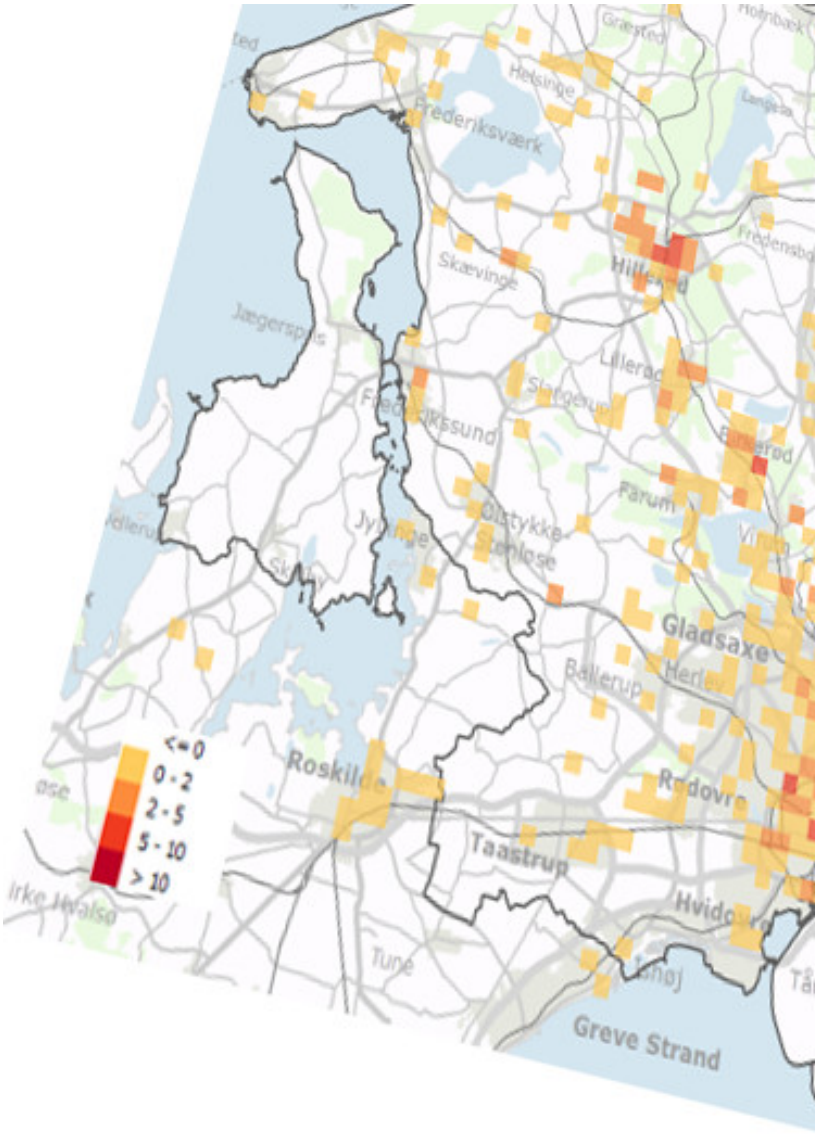
IT Diplomuddannelsen

Grundlæggende Objektorienteret Programmering - Forår 2017

Studerende: Allan Køtter
Studienummer: s140381
Underviser: Inge-Lise Salomon
Dato: 25. november 2017
Ordtælling: 4400 ord



RasterLib: Et .NET Assembly til Raster Data

Indhold

1 Indledning **1**

2 Analyse og design **2**
2.1 Kravspecifikation 2
2.2 Use Cases 2
2.3 Domænemodel 2
2.4 Design af klasser 4

3 Bemærkninger til kodens indhold og opbygning **7**
3.1 Klassen Raster 7
3.2 Klassen Griddef 7
3.3 Klassen Band 8
3.4 Klassen Cell 8

4 Test **9**
4.1 Funktionstest 9

5 Selvstudie: Udvikling af Powershell *Cmdlets* med C# og .NET **10**
5.1 Krav til udvikling af en Cmdlet 10
5.2 Fremgangsmåde ved udvikling af af PSRaster 12
5.3 Distribution af Cmdlets 13

6 Konklusion **14**
6.1 Resultat 14
6.2 Perspektivering 15

Bilag **16**

A Anvendte værktøjer **17**

B Brugervejledning **18**

C Use Cases **22**

D Funktionstest af RasterLib **28**

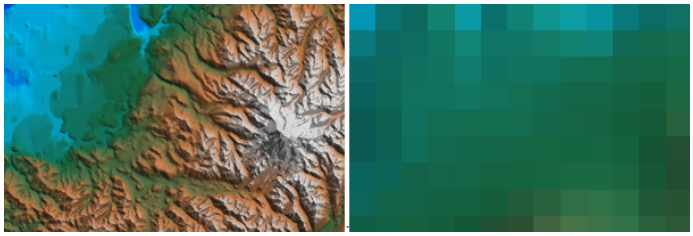
E Udskrift af koden: RasterLib **35**

F Udskrift af koden: PSRaster **59**

1. Indledning

Inden for arbejdet med Geografiske Informations Systemer (GIS) anvendes overordnet set to typer af data – vektorbaserede data og rasterbaserede data. Et rasterdatasæt kan kort beskrives som et eller flere geografisk refererede grid eller "net"af celler, der dækker et bestemt område på kortet. Hver celle indeholder en værdi der repræsenterer aggregeret eller interpoleret information for det område, som cellen dækker. Et grid i et rasterdatasæt, kaldes for et 'bånd', hvor hvert bånd repræsenterer en specifik informationstype.

Rasterdata kendes f.eks. fra vejrudsigten og gammeldags atlas'er, hvor informationstyper som temperatur, nedbør eller lufttryk bliver vist med farveintervaller på et kort. Rasterdata bruges også i mange andre sammenhænge f.eks. luftfotos, hvor billedets RGB værdier findes i hvert sit bånd, og hvor hver pixel i billedet repræsenteres af en celle i hvert af båndene.



*Figur 1: Eksempel på rasterbaseret terrænkort [**PB2015**]. I billedet til højre er der zoomet så langt ind, så de enkelte celler i grid'et kan ses. Hver celle har en værdi svarende til terrænets højde over havoverfladen.*

Jeg arbejder typisk med rasterdata via grafiske brugerflader, som ofte er isolerede analysemiljøer og derfor svære at koble med mere generelle dataprocesseringsopgaver, herunder datavask samt ind- og udlæsning af data i forskellige fil-formater. Dette udføres ofte med PowerShell og det kunne derfor være nyttigt at have muligheden for at arbejde med rasterdata direkte fra PowerShell.

I dette projekt vil jeg programmere et .NET assembly, med basale funktioner til oprettelse, processering og skrivning af rasterdatasæt, hvorved funktionerne kan anvendes i andre programmeringsopgaver. Efterfølgende vil funktionerne blive forsøgt implementeret i et powershell-modul og derigennem blive stillet til rådighed for brugeren. C# og .NET vurderes at være velegnede værktøjer til udvikling af PowerShell moduler, da Powershell er objektorienteret og i forvejen bygget på .NET frameworket.

Selvstudie: udvikling af Powershell moduler med C# og .NET

2. Analyse og design

2.1 Kravspecifikation

Den ønskede funktionalitet skal omfatte oprettelse af rasterdatasæt ud fra en specificeret geografisk udbredelse og cellestørrelse eller ud fra et eksisterende raserdatasæt. I rasterdatasættet skal der kunne oprettes nye bånd med samme grid-dimensioner og eksisterende bånd skal kunne slettes. Der skal være mulighed for at vise, ændre eller slette værdierne i det nye bånds celler. Ved sletning skal cellens værdi opdateres til default null-værdi for rasterdatasættet. Der skal kunne udføres beregninger mellem to eller flere eksisterende bånd, og de beregnede værdier skal kunne lagres i et nyt bånd.

Rasterdatasættet skal kunne eksporteres til et ASCII-grid, der kan indlæses af almindelige GIS-programmer, og der skal være mulighed for at importere ACSII-grids med samme grid-dimentioner som rasterdatasættet. Disse basale rasterfunktioner skal kompileres til et dll-assembly, der kan anvendes i andre projekter. Assemblyet anvendes i dette projekt til at programmere et powershell-modul, der stiller rasterfunktionaliteten til rådighed for brugeren via standardiserede powershell commandlets. I det følgende beskrives Use Cases og det resulterende klasse-diagram for applikationen gennemgås.

