmet har gjort at lite i programmet har fungert før nesten alle moduler ble ferdig konvertert. Testing av grensesnittet har derimot gått greit underveis, og ny kode i programmet er utviklet og testet i egne prosjekter før det ble tatt inn i programmet.

De store likhetene mellom Windows CE og Win32 API har lettet portingen betraktelig. Vi har tatt direkte utgangspunkt i eksisterende kode, og tilpasset denne til Windows CE. Porting til Windows CE startet med å videreutvikle originalversjonen. Fordelen med dette er klar: Ved å videreutvikle direkte i desktop versjonen, i samme utviklingsverktøy og kompilator, fortsetter en med noe som allerede fungerer. Feilsøking blir lettere og hele programmet kan testes i sin helhet før resten av portingen tar til. Med dette vil typisk en stor del av feilene være eliminert, før en engang har startet med å kompilere for Windows CE og PocketPC.

7.2 Videreutvikling av Seislog

Kildekodestyling

For å skille Win32 og Windows CE kode er det benyttet kompilatorsvisjer.

```
#define WINCE          // defineres ved kompilering for Pocket PC
...
#ifdef WINCE

    // Pocket PC spesifikk kode

#else

    // Eventuell desktop PC spesifikk kode

#endif

...
// Felles kode
...

#ifndef WINCE

    // Desktop PC spesifikk kode

#endif
...
```

Det er ikke gjort forsøk på å kompilere for Windows, altså uten å definere WINCE. Med tanke på videre vedlikehold av programmet, vil vi anbefale å *ikke* kompilere Windows versjon ut fra den denne versjonen, men å beholde den gamle. Argumentene for dette er følgende:

- Vi har ikke testet å kompilere for Windows
- Versjonen som finnes for Windows fungerer og er godt testet.
- Windows og PocketPC versjonene av Seislog er svært like (bortsett fra Unicode og grensensitt), slik at det er lett å endre kode begge steder.
- Å skifte mellom de to versjonene, særlig ved små endringer i kode, kan være tidkrevende.

Manglende drivere

Seislog for PocketPC mangler støtte for noen av enhetene som støttes i Seislog for Windows. Vi har kun portet drivere for enheter som vi har hatt tilgjengelig under utviklingsarbeidet, og derfor har kunnet testet. Dette er følgende enheter:

- GeoSIG AG - GBV-x16
- Nanometrics - RD3 / RD6, software rev 3/5
- Internal Wave-Generator
- Earth Data Ltd - PS2400

Earth Data Ltd - EDM006 er også portet men er ikke testet. Denne kan derfor inneholde feil. Enheten laget av ComputerBoards er ikke mulig å porte da den krever et spesielt kort som ikke kan benyttes i PocketPC.

I Seislog for Windows og PocketPC ligger driverne for hver av enhetene egne filer. For å få inn støtte for disse, må driverfilene for de manglende enhetene portes, og det må gjøres noen endringer i `devices.cpp` og `devices.h`

I `devices.h` må forward deklarasjon av manager trådene tas med. Utdraget under, vises inndelingen. Flytt ganske enkelt forward deklarasjon til WINCE blokken for å ta med en driver.

```
#ifdef WINCE
DWORD WINAPI PS2400MngrThrd(PVOID);
DWORD WINAPI NullDevMngrThrd(PVOID);
DWORD WINAPI GBVx16MngrThrd(PVOID);
DWORD WINAPI NMRDxMngrThrd(PVOID);
#else
DWORD WINAPI EDM006MngrThrd(PVOID);
DWORD WINAPI NMHRD24MngrThrd(PVOID);
DWORD WINAPI CBDAS640216MngrThrd(PVOID);
DWORD WINAPI KMK2MngrThrd(PVOID);
#endif
```

I `devices.cpp` må funksjonene `StartDevice`, `StartDeviceManager` og `AddDeviceDlgProc` også endres. Her det benyttet kompilatorsvitsjer for å kommentere ut oppstart av manager-tråder og elementer i listeboksen for enheter som mangler. Flytt nødvendig kode inn i WINCE blokkene.

Driverne for de ulike enhetene har store likhetstrekk, så de driverne som allerede er portet kan benyttes som mal. Endringer som må gjøres i driverne for enhetene er følgende:

- Porting til Unicode
- Konkatinere tegnet ':' på strengen som angir portnummer og er parameter til CreateFile. Følgene eksempel viser hvordan dette er gjort i driveren for GeoSig 316:

```
#ifdef WINCE
    // All ports in Windows CE must end with ':'
    TCHAR CPortName[7];
    wcscpy(CPortName, DeviceConf.PortName);
    wcscat(CPortName, TEXT(":"));
    hComm=CreateFile(CPortName, GENERIC_READ, 0, NULL,
        OPEN_EXISTING, 0, NULL);
#else
    hComm=CreateFile(DeviceConf.PortName, GENERIC_READ, 0, NULL,
        OPEN_EXISTING, 0, NULL);
#endif
```

- Bruke en struktur SHINITDLGINFO og kalle SHInitDialog() i WM_INITDIALOG i dialogboksprosedyren for dialogen der enheten konfigureres. Sørg for at OK knappen har IDen ID_OK. Se hvordan dette er gjort i andre drivere
- Device Control Block structen (DCB) må fylles ut manuelt, da BuildCommDCB() ikke kan benyttes på Windows CE. Her er et eksempel som også er hentet fra driveren til for GeoSig 316:

```
#ifdef WINCE
    dcb.BaudRate      = _wtoi(SI->CommSpeed);
    dcb.fParity        = TRUE;
    dcb.Parity         = NOPARITY;
    dcb.StopBits       = ONESTOPBIT;
    dcb.ByteSize       = 8;

    dcb.fInX           = FALSE;
    dcb.fOutX          = FALSE;
    dcb.fOutxDsrFlow   = FALSE;
    dcb.fOutxCtsFlow   = FALSE;
    dcb.fDtrControl    = DTR_CONTROL_ENABLE;
    dcb.fRtsControl    = RTS_CONTROL_ENABLE;
#else
    wcscpy(DevCntrlStr, TEXT("baud="));
    wcscat(DevCntrlStr, SI->CommSpeed);
    wcscat(DevCntrlStr, TEXT(" parity=N data=8 stop=1"));
    BuildCommDCB(DevCntrlStr, &dcb);
#endif
```

Referanser

Richter, Jeffrey. Programming Applications for Microsoft Windows. Microsoft Press, 1999. ISBN 1-57231-996-8.

Boling, Douglas. Programming Windows CE. Microsoft Press. ISBN 1-57231-856-2.

Havskov, Jens et al. Seisan The Earthquake Analysis software for Windows Solaris and Linux. Bergen, May 2000.

Microsoft Developer Network. Microsoft Corporation. <http://msdn.microsoft.com>.

Pappas, Chris H. et al. Debugging C++. Osborne/McGraw-Hill. ISBN 0-07-212519-5.

Petzold, Charles. Programming Windows. Microsoft Press. ISBN 1-57231-995-X.

Berg, Lars E. Programutvikling for tekniske anvendelser. ISBN 82-7674-393-5.