SINTEF IKT

Postadresse: 7465 Trondheim
Besøksadresse: O S Bragstads plass 2C
7034 Trondheim
Telefon: 73 59 30 00
Telefaks: 73 59 10 39

Foretaksregistreret: NO 948 007 029 MVA

SINTEF RAPPORT

TITTEL

Støyberegner – Teknisk dokumentasjon

FORFATTER(E)

Rolf Tore Randeberg, Herold Olsen

OPPDRAGSGIVER(E)

Statens vegvesen

RAPPORTNR.	GRADERING	OPPDRAGSGIVERS REF.	
SINTEF A4142	Åpen	Ingunn Milford	
GRADER. DENNE SIDE	ISBN	PROSJEKTNR.	ANTALL SIDER OG BILAG
Åpen	978-82-14-04380-8	90E265	50
ELEKTRONISK ARKIVKODE		PROSJEKTLEDER (NAVN, SIGN.)	VERIFISERT AV (NAVN, SIGN.)
RAPPORT Støyberegner - Teknisk.doc		Herold Olsen	Svein Å. Storeheier
ARKIVKODE	DATO	GODKJENT AV (NAVN, STILLING, SIGN.)	
	2007-12-14	Truls Gjestland, forskningssjef	

SAMMENDRAG

Denne rapporten er en teknisk beskrivelse av støyberegningsløsningen som er utviklet av SINTEF for Statens vegvesen. Hensikten med rapporten er primært å gjøre framtidig utviklingsarbeid og vedlikehold enklere ved at sentrale komponenter og rutiner beskrives. Rapporten er også en dokumentasjon av utviklingsarbeidet som er utført i løpet av prosjektet.

Rapporten inneholder også en installasjonsveiledning for støyberegningsløsningen, samt en brukerveiledning for brukergrensesnittet til ”statusmonitor”-komponenten.

STIKKORD	NORSK	ENGELSK
GRUPPE 1	Akustikk	Achoustics
GRUPPE 2	Programvare	Software
EGENVALGTE	Vegtrafikk	Road Traffic

INNHOLDSFORTEGNELSE

1 Introduksjon

Denne rapporten er en teknisk beskrivelse av støyberegningsløsningen som er utviklet av SINTEF for Statens vegvesen. Hensikten med rapporten er primært å gjøre framtidig utviklingsarbeid og vedlikehold enklere ved at sentrale komponenter og rutiner er beskrevet. Rapporten er også en dokumentasjon av utviklingsarbeidet som er utført i løpet av prosjektet. Programmenes kildekode er for øvrig i stor grad kommentert og gir ytterligere dokumentasjon. Merk at rapporten beskriver programmene slik de var i medio juni 2007, da støyberegningsløsningen ble godkjent av Statens vegvesen. På grunn av feilrettinger som har vært utført i ettertid, vil det kunne være små avvik mellom det som beskrives her og programmene slik de framstår per dags dato.

Rapporten er *ikke* en brukerveiledning[1] for sluttbrukere; for disse har Statens vegvesen laget en egen brukerveiledning. For en teknisk beskrivelse av bruk av støyberegningsløsningen og en detaljert beskrivelse av datagrunnlagfiler og resultatfiler vises det til dokumentet ”NorStøy detaljspesifikasjon” som er utarbeidet i samarbeid mellom SINTEF og Statens vegvesen.

Hele støyberegningsløsningen, unntatt beregningskjernen Nord 2000 Road, er utviklet i programmeringsspråket C# 2.0 fra Microsoft. Utviklingsmiljøet har vært Microsoft Visual Studio 2005. Beregningskjernen Nord 2000 Road er utviklet i Intel Fortran i et tidligere prosjekt for Statens vegvesen. Støyberegningsløsningen er utviklet og kompilert for kjøring på 32-bits prosessorer. Eventuell bruk på nyere 64-bits datamaskiner er derfor foreløpig ikke støttet.

Støyberegningsløsningen består av fire programmer (.exe) og to rutinebibliotek (.dll). De fire programmene er:

- **Statusmonitor**. Koordinerer aktiviteten mellom de andre programmene.
- **StatusmonitorTray**. Gir et brukergrensesnitt for Statusmonitor.
- **Støyberegner**. Behandler et støyberegningsoppdrag, og deler dette i deloppdrag.
- **Regnecelle**. Behandler et deloppdrag

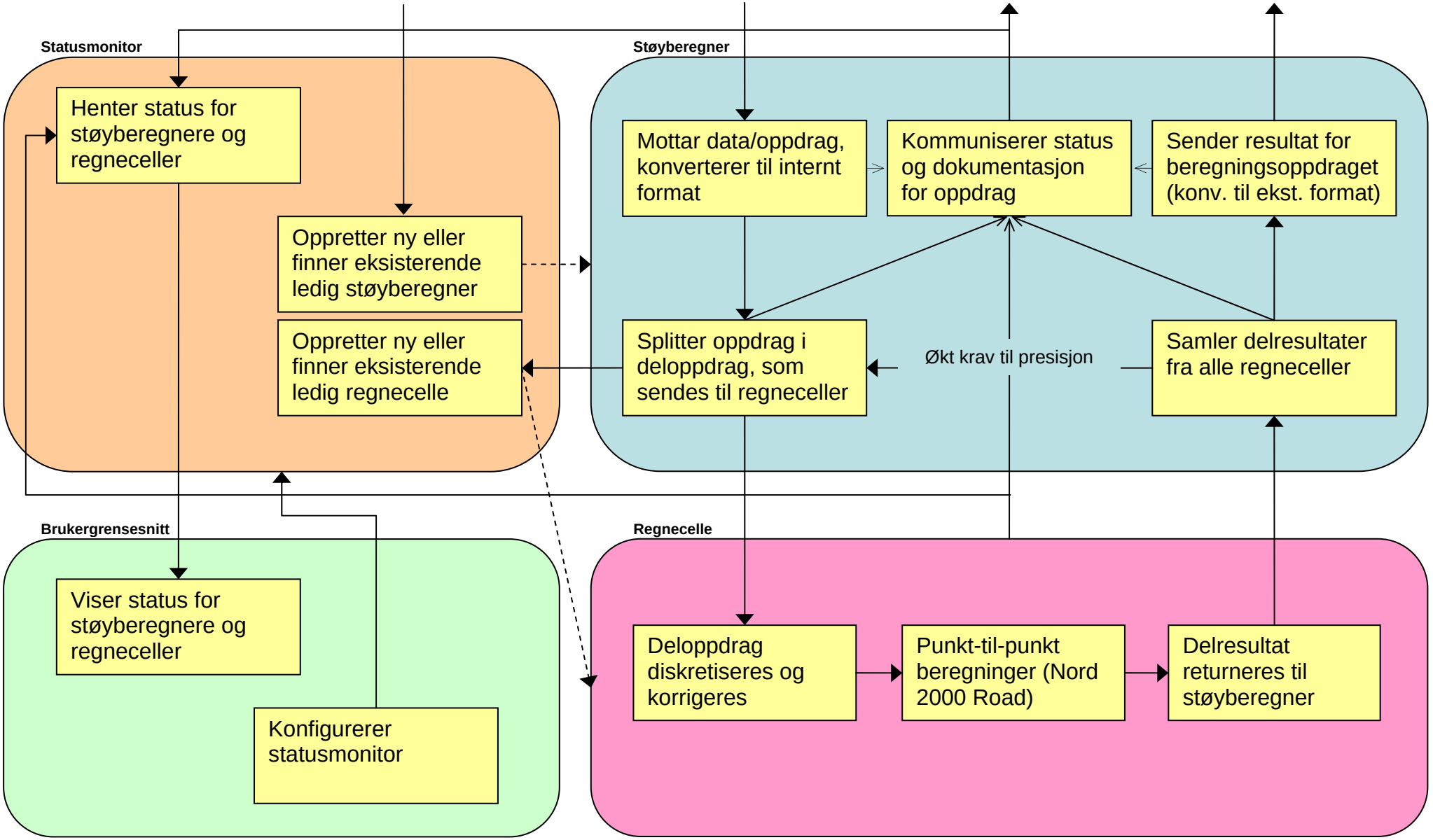
De to rutinebibliotekene (.dll-ene) er:

- **SoundKernel**. Utfører beregning av et deloppdrag, og deler dette i enkle punkt-til-punkt-beregninger.
- **ProP2P**. Utfører punkt-til-punkt-beregninger.

Figuren på neste side gir en oversikt over dataflyten mellom de fire programmene.

De følgende kapitlene beskriver hver komponent etter tur (kapittel 2 – 7). Kapittel 8 gir en oversikt over kommando-grensesnittene til programmene. Kapittel 9 inneholder en installasjonsveiledning. Kapittel 10 forklarer bruken av brukergrensesnittet StatusMonitorTray.

[1] Unntatt for StatusmonitorTray, se avsnitt 10.


Statusmonitor
Henter status for støyberegnere og regneceller
Oppretter ny eller finner eksisterende ledig støyberegner
Oppretter ny eller finner eksisterende ledig regnecelle
Støyberegner
Mottar data/oppdrag, konverterer til internt format
Kommuniserer status og dokumentasjon for oppdrag
Sender resultat for beregningsoppdraget (konv. til ekst. format)
Splitter oppdrag i deloppdrag, som sendes til regneceller
Økt krav til presisjon
Samler delresultater fra alle regneceller
Brukergrensesnitt
Viser status for støyberegnere og regneceller
Konfigurerer statusmonitor
Regnecelle
Deloppdrag diskretiseres og korrigeres
Punkt-til-punkt beregninger (Nord 2000 Road)
Delresultat returneres til støyberegner

2 Statusmonitor

Type: Kjørbart program (Windows service) uten grafisk brukergrensesnitt
Filnavn: Statusmonitor.exe
Grensesnitt: Socket (port 50000)
Oppsett: Registeroppføringer[2]

2.1 Funksjon og struktur

Programmet består i hovedsak av én klasse (`StatMon`). I tillegg lenkes det til biblioteket SoundKernel (se avsnitt 6), hvorifra det opprettes en instans av `SokServer` (avsnitt 6.5).

Programmet vedlikeholder blant annet følgende informasjon

- Liste over støyberegnere på lokal maskin (i minnet)
- Liste over regneceller på lokal maskin (i minnet)
- Liste over statusmonitorer på *andre* maskiner i nettverket (i minnet og i registeret)
- Liste over status for regneceller på *andre* maskiner i nettverket (i minnet)
- IP og port til "master" statusmonitor[3] (i minnet og i registeret)
- Flagget `IAmTheMaster` (i minnet)

Ved oppstart hentes oppsettet fra registeret (inkl. portnummer for socket-grensesnittet og standard portnummer for støyberegnere og regneceller). IP-adressen for lokal maskin bestemmes dersom den ikke allerede finnes i registeret. Dersom IP-adressen er feil, kan den settes fra brukergrensesnittet, via kommandoen "setIP".

Etter oppstart er all aktivitet initiert av `SokServer` sine kall til `DoCommand()`[4], samt to timere som hvert sekund anroper hhv. `tmrCheckSCCC_Elapsed()` og `tmrCheckRCC_Elapsed()`.

2.2 StatMon – Beskrivelse av metoder (et utvalg)

2.2.1 tmrCheckSCCC_Elapsed()

Denne timeren utfører følgende hvert sekund:

- Tømmer feilmeldingslista for gamle feilmeldinger (beholder maks. 1000 stk.)
- Løkker gjennom lista over støyberegnere
  - Henter status og informasjon for denne støyberegneren
  - Hvis kontakt med støyberegneren feiler, forsøkes gjenoppretting av kontakt
    - For hvert gjenopprettingsforsøk økes en teller
    - Gjenoppretting av kontakt oppgis dersom telleren overstiger 30
    - Hvis kontakt oppgis, avsluttes tilhørende regneceller (anroper `CloseCells()`)
  - Hvis status er forskjellig fra forrige status, oppdateres tidspunkt for siste statusendring
  - Hvis lista over støyberegnere ikke er endret på annet hold (via `DoCommand()`), blir den oppdatert med status og informasjon for denne støyberegneren
- Løkker gjennom lista over regneceller
  - Henter status og informasjon for denne regnecella
  - Hvis kontakt med regnecella feiler, forsøkes gjenoppretting av kontakt

[2] Finnes under HKEY_LOCAL_MACHINE/Software/SINTEF/StøyBeregner/
[3] "Master" statusmonitor har hoved- og eneansvar for tildeling av unike beregningsID til støyberegnere
[4] Se avsnitt 8.1 for en oversikt over hvilke kommandoer som støttes av `DoCommand()`.

- For hvert gjenopprettingsforsøk økes en teller
- Gjenoppretting av kontakt oppgis dersom telleren overstiger 30
- Hvis kontakt oppgis, fjernes regnecella fra lista over regneceller

- Hvis status er forskjellig fra forrige status, oppdateres tidspunkt for siste statusendring
- Hvis lista over regneceller ikke er endret på annet hold (via `DoCommand()`), blir den oppdatert med status og informasjon for denne regnecella

2.2.2 tmrCheckRCC_Elapsed()

Denne timeren utfører følgende hvert sekund:

- Løkker gjennom lista over statusmonitorer på andre maskiner
  - Henter IP og port for en statusmonitor på en annen maskin
  - Anroper denne statusmonitoren med kommandoen ”getCCInfo”
  - Legger statusinformasjon for hver regnecelle til listen over status for regneceller på andre maskiner

2.2.3 GetSC()

Bruker enten oppgitt beregningsID, eller anroper `GetNewTaskID()(`med `Local` = true`)` for å få tildelt en ny.

Løkker gjennom den interne lista over støyberegnere. Hvis en ledig støyberegner finnes, sjekkes det om den virkelig er ledig (”getStatus”). I så fall settes støyberegnerens beregningsID, og dens status settes til ”occupied”. Lista oppdateres med tilsvarende data.