2.2 Use Cases

I tabel 1 ses oversigt over use cases, der også er vist som use case diagram på figur 2. Tabellen indeholder en reference til det domæne, som de enkelte use cases vurderes at høre til. Use Case ID i tabellen refererer til uddybende beskrivelser i bilag C.

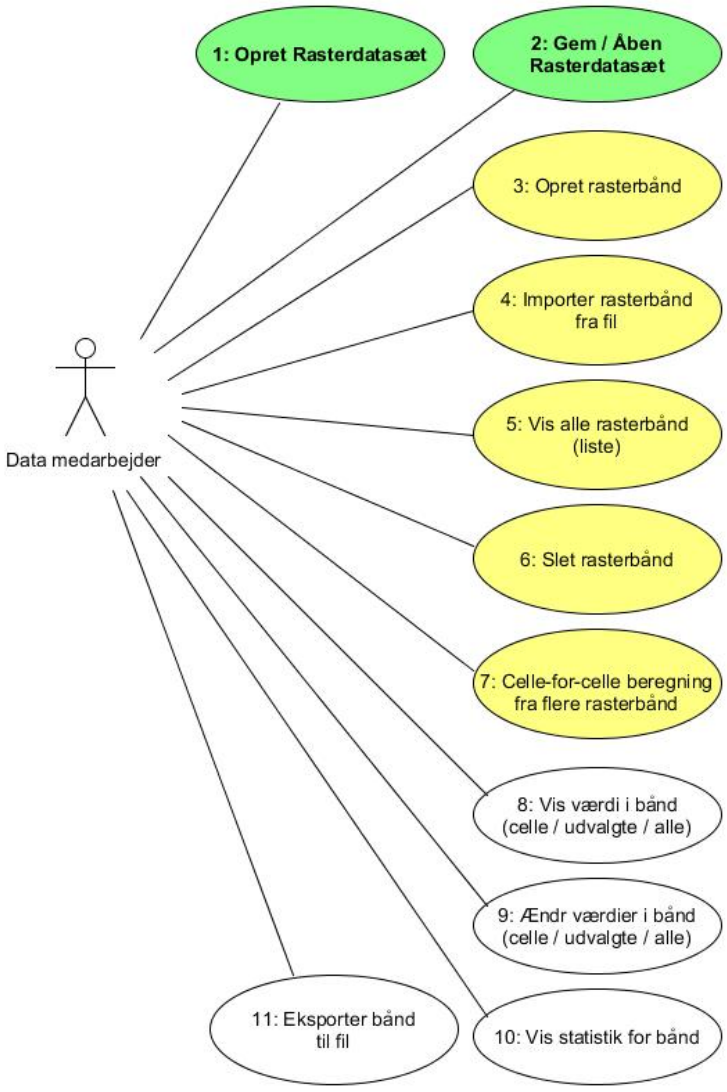
2.3 Domænemodel

Domænet Raster vil være den primære indgang til assembly'et og skal bruges til at oprette et nyt rasterdatasæt, samt arbejde med samlingen af bånd. Domænet raster vil også skulle indehold metadata, der er fælles for alle raserbånd bl.a. en griddefinition, der georefererer rasterdatasættet, og fastlægger dimentionen på båndenes matricer samt cellestørrelsen i matricerne.

Domænet Rasterbånd kan betragtes som værende én "beholder" til bånd, hvori nye bånd kan oprettes og eksisterende bånd ændres, slettes eller hentes fra. Domænet

Tabel 1: Use case oversigt med reference til domæne

ID	Navn	Domæne	Kompleksitet	Prioritet
1	Opret Rasterdatasæt	Raster	Lav	Høj
2	Gem / Åben Rasterdatasæt	Raster	Mellem	Høj
3	Opret rasterbånd	Rasterbånd	Lav	Høj
4	Importer bånd fra fil	Rasterbånd	Høj	Lav
5	Vis alle rasterbånd	Rasterbånd	Høj	Lav
6	Slet Rasterbånd	Rasterbånd	Lav	Høj
7	Beregning af bånd	Rasterbånd	Høj	Mellem
8	Vis værdier i bånd	Bånd	Høj	Mellem
9	Ændre værdier i bånd	Bånd	Høj	Høj
10	Vis statistik for bånd	Bånd	Mellem	Lav
11	Eksporter bånd til fil	Bånd	Mellem	Høj



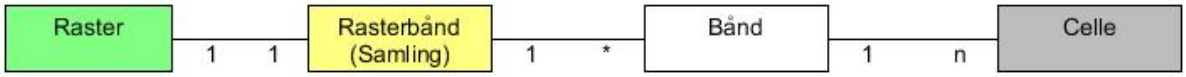
Figur 2: Use Case diagram med 11 Use Cases. Use Cases er oplistet i tabel 1 og er nærmere beskrevet i C

rasterbånd varetager funktioner, der opererer på samlingen af bånd. Der skelnes således mellem use cases, der opererer på samlingen af bånd (rasterbånd) og use cases, der opererer på et enkelt bånd og dette er grunden til opdelingen mellem domænet Rastebånd (samling) og domænet Bånd (det enkelte bånd).

Domænet Bånd varetager alle operationer på det enkelte bånd, herunder operationer der henter, filtrerer eller ændrer båndets data. Båndet er en samling af cellers, hvor operationer på den enkelte celles værdi - f.eks. hent eller opdatér cellens værdi - også håndteres i domænet bånd.

Domænet celle indeholder værdier for cellen og varetager læsning og skrivning af disse.

Domænemodellen er vist på figur 3.



Figur 3: Domænemodel for RasterLib. Domænerne er refereret i tabel 1

2.4 Design af klasser

I figur 4 er designklassediagrammet for *RasterLib* assembly'et vist og i figur 5 er designklassediagrammet for PowerShell modulet *PSRaster* vist.

RasterLib assembly'et

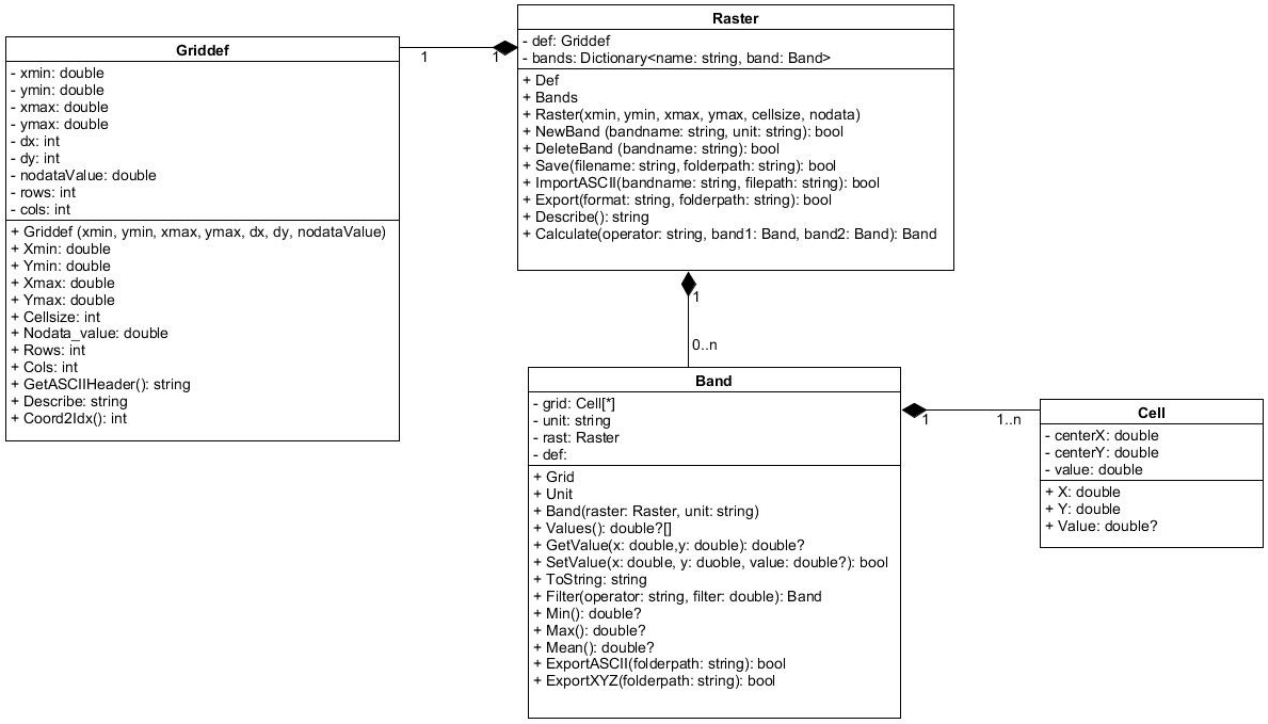
Assembly'et *RasterLib* består af 4 klasser, hvoraf klasserne *Griddef* og *Raster* knytter sig til domænerne *Raster* og *Rasterbånd* og varetager relaterede usecases. Klassen *Bånd* knytter sig til domænet *Bånd* og klassen *Cell* knytter sig til domænet *Celle*, og varetager hver især use cases, der relaterer sig til disse domæner (se tabel 1 og figur 3).