Hvis ingen ledige støyberegnere ble funnet i lista, opprettes det (vanligvis) en ny. Det søkes først blant portnummer som ikke finnes i lista. Dersom et ledig portnummer finnes, opprettes en ny instans av støyberegner. Dersom porten derimot svarer på anrop, brukes kommandoen ”WhatAreYou” til å sjekke at det ikke er en ”glemt” støyberegner. Hvis det er tilfelle, og denne er ledig, kapres denne og blir lagt til lista over støyberegnere. I begge tilfeller blir beregningsID og status satt som over, og lista oppdateres med tilsvarende data.

2.2.4 GetCell()

Sjekker antall regneceller (på lokal maskin) mot antall CPU’er. Dersom det er flere CPU’er enn regneceller, opprettes det (vanligvis) en ny instans av regnecelle. På samme måte som for støyberegnere, sjekkes det først om neste ubrukte port er ledig eller i bruk av en ledig regnecelle som kan kapres. I alle tilfeller settes regnecellens status til ”occupied”, og den interne lista over regneceller oppdateres tilsvarende, inkl. beregningsID og deloppdragID.

Dersom alle CPU’er er i bruk, løkkes det gjennom lista over regneceller. Hvis en ledig regnecelle finnes, sjekkes det om den virkelig er ledig (”getStatus”). I så fall settes regnecellas status, og lista oppdateres som over.

Hvis alle CPU’er er i bruk, og ingen ledige regneceller kan finnes, og hvis `Local` = true, anropes `FindRemoteCell()`. Hvis anropet til `GetCell()` kommer fra en annen maskin vil `Local` være false. Anrop til andre maskiner blir altså *ikke* ”viderekoblet”.

Hvis ingen ledige regnecelle kunne finnes, verken på lokal maskin eller på andre maskiner, returneres ”0:0”.

2.2.5 FindRemoteCell()

Løkker gjennom lista over *andre* statusmonitorer, og anroper dem med kommandoen

GetCell false <beregningsID> <deloppdragID>

inntil alle er kontaktet, eller en av dem evt. returnerer gyldig IP:port.

2.2.6 CloseCells()

Løkker gjennom lista over regneceller og stenger ned regneceller som har oppgitt beregningsID. Dersom `Local` = true anropes også `CloseRemoteCells()`. Hvis anropet til `CloseCells()` kommer fra en annen maskin vil `Local` være false.

2.2.7 CloseRemoteCells()

Løkker gjennom lista over *andre* statusmonitorer, og anroper dem med kommandoen

CloseCells false <beregningsID>

inntil alle er kontaktet.

2.2.8 UpdateSMList()

Erstatter den interne lista over andre statusmonitorer med den oppgitte lista.

2.2.9 UpdateSMMaster()

Lagrer oppgitt statusmonitor som ”master” statusmonitor, dvs. den statusmonitor som har ansvar for tildeling av unike beregningsID’er. Dersom dette er lokal maskin settes flagget `IAmTheMaster` til true, ellers er dette flagget false.

2.2.10 GetNewTaskID()

Hvis flagget `IAmTheMaster` = true, hentes forrige beregningsID fra registeret og økes med én.

Hvis flagget `IAmTheMaster` = false, og `Local` = true sendes kommandoen

getNewTaskID false

til ”master” statusmonitor.

I begge tilfeller blir registeret oppdatert med den nye, siste beregningsID.

Hvis `IAmTheMaster` = false og `Local` = false flagges det feil (indikerer feil i oppsettet på lokal eller annen maskin).

3 StatusmonitorTray

Type: Kjørbart program (Tray-applikasjon)
Filnavn: StatusmonitorTray.exe
Grensesnitt: Grafisk brukergrensesnitt
Oppsett: Registeroppføringer

3.1 Funksjon og struktur

Programmet består av to klasser; én klasse (`MainForm`) som opprettes når programmet starter, og én klasse (`TrayForm`) som håndterer det grafiske brukergrensesnittet. Den første klassen håndterer komponenter som `NotifyIcon` (ikonet nederst til høyre på skjermen) og `ContextMenuStrip` (menyen som framkommer ved høyreklikk på ikonet). Når brukergrensesnitt-vinduet skal åpnes, opprettes en instans av `TrayForm` (hvis den ikke allerede eksisterer), og de to timerne `tmrGetNCCCInfo` og `tmrShowNCCCInfo` aktiveres. Disse timerne blir deaktivert når vinduet lukkes.

Ved oppstart av `TrayForm` hentes liste over statusmonitorer på *andre* maskiner i nettverket, samt IP og port til ”master” statusmonitor fra registeret.

Etter oppstart er all aktivitet i klassen `TrayForm` initiert av timerne `tmrGetNCCCInfo` og `tmrShowNCCCInfo`.

Se for øvrig avsnitt 10 for en beskrivelse av brukergrensesnitt-vinduet.

3.2 TrayForm – Beskrivelse av metoder (et utvalg)

3.2.1 tmrGetNCCCInfo_Elapsed()

En `Client`-forbindelse mot lokal statusmonitor (gjen-)opprettes (hvis den ikke allerede finnes og er i live). `CellProxy.Comm2Cell()` anropes med `Client`’en og kommandoene

GetNCInfo
GetCCInfo
GetRCCInfo

for å hente statusinformasjon for hhv. støyberegnere, og regneceller på lokal maskin og andre maskiner. Resultatene fra kallene lagres i variablene `NCInfo`, `CCInfo`, `RCCInfo`.

3.2.2 tmrShowNCCCInfo_Tick()

Først hentes og dekodes informasjonen i `NCInfo`. Informasjonen brukes til å fylle ut diverse `Label`’er i `TabPage`’en ”Støyberegnere”. Deretter hentes og dekodes informasjonen i `CCInfo` og `RCCInfo`. Denne informasjonen brukes til å fylle ut `ListView`-tabellen i `TabPage`’en ”Regneceller”.

3.2.3 cmdTerminate_Click()

Denne metoden aktiveres ved klikk på knappen ”Avslutt”. Etter at bruker bekrefter at han ønsker å stenge ned støyberegneren, anropes `CellProxy.Comm2Cell()` med `Client`’en som er koblet opp mot statusmonitoren og kommandoen

CloseCalculation <beregningsID>

3.2.4 cmdMulti_Click()

Denne metoden aktiveres ved klikk på en av knappene ”Start”, ”Pause”, ”Stopp”, ”Nullstill”. Etter at bruker bekrefter at han ønsker å utføre kommandoen, opprettes en `Client` mot støyberegneren. Deretter anropes `CellProxy.Comm2Cell()` med `Client`’en og den kommandoen som skal utføres;

StartCalc
PauseCalc
StopCalc
ClearCalc

Siden de to siste kommandoene kan ta litt lang tid å utføre (de må begge vente på at kall til `CellProxy.CloseMyCells()` skal returnere), settes ekstra lang timeout i disse tilfellene.

3.2.5 cmdUpdateSM_Click()

Denne metoden aktiveres ved klikk på knappen ”Oppdater SM”. Etter at bruker bekrefter at han ønsker å oppdatere statusmonitoren med informasjonen i `TabPage`’en ”Statusmonitorer”, anropes `CellProxy.Comm2Cell()` med `Client`’en som er koblet opp mot statusmonitoren og kommandoene

SetLocalCalc <True | False>
SetIP <IP>
UpdateSMList <SMListe>
UpdateSMMaster <IP> <vertsnavn> <port>

4 Støyberegner

Type: Kjørbart program uten grafisk brukergrensesnitt
Filnavn: Stoyberegner.exe
Grensesnitt: Socket (port 51000 og høyere)
Filer (shp, asc, xml)
Oppsett: Registeroppføringer og kommandolinjeargumenter

4.1 Funksjon og struktur

Programmet består i hovedsak av en klasse (`NoiseCalculator`) som oppretter instanser av fire klasser med definerte oppgaver (`Preprocess`, `Splitter`, `Collector`, `Results`). Programmet har også en statisk klasse (`Database`). I tillegg lenkes det til biblioteket SoundKernel (se avsnitt 6), hvor den statiske klassen `SK` holder felles variable, den statiske klassen `CellProxy` håndterer kommunikasjon med statusmonitor og regneceller, og hvorifra det opprettes en instans av `SokServer`.

Ved oppstart hentes standard databasekatalog fra registeret, mens portnummer for `SokServer` hentes fra første kommandolinjeargument. Andre kommandolinjeargument inneholder normalt portnummer til statusmonitor. Hvis en eller begge disse er utelatt vil standardverdier være hhv. 51000 og 50000.

Etter oppstart er all aktivitet i klassen `NoiseCalculator` initiert av `SokServer` sine kall til `DoCommand()`[5] og `KillMe()`. I de andre klassene er det diverse timere og separate tråder som blir styrt av en rekke tilstandsflagg.

4.2 NoiseCalculator – Beskrivelse av metoder (et utvalg)

4.2.1 KillMe()

Anroper `ClearModules(CloseAllCellsToo` = true`)` for å nullstille alle underklasser og avslutte tilhørende regneceller. Avslutter så applikasjonen.

4.2.2 ClearModules()

Setter `SK.MainStatus` til ”interrupted” for å initiere avslutning av aktiviteten i trådene i undermodulene. Anroper `.Clear()` i alle klasser. Kallene blokkerer inntil aktiviteten i hver klasse er avsluttet. Hvis flagget `CloseAllCellsToo` er true, anropes `CellProxy.CloseMyCells()` for å stenge ned tilhørende regneceller. Setter så `SK.MainStatus` til ”free”.

4.2.3 GetElapsed()

Returnerer total tid fra beregningen ble startet til nåværende tidspunkt, eller evt. til beregningen ble avsluttet eller var ferdig. Returnerer også samlet CPU-tid, ved å løkke gjennom intern liste over deloppdrag og summere medgått CPU-tid for hvert deloppdrag.

4.2.4 GetRemaining()

Løkker gjennom intern liste over deloppdrag (dog bare deloppdrag med fin presisjon). Summerer CPU-tidsforbruk og antall mottakerpunkt for ferdige deloppdrag, og antall mottakerpunkt for gjenstående deloppdrag (som skal beregnes med fin presisjon). Estimerer gjenstående CPU-tidsforbruk ved lineær ekstrapolering, og returnerer dette.

[5] Se avsnitt 8.3 for en oversikt over hvilke kommandoer som støttes av `DoCommand()`.

4.2.5 MakeDocument()

Oppretter et `XMLDocument`, og legger til elementer for `NoiseCalculator`, inkl. feilmeldinger i `SokServer` og `Database`. Anroper deretter `.Doku()` i klassene `Preprocess`, `Splitter`, `Collector`, og `Results` for å legge til elementer for disse klassene. Lagrer XML-dokumentet til "report_<ID>.xml", hvor <ID> er beregningsID.

4.3 Preprocess

Denne klassen er ansvarlig for innlesning av grunnlagsdata og oppdragsbeskrivelse. Aktiviteten i klassen er styrt av en timer som hvert sekund anroper `CheckStatus_Elapsed()`. Denne sjekker flaggene `StartReading` og `StartReLoad` (settes av kommandoene "Preprocess" og "ReLoad"), og hvis et av flaggene er lik true, anropes hhv. `ReadFiles()` og `ReLoadByTaskID()`.

4.3.1 ReadFiles()

Løkker gjennom de kjente filtypene[6]. For hver filtype søkes i `SK.MainPath` etter filer. For hver fil igangsettes passende metode for innlesing (`ProcessRasterFile()`, `ProcessShapeFile()`, `ProcessXMLFile()`). Etter at alle filer er lest inn, sjekkes det at minimumskravet til en beregning er oppfylt: minst én kilde, en oppdragsspesifikasjon med minst ett mottakerpunkt, og en topografi. Hvis det ikke oppsto feil under innlesning lagres det innleste beregningsgrunnlaget til filer[7] på internt format til den interne databasen (se avsnitt 4.7), og `SK.DataInStatus` settes til "finished".

4.3.2 ReLoadByTaskID()

Henter (fra den interne databasen), et eksisterende beregningsgrunnlag med gitt beregningsID. Dersom liste over deloppdrag finnes for gitt beregningsID, leses også denne inn, og `UpdateSubTasksStatus()` anropes. Hvis ingen feil oppsto settes tilslutt `SK.DataInStatus` til "finished".

4.3.3 UpdateSubTasksStatus()

Løkker gjennom listen over deloppdrag (`SK.SubTasks`), leser inn hvert deloppdrag fra databasen, og oppdaterer status, presisjon og CPU-tidsforbruk i listen over deloppdrag.

Merk at status i selve deloppdraget og status i lista over deloppdrag følger to ulike prinsipper; status i deloppdraget er "absolutt", det vil si at deloppdraget faktisk har denne statusen. Status i lista er mer reversibel, det vil si at deloppdraget kan være i overgangen fra en "absolutt" status til en annen, og kan evt. under visse betingelser (feil) gå tilbake til forrige status. Status i deloppdraget følger alltid rekkefølgen:

"not_started" ➔ "finished_coarse" ➔ "finished"

Status i lista over deloppdrag følger *i utgangspunktet* rekkefølgen:

"not_started" ➔ "is_calculating_coarse" ➔ "finished_coarse"

[6] Filtypene leses inn i denne rekkefølgen: **Task** (oppdrag), **FacP** (fasadepunkt), **TopG** (topografigrid), **TopP** (topografipunkt), **TopS** (marktypeflater), **TopF** (skogflater), **Scrn** (skjermer), **Mnd** (voller) , **Road** (veger), **BldS** (punkt-bygninger), **BldE** (bygningsomriss), **BldL** (bygningstaklinjer), **Srce** (kilder), **MET** (meteorologidata).

[7] Det lagres filer for samlet topografi, oppdragsspesifikasjon og kilder.

Deretter (etter signifikanstest, se avsnitt 4.5.2):

➔ ”finished” *eller*
➔ ”to_be_recalculated_fine” ➔ ”is_calculating_fine” ➔ ”finished”

4.3.4 ProcessRasterFile

Leser inn en fil på ArcInfo .ASC-format. Tilordner data til rett klasse basert på filtype. Inntil videre er bare høydegrid, TopG, støttet:

Én TopG-fil ➔ Én instans av `SK.skTopGrid`

4.3.5 ProcessShapeFile()

Leser inn en fil på ESRI Shape-format. Avhengig av geometritype (punkt, polylinjer, polygoner, etc) anropes metodene `ReadShpPntZ()` og `ReadShpPntZ()` med ulike flagg satt. Begge disse metodene utfører i sin tur:

- Innlesing av indeksfil (.shx) for å holde rede på start av hver oppføring
- Innlesing av geometrifil (.shp) for å definere geometri for hver oppføring
- Innlesing av attributtfil (.dbf) for evt. å knytte attributter til hver oppføring

Basert på filtype blir geometri og evt. attributter brukt til å opprette og konfigurere en eller flere instanser av interne klasser som svarer til de ulike filtypene:

- Én FacP-fil ➔ Utvider lister i `SK.Task` for å gi plass til nye mottakerpunkt. Geometri for hvert punkt lagt til listen, med bygningsID gitt av attributter
- Én TopP-fil ➔ Én instans av `SK.skTopPoints` med punktenes høyde hentet fra geometri, og markslag / skoghøyde / ruhet gitt av attributter
- Én TopS-oppføring ➔ Én instans av `SK.skTopSurface` med markslag gitt av attributter
- Én TopF-oppføring ➔ Én instans av `SK.skTopSurface` med skoghøyde gitt av attributter
- Én Scrn-oppføring ➔ Én instans av `SK.skTopScreen` per del-polylinje
- Én Mnd-oppføring ➔ Én instans av `SK.skTopScreen` per del-polylinje
- Én Road-oppføring ➔ Én instans av `SK.skTopRoad`.
- Én BldS-oppføring ➔ Én instans av `SK.skBuilding` (enkel bygning)
- Én BldE-oppføring ➔ Én instans av `SK.skBuilding` (generell bygning). Geometri for omrisspolygoner blir lagt til. ID for bygningen og instans-nummer blir lagt til en `Hashtable` som heter `TopBldsIdx`
- Én BldL-oppføring ➔ Nummer for rett instans av `SK.skBuilding` blir hentet fra `TopBldsIdx`, basert på ID for bygningen. Geometri for taklinjer blir lagt til

4.3.6 ProcessXMLFile

Leser inn en XML-fil i et internt `XMLDocument`. Tilordner data til rett klasse basert på filtype:

- Én Task-fil ➔ Én instans av `SK.Task`. Alle elementer legges til denne instansen, unntatt standardverdi-elementene, som blir lagt til tabellen `SK.Defaults`.

- Én Srce-fil ➔ Én instans av `SK.skSource` per `<source>`-element.

I senere versjoner vil meteorologi (MET) også angis på XML-format.

4.4 Splitter

Aktiviteten i denne klassen er styrt av en timer som hvert sekund anroper `CheckStatus_Elapsed()`. Denne gjør ikke noe før innlesing av beregningsgrunnlag er ferdig (`SK.DataInStatus` = ”finished”). Hvis splitting av oppdrag i deloppdrag ikke er ferdig (og heller ikke påbegynt) anropes `Split()`. Hvis splitting er ferdig blir aktiviteten i en separat tråd, `threadFunction()` regulert av `SK.MainStatus`.

4.4.1 Split()

Denne og tilhørende metoder besørger oppdeling av et beregningsoppdrag i mange deloppdrag. Hvis oppdraget har definert en mottakergrid, blir denne delt inn i delgrider på opptil 16 x 16 punkt. For hver delgrid anropes `CreateSubTasks()` for å lage deloppdrag for delgridet. Hvis oppdraget har definert spredte mottakerpunkt (fasadepunkt, etc.), anropes `CreateSubTasks()` for å lage deloppdrag basert på kildene som er definert i oppdraget. Når alle deloppdrag er generert uten feil, settes `SK.SplitterStatus` = ”finished”.

4.4.2 CreateSubTasks()

Dette er to ulike metoder; en for deloppdrag basert på delgrid, og en for deloppdrag basert på kilder.

For deloppdrag basert på delgrid blir det lagt til et rektangulært område som ligger minst 300 meter utenfor delgridet. Alle kilder som ligger helt eller delvis innenfor dette området blir inkludert i genereringen av deloppdrag. `CreateSubTopography()` anropes for evt. å opprette en subtopografi for deloppdraget. Deretter anropes `CreateSingleSubTask()`for å opprette ett deloppdrag for hver av de inkluderte kildene.

For deloppdrag basert på kilder blir det for hver kilde definert et rektangulært område som ligger minst 300 meter utenfor kildens utstrekning. Innenfor dette området søkes det etter spredte mottakerpunkt. `CreateSubTopography()` anropes for evt. å opprette en subtopografi for deloppdraget. Deretter anropes `CreateSingleSubTask()`for å opprette ett deloppdrag for hver kilde og for alle spredte mottakerpunkt som ligger innenfor området. Det opprettes ikke noe deloppdrag dersom det ikke er noen spredte mottakerpunkt innenfor området.

I begge tilfeller anropes `UpdateSingleSubTask()` for å sjekke om deloppdraget allerede er beregnet, og evt. gjenbruke resultatene.

Tilslutt blir listen over deloppdrag oppdatert med diverse informasjon og status om hvert deloppdrag.

4.4.3 CreateSubTopography()

For hvert deloppdrag blir det opprettet en subtopografi. Det legges til en margin på 100 meter utenfor den kombinerte utstrekningen til kilden og delgridet/mottakerpunktene. For å kunne gjenbruke flest mulig subtopografier blir koordinatene for utstrekningen til subtopografien avrundet utover til nærmeste hele kilometer. Koordinatene blir sjekket mot en tabell over subtopografier. Dersom subtopografien ikke allerede finnes der, anropes `skTopography.GetSubTopography()` for å hente ut de deler av topografien som ligger innenfor det gitte området. Koordinatene blir lagret i tabellen over subtopografier.

4.4.4 UpdateSingleSubTask()

For et gitt deloppdrag sjekkes det mot databasen om deloppdraget allerede finnes. I så fall hentes det ”gamle” deloppdraget fra databasen. Hvis status for det ”gamle” deloppdraget” er ”finished” eller ”finished_coarse” kan resultatene gjenbrukes, og `AdjustToNewQ()` anropes for å oppdatere resultatene i tråd med angitt mengde for kilden i det ”nye” deloppdraget. Siden det ”gamle” og ”nye” deloppdraget er like i alle andre henseender enn mengde, kan det ”gamle”, nå oppdaterte, deloppdraget lagres tilbake til databasen.

4.4.5 threadFunction()

Denne metoden besørger utsending av deloppdrag til regneceller. Den løkker gjennom listen over alle deloppdrag inntil

- *enten* `SK.MainStatus` settes til ”interrupted”, ”finished” eller ”error”
- *eller* alle deloppdrag er ferdig beregnet (eller har feil)
- *eller* det har gått en time uten at noen ledige regneceller er funnet

For hvert deloppdrag utføres følgende

- Status sjekkes; hvis status for deloppdraget er verken ”not_started” eller ”to_be_recalculated_fine”, gjøres det ikke noe mer med deloppdraget
- Statusmonitoren anropes for å finne en ledig regnecelle (`CellProxy.FindFreeCell()`). Hvis ingen ledig regnecelle kunne finnes, venter tråden i ett sekund før en ny gjennomløkking av deloppdragene begynner
- Status for deloppdraget endres til ”is_calculating_coarse” eller ”is_calculating_fine”
- Det opprettes en `Client` for kommunikasjon med regnecella
- Regnecella kontaktes og hvis korrekt subtopografi ikke finnes i regnecella fra før, oversendes denne (`CellProxy.File2Cell()`)
- Regnecella kontaktes og deloppdrag oversendes (`CellProxy.File2Cell()`)
- Regnecella kontaktes og beregning igangsettes (`CellProxy.Comm2Cell()`, med kommandoen ”StartCalc”)
- Enhver feil som måtte oppstå fører til at deloppdragets status i lista over deloppdrag settes tilbake til forrige status (altså ”not_started” eller ”to_be_recalculated_fine”)

4.5 Collector

Aktiviteten i denne klassen er styrt av en timer som hvert sekund anroper `CheckStatus_Elapsed()`. Denne gjør ikke noe før splitting av oppdrag i deloppdrag er ferdig (`SK.SplitterStatus` = ”finished”). Hvis splitting er ferdig blir aktiviteten i en separat tråd, `threadFunction()` regulert av `SK.MainStatus`.

4.5.1 threadFunction()

Denne metoden besørger innsamling av ferdige deloppdrag fra regneceller. Den løkker gjennom liste over alle deloppdrag inntil

- *enten* `SK.MainStatus` settes til ”interrupted”, ”finished”, ”error”
- *eller* alle deloppdrag er ferdig beregnet (eller har feil)

For hvert deloppdrag utføres følgende

- Status sjekkes; hvis status for deloppdraget er verken ”is_calculating_coarse” eller ”is_calculating_fine”, gjøres det ikke noe mer med deloppdraget
- Det opprettes en `Client` for kommunikasjon med regnecella som har deloppdraget
- Regnecella kontaktes for å sjekke status for beregningen (`CellProxy.GetStatus()`); hvis beregningen pågår ennå gjøres det ikke mer med deloppdraget
- Deloppdraget leses inn fra regnecella (`CellProxy.Cell2File()`) og lagres i databasen (`Database.PutFile()`)
- Enhver feil som måtte oppstå fører til at deloppdragets status i lista over deloppdrag settes til ”error”
- Regnecella nullstilles (`CellProxy.ClearCell()`) hvis deloppdraget var ferdig beregnet eller hvis det oppsto en feil.

Når alle deloppdrag har status ”finished”, ”finished_coarse” eller ”error”, sjekkes det om det er noen deloppdrag som har status ”finished_coarse”. I så fall anropes `SignificanceCheck()` for å utføre en signifikanstest. Etter at dette kallet returnerer, restartes løkka over deloppdrag. Når (og hvis) alle deloppdrag har status ”finished” settes `SK.CollectorStatus` = ”finished”.

4.5.2 SignificanceCheck()

Denne metoden sjekker om et deloppdrag er signifikant, det vil si om deloppdragets støykilde bidrar signifikant til de beregningspunktene som deloppdraget omfatter. Deloppdrag som bidrar signifikant blir beregnet om igjen, mer nøyaktig (og mer tidkrevende). Ikke-signifikante deloppdrag blir flagget som ferdige. Metoden består av to deler:

- Innlesing av totalresultat og validering av alle deloppdrag
- Signifikanstesting av hvert deloppdrag

I første del løkkes det gjennom listen over alle deloppdrag. Hvis et deloppdrag har status ”error” blir de mottakerpunkt[8] i oppdraget som svarer til deloppdragets mottakerpunkt, flagget som ”skitne”. Deloppdraget behandles da ikke videre. Hvis et deloppdrag har status ”finished” eller ”finished_coarse”, men svarer til et eller flere ”skitne” punkt i oppdraget, blir deloppdraget heller ikke behandlet videre[9]. Ikke-”skitne”, ferdige deloppdrag blir lest inn fra databasen, og delresultatene blir samlet til resultat for oppdraget.

Deretter blir hvert mottakerpunkt i samlet resultat flagget som ikke-signifikant dersom resultatene for alle støyenheter ligger mer enn 10 dB under nedre beregningsterskel (som vanligvis er 50 dB).

Så løkkes det gjennom listen over alle deloppdrag på nytt. Følgende deloppdrag blir ikke behandlet:

- Deloppdrag med status ”finished” eller ”error”
- Deloppdrag som svarer til et eller flere ”skitne” punkt i oppdraget

For deloppdrag hvor *alle* tilsvarende mottakerpunkt i oppdraget er ikke-signifikante, blir status i listen over deloppdrag satt til ”finished”; ingen videre signifikanstesting utføres. De resterende deloppdragene blir lest inn fra databasen og testet. Hvert deloppdrag blir i utgangspunktet tildelt status ”finished”. Hvis følgende test er sann for minst ett mottakerpunkt og minst én støyenhet settes status i stedet til ”to_be_recalculated_fine”:

[8] Mottakerpunkt brukes her både om gridpunkt og spredte punkt.
[9] Det har nemlig ingen hensikt å utføre signifikanstesting dersom bidraget fra et eller flere deloppdrag ikke er inkludert pga. feil i deloppdraget.

$$\frac{\sum_{i=1, i\neq j}^{N} p_i^2 + 10\,p_j^2}{\sum_{i=1}^{N} p_i^2} > 1.05,$$

Hvor p_i^2 er proporsjonal med lydenergien i punktet fra det *i*'te deloppdraget, *j* er indeks til deloppdraget som testes og *N* er antall deloppdrag som er inkludert.

4.6 Results

Aktiviteten i denne klassen er styrt av en timer som hvert sekund anroper `CheckStatus_Elapsed()`. Denne gjør ikke noe før alle deloppdragene er ferdig beregnet (`SK.CollectorStatus` = "finished"). Hvis resultatene ikke er lagret (`SK.SavedStatus` = "not_finished"), anropes `SaveResults()`. Hvis lagringen var vellykket settes `SK.SavedStatus` = "finished" og `SK.MainStatus` = "finished". ***Beregningen er da ferdig.***

4.6.1 SaveResults()

Først hentes alle deloppdragene fra databasen, og samles til et totalresultat for oppdraget. Totalresultatet konverteres til dB. Negative totalresultat angir feil eller manglende resultat, og flagges med spesielle dB-verdier. Deretter anropes `SaveUnit()` for å lagre resultater for hver enkelt støyenhet til separate filer.