Klassen *Raster* er den overordnede klasse i assembly'et og fungerer som en beholder for alle de Bånd der indgår i Rasterdatasættet. Dette er imlementeret i klassen med attributten *bands* i form af et Dictionary, der indeholder en samling af navne (keys) og bånd (values), som instanser af klassen *band*. Desuden indeholder klassen attributten *def*, som er en instans af klassen *Griddef*. Klassen indeholder metoder til import af et bånd, og eksport af alle bånd samt visning af information om rasterdatasættet.

Klassen *Griddef* varetager alle informationer og funktioner, der relaterer sig til at orientere rasterdatasættet geografisk. Det indeholder attributter til minimums- og

maksimumskoordinater, størrelsen på cellerne, der indgår i rasterdatasætter, samt en metode til at oversætte en koordinat til et index i båndet (Coord2Idx). Desuden indeholder klassen nogle hjælpemetoder til udskrivning (GetASCIIHeader()) og visning (Describe()).

Klassen *band* varetager informationer og funktioner, der relaterer sig til de enkelte bånd i datasættet. Klassen indeholder attributten grid, der er et 1-dimentionelt array af celle-objekter og attributten unit, der indeholder information om enheden på værdierne i båndet. Desuden indeholder klassen metoder til at returnere information om båndet (Min(), Max() og Mean()) samt en metode (GetValue(x,y) der returnerer en værdi fra en celle i arrayet ud fra en koordinat. Båndet indeholder også metoden Filter() til at vise udvalgte værdier i båndet, samt to metoder til at eksportere båndet til tekstfiler - henholdsvist ASCII-format [**ESRI2016:2**] og XYZ-format.



Figur 4: Designklassediagram for RasterLib

Klassen *Cell* varetager informationer og funktioner, der relaterer sig til den enkelte celle i rasterdatasættet. Klassen indeholer attributterne centerX og centerY, der indeholder cellens centerkoordinat samt attributten value, der indeholder cellens værdi. Cellens værdi er implementeret med typen *double?*, der understøtter NULL-værdier (nullable). I klassen er der ikke implementeret nogle metoder.

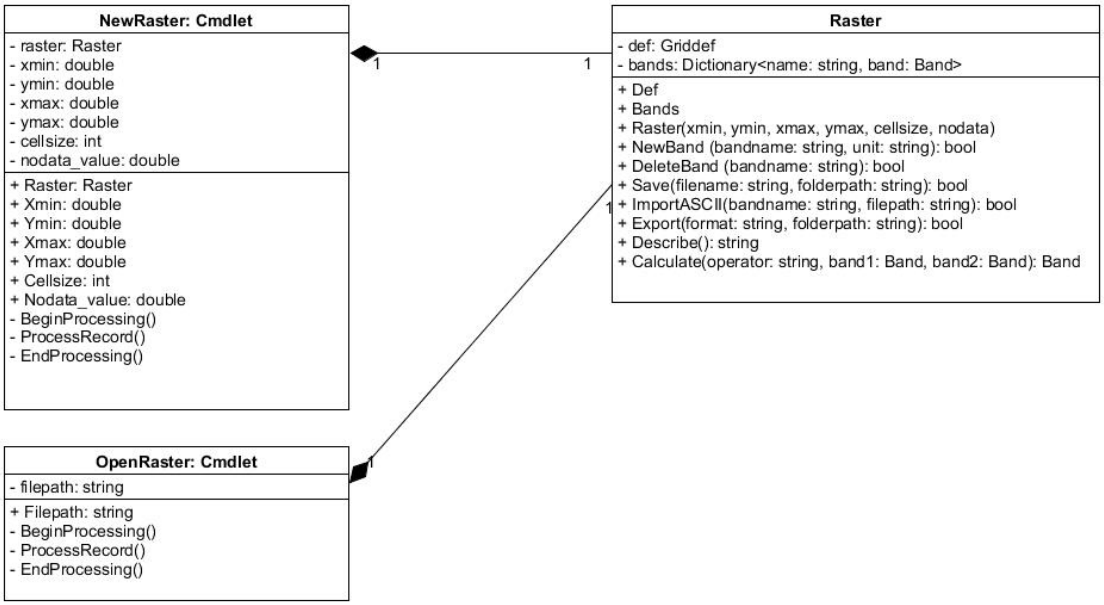
Alle klasserne indeholder desuden properties, med get'ere og set'ere for klassernes attributter.

Powershell modulet PSRaster

Modulet *PSRaster* består af 2 klasser, som hver implementerer en powershell Cmdlet ved at nedarve fra superklassen *Cmdlet* fra .NET assemblyet *System.Management.Automation*.

Klassen *NewRaster* stiller Cmdlet'en `New-Raster` til rådighed i PowerShell modulet. Klassen opretter et nyt rasterobjet og returnerer dette. Klassen har attributter svarende til de inputparametre, der er nødvendige, for at kunne oprettet en griddefenition til det nye raster-objekt. Klassen override'er metoderne *BeginProcessing()*, *ProcessRecord()* og *EndProcessing()*.

Klassen *OpenRaster* stiller Cmdlet'en `Open-Raster` til rådighed i PowerShell modulet. Klassen åbner en gemt rasterobjet-fil og returnerer denne til brugeren. Klassen har en attribut til stien til objekt-filen, som er den inputparameter, som Cmdlet'en kræver. Klassen override'er, lige son den anden Cmdlet-klasse, metoderne *BeginProcessing()*, *ProcessRecord()* og *EndProcessing()*.



Figur 5: Designklassediagram for PSRaster, der implementerer RasterLib assemblyet i et PowerShell modul

3. Bemærkninger til kodens indhold og opbygning

I det følgende beskriver jeg de overvejelser jeg har haft i forbindelse med design af klasserne og de valg jeg har truffet.

3.1 Klassen Raster

Rasterbånd (samling)

Jeg ønsker at kunne referer direkte til et specifikt bånd i samlingen vha. et navn (f.eks. *myRaster.Bands["terraen"]*), idet dette er mere brugervenligt end f.eks. et index. Denne logik er allerede implementeret i den generiske type *Dictionary<Key, Value>*, hvor Key dermed kan sættes til båndets navn og Value sættes til selve båndet. Samtidig vil den grundlæggende listerelaterede funktionalitet være umiddelbart tilgængeligt i form af metoder f.eks. Add, Delete, Count osv.

Til at starte med overvejede jeg i stedet at programmere rasterbånd som en klasse, der nedarvede fra Dictionary, hvorved metoder der hører under rasterbånd-domænet kunne implementeres her. Ved nærmere eftertanke giver det dog bedre mening at implementere rasterbånd-domænets metoder i selve rasterklasse. Dette giver mening bl.a. fordi relationen mellem Raster-klassen og Rasterbånd (samlingen af bånd) er 1-1. Jeg har således valgt at implementere Rasterbånd (samling) direkte som en attribut i Raster-klassen i form af et Dictionary.

3.2 Klassen Griddef

3.2.1 Griddefinitionen: Attributter, Struct eller Klasse?

Argumentet for at arbejde med griddefinitionen som et objekt, frem for at implementere attributter og metoderne direkte i klassen Raster, er princippet om indkapsling / høj kobling. Felter og metoder, der indgår i en griddefinition vil altid skulle bruges sammen og giver ikke mening hver for sig. Da en griddefinition kun skal initieres en gang pr. rasterobjekt, der som udgangspunkt ikke ændres, kan der argumenteres for at implementere griddefinitionen som en Struct.

Dog forventes størrelsen at overstige 16 byte, som er den anbefalede maksimale størrelse for struct'er [**MS2009:1**], og definitionen forventes desuden at skulle bruges af Raster-objektets nestede objekter, og bør derfor være en reference-type lagret på *heap'en*, frem for en værdi-type legret på *stack'en*. Det er derfor valgt at implementerer griddefinitionen som en klasse.