4.6.2 SaveUnit()

Koordinater og resultater for spredte mottakerpunkt lagres til en shp-fil. Støyverdiene lagres i "measure"-verdien for hvert punkt. Attributter (for eksempel punkt-ID) lagres til den tilhørende dbf-filen.

Griddefinisjon og resultater for gridpunkt lagres til en asc-fil. For å unngå problemer med desimalskilletegnkonvensjoner er alle verdier angitt som heltall. Støyverdiene er angitt i cB.

4.7 Database

Denne statiske klassen inneholder metoder for å legge til og hente ut innhold fra en intern database. Databasen lagrer oppdragsbeskrivelse, grunnlagsdata, deloppdrag, osv. Allerede innleste oppdrag kan dermed hentes fram basert på beregningsID, og allerede beregnete deloppdrag kan hentes fram basert på hash-kode for deloppdraget, slik at unødig gjentatt beregning unngås. Databasen er implementert som en hierarkisk katalogstruktur. Innholdet er binære filer, som er en dump av de enkelte klassenes .Save()-metoder. Se kapittel 6 for en beskrivelse av klassene.

Filklasse (enFC)	Innhold	Underkatalog
`MET`	En instans av `skMetClass`	MET
`METList`	En liste av `skMetClass`'er	METLST
`Source`	En instans av `skSource`	SRCE
`SourceList`	En liste av `skSource`'er	SRCELST
`SubTask`	En instans av `skSubTask`	STSK
`SubTaskList`	En liste av `typSubTask`'er	STSKLST
`SubTopo`	En instans av `skTopography`	STOP
`Task`	En instans av `skTask`	TASK
`Topo`	En instans av `skTopography`	TOPO

Metodene `PutFile()`, `GetFile()` og `FileExists()` lagrer, henter og sjekker eksistensen til en fil basert på en hash-kode som (nesten) unikt identifiserer objektet som filen inneholder. Metodene `PutFileByID()`, `GetFileByID()` og `FileExistsByID()` lagrer, henter og sjekker eksistensen til en fil basert på oppgitt beregningsID.

5 Regnecelle

Type: Kjørbart program uten grafisk brukergrensesnitt
Filnavn: Regnecelle.exe
Grensesnitt: Socket (port 52000 og høyere)
Oppsett: Kommandolinjeargumenter

5.1 Funksjon og struktur

Programmet består av én klasse (`CalcCell`). I tillegg lenkes det til biblioteket SoundKernel (se avsnitt 6), hvor den statiske klassen `SK` holder felles variable, og hvorifra det opprettes en instans av `SokServer` (avsnitt 6.5).

Ved oppstart hentes portnummer for `SokServer` fra første kommandolinjeargument. Hvis denne er utelatt vil standardverdi være 52000.

Etter oppstart er all aktivitet i klassen `CalcCell` initiert av `SokServer` sine kall til `DoCommand()`[10], `Data2Object()` og `KillMe()`, samt en timer som hvert sekund anroper `CheckStatus_Elapsed()`. Sistnevnte sjekker hvert sekund `SK.MainStatus` og regulerer aktiviteten i en separat tråd, `SK.RunSubTask()`, basert på denne statusen.

5.2 CalcCell – Beskrivelse av metoder (et utvalg)

5.2.1 Data2Object()

Etter at `SokServer` har mottatt et binært objekt (via kommandoen ”PutObject”, se 6.5.3), overføres en `MemoryStream` til denne metoden, samt en enumerator som angir hvilken klasse objektet tilhører. Basert på dette anropes klassens `.Load()`-metode for å opprette en ny instans av klassen, med innhold gitt av det binære objektet.

5.2.2 KillMe()

Anroper `ClearModules()` for å nullstille alle underklasser. Avslutter så applikasjonen.

[10] Se avsnitt 8.4 for en oversikt over hvilke kommandoer som støttes av `DoCommand()`.

6 SoundKernel

Type: Rutinebibliotek
Filnavn: SoundKernel.dll

Dette rutinebiblioteket inneholder en rekke klasser som brukes av flere av komponentene i støyberegningsløsningen. Det inneholder klasser som definerer de objektene som utgjør et komplett beregningsgrunnlag, slik som terreng, veger, skjermer og spesifikasjon av beregning. Videre har det metoder som diskretiserer beregningsgrunnlaget ned til isolerte punkt-til-punkt-tverrsnitt som overføres til ProP2P (se avsnitt 7) for beregning av lydutbredelse. De viktigste klassene er kort beskrevet under.

6.1 SK

Dette er en statisk klasse som inneholder alle variable som er delt mellom andre klasser (for eksempel status-flagg og beregningsgrunnlag). Den inneholder også enkelte felles rutiner (for eksempel rutiner for feil-logging og lesing/skriving av filer). Den inneholder også metoden `RunSubTask()` som kjøres som en egen tråd av regnecelle-applikasjonen.

6.1.1 RunSubTask()

Denne metoden behandler deloppdraget og subtopografien innlest av regnecella. For hvert mottakerpunkt i deloppdraget bestemmes posisjon og høyde, og `RoadSourceToPoint()` anropes for å bestemme støybidraget til punktet fra deloppdragets kilde.

6.1.2 RoadSourceToPoint()

Denne metoden bestemmer støybidraget fra én kilde (veg) til ett mottakerpunkt.

- Støydata for kilden blir bestemt i henhold til kildedelen av Nord 2000 Road. Dette skjer ved kall til rutinene `RoadSpectra()` og `BaseRoadSpectra()`.
- Metoden `skTopography.BuildMirrors()` anropes for å bestemme alle refleksjonsbanene fra et sett representative punkter langs vegen til beregningspunktet.
- Kildepunktene splittes ytterligere for å representere kjørefelt, hjulspor og delkildehøyder for kjøretøy.
- Terrengprofiler bestemmes for lydutbredelsen fra hvert kildepunkt til mottakerpunktet.
- `PointSourceToPoint()` anropes for å beregne støybidraget fra hvert representative kildepunkt til beregningspunktet.
- Støybidraget fra hvert kildepunkt blir akkumulert opp til samlet støy i beregningspunktet fra kilden.

6.1.3 PointSourceToPoint()

Denne metoden beregner støybidraget fra et generalisert kildepunkt til et mottakerpunkt over en oppgitt terrengprofil, for hver av de ønskede enhetene. Dette gjøres ved hjelp av algoritmene for luftabsorpsjon, skjermdemping og bakkedemping som er definert i Nord 2000 Road, slik de er implementert i rutinene `AirAbs()` og `N2kR()` i biblioteket ProP2P (se avsnitt 7).

6.2 skTopography

Inneholder en topografi eller en subtopografi, dvs. lister med instanser av `skTopPoints`, `skTopGrid`, `skTopSurface`, `skTopScreen`, `skTopRoad` og `skTopBuilding`. Inneholder også en instans av `skTopIdx`. Se 6.8 for en kort beskrivelse av disse klassene.

Klassen inneholder en `Hashtable` (`TopRoadsIdx`) som brukes til å koble en veg-ID til indeks i listen med instanser av `skTopRoad`. Dette er nødvendig for at en kilde skal kunne kobles til den korrekte fysiske vegen, uavhengig av i hvilken rekkefølge vegene er lest inn.

6.2.1 GetSubTopography()

Denne metoden returnerer en ny instans av `skTopography`, basert på et oppgitt rektangel. Det løkkes gjennom listene med instanser av `skTopPoints`, `skTopGrid`, `skTopSurface`, `skTopScreen`, `skTopRoad` og `skTopBuilding`, og bare objekter som helt eller delvis ligger innenfor det oppgitte rektangelet blir inkludert i den nye subtopografien. For `skTopPoints` og `skTopGrid` blir instanser som ligger delvis utenfor rektangelet klippet, slik at de punkter eller deler av et grid som ligger utenfor ikke blir inkludert i subtopografien.

6.2.2 BuildSection()

Dette er en metode som beregner et vertikalt terrengsnitt mellom et to punkter. Rutinen finner først profilet for grunnterrenget ved kall til terrengegridets metode for profilgenerering (`skTopGrid.GetSection()`). Deretter gjennomgår rutinen alle øvrige berørte geografiske objekter (terrengoverflater, veger, skjermer og bygninger), og henter ut delprofiler fra hvert av dem. Disse delprofilene klippes så inn i det opprinnelige terrengprofilet ved hjelp av metoden `JoinProfile()`.

6.2.3 JoinProfile()

Denne metoden legger et tverrsnitt av et objekt inn i et grunnleggende terrengprofil. Hvordan dette gjøres, er avhengig av typen objekt:

- Terrengoverflater påvirker markslaget innenfor profilets grenser. Terrenghøyden blir justert kun når objektets flagg for dette er satt.
- Veger klippes inn i terrengprofilet, og erstatter alle eksisterende data mellom profilets grenser. Det settes inn soner på hver side av vegen (skjæring/fylling) for å justere inn eventuelle høydeavvik.
- Skjermer og voller klippes inn i profilet på samme måte som for veger. Også her vil det ved behov settes inn soner på hver side for å justere inn avvik i vollhøyde og/eller bakkehøyde ved foten av skjermen.
- Bygninger klippes inn i profilet. Veggene antar å være vertikale, og avsluttes i skjeringspunktet mot hovedprofil.

6.2.4 AppendSection()

Brukes til å koble sammen to terrengprofiler som skal henge sammen. Dette brukes for de linjestykkene som utgjør refleksjonsveiene dersom refleksjoner er involvert.

6.2.5 BuildMirrors()

`BuildMirrors()` er en metode som anvender klassen `skMirror2D` til å bygge opp et sett med lydbaner fra en linjekilde til et mottakerpunkt i omgivelser med bygninger og skjermer. Denne klassen er en omfattende modul som løser oppgaven med å beregne refleksjoner ved hjelp av såkalt BeamTracing fra mottakerpunktet mot linjekilden. All geometrisk betraktning skjer i x/y-planet, altså 2D. Det tredje dimensjonen (høyden) regnes inn på et senere tidspunkt, i forbindelse med etablering av terrengprofiler. Det forutsettes derfor at alle reflekterende flater er vertikale, med vertikale grenser (hjørner og endepunkter). Ved søk etter refleksjonsveier, og senere beregning av lydrefleksjon tas det hensyn til prinsippet om Fresnel-soner som er nedfelt i Nord 2000. Fresnel-sonen dekomponeres i en horisontal del (i denne rutinen) og en vertikal del (senere behandling).

Sentrale metoder i klassen `skMirror2D` er:

- `TraceBeams()`, som søker i hierarkisk trestruktur etter speil og bakenforliggende speil, med utgangspunkt i beregningspunktet. For å begrense oppgaven avsluttes søket for hver gren i treet dersom:
    - Det oppstår mer enn 3 speil (inntil 3. ordens refleksjon)
    - Samlet horisontal utbredelseslengde overstiger 1000 meter
    - Speilets åpningsvinkel mot bakenforliggende speil eller kilder er mindre enn 2 grader.

- `TraceRecievers()`, som søker gjennom alle speilsoner (Beams) for å finne om den belyser (krysser) hele eller en del av linjekilden. Alle slike deler av kildelinja som belyses diskretiseres etter de kriterier for vinkeløpning og innbyrdes avstandsforskjell som er nedfelt i Nord 2000 Road. Disse utgjør potensielle punktkilder for beregningen. Hvert kildepunkt tilordnes en andel av støykilden tilsvarende lengden av den vegbiten den representerer.

- `TraceTracks()` er en metode som trekker alle linjene fra kildepunktene til beregningspunktet, via de respektive speilene. For å begrense antall nødvendige senere beregninger med Nord 2000 kjernen, gjøres følgende forenkling:
    - Dersom mer enn ett linjestykke mellom kilden og mottakeren er fullstendig blokkert av en skjerm eller bygning, ignoreres tilhørende kildepunkt.
    - Dersom to eller flere linjesett mellom sine respektive kilder og felles mottaker er tilstrekkelig like, slås de sammen til ett. Den ene av dem overtar kildebidraget fra den/de som faller bort. Kriteriene for likhet er satt til:
        - Samme antall linjestykker (samme refleksjonsorden)
        - Samme antall kryssende skjermer/bygninger
        - Horisontal vinkelforskjell ved mottakeren er mindre enn 5 grader
        - Lengden for hvert linjestykke avviker med mindre enn 25 %

6.3 skGeo

Dette er en statisk klasse som inneholder en samling metoder for geometriske beregninger, for eksempel test av om et punkt er innenfor et polygon, krysningspunkt mellom to linjer, beregning av vektor basert på endepunkt, og lignende.

6.4 skProp

Dette er en statisk klasse som fungerer som nærmeste overbygning til Nord 2000 Road biblioteket (se avsnitt 7). Den primære oppgaven er å oversette detaljene i et beregningsoppdrag til Nord 2000 Road terminologi, og kjøre hovedrutinen for Nord 2000. Ved oversettelse av inngangsdata gjøres følgende tilpasninger:

- Nord 2000 markslag mappes til strømningsresistans
- Ruhetslengde settes til 0.025
- Standardavvik for vind settes til 0
- Standardavvik for temperatur settes til 0
- Antall scatteringsoner settes til 0. *Dette betyr at skogsdemping ikke beregnes*
- Vertikale linjer i terrengprofilet skråstilles tilsvarende minst 25 cm horisontal lengde

6.5 SokServer

Denne klassen oppretter og vedlikeholder en generell socket server. Klassen har en timer som fire ganger per sekund anroper `tmrServer_Tick()` for å sjekke om klienter ønsker kontakt, og for å sjekke status for eksisterende oppkoblinger.

Ved oppstart av klassen defineres tre delegates:

- `CommandDelegate()`. Formidler en kommando
- `Data2ObjectDelegate()`. Formidler et binært objekt
- `KillMeDelegate()`. Formidler anmodning om å stenge ned applikasjonen

Når en klient oversender en kommando (terminert med ”\r\n” eller ”\n”) anropes vanligvis `CommandDelegate()`. Resultatet fra denne er en byte array, som returneres til klienten ved hjelp av `Send()`. Unntaket fra dette er kommandoen ”Exit”, som stenger ned forbindelsen på en pen måte, og kommandoen ”PutObject” (se 6.5.3), som anroper `Data2ObjectDelegate()`.

6.5.1 tmrServer_Tick()

Først sjekkes det om det er klienter som ønsker kontakt (`TcpListener.Pending()`). I så fall opprettes en asynkron forbindelse (`Socket`) til denne klienten, og det opprettes en ny instans av `clsLine`. Denne klassen inneholder selve `Socket`-instansen og all informasjon om forbindelsen. Klassen blir lagt til en liste over forbindelser (`Lines`).

Deretter sjekkes det hvor lang tid det er gått siden siste kommunikasjon med en klient. Dersom denne tiden overskrider størrelsen `TimeToKill` (vanligvis 90 sekunder) anropes `KillMeDelegate()`. Hvis `TimeToKill` er null anropes aldri `KillMeDelegate()`.

Til slutt løkkes det gjennom listen over forbindelser (`Lines`). For hver forbindelse sjekkes det om forbindelsen til klienten er ”i live”, og at ikke tid siden siste kommunikasjon med klienten overskrider `TimeOut` (vanligvis 60 sekunder). Hvis det er problemer med forbindelsen eller `TimeOut` er overskredet, stenges forbindelsen ned på en pen måte, og forbindelsen fjernes fra listen `Lines`.

6.5.2 Send()

Denne metoden returnerer en byte array til en klient. Først konverteres antall bytes i byte array’en til en 4 bytes representasjon av `int`. Disse 4 bytes oversendes til klienten. Deretter oversendes selve byte array’en.

6.5.3 ”PutObject”

Denne kommandoen setter forbindelsen i en spesiell modus for overføring av binære objekter. Syntaks for kommandoen, og protokoll for overføring, er som følger:

Klient:	Sender kommando: PutObject <type> <numbytes>
Server:	Oppretter en `MemoryStream` hvor objektet lagres midlertidig. Svarer ”OK\r\n”
Klient:	Sender objekt
Server:	Leser fra `Socket` og skriver til `MemoryStream` inntil gitt antall bytes er lest. Hvis *for mange* bytes blir mottatt, flagges feil til klient (”Error\r\n”). Hvis antall bytes er korrekt, anropes `Data2ObjectDelegate()` Svarer ”OK\r\n”

6.6 Client

Denne klassen oppretter en `TcpClient` som anroper en server på oppgitt IP-adresse og port, og oppretter en `NetworkStream` for å håndtere data til og fra. Metoden har ingen aktiv kontroll av tilstanden til forbindelsen.

6.6.1 Send()

Dette er to ulike metoder.

Den ene er for overføring av en kommando; kommandoen + ”\r\n” konverteres til en byte array, og skrives til forbindelsens `NetworkStream`. Deretter anropes `GetResponse()` for å hente svaret fra serveren.

Den andre metoden er for overføring av et binært objekt, og følger protokollen angitt i 6.5.3.

6.6.2 GetResponse()

Denne metoden leser først antall bytes som skal mottas, representert ved de 4 første bytes i forbindelsens `NetworksStream`. Disse konverteres til `int`. Hvis dette tallet er 0 flagges feil, ellers leses det fra `NetworkStream` inntil korrekt antall bytes er lest.

6.7 CellProxy

Denne statiske klassen inneholder en samling metoder som ligger ”mellom” en applikasjon og `Client`-klassen. De viktigste metodene er beskrevet under.

6.7.1 FindFreeCell()

Oppretter en `Client` mot lokal statusmonitor, og anroper `Comm2Cell()` med denne `Client`’en og kommandoen

> GetCell true <beregningsID> <deloppdragID>

for å få IP-adresse og portnummer til en ledig regnecelle på lokal eller annen maskin.

6.7.2 CloseMyCells()

Oppretter en `Client` mot lokal statusmonitor, og anroper `Comm2Cell()` med denne `Client`’en og kommandoen

> CloseCells true <beregningsID>

for at statusmonitor skal avslutte alle regnecelle-applikasjoner som er har deloppdrag med gitt beregningsID (på lokal eller annen maskin).

6.7.3 ClearCell()

Anroper `Comm2Cell()` med oppgitt `Client` og kommandoen

> ClearCalc

for å nullstille støyberegner- eller regnecelle-applikasjonen som `Client`’en er koblet opp mot.

6.7.4 SetCellStatus() / GetCellStatus()

Anroper `Comm2Cell()` med oppgitt `Client` og kommandoen

SetStatus <status>

eller

GetStatus

for å sette eller hente status for støyberegner- eller regnecelle-applikasjonen som `Client`'en er koblet opp mot.

6.7.5 Comm2Cell()

Anroper `Client.Send()` med gitt kommandostreng for å sende kommando til serveren. Svaret fra serveren (en byte array) blir konvertert til en streng og returnert.

6.7.6 File2Cell()

Oppretter og fyller en byte array med innholdet i en fil med oppgitt filnavn. Anroper `Client.Send()` for å sende byte array'en til serveren (i hht. protokoll beskrevet i 6.5.3).

6.7.7 Cell2File()

Anroper `Client.Send()` med kommandoen "GetObject <type>" for å hente binært objekt av gitt type fra server. Svaret fra serveren (en byte array) blir lagret til fil med oppgitt filnavn.

6.8 Andre klasser

SoundKernel-rutinebiblioteket inneholder flere klasser, samt felles enumeratorer og strukturer. De viktigste strukturene og klassene som brukes til å lagre oppdrag og beregningsgrunnlag er kort beskrevet her:

- `typSubTask`. Inneholder informasjon om et deloppdrag (men ikke selve deloppdraget).
- `AlistWrap`. Er en overbygning til en `ArrayList` for å tilføre metoder som `.Load()` og `.Save()`.
- `skTopPoints`. Inneholder en punktsamling for topografi (posisjon, høyde, markslag, skoghøyde, ruhet).
- `skTopGrid`. Inneholder definisjon av en grid, og en grid for hver av høyde, markslag, skoghøyde og ruhet.
- `skTopSurface`. Inneholder en topografiflate med gitt geometri, markslag, skoghøyde og ruhet.
- `skTopScreen`. Inneholder geometri og attributter for en (sammenhengende) skjerm eller voll.
- `skTopRoad`. Inneholder geometri og attributter for en veg (kan bestå av flere deler)
- `skTopBuilding`. Inneholder geometri og attributter for enten en enkel bygning med gitt senterposisjon, størrelse og fasong, eller en generell bygnings omrisspolygon og taklinjer.
- `skSource`. Inneholder spesifikasjon av en kilde (veg). Fysisk beskrivelse av kilden er gitt i en instans av `skTopRoad`. En unik ID for vegen blir brukt til å lenke til denne.
- `skTask`. Inneholder definisjon av et beregningsoppdrag og samlete resultater.
- `skSubTask`. Inneholder definisjon av et deloppdrag, inkl. en instans av `skSource`. Inneholder også hash-kode for tilhørende subtopografi.

- `skTopIdx`. Inneholder topografiske indekser i form av R-trær[11] for topografiflater, skjermer/voller, veger, bygninger og kilder. Disse brukes til å bestemme hvilke objekter som ligger innenfor et gitt område eller langs en gitt linje.

[11] Dette er en delvis implementering av Guttmann A., *R-Trees: A dynamic index structure for spatial searching*, 1984. Metoder for innsetting og søking er implementert. Metoder for sletting av noder er *ikke* implementert, da det ikke er nødvendig for denne applikasjonen.

7 ProP2P – Nord 2000

Type: Rutinebibliotek
Filnavn: prop2p.dll

7.1 Funksjon og struktur

Rutinebiblioteket har én funksjon som er synlig utad; `N2000`. Ved bruk av en lang rekke interne funksjoner beregner `N2000` markdemping i henhold til Nord 2000-metoden.

Dagens versjon av rutinebiblioteket skal være i overensstemmelse med MATLAB-kildekoden i compro16.m, med unntak av luftabsorpsjon. Det forutsettes at luftabsorpsjon blir beregnet utenfor `N2000`. I Regnecelle-applikasjonen gjøres dette i `SK.PointSourceToPoint()`.

I framtidige utgaver av rutinebiblioteket kan andre metoder for beregning av markdemping bli inkludert.

8 Oversikt over støttede kommandoer

8.1 Statusmonitor

Dette programmets `DoCommand()`-delegate støtter følgende kommandoer:

Kommando	Argument	Arg. type	Resultat	Res. Type	Beskrivelse
getNewCalculation			ID + ”\r\n” + Portnummer	2 x heltall	Anroper `GetSC()` for å få ID og portnummer til en ledig støyberegner.
getNewCalculation	ID	heltall	ID +”\r\n” + Portnummer	2 x heltall	Anroper `GetSC()` for å få ID og portnummer til en ledig støyberegner (med oppgitt ID).
getCalculation	ID	heltall	Portnummer	heltall	Sjekker lista over støyberegnere for å finne portnummer til støyberegner med oppgitt ID. Kontakter støyberegner på oppgitt port for å sjekke at ID er korrekt.
CloseCalculation	ID	heltall	OK | Error	streng	Sjekker lista over støyberegnere for å finne portnummer til støyberegner med oppgitt ID, og sender ”CloseApp” til denne. Anroper også `CloseCells()` for å stenge ned regneceller som tilhører denne støyberegneren.
getNewTaskID	Local	bool	ID	heltall	Anroper `GetNewTaskID()` for å få tildelt en ny unik beregningsID.
getCell	Local, ID, DeloppdragID	bool, heltall, heltall	IP:port	IP:heltall	Anroper `GetCell()` for å få IP og portnummer til en ledig regnecelle.
CloseCells	Local, ID	bool, heltall	OK | Error	streng	Anroper `CloseCells()` for å avslutte regnecellene som tilhører støyberegner med oppgitt ID.
getNCInfo			Info-liste	streng	Løkker gjennom lista over støyberegnere og returnerer en linje med informasjon for hver.
getCCInfo			Info-liste	streng	Løkker gjennom lista over regneceller og returnerer en linje med informasjon for hver.
getRCCInfo			Info-liste	streng	Returnerer listen over status for regneceller på andre maskiner
getIP			IP	IP	Oppdaterer og returnerer `MyIP`, dvs. den IP-adressen som returneres av lokale anrop til `getCell()`

Kommando	Argument	Arg. type	Resultat	Res. Type	Beskrivelse
setIP	IP	IP	OK | Error	streng	Setter ditto IP-adresse
getLocalCalc			True | False	streng	Returnerer om statusmonitor oppretter regneceller på lokal maskin
setLocalCalc	True | False	streng	OK | Error	string	Angir om statusmonitor skal opprette regneceller på lokal maskin
UpdateSMList	SM-liste	streng	OK | Error	streng	Anroper `UpdateSMList()` for å erstatte lista over andre statusmonitorer.
UpdateSMMaster	IP, vertsnavn, port	IP, streng, heltall	OK | Error	streng	Anroper `UpdateSMMast()` for å angi hvilken statusmonitor som er ”master”.
ClearErrors			OK	streng	Tømmer liste med feilmeldinger
getErrors			Liste med feilmeldinger	streng	Returnerer liste med alle feilmeldinger.
WhatAreYou			Identifikator	streng	Returnerer ”Statusmonitor”.
WhatsYourID			PID	heltall	Returnerer prosess-ID for instansen.

8.2 StatusmonitorTray

Dette programmet har ikke noe server-grensesnitt.

8.3 Støyberegner

Dette programmets `DoCommand()`-delegate støtter følgende kommandoer:

Kommando	Argument	Arg. type	Resultat	Res. type	Beskrivelse
setID	ID	heltall	OK | Error	streng	Setter `SK.TaskID` (beregningsID).
getID			ID	heltall	Returnerer ditto.
setPath	katalog	streng	OK | Error	streng	Setter `SK.MainPath` (katalog hvor data, oppdrag, dokumentasjon og resultater legges).
getPath			katalog	streng	Returnerer ditto katalog.
setDBPath	katalog	streng	OK | Error	streng	Kaller `Database.SetDBRoot()` (rot-katalog for intern database).

Kommando	Argument	Arg. type	Resultat	Res. type	Beskrivelse
getDBPath			katalog	streng	Returnerer ditto katalog.
Preprocess			OK | Error [12]	streng	Endrer `SK.MainStatus` fra ”occupied” til ”unstarted”, og setter `Preprocess.StartReading` til true.
StartCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”unstarted” eller ”inactive” til ”active”.
PauseCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”active” til ”inactive”.
StopCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”active”, ”inactive” eller ”unstarted” til ”interrupted”. Venter så til `Splitter` har avsluttet utsendelse av deloppdrag til regneceller. Anroper deretter `CellProxy.CloseMyCells()` for å stenge ned tilhørende regneceller.
ClearCalc			OK	streng	Anroper `ClearModules((CloseAllCellsToo` = true`)` for å nullstille alle underklasser og avslutte tilhørende regneceller.
ReLoad			OK | Error	streng	Endrer `SK.MainStatus` fra ”occupied” til ”unstarted”, og setter `Preprocess.StartReLoad` til true.
MakeMap			OK | Error	streng	Anroper `Results.SaveResults()` for å generere og lagre støykart (uavhengig av `SK.MainStatus`).
NoKill			OK	streng	Setter `SokServ.TimeToKill` = 0, slik at `SokServer` aldri anroper `KillMe()`.
CloseApp					Anroper `KillMe()` for å stenge ned applikasjonen på en ryddig måte.

[12] Disse kommandoene er asynkrone; ”OK” eller ”Error” angir bare om kommandoen er akseptabel i konteksten. ”OK” betyr altså ikke at handlingen som kommandoen iverksetter er ferdig.