3.3 Klassen Band

Klassen Band er som udgangspunkt et 2-dimensionelt array af objekter af typen Cell. Standard metoderne, der hører til klassen *Array* samt nogle af metoderne, der kan anvendes på et array via *System.Linq*, vil som udgangspunkt være tilstrækkelige i forhold til klassens funktionalitet. Indledningsvist blev muligheden for bare at implementere et 2-dimentionalt array af celler (Cell[,]), eller evt. lade klassen nedarve fra klassen Array overvejet, men klassen Array er en static klasse, og kan derfor ikke nedarves.

Derudover blev jeg opmærksom på, at der også er behov for at *Band* kan indeholde information om enheden (unit) for værdierne i array'ets celler og evt. også andre parametre. Nedarvning fra klassen list blev også overvejet, men nogle af nøglefunktionerne i de generiske typer er uhensigtsmæssige i forhold til klassen Band. Dimensionerne af alle bånd i et raster objekt vil vil være statisk og defineret af klassen GridDef når objektet instantieres. Mulighederne for f.eks. at ændre på størrelsen / dimentionerne på de generiske typer vil derfor være uhensigtsmæssige i relation til klassen *Band*.

3.4 Klassen Cell

Klassen Celle kunne i princippet godt erstattes med en simpel type, der indeholdt cellens værdi. Jeg har dog valgt at implementere cellerne som objekter, idet det giver mulighed for at tilknytte ekstra data og metoder til cellen. I nuværende version af raster-assemblyet har jeg valgt kun at tilknytte attributter og properties for cellens værdi og centerkoordinater, men dette kunne godt udvides. Desuden kunne der etableres polymorfi for klassen Cell, således at cellens værdi kunne indeholde andre typer en double.

4. Test

4.1 Funktionstest

Der er gennemfør funktionstest af Raster-assembly'et *RasterLib*, ved at programmere et C# kommandolinje-program, der anvender assembly'et. Testprogrammet findes i mappen Test og kan køres i Visual Studio, hvis projektets filer oprettes på PC'en som anvist under bilag B.

Testprogrammet tager udgangspunkt i de Use Cases, der er vist i tabel 1 og figur 2. Fra programmets output kan det ses, at alle Use Cases kunne gennemføres, og funktionstesten vurderes derfor at være bestået. Udskrift af testprogrammets output, samt koden til testprogrammet, kan ses i bilag D.

Test af PSRaster

Der er som en del at projektet lavet et PowerShell modul, der anvender assembly'et til at arbejde med rasterdata interaktivt i PowerShell (se afsnit 5). Modulet er testet ved at anvende Cmdlet'en `New-Raster`. Dette genererede et raster-objekt ind i variablen *$r*. Objektet kunne inspiceres med Cmdlet'en `Get-Members` (standard i PowerShell), hvorved alle objektets properties og metoder blev vist.

Det blev forsøgt at tilføje et bånd til rasterobjektet med metoden `r.NewBand(<navn>, <enhed>)`, hvilket fejlede. Fejlmeddelelsen tyder på at metodens parametre ikke parses korrekt til objektet. Metoder, der ikke tager parametre, kunne godt anvendes som f.eks. `r.Describe()`.

Det blev tilsidst forsøgt at instantiere et raster-objekt i PowerShell med standard Cmdlet'en New-Object direkte fra RasterLib assembly'et. Dette lykkedes også og her kunne metoden `r.NewBand(<navn>, <enhed>)` godt anvendes. Efterfølgende kunne en af båndets celler også opdateres ved at ændre en værdi i en af båndets celler.
Det må på den baggrund konkluderes, at det ikke er lykkedes at lave en fuldt implementering af RasterLib assembly'et i et PowerShell modul. Det er dog lykkedes at lave et modul, hvorigennem et raster-objekt kan instantieres.

5. Selvstudie: Udvikling af Powershell *Cmdlets* med C# og .NET

PowerShell er Microsofts standard kommandolinje program, der afløser programmet Cmd og VB-script som det primære automatiserings- og administrationsværktøj i Windows. Til forskel fra traditionelle kommandolinje programmer arbejder PowerShell med .NET objekter i stedet for tekst til input og output. PowerShell-funktionalitet stilles til rådighed for brugeren i form af Cmdlets, der er kommandoer som udfører en specifik opgave. Cmdlets kan udvikles i C# som klasse moduler, der kompileres til dynamic link libraries (DLL) [**Holmes2013**].

.NET namespace'et *System.Management.Automation* indeholder Base-klasser med Powershells kernefunktionalitet, der kan nedarves i forbindelse med udvikling af Cmdlets. Namespacet er relativt omfattende, og i det følgende vil fokus være på klassen Cmdlet, der er relevant for udvikling af powershell Cmdlets [**MS2016:2**]. Overordnet set skal en Cmdlet-klasse nedarves fra en af Base-klasserne *Cmdlet* eller *PSCmdlet* og klassen skal dekoreres med forskellige Cmdlet-attributter, for at PowerShell genkender klassen og dens members.

5.1 Krav til udvikling af en Cmdlet

Ved udvikling af Cmdlets har Microsoft beskrevet en række krav og retningslinjer, der skal overholdes for at udvikle "velformede"og robuste Cmdlets. Formålet med dette er dels en optimal udnyttelse af den generelle funktionalitet, der er indbygget i PowerShell, og dels at give brugeren af de udviklede Cmdlets en konsistent og genkendelig brugeroplevelse [**MS2017:2**]. Nedenfor vil de vigtigste krav blive gennemgået.

Navngivning:

Generelt skal navngivning af Cmdlets følge en specifik systematik, der består af et Verb-Noun par forbundet med en bindestreg [**Holmes2013**, **MS2017:2**]. Verb-delen angiver den "aktion" som Cmdlet'en skal udføre og Noun-delen angiver hvad aktionen skal udføres på eller for. F.eks. vil Cmdlet'en Get-Process hente alle styresystemets kørende processer. Cmdlet'en Import-Module vil importere et PowerShell modul, så det kan anvendes i den aktuelle session osv. En liste over alle tilgængelige moduler i den aktuelle session kan fås ved at bruge Cmdlet'en Get-Command.

Ved udvikling af PowerShell Cmdlets skal denne navngivnings-systematik overholdes, og godkendte Verb'er skal anvendes. Desuden er der en række karakterer, som ikke må indgå i navngivningen [**MS2017:2**]. Navnet på Cmdlet'en skal identificeres, ved at dekorere klassen med `[Cmdlet(Verb, ''Noun'')]`.

Verb angives her med en af PowerShell's navngivne enumeration-klasser, som er lister med de godkendte Verb'er. F.eks. angives *VerbsCommon.Get* som Verb, hvis aktionen skal være "Get", fordi Get findes under enumeration-klassen *VerbsCommon*, som definerer generelle aktioner, der kan bruges i mange sammenhænge. Der findes i alt 7 enumeration-klasser med godkendte Verb'er (se [**MS2017:2**]). Selve klassen navngives typisk med syntaksen `VerbNounCommand`.

Parametre:

Cmdlet-klassens properties, der skal indeholde de parametre, som Cmdlet'en kan eller skal modtage, dekoreres med `[Parameter()]`. I parentesen kan en række attributter til parameteren angives for at modificere parametrenes egenskaber. F.eks. kan *Position = 0..n* præcisere rækkefølgen, som parameteren skal angives i, og "Mandatory = true/false" angiver, om parameteren er påkrævet eller ej. "ValueFromPipeline = true/false" angiver om parameteren kan modtages fra en anden Cmdlet via en pipeline. Dette er de oftest anvendte attributter, men der findes en række andre attributter som kan sættes for at modificere egenskaberne for parameteren (se [**MS2017:2**]).

En Cmdlet-klasses properties, der på den måde "bindes" til parametre, deklareres derudover på normal vis med en *Get* og en *Set* metode.

I Powershell-Cmdlets er der indbygget en række standard parametre, der relaterer sig til PowerShell's basale funktionalitet. Disse parametre kan ikke overskrives eller ændres, og navngivningen af parametre i Cmdlet-klasser må derfor ikke være sammenfaldende med disse. Der er desuden indbygget en række standardparametre i Powershell-Cmdlets, der relaterer sig til interaktive sessioner. Disse bør overskrives, hvis Cmdlet'en man udvikler skal bruges interaktivt - dvs. fra kommandolinjen [**MS2017:2**, **MS2017:4**].

Processerings-metoder

PowerShell Cmdlets indeholder tre *Input Processerings Metoder*, der bruges til at processere det objekt, som Cmdlet'en skal bearbejde. Metoderne er *BeginProcessing*, *ProcessRecord* og *EndProcessing*. For at Cmdlet'en kan blive en del at PowerShell-miljøet, skal mindst en af disse metoder override's.