Kommando	Argument	Arg. type	Resultat	Res. type	Beskrivelse
setStatus	Status	streng	OK	streng	Setter `SK.MainStatus` (“free”, “occupied”, “unstarted”, “active”, “inactive”, “interrupted”, “finished”, “error”).
getStatus			Status	streng	Returnerer `SK.MainStatus`.
getStatus	Preprocess | Splitter | Collector | Results | All	streng	Status	streng	Returnerer status for en gitt modul (”not_finished”, ”finished”, ”error”), eventuelt status for alle moduler.
getElapsedTime			Total tid + ”\r\n” + CPU-tid	streng (2 x hh:mm:ss)	Anroper `GetElapsed()` for å hente oppdragets totaltid og CPU-tid.
getRemainingTime			CPU-tid | N/A	streng (hh:mm:ss)	Anroper `GetRemaining()` for å hente estimert gjenstående CPU-tid. Returnerer ”N/A” dersom gjenstående tid ikke kan estimeres ennå.
getNumParts			Ntotal	heltall	Returnerer antall oppføringer i intern liste over deloppdrag.
getNumFinished			Nfinished	heltall	Returnerer antall deloppdrag flagget som ”finished” i intern liste over deloppdrag.
getErrors			Liste med feilmeldinger	streng	Returnerer liste med alle feilmeldinger i alle moduler.
getErrors	Main | Preprocess | Splitter | Collector | Results | Sockets | Database		Liste med feilmeldinger	streng	Returnerer liste med alle feilmeldinger for en gitt modul.
GetInfo			Div. info	streng	Returnerer en kodet streng med informasjon om deloppdragene, status og tidsforbruk.
WhatAreYou			Identifikator	streng	Returnerer ”NoiseCalculator”
WhatsYourID			PID	heltall	Returnerer prosess-ID for instansen.
getDocument			OK | Error	streng	Anroper `MakeDocument()` for å generere dokumentasjon for beregningen.

8.4 Regnecelle

Dette programmets `DoCommand()`-delegate støtter følgende kommandoer:

Kommando	Argument	Arg. Type	Resultat	Res. type	Beskrivelse
getTaskID			ID	heltall	Returnerer `SK.Task.TaskID` (beregningsID).
getSubTaskHash			Hash-kode	streng	Returnerer hash-kode for innlest deloppdrag.
getSubTopoHash			Hash-kode	streng	Returnerer hash-kode for innlest subtopografi.
StartCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”unstarted” eller ”inactive” til ”active”.
PauseCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”active” til ”inactive”.
StopCalc			OK | Error	streng	Endrer `SK.MainStatus` fra ”active”, ”inactive” eller ”unstarted” til ”interrupted”.
ClearCalc			OK	streng	Anroper `ClearModules()` for å nullstille alle underklasser.
getObject	type	streng	Deloppdrag	binært objekt	Hvis type er `enFC.SubTask`, returneres en serialiseringen av deloppdraget (`skSubTask.Save()`).
NoKill			OK	streng	Setter `SokServ.TimeToKill` = 0, slik at `SokServer` aldri anroper `KillMe()`.
CloseApp					Anroper `KillMe()` for å stenge ned applikasjonen på en ryddig måte.
setStatus	Status	streng	OK	streng	Setter `SK.MainStatus` (“free”, “occupied”, “unstarted”, “active”, “inactive”, “interrupted”, “finished”, “error”).
getStatus			Status	streng	Returnerer `SK.MainStatus`.
getElapsedTime			CPU-tid	streng (hh:mm:ss)	Anroper `GetElapsed()` for å hente deloppdragets CPU-tid.
getErrors			Liste med feilmeldinger	binært objekt	Returnerer en serialisering av en `ArrayList` med alle feilmeldinger i alle moduler.
GetInfo			Div. info	streng	Returnerer en kodet streng med informasjon om deloppdraget.
WhatAreYou			Identifikator	streng	Returnerer ”CalculationCell”
WhatsYourID			PID	heltall	Returnerer prosess-ID for instansen.

9 Installasjonsveiledning

Dette avsnittet er stort sett identisk med tidligere utsendt notat ”Installasjonsveiledning for Støyberegner” av 2007-02-22.

9.1 Introduksjon

Her beskrives installasjon og oppsett av Støyberegningsløsningen utviklet av SINTEF for Vegdirektoratet. Bruk av Støyberegningsløsningen beskrives ikke, da dette er beskrevet i egen brukerveiledning og i prosjektets detaljspesifikasjon. Kort oppsummert består løsningen av fire komponenter:

- **Statusmonitor**. Dette er en service som tar i mot forespørsler etter Støyberegnere (fra NorStøy-applikasjonen) og/eller Regneceller (fra Støyberegnere), og returnerer nettverksadressen (IP) til nyopprettete (eller eksisterende, ledige) slike.

- **Støyberegner**. Dette er et program som opprettes av Statusmonitor og styres av NorStøy-applikasjonen. Programmet leser inn oppgitte grunnlagsdata for et beregningsoppdrag og deler oppdraget i mindre biter (deloppdrag). Deloppdragene sendes etter tur ut til Regneceller for beregning. Etter beregning samles og lagres resultatene.

- **Regnecelle**. Dette er et program som opprettes av Statusmonitor og styres av Støyberegner. Programmet utfører selve beregningen av alle deloppdragene.

- **StatusmonitorTray**. Dette er et program som kan startes av lokal, innlogget bruker. Programmet er et brukergrensesnitt for Statusmonitor-service’en, og brukes for å konfigurere denne. Det gir også en oversikt over status for lokale Regneceller og Støyberegnere, og kan i en viss grad styre sistnevnte. Merk at programmet bare er et hjelpemiddel; det er ikke kritisk nødvendig for å bruke støyberegningsløsningen.

9.2 Installasjon

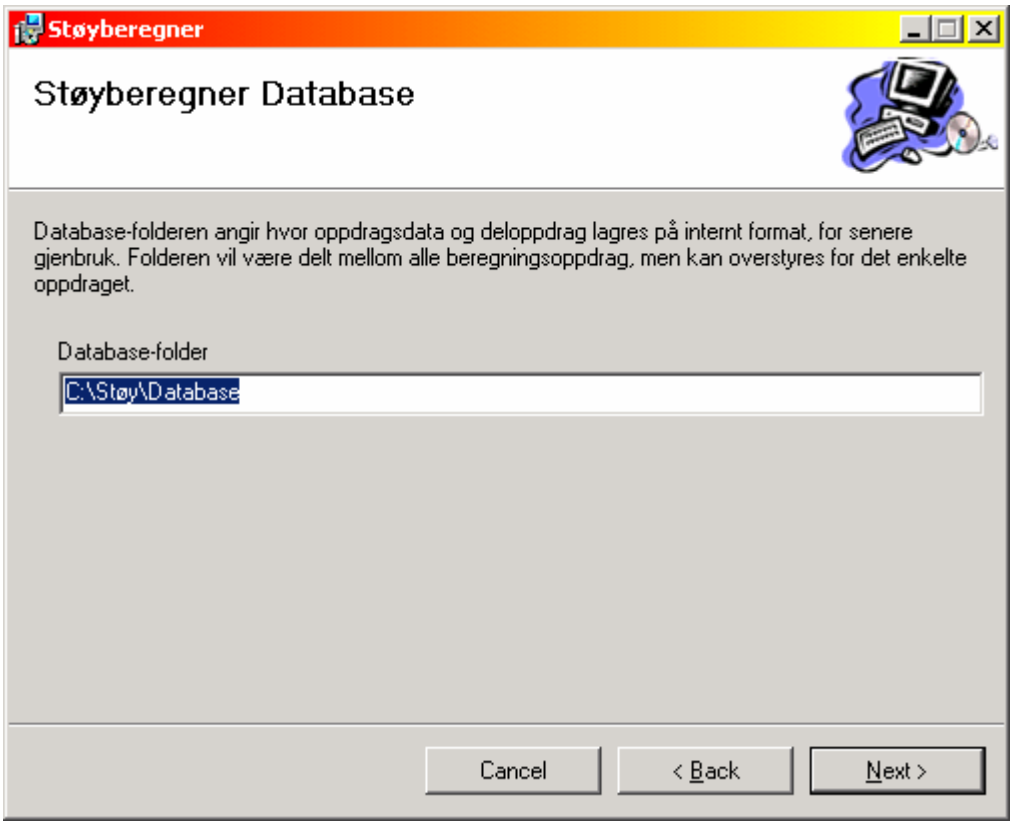
Denne beskrivelsen gjelder en vanlig PC hvor man er innlogget med administratorrettigheter. Installasjonen (og senere konfigurering) forutsetter at man har (skrive-/endre-)tilgang til:

- Lokalt filsystem
- Registeret (nøkkel HKEY_LOCAL_MACHINE/Software)
- Services
- Brannmur

Installasjonspakken består av filene setup.msi og setup.exe, og igangsettes ved dobbelklikk på sistnevnte program.

Installasjonsprogrammet er i stor grad selvforklarende. Dersom maskinen det installeres på ***kun*** skal brukes til Regneceller kan alle standardverdier aksepteres.

Dersom maskinen det installeres på skal brukes til Støyberegnere ***må*** det i vinduet under angis full sti til en katalog hvor interne filer skal kunne lagres for senere gjenbruk. Katalogen blir opprettet hvis den ikke finnes.



Omstart etter installasjon er ikke nødvendig; Statusmonitor-service'en blir automatisk startet etter installasjon. For å starte/stoppe Statusmonitor-service'en manuelt, høyreklikk på "My Computer" og velg

Manage ➔ Services and Applications ➔ Services

I listen over services, høyreklikk på "Statusmonitor" og velg "Start" eller "Stop".

9.3 Konfigurering av brannmur

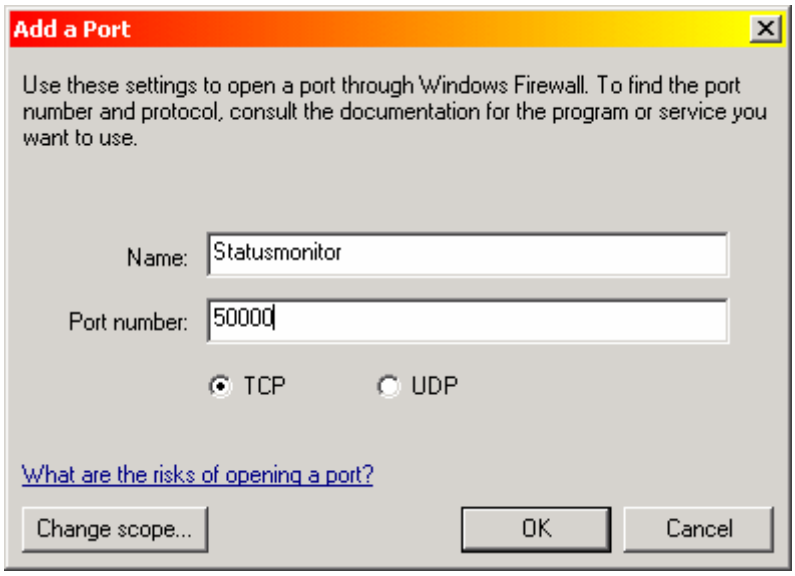
9.3.1 Generelt

Støyberegningsløsningen bruker nettverket til kommunikasjon mellom komponentene, og er derfor avhengig av at bestemte porter blir åpnet i brannmuren på de ulike maskinene. For å åpne porter, velg:

Control Panel ➔ Security Center ➔ Windows Firewall ➔ Exceptions

For alle installasjoner av Støyberegningsløsningen kreves det tilgang til Statusmonitor på lokal maskin. Denne bruker som standard[13] port 50000. Denne porten åpnes ved å velge "Add Port…", og fylle ut vinduet som vist under.

[13] Dersom standard port er i bruk av andre programmer, kan den forandres ved å endre registernøkkelen HKEY_LOCAL_MACHINE/Software/SINTEF/Støyberegner/StatusMonitorPort.



9.3.2 For kjøring av Støyberegnere (og Regneceller)

Port-nummeret til Støyberegner og Regnecelle blir tildelt av Statusmonitor når Støyberegner eller Regnecelle startes. I dette tilfellet må det derfor åpnes for Støyberegner- og Regnecelle-applikasjonene, og ikke for en bestemt port. Velg:

Control Panel ➔ Security Center ➔ Windows Firewall ➔ Exceptions ➔ Add Program…

I vinduet som kommer opp, velg ”Browse…” og naviger til programmet det skal åpnes for (***stoyberegner.exe,*** som standard installert i katalogen C:\Program Files\SINTEF\Støyberegner), og lukk vinduene ved å velge ”Open” og ”OK”. Gjenta operasjonen fra ”Add Program…” for å legge til programmet ***regnecelle.exe***.

9.3.3 For kjøring av kun Regneceller

Åpning av brannmuren for Regnecelle gjøres som beskrevet i avsnitt 9.3.2, men bare for programmet ***regnecelle.exe***.

9.4 Konfigurering av Statusmonitor

Se kapittel 10, spesielt avsnitt 10.3.

10 Brukerveiledning for StatusMonitorTray

Brukergrensesnittet til statusmonitor-servicen gir brukeren mulighet til å:

- observere status for en eller flere støyberegninger
- starte/pause/stoppe/nullstille/avslutte en eller flere støyberegninger
- konfigurere statusmonitor for kjøring av parallelle beregninger.

Brukergrensesnittet aktiveres gjennom ikonet  som finnes i nedre høyre del av skjermen[14]. Dersom dette ikonet ikke finnes, er brukergrensesnittet ikke startet. Programmet finnes i undermenyen ”SINTEF” i Start-menyen, og kan startes derfra.

Ved høyreklikk på ikonet framkommer følgende valg:

- Vis Statusmonitor – Viser vinduet med brukergrensenittet. Vinduet vises også ved å dobbelklikke på ikonet.
- Skjul Statusmonitor – Skjuler vinduet med brukergrensesnittet. Man kan også stenge ned vinduet på vanlig måte (ved klikk på kryss)
- Avslutt Statusmonitor – Avslutter brukergrensesnitt-programmet. Ikonet  vil da forsvinne. Programmet må da startes på nytt fra Start-menyen.