BeginProcessing bliver kaldt én gang når Cmdlet'en aktiveres, og bruges til præprocessering af data og objekter. *ProcessRecord* kaldes flere gange når en Cmdlet aktiveres, og bruges til iterationer - f.eks. hvis et dataobjekt har en liste af indlejrede objekter, der hver især skal processeres. *EndProcessing* kaldes livesom BeginProces-

sing kun én gang, og bruges til efterbehandling af data og objekter.

Output Objekter

Hvis en PowerShell Cmdlet skal returnere et objekt, anvendes metoden `WriteObject()` fra en af de 3 processerings-metoderne, der er nævnt ovenfor. For at brugeren kan anvende standart PowerShell værktøjer på det returnerede objekt, skal objektet dokumenteres.

Dels skal typen af output objektet deklareres, hvilket gøres ved at dekorere klassen med `[OutputType()]` (se "OutputType Attribute Declaration"under [**MS2017:3**]). Dels skal objektets *members* dokumenteres. Dokumentationen af output objektet sker ved at lave et modul manifest,

Robust fejlhåndtering

Hvis der opstår en fejl i koden, der forhindrer Cmdlet'en at færdiggøre processeringsmetoderne således at Cmdlet'en afsluttes, skal metoden *ThrowTerminationError* anvendes. Fejl, der ikke får Cmdlet'en til at stoppe ("ikke-terminerende fejl"), fanges med metoden *WriteError*. Begge typer af fejl kan have alvorlige konsekvenser, hvis f.eks. Cmdlet'en skal gennemføre nogle system-relaterede ændringer eller andre transaktioner, der kræver, at alle dele af processen kører til ende for at lykkes.

Begge metoder kan referere til et *ErrorRecord* objekt, der indeholder værdifuld information, som brugeren kan bruge til at få indsigt i, hvad der gik galt. *ErrorRecord* kræver at der opstår en exception, og standardmetoderne i C# til at fastslå typen af exeption kan bruges til at aktivere den rette metode.

5.2 Fremgangsmåde ved udvikling af af PSRaster

Udvikling af et Powershell modul starter således med oprettelsen af et C# projekt af typen Class Library. I dette projekt er det oprettet under namespacet *PSRaster* under mappen PSRaster i eksamensprojektmappen.

Namespace'et *System.Management.Automation* er ikke er en del af .NET core libraries, og for at kunne anvende det, skal namespace'et installeres i projektet, hvilket gøres med NuGet, som er Visual Studios Package Manager. Installationen er sket ved at åbne NuGet-konsollen og indtaste kommandoen:
`Install-Package System.Management.Automation`.

Ud over at etablere en reference til *System.Management.Automation*, er der også blevet etableret en reference til RasterLib.dll, der for nemheds skyld er kopieret til mappen PSRaster. Begge namespaces kan herefter aktiveres på normal vis med `Using <namespace>`. I PSRaster-projektet er der oprettet en klasse-fil til hver af de to Cmdlets hhv. *NewRaster.cs* og *OpenRaster.cs*. I hver af filerne er der program-

meret en Cmdlet-klasse, der nedarver fra baseklassen *Cmdlet*.

Cmdlet'en *New-Raster* skal instantierer et nyt Raster-objekt, og skal derfor kaldes med parametrene som er nødvendige for at etablere grid-definitionen. Cmdlet'en *Open-Raster* skal åbne en gemt Raster objektfil, og skal derfor kaldes med en sti til den pågældende fil som parameter. Disse parametre er blevet indbygget i de respektive Cmdlet-klasser som properties med tilhørende attributter som beskrevet ovenfor.

Processeringen i de to Cmdlets er lavet meget simpelt. Der er lavet en override af metoden `ProcessRecord()`, hvori Raster objektet genereres. Der er desuden lavet en override af metoden `EndProcessing()` hvori rasterobjektet returneres med metoden `WriteObject()`. Der er samtidig etableret en meget simpel fejlhåndtering med `try` og `catch`, men der er ikke skelnet mellem forskellige typer af exeptions.

Koden til PSRaster projektet, der til sidst er kompileret til et PowerShell modul-assembly, kan ses i bilag F

5.3 Distribution af Cmdlets

Det kompilerede assembly, der indeholder de pågældende Cmdlet klasser, kan bruges direkte som en binær modulfil, og importeres med kommandoen: `Import-Module <sti-til-assembly>`, hvilket er anvendeligt i forhold til test af Cmdlet-klasserne.

Men hvis man ønsker at stille de udviklede Powershell Cmdlets til rådighed for andre brugere, gøres dette dog typisk i form af et egentligt PowerShell modul. Modulet oprettes ved at kopiere assemblyet til en mappe med samme navn som modulet, hvori der også oprettes et modul-manifest med samme navn og en hjælpefil. Denne pakke kan så distribueres til brugeren, der skal kopiere mappen til en af de systemspecifikke modul-mapper hos brugeren, og aktiveres med kommandoen `Import-Module <modulnavn>`.

Modul-manifestet dokumenterer modulet med metadata og sikrer samtidig, at andre assembly'er, som modulet er afhængige af, åbnes inden selve den binære modulfil importeres i PowerShell. I forhold til *PSRaster* er der en afhængighed til *RasterLib*, hvilket er registreret i modul-manifestet. Der er i dette projekt lavet et egentlig powershell-modul inkl. modul-manifest, der kan distribueres til andre brugere. Modulet findes i projektmappen under mappen med navnet *Modules*.

6. Konklusion

Målet med projektet har været at udvikle et .NET assembly, der stiller funktionalitet til oprettelse, processering og skrivning af rasterdatasæt til rådighed, således at funktionaliteten kan anvendes i andre programmeringsopgaver. Projektet har desuden haft til formål at anvende det udviklede assembly til udvikling af et Powershell modul, der stiller raster-funktionaliteten til rådighed via en eller flere Powershell Cmdlets.

Projektet er blevet gennemført ved at opstille kravspecifikationer til funktionaliteten af assembly'et, og ud fra disse opstille en række Use Cases. Ud fra de formulerede Use Cases, er der opstillet UML designklassediagrammer, der viser hvordan den ønskede funktionalitet forventes at blive implementeret som klasser i det objektorienterede paradigme.

Assemblyet er derefter blevet programmeret i C#, og testet med et simpelt kommandolinjeprogram. Testen er gennemført som en Funktionstest og har taget udgangspunkt i funktionaliteten beskrevet i de formulerede Use Cases. Designprocessen, implementering af klasserne i C# og funktionstest er gennemført som en Unified Process (UP) med flere iterationer og tilhørende justeringer af både klassedesign, kode og test. Kravspecifikationen og Use Cases er der ikke blever ændret på.

Efter gennemført funktionstest af assembly'et, er PowerShell modulet blevet programmeret. Der er implementeret 2 Cmdlets i modulet, der kan oprette og returnere et nyt Raster-objekt og åbne et eksisterende, gemt Raster-objekt. Raster-objektets properties og metoder er automatisk til rådighed, når objektet er oprettet.

6.1 Resultat

I projektet er der blevet udviklet et .NET assembly med navnet RasterLib.dll, der indeholder funktionalitet til at arbejde med rasterdatasæt. Assembly'et indeholder funktionalitet til at *oprette* nye rasterdatasæt og *gemme* disse som objektfiler, så de efterfølgende kan *åbnes og indlæses*. Assemblyet kan desuden tilføje nye bånd til rasterdatasættet, og *ændre* værdien i båndets celler. Assemblyet giver mulighed for at vise information om datasættet, herunder vise liste med alle bånd med statistik for disse, *vise alle* data i et bånd, *vise udvalgte* data i et bånd samt udføre beregninger på to bånd. Et bånd kan *slettes* fra datasættet og samlingen af bånd kan eksporteres til et ASCII-grid, som er et standard tekstbaseret filformat. ASCII-grids kan desuden importeres til nye bånd i rasterdatasættet

Der er desuden udviklet et PowerShell modul, der stiller funktionaliteten til rådighed via to Cmdlets. Cmdlet'en *New-Raster* kan oprette et nyt rasterobjekt, og returnere dette i en PowerShell session, således at rasterobjektets properties og metoder er tilgængelige. Cmdlet'en *Open-Raster* kan åbne en gemt raster objektfil, og returnere dette i form af et rasterobjekt i en PowerShell session.

Anvendelse af nogle af rasterobjektets metoder - bl.a. de metoder der skal modtage parametre - har jeg ikke kunne få til at fungere via de udviklede Cmdlets. Årsagen til dette vurderes at skyldes den måde at PowerShell overleverer (parser) parametre til objektet. Ved instantiering af et rasterobjekt direkte fra RasterLib assembly'et med Cmdlet'en `New-Object` kan de samme metoder dog godt anvendes. Det har inden for projektets tidsramme ikke været muligt at forfølge dette yderligere.