Brukergrensesnittet er inndelt i fire faner som hver er beskrevet under.

10.1 Støyberegnere

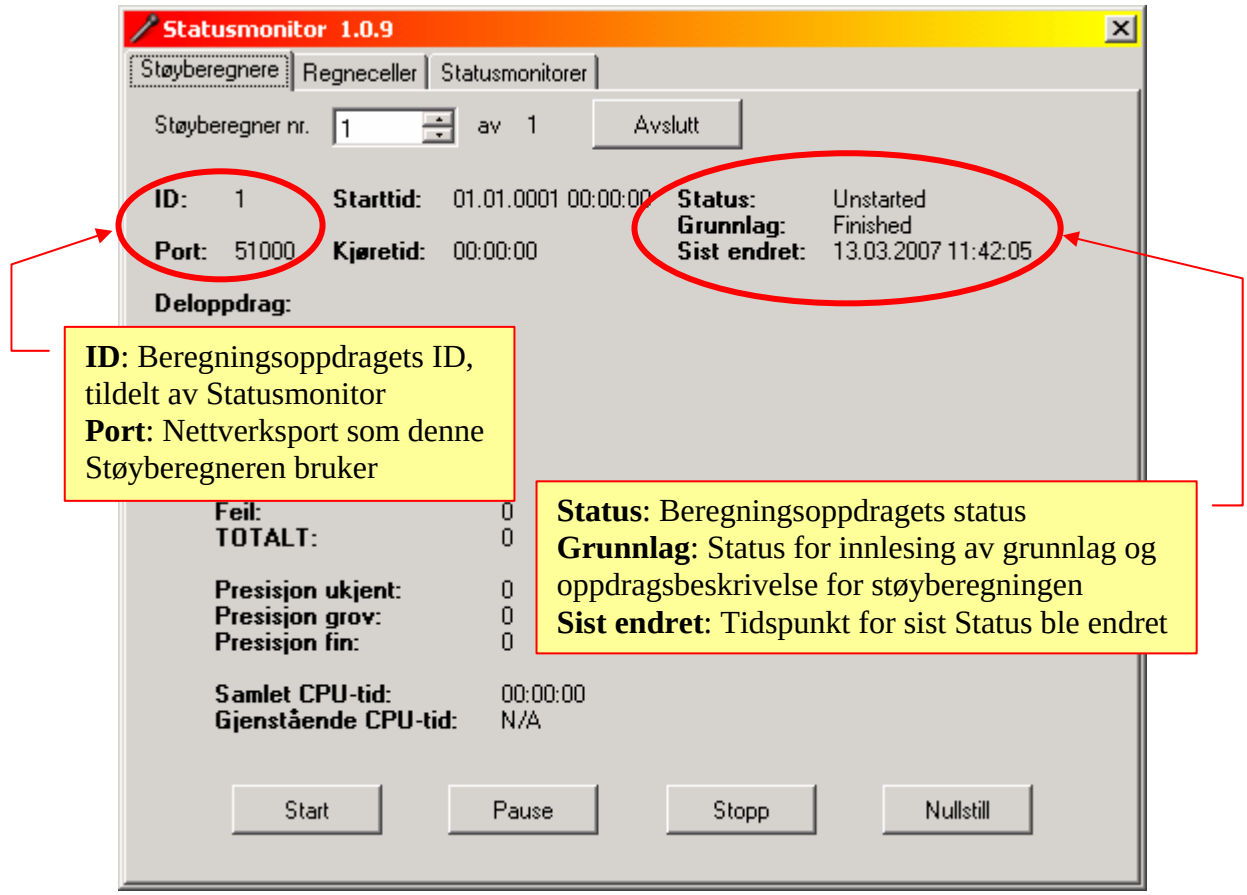
Denne fanen inneholder de elementene som framgår av de to eksemplene på neste side. Øverst i fanen vises det hvilken støyberegner som det vises informasjon for. Det er fullt mulig å ha flere støyberegnere kjørende samtidig på samme maskin. Bruk kontrollen  for å velge mellom de ulike støyberegnerne. Støyberegnere på andre maskiner vil ikke være synlige her.

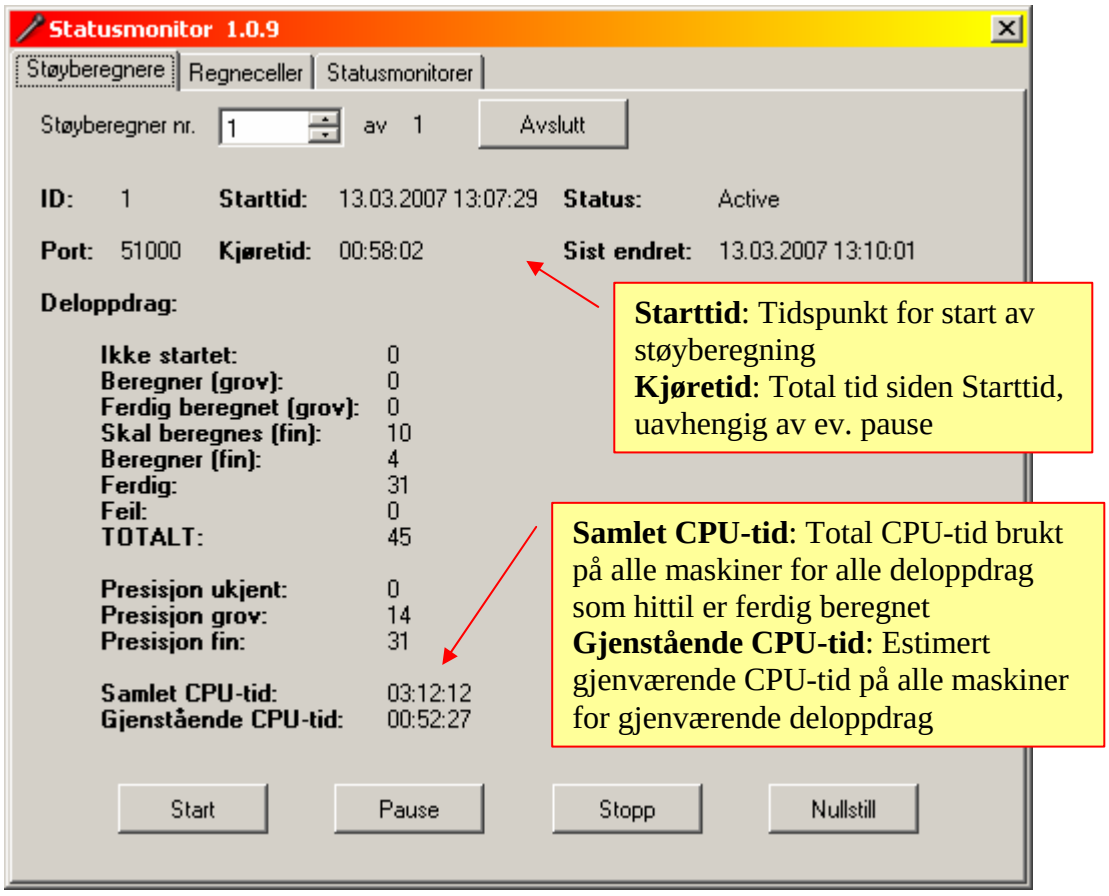
Det første eksempelet viser hvordan grensesnittet ser ut for en støyberegner som er ferdig med innlesning av beregningsgrunnlag og oppdragsbeskrivelse. De fleste felt har verdien 0, men beregningsoppdragets ID og støyberegnerens port er begge definert. Feltet ”Status” angir status for støyberegneren, og kan ha verdiene

- Free – Støyberegneren er ledig
- Occupied – Støyberegneren er reservert, men ikke tatt i bruk ennå
- Unstarted – Innlesing av beregningsgrunnlag og oppdragsbeskrivelse er igangsatt, men ikke nødvendigvis ferdig. Feltet ”Grunnlag” viser status for innlesing.
- Active – Støyberegneren er aktiv, beregning pågår
- Inactive – Støyberegneren er satt i pause-modus
- Interrupted – Støyberegneren er avbrutt før beregning var ferdig
- Finished – Støyberegneren er ferdig med beregning
- Error – En (mer eller mindre alvorlig) feil har oppstått slik at beregning er avbrutt.
  NB: Dette kan også være et tegn på at statusmonitoren har (midlertidige) problemer med forbindelsen til støyberegneren

Feltet ”Grunnlag” vises bare før, under, og etter innlesing (med verdi ”Not finished”/”Finished”).

[14] Merk at Windows i enkelte tilfeller skjuler lite brukte ikoner. Klikk på  for å vise alle ikonene.


Statusmonitor 1.0.9
Støyberegnere
Regneceller
Statusmonitorer
Støyberegner nr. 1 av 1
Avslutt
ID: 1
Starttid: 01.01.0001 00:00:00
Status: Unstarted
Grunnlag: Finished
Sist endret: 13.03.2007 11:42:05
Port: 51000
Kjøretid: 00:00:00
Deloppdrag:
Feil: 0
TOTALT: 0
Presisjon ukjent: 0
Presisjon grov: 0
Presisjon fin: 0
Samlet CPU-tid: 00:00:00
Gjenstående CPU-tid: N/A
Start
Pause
Stopp
Nullstill
ID: Beregningsoppdragets ID, tildelt av Statusmonitor
Port: Nettverksport som denne Støyberegneren bruker
Status: Beregningsoppdragets status
Grunnlag: Status for innlesing av grunnlag og oppdragsbeskrivelse for støyberegningen
Sist endret: Tidspunkt for sist Status ble endret


Statusmonitor 1.0.9
Støyberegnere
Regneceller
Statusmonitorer
Støyberegner nr. 1 av 1
Avslutt
ID: 1
Starttid: 13.03.2007 13:07:29
Status: Active
Port: 51000
Kjøretid: 00:58:02
Sist endret: 13.03.2007 13:10:01
Deloppdrag:
Ikke startet: 0
Beregner (grov): 0
Ferdig beregnet (grov): 0
Skal beregnes (fin): 10
Beregner (fin): 4
Ferdig: 31
Feil: 0
TOTALT: 45
Presisjon ukjent: 0
Presisjon grov: 14
Presisjon fin: 31
Samlet CPU-tid: 03:12:12
Gjenstående CPU-tid: 00:52:27
Start
Pause
Stopp
Nullstill
Starttid: Tidspunkt for start av støyberegning
Kjøretid: Total tid siden Starttid, uavhengig av ev. pause
Samlet CPU-tid: Total CPU-tid brukt på alle maskiner for alle deloppdrag som hittil er ferdig beregnet
Gjenstående CPU-tid: Estimert gjenværende CPU-tid på alle maskiner for gjenværende deloppdrag

Det andre eksempelet viser hvordan grensesnittet ser ut for en støyberegner som er aktiv med en beregning. I tillegg til tidsforbruk, som forklart i figuren, vises også en opptelling av status og presisjon for deloppdragene. Alle deloppdrag beregnes først (hurtig) med en grov presisjon, og deretter, hvis de bidrar signifikant til sluttresultatet, med en langsommere, fin presisjon. Status til hvert deloppdrag følger følgende skjema i løpet av beregningen:

Første del av beregningen:
Ikke startet ➔ Beregner (grov) ➔ Ferdig beregnet (grov)

Når alle deloppdrag har status ”Ferdig beregnet (grov)” eller ”Ferdig” blir en signifikanstest utført:
Deloppdrag som ikke er signifikante får status ➔ Ferdig
Deloppdrag som er signifikante får status ➔ Skal reberegnes (fin)

Andre del av beregningen:
Skal reberegnes (fin) ➔ Beregner (fin) ➔ Ferdig

I tillegg kan deloppdragene få status ➔ Feil, dersom det oppstår en feil under beregningen av deloppdraget (feil i beregningskjernen, nettverksfeil, osv.).

Knappene som er tilgjengelige i denne fanen har følgende funksjon:

- Avslutt – Avslutter denne støyberegneren og nullstiller alle tilhørende regneceller.
- Start – Starter støyberegningen etter innlesing av oppdrag og beregningsgrunnlag, eller restarter støyberegningen etter ”Pause”. Dette er bare mulig hvis Status er hhv. ”Unstarted” eller ”Inactive”. Hvis Status er ”Unstarted” må også status for innlesing av beregningsgrunnlag og oppdrag (Grunnlag) være ”Finished”.
- Pause – Setter støyberegningen i pausemodus (Status ”Inactive”). Dette medfører bl.a. at det ikke sendes ut nye deloppdrag til regneceller. Foranledningen er at brukeren ønsker å gi prioritet til en annen regnekrevende prosess, for eksempel en annen støyberegning. Brukeren kan fortsette støyberegningen ved å trykke på ”Start”.
- Stopp – Avbryter støyberegningen, og nullstiller alle tilhørende regneceller. Dette er en ikke-reversibel handling, dvs. at støyberegningen kan *ikke* restartes ved trykk på ”Start”.
- Nullstill – Nullstiller støyberegeren og alle tilhørende regneceller. Dette vil også selvsagt avbryte støyberegningen på samme måte som ”Stopp”. I tillegg vil (nesten) all informasjon om oppdraget slettes, slik at støyberegneren er ledig og kan tas i bruk for et nytt oppdrag.

I tilfellene ”Avslutt”, ”Stopp” og ”Nullstill” blir brukeren først spurt om han ønsker å utføre den valgte handlingen / kommandoen.