6.2 Perspektivering

Inden for .NET udvikling er muligheden for at udvide frameworkets kernefunktionalitet med egenudviklede klasser og assembly'er en central og væsentlig styrke ved .NET. Det udviklede *RasterLib* assembly, der er programmeret i dette projekt, er funktionsdygtigt og vil kunne anvendes til mange fremtidige opgaver og projekter. I dette projekt er assembly'et forsøgt anvendt til at stille raster-funktionalitet til rådighed i et PowerShell modul, men det kunne lige så godt have været i et kommandolinje-program, en grafisk Windows applikation eller en ASP.NET hjemmeside, at der var blevet indbygget raster-funktionalitet vha. *RasterLib*.

Funktionaliteten der i øjeblikket ligger i *RasterLib*, er relativ simpel, men den kunne godt udvides. Muligheder for udbygning af assembly'et kunne være grafisk visualisering af bånd, algoritmer til at interpolere punkter inden for det geografiske område, algoritmer til at genererer iso-kurver fra cellerne i et bånd osv. Udvidede muligheder for at importere andre rasterdata - f.eks. rasterdata med en anden geografisk udbredelse - ville også være en værdifuld tilføjelse.

Muligheden for at udvikle kompilerede PowerShell Cmdlets vha. C# og .NET frameworket, og der igennem udvikle special funktionalitet i forhold til specifikke behov, er efter min mening en lige så stor styrke ved PowerShell, som ikke kendes i andre kommandolinje-programmer. PowerShell har i forvejen muligheden for at tilgå .NET assembly'er direkte, og da programmering af Cmdlets er relativt kompliceret, bør denne mulighed overvejes. Det Powershell modul, som jeg har udviklet i dette projekt, har så enkelt en funktionalitet, at det lige så godt kunne have været lavet som et almindeligt scriptbaseret modul.

Men med de kompilerede / programmerede moduler kan man opnå en række fordele, som man ikke opnår med scriptbaserede moduler. F.eks. vil et kompileret modul kunne optimere hastigheden på tung og langvarig dataprocessering. C# og .NET giver også mulighed for at implementere avancerede metoder, som f.eks. multithreading i kompilerede Cmdlets, hvilket er svært at implementere med et PowerShell script [**MS2017:5**]

Bilag

A. Anvendte værktøjer

Projektet er gennemført ved at anvende følgende værktøjer:

UMLet (v. 14.2): Open Source program til at oprette UML Diagrammer. Programmet er blevet anvendt til processen omkring analyse og design af klasserne. For yderligere information se: `http://www.umlet.com`

Visual Studio Community 2017: Gratis udgave af Microsofts Integrated Developer Environment (IDE) til udvikling af Windows-, web- og mobilapplikationer. For yderligere information se: `http://www.visualstudio.com/`

Udvikling og test er sket på en Lenovo ThinkPad W540 PC med Windows 10 64 bit som styresystem og .NET version 4.5 installeret. Funktionstesten er desuden med held kørt på en Macbook Pro med Visual Studio Community 2017 samt mono version version 5.0.

B. Brugervejledning

RasterLib

For at anvende assembly'et Rasterlib i et Visual Studio projekt, kopieres filen RasterLib.dll til mappen *packages* under den oprettede projektmappe, og der tilføjes en reference i projektet til assembly'et. Herefter kan assembly'et anvendes i koden med *using*-direktivet.

Raster-objektet

Der initialiseres en ny instans af et raster-objekt til variablen r med constructoren:
`Raster r = new Raster(xmin, ymin, xmax, ymax, cellsize, nodata_value)`

... hvor xmin, ymin, xmax og ymax definerer hjørnekoordinaterne for det område, som rasterdatasættet skal dække. Cellsize er størrelsen på cellerne i meter og nodata_value er den værdi, som identificerer manglende data, når et bånd skal eksporteres som ASCII-filer. Sammen med raster-objektet initieres også en ny instans af et Griddef-objekt, ud fra de angivne parametre med hjørnekoordinater, cellestørrelse og nodata_value

Properties:

r.Def: Returnerer raster-objektets griddefinition i form af et griddefinitionsobjekt. Griddefinitionen oprettes som et led i instantieringen af et nyt rasterobjekt. Set metoden er private og griddefenitionen kan dermed ikke ændret.

r.Bands: Returnerer et *Dictionary* objekt med en samling af navne-strenge som `<key>` og bånd-objekter som `<values>`. Metoder, der hører til et *Dictionaty* er tilgængellige f.eks. vil metoden `r.Bands.Remove("<bandname>")` slette det pågældende bånd fra samlingen. Bands er tomt, når en nyt raster-objekt instantieres.

Metoder:

r.NewBand([string] bandname, [string] unit): Opretter et nyt bånd i samlingen med navnet <bandname> og enheden <unit>.

r.Save([string] filename, [string] folderpath): Gemmer raster-objektet som en binær objektfil i mappen <folderpath> med navnet <*filename*>.*ras*, der efterfølgende kan åbnes og cast'es til et rasterobjekt.

r.ImportASCII([string] bandname, [string] filepath): Importerer et ASCII-grid (se [**ESRI2016:2**]) ind i et nyt bånd med navnet <bandname>.

r.Export([string] format, [string] folderpath): Eksporterer alle bånd i rasterdatasættet til mappen <folderpath>. På nuværende tidspunkt understøtter RasterLib kun formaterne XYZ og ASCII-grid.

r.Describe(): Returnerer en tekststreng med en beskrivelse af datasættet, herunder griddefinitionen samt en oplistning af båndene i datasættet med statistik for cellernes data-værdier.

r.Calculate([string] operator, [Band] band1, [Band] band2): Udfører beregninger mellem to bånd og returnerer et nyt båndobjekt med resultaterne. Understøtter på nuværende tidspunkt kun beregninger med 2 bånd og operatorerne +, -, *, /.

Griddef-objektet

Griddef-objektet instantieres sammen med raster-objektet og kan efterfølgende ikke ændres. Som udgangspunkt er der ikke behov for at anvende properties og metoder i denne klasse, når man anvender RasterLib - klassens members er primært støttefunktioner til de øvrige klassers metoder.

Properties:

r.Def.Xmin: X-koordinaten på nederste venstre punkt i griddet

r.Def.Ymin: Y-koordinaten på nederste venstre punkt i griddet

r.Def.Xmax: X-koordinaten på øverste højre punkt i griddet

r.Def.Ymax: Y-koordinaten på øverste højre punkt i griddet

r.Def.Cellsize: Returnerer et heltal, med cellernes geografiske størrelse i meter. Assemblyet understøtter på nuværende tidspunkt kun kvadratiske celler

r.Def.NodataValue: Værdi som indikerer manglende data, når et bånd skal eksporteres til et ASCII-grids

r.Def.Rows: Returnerer et heltal, med antallet af rækker, som grid'et er inddelt i

r.Def.Cols: Returnerer et heltal, med antallet af kolonner, som grid'et er inddelt i

Metoder:

r.DefGetASCIIHeader(): returnerer en tekststreng med den header-information, som bruges i et ASCII-grid (se [**ESRI2016:2**])

r.Describe(): Returnerer en tekststreng med en beskrivelse af griddefenitionen.

r.Def.Coord2Idx([double] x, [double] y: omregner en xy-koordinat til et index i båndenes array med celler, som returneres. Bruges bl.a. internt til at opdaterer værdierne i et bånd ud fra en koordinat. Genererer en exeption hvis koordinaten ligger uden for rasterend geografiske udbredelse.

Band-objektet

Band-objekter oprettes med metoden `r.NewBand(<bandname>, <unit>)` som beskrevet ovenfor, og bliver derefter tilgængelige via Dictionary-objektet Bands i Rasterobjektet: `r.Bands[<bandname>]`

Properties:

r.Bands[<bandname>].Grid: Returnerer et 1-dimentionelt array af celle-objekter af størrelsen *r.Def.Rows * r.Def.Cols*. Kan bruges til at tilgå de enkelte celler i griddet vha. index.

r.Bands[<bandname>].Unit: Returnerer en streng med enheden for båndets værdier.