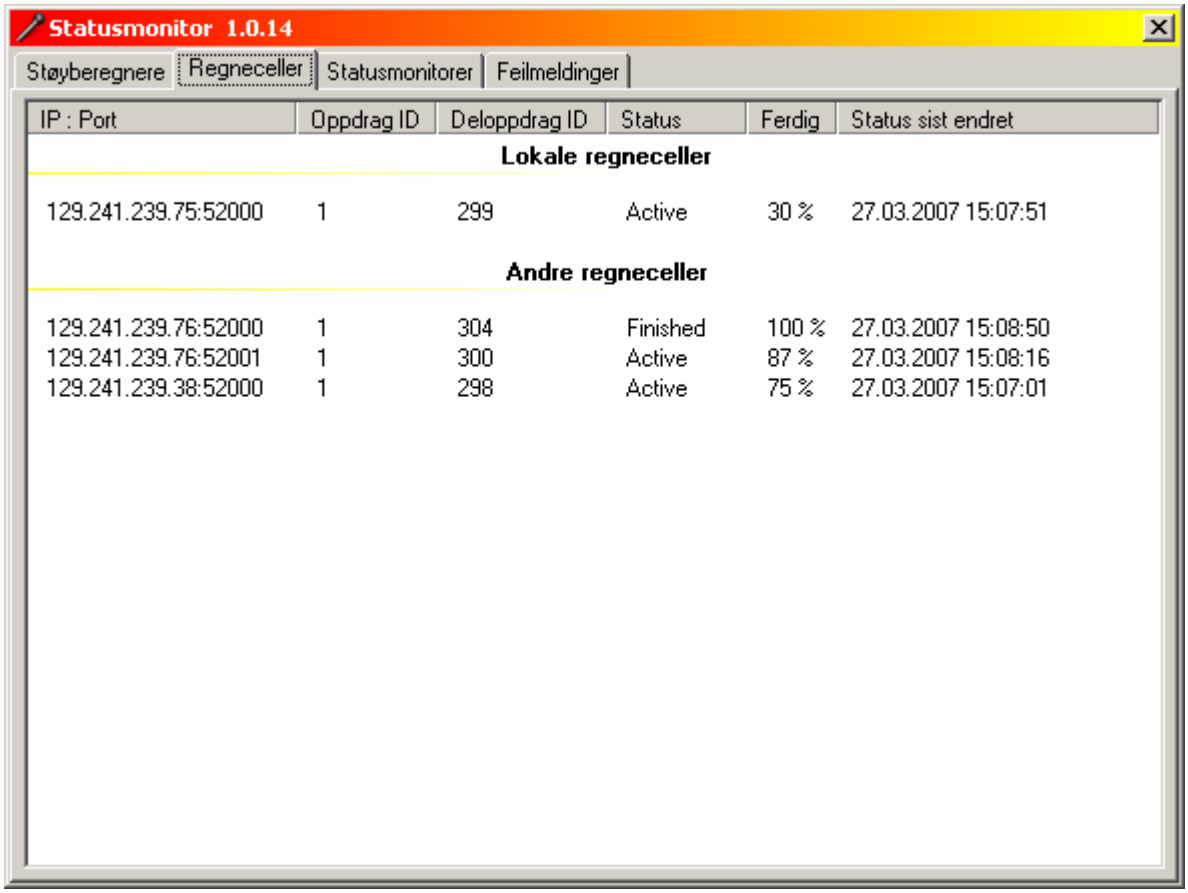
Dersom brukeren ønsker å ***restarte*** en beregning som har blitt avbrutt, må han først nullstille støyberegneren, ved kommandoen ”ClearCalc” eller trykk på ”Nullstill”-knappen. Merk at nullstilling med hensikt ikke sletter verken ID for beregningsoppdraget eller katalog for grunnlagsdata. Derimot settes status for støyberegneren til ”Free”. Før støyberegneren kan tas i bruk igjen må status settes til ”Occupied” ved kommandoen ”SetStatus”.

Deretter må beregningsgrunnlaget leses inn. Dette kan enten gjøres ved kommandoen ”Preprocess” (som leser inn beregningsgrunnlaget på nytt fra de originale grunnlagsfilene), eller kommandoen ”ReLoad” (som leser inn et eksisterende grunnlag på internt format, basert på beregningsoppdragets ID).

Når innlesing av beregningsgrunnlag er ferdig, kan støyberegningen startes ved kommandoen ”StartCalc” eller trykk på ”Start”-knappen.

10.2 Regneceller

Denne fanen viser en liste over regneceller på lokal maskin og på de maskiner som er oppført i listen over andre statusmonitorer (se 10.3):

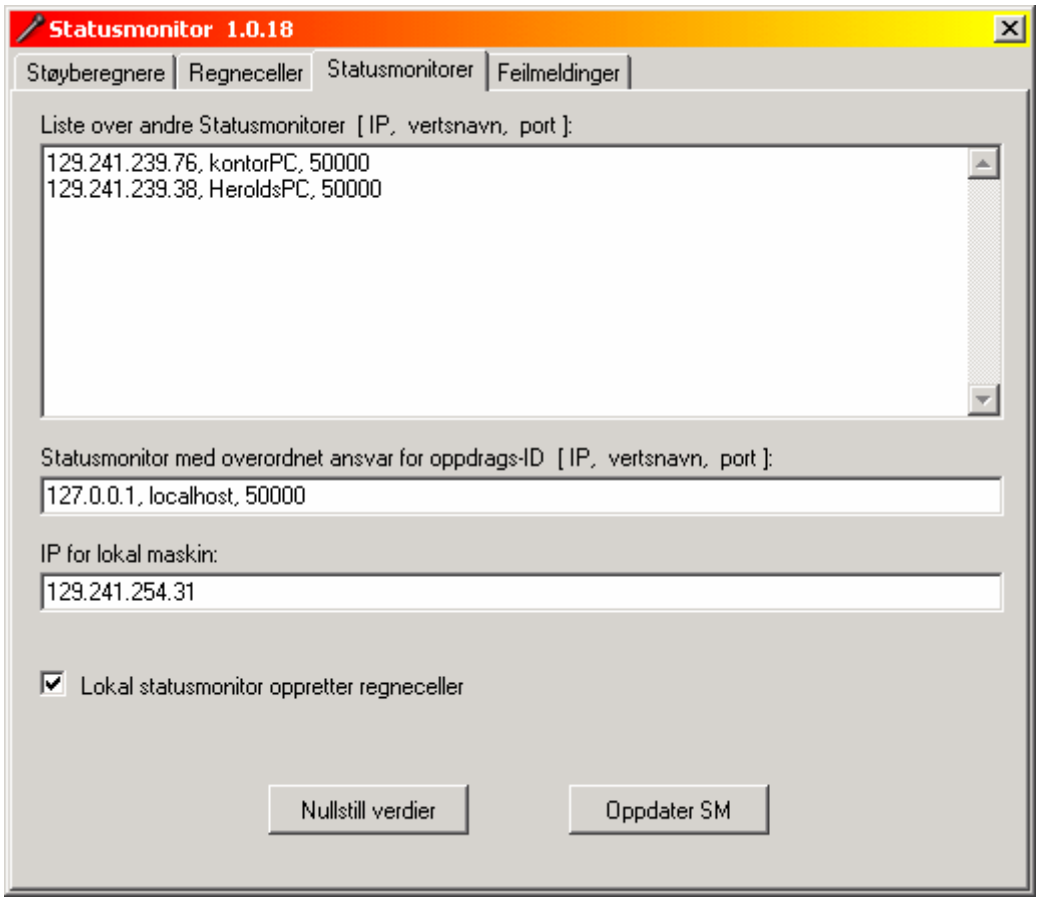


For hver regnecelle vises:

- IP : Port – IP-adresse og portnummer til maskinen og regnecella
- Oppdrag ID – Oppdragets beregningsID
- Deloppdrag ID – Deloppdragets ID
- Status – Status for regnecella. Verdiene er de samme som for støyberegnerne
- Ferdig – Viser hvor mye av deloppdraget som er ferdig
- Status sist endret – Viser tidspunkt for sist Status ble endret

10.3 Statusmonitorer

Denne fanen brukes til å konfigurere statusmonitor-service’en for å fordele deloppdrag ut på flere maskiner, slik at total regnetid reduseres. *Dersom maskinen kun skal brukes til kjøring av regneceller, er det bare nødvendig å kontrollere og evt. korrigere feltet* ”IP for lokal maskin”.



Under ”Liste over andre Statusmonitorer…” kan det fylles ut adresse til andre maskiners statusmonitorer[15]. Dette er valgfritt, men vil bidra til redusert regnetid. Når en støyberegner på lokal maskin spør statusmonitoren etter ledige regneceller, vil lokal statusmonitor videreformidle forespørselen til statusmonitorene oppgitt i denne listen. Dersom denne listen er tom vil støyberegneren bare kunne bruke regnecelle(r) på lokal maskin. Merk at adressen må oppgis i form av IP (nnn.nnn.nnn.nnn). IP-adressen må derfor være fast for de maskiner som angis i listen. Oppgitt vertsnavn brukes (foreløpig) ikke til noe, men bør beskrive maskinen. Port vil som regel være 50000, dersom dette ikke er endret for den aktuelle statusmonitoren.

Under ”Statusmonitor med overordnet ansvar…” skal det angis adresse til ***én*** bestemt maskin som har ansvaret for tildeling av oppdrags-ID. Dette må gjøres for å unngå konflikter i tilfeller der flere støyberegnere deler beregningsressurser fra felles regnecelle-maskiner. I miljøer hvor kun én maskin kjører støyberegner(e) kan for eksempel lokal maskin ha dette overordnete ansvaret. I så fall kan feltet fylles ut med

127.0.0.1, localhost, 50000

I feltet ”IP for lokal maskin” vil det automatisk komme fram en IP-adresse for den lokale maskinen. Dette vil som regel være korrekt, og kan forbli uendret. I enkelte tilfeller, for eksempel hvis maskinen har flere virtuelle og/eller fysiske nettverksforbindelser, kan IP-adressen som er oppgitt her være feil. I så fall må denne verdien endres til korrekt IP-adresse.

[15] De andre Statusmonitorene må selvsagt også være installert, startet og konfigurert som beskrevet i avsnitt 9 og 10.

Dersom man ikke ønsker at lokal maskin brukes til regneceller, kan man fjerne avkrysningen ”Lokal statusmonitor oppretter regneceller”. Denne er som standard avkrysset, for å minimalisere nødvendig konfigurering av maskiner som bare skal brukes som regneceller.

Når alle feltene er ferdig utfylt, velges ”Oppdater SM” for å konfigurere Statusmonitor-service’en med de nye verdiene.

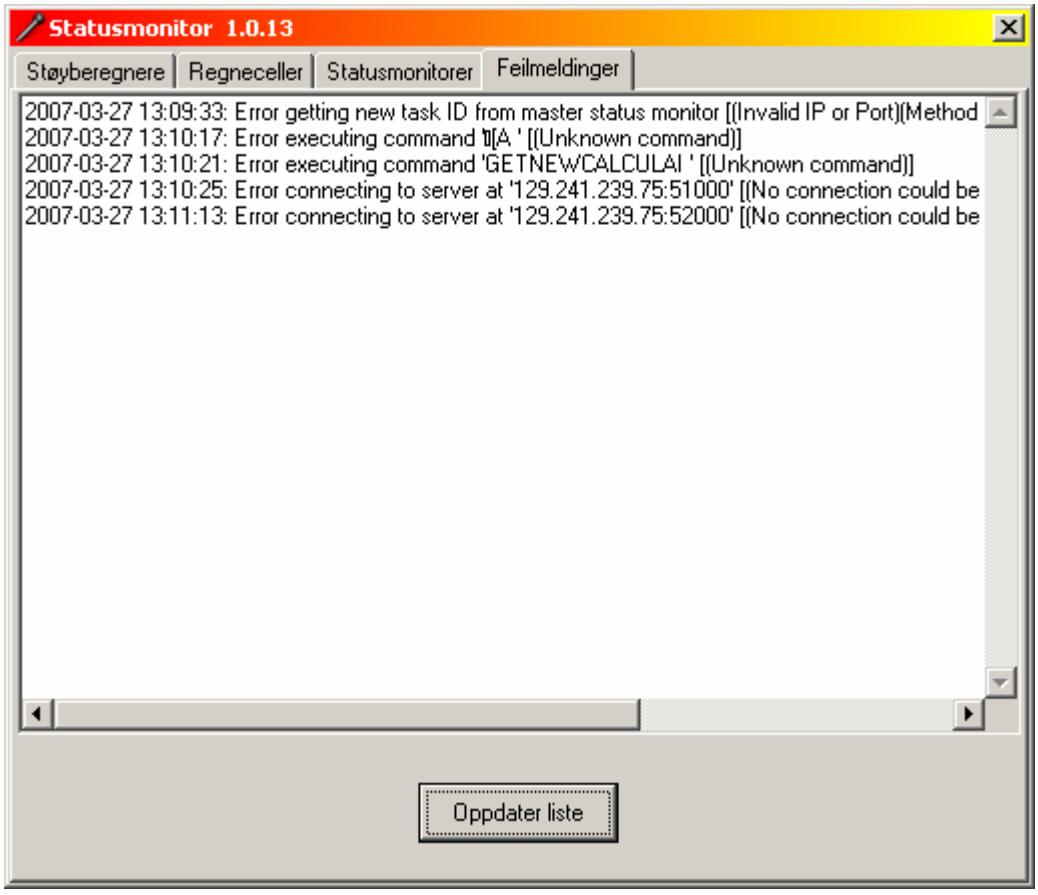
Knappen ”Nullstill verdier” utfører følgende:

- Tømmer listen over andre statusmonitorer
- Tømmer informasjon om statusmonitor med overordnet ansvar
- Henter IP for lokal maskin, og fyller ut dette feltet

Merk at etter nullstilling må IP for lokal maskin kontrolleres på nytt og evt. korrigeres.

10.4 Feilmeldinger

Denne fanen viser feilmeldinger som er oppstått i Statusmonitor-service’en:



Listen oppdateres *ikke* fortløpende, så man må trykke på ”Oppdater liste” for å oppfriske listen over feilmeldinger. Merk også at statusmonitoren lagrer maksimalt 1000 feilmeldinger, slik at svært gamle feilmeldinger i enkelte tilfeller kan være falt ut av listen.