Metoder:

r.Bands[<bandname>].Values(): Returnere værdierne i et bånd i form af et 1D array af type [double?].

r.Bands[<bandname>].GetValue([double] x, [double] y): Returnerer værdien fra den celle, som xy-koordinater ligger inden for. returtypen er [double?]

r.Bands[<bandname>].SetValue([double] x, [double] y, [double] value): Ændre værdien i den celle, som xy-koordinater ligger inden for til `<value>`. Returtnerer *True* hvis cellen blev ændret.

r..Bands[<bandname>].ToString(): Returnerer en tekststreng, med værdier i båndet, formateret i rækker og kolonner.

r..Bands[<bandname>].Filter([string] operator, [double] filter): Celle-for-celle sammenligning med filterværdien i båndet givet ved sammenligningsoperatoren f.eks. <, >, ==, !=. Returnerer et nyt bånd-objekt, med værdien 1 i de celler, hvor sammenligning returnerer True og null i de celler, hvor sammenligningen returnerer False. Anvendes til at identificerer celler i et bånd, der f.eks. er større end en givet filterværdi.

r..Bands[<bandname>].Min(): Returnere minimumværdien i båndet af typen [double?].

r..Bands[<bandname>].Max(): Returnere maksimumværdien i båndet af typen [double?].

r..Bands[<bandname>].Mean(): Returnere middelværdien i båndet af typen [double?]. Middelværdien beregnes på baggrund af de celler med en værdi - dvs. celler der ikke er NULL.

r..Bands[<bandname>].ExportASCII([string] folderpath): Eksportere båndet som et ASCII-grid til mappen <folderpath> - dvs. gemmer båndet i et tekstbaseret raster filformat.

r..Bands[<bandname>].ExportXYZ([string] folderpath): Eksportere båndet som en CSV-fil med x-, y- og z-værdier i mappen <folderpath>

Cell-objektet

Cell-objekterne i et bånd kan tilgås ved `r.Bands[<bandname>].Grid[<idx>]`, hvor <idx> er et index i array'et Grid.

Properties:

r.Bands[<bandname>].Grid[<idx<].CenterX: Returnerer en [double] med x-værdien for cellens centerkoordinat.

r.Bands[<bandname>].Grid[<idx<].CenterY: Returnerer en [double] med y-værdien for cellens centerkoordinat.

r.Bands[<bandname>].Grid[<idx<].Value: Returnerer en [double?] med værdien i cellen. Værdien kan være NULL.

C. Use Cases

UC1: Opret rasterdatasæt

Aktør: Medarbejder (Dataanalytiker)

Betingelser: Medarbejder har installeret powershell modulet psRaster under sin brugerprofil.

Succes kriterier: Medarbejder kan oprette et raster-objekt i en ny varaibel i Powershell og objektet kan inspiceret med powershell-kommandoen Get-Member.

Primær arbejdsgang:

1. Medarbejder åbner Powershell
2. Medarbejder indtaster kommandoen Import-Module psRaster
3. Beskeden "psRaster was loaded" vises på kommandolinjen
4. Medarbejder indtaster kommandoen:
   $myRaster = New-Raster 670004.32, 6161030.5, 694990.2, 6181990.4, 1000, -9999
5. Medarbejder indtaster kommandoen: Get-Member $myRaster
6. Rasterobjektets members vises
7. Medarbejder taster: $myRaster.Defenition.GetASCIIHeader()
8. ASCII-header viser at rasterdatasættet har 21 kolonner og 25 rækker "

UC2: Gem / Åben rasterdatasæt

Aktør: Medarbejder

Betingelser: Medarbejder har i powershell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1. Mappen C:
Temp eksisterer.

Succes kriterier: Medarbejder kan gemme et raster-objekt til disk med metoden Save og efterfølgende åbne det igen med Powershell commandlet'en Open-Raster

Primær arbejdsgang:

1. Medarbejder indtaster kommandoen:
   $myRaster.Save C:\Temp\myraster.ras

2. Medarbejder lukker Powershell
3. Medarbejder åbner Powershell
4. Medarbejder indtaster kommandoen: Import-Module psRaster
5. Beskeden ”psRaster was loaded” vises på kommandolinjen
6. Medarbejder indtaster kommandoen:
   $savedRaster = Open-Raster C:\Temp\myraster.ras
7. Medarbejder taster:
   $savedRaster.Defenition.GetASCIIHeader()
8. Information på skærmen viser at rasterdatasættet har 21 kolonner og 25 rækker

UC3: Opret rasterbånd

Aktør: Medarbejder

Betingelser: Medarbejder har i powershell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1.

Succes kriterier: Medarbejderen kan se det nyoprettede bånd med kommandoen $myRaster.Describe()

Primær arbejdsgang:

1. Medarbejderen kontrollere rasterdatasætte med kommandoen $myRaster.Describe()
2. Medarbejder indtaster kommandoen:
   $myRaster.Bands.Add("MitNyeBaand", New Band(self, "m"))
3. Beskeden "Band was added"vises
4. Medarbejderen indtaster kommandoen $myRaster.Describe()
5. Information på skærmen viser at der er tilføjet et nyt bånd med navnet "MitNyeBaand"

UC4: Importer rasterbånd fra fil

Aktør: Medarbejder

Betingelser: Medarbejder har i powershell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1.

Succes kriterier: Medarbejderen kan se det importerede bånd med kommandoen $myRaster.Describe()

Primær arbejdsgang:

1. Medarbejderen kontrollere rasterdatasætte med kommandoen $myRaster.Describe()

2. Medarbejder indtaster kommandoen:
   $myRaster.Import("C:tempimportras.asc", "ImportBaand", "m"))
3. Beskeden "Band was imported"vises
4. Medarbejderen indtaster kommandoen $myRaster.Describe()
5. Information på skærmen viser at der er tilføjet et nyt bånd med navnet "ImportBaand"

UC5: Vis alle rasterbånd (liste)

Aktør: Medarbejder

Betingelser: Medarbejder har i powershell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har 2 eller flere bånd liggende i rasterdatasættet.

Succes kriterier: Medarbejderen kan få vist alle de eksisterende bånd med kommandoen $myRaster.Describe()

Primær arbejdsgang:

1. Medarbejderen indtaster kommandoen $myRaster.Describe()
2. Information på skærmen viser alle bånd i rasterdatasættet.

UC6: Slet rasterbånd

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har 1 bånd med navnet "MitNyeBaand"liggende i rasterdatasættet.

Succes kriterier: Medarbejder får ikke vist det slettede bånd, når kommandoen $myRaster.Describe() anvendes.

Primær arbejdsgang:

1. Medarbejderen indtaster kommandoen:
   $myRaster.Bands.Delete("MitNyeBaand")
2. Meddelelsen "Båndet "MitNyeRaster"blev slettet"vises
3. Medarbejderen indtaster kommandoen $myRaster.Describe()
4. Information på skærmen viser at båndet ikke længere findes i datasættet

UC7: Celle-for-celle beregning

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har 2 eller flere bånd liggende i rasterdatasættet med samme enhed.

Succes kriterier: Der er oprettet et nyt bånd i rasterdatasættet, hvor resultatet af beregningen gemmes.

Primær arbejdsgang:

1. Medarbejder indtaster kommandoen:
   $resultat = $myRaster.Calculate("baand1","baand2", -")
2. Medarbejder indtaster kommandoen:
   $myRaster.Bands.Add("ResultatBaand", $resultat)
3. Beskeden "Band was added"vises
4. Medarbejderen indtaster kommandoen:
   $myRaster.Describe()
5. Information på skærmen viser at der er tilføjet et nyt bånd med navnet "ResultatBaand"
6. Medarbejderen indtaster kommandoen:
   $myRaster.Bands("ResultatBaand").View()
7. Resultatet af beregningen vises på skærmen

UC8: Vis værdi i bånd (celle / udvalgte / alle)

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har mindst 1 bånd i rasterdatasættet.

Succes kriterier: Medarbejderen kan få returneret en værdi fra en celle vha. koordinater inden for rasterdatasættets geografiske område.

Primær arbejdsgang:

1. Medarbejder indtaster kommandoen:
   $myRaster.bands("Baand1").GetValue(682258.42, 6175223.1)
2. Værdien for den celle som koordinaten 670004.32, 6161030.5 ligger inden for vises på skærmen.

UC9: Ændre værdier i bånd

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har mindst 1 bånd i rasterdatasættet.

Succes kriterier: Værdien i en specifik celle i båndet er ændret

Primær arbejdsgang:

1. Medarbejderen indtaster kommandoen:
   $myRaster.Bands("Baand1").Viev()
2. Båndets værdier vises på skærmen
3. Medarbejderen indtaster kommandoen:
   $myRaster.Bands("Baand1").Grid[0] = 199.99
4. Medarbejderen indtaster kommandoen:
   $myRaster.Bands("Baand1").Viev()
5. Båndets værdier vises på skærmen og første celles værdi er ændret til 199.99

UC10: Vis statistik for bånd

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har mindst 1 bånd i rasterdatasættet.

Succes kriterier: Statistiske oplysninger for båndene fremgår, når kommandoen $myRaster.Describe() anvendes.

Primær arbejdsgang:

1. Medarbejderen indtaster kommandoen: $myRaster.Describe()
2. Information på skærmen viser minimum, maksimum og gennemsnit for båndene i rasterdatasættet.

UC11: Eksporter bånd til fil

Aktør: Medarbejder

Betingelser: Medarbejder har i PowerShell oprettet et raster-objekt i variablen $myRaster som beskrevet under UC1 og har mindst 1 bånd i rasterdatasættet.

Succes kriterier: Der er oprettet en fil af typen ASCII-grid [**ESRI2016:2**] i den specificerede mappe, som indeholder værdierne fra det eksporterede bånd

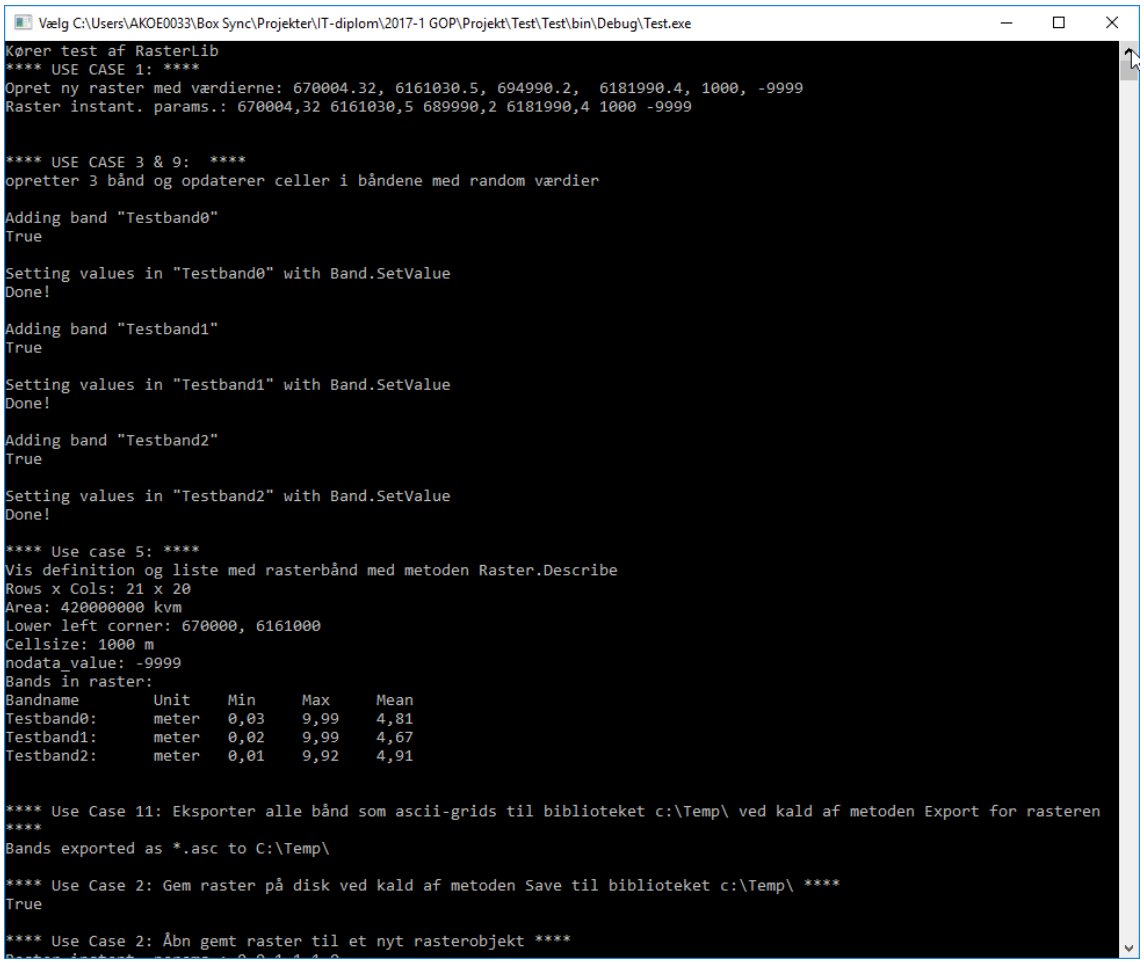
Primær arbejdsgang:

1. Medarbejderen indtaster kommandoen:
   $myRaster.Export("C:temp", "ASCII")

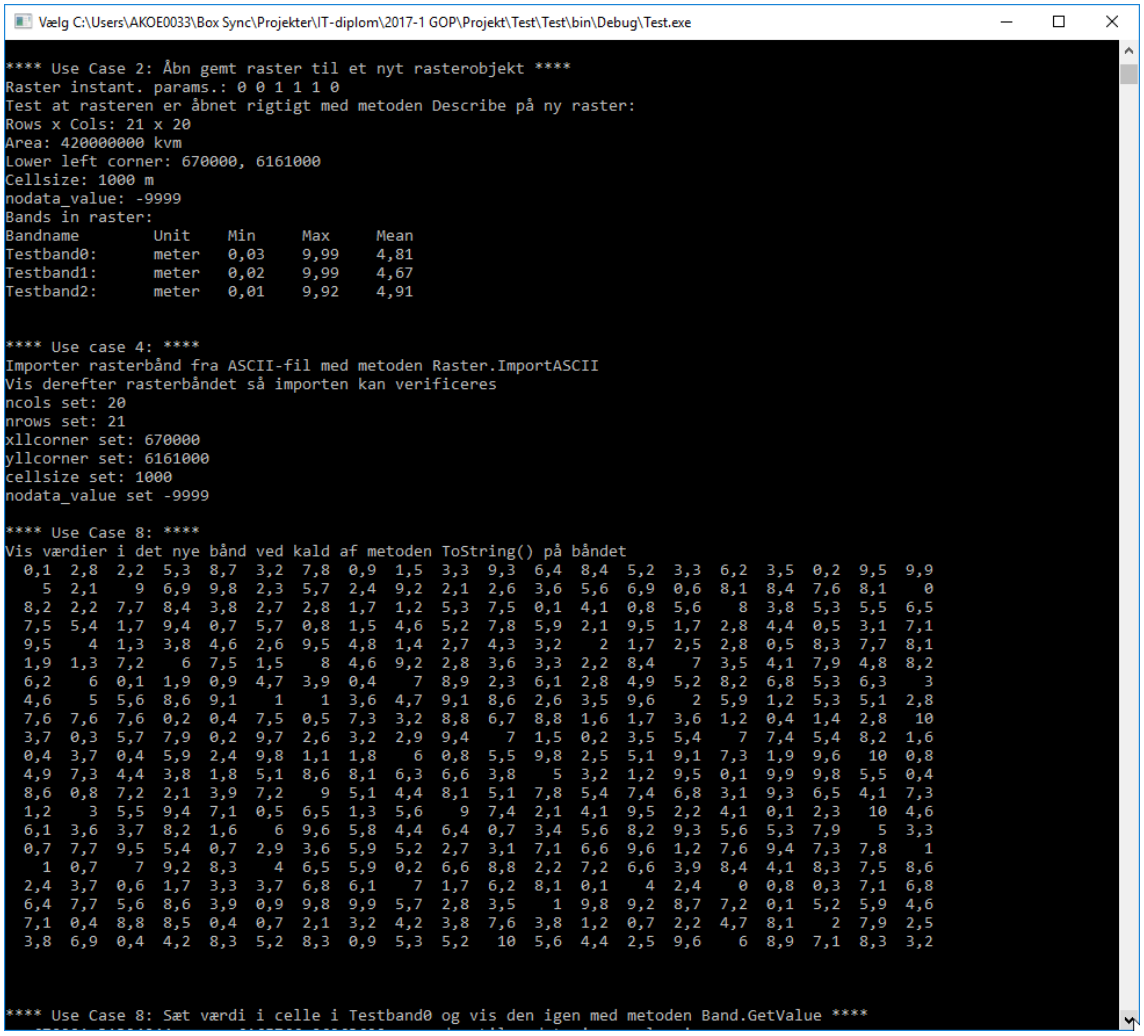
2. Beskeden "Rasterdata was exported to C:
   temp
   "vises på skærmen
3. I mappen C:
   temp er der nu oprette en fil for hver bånd i rasterdatasættet, navngivet med "[båndets navn].asc"

D. Funktionstest af RasterLib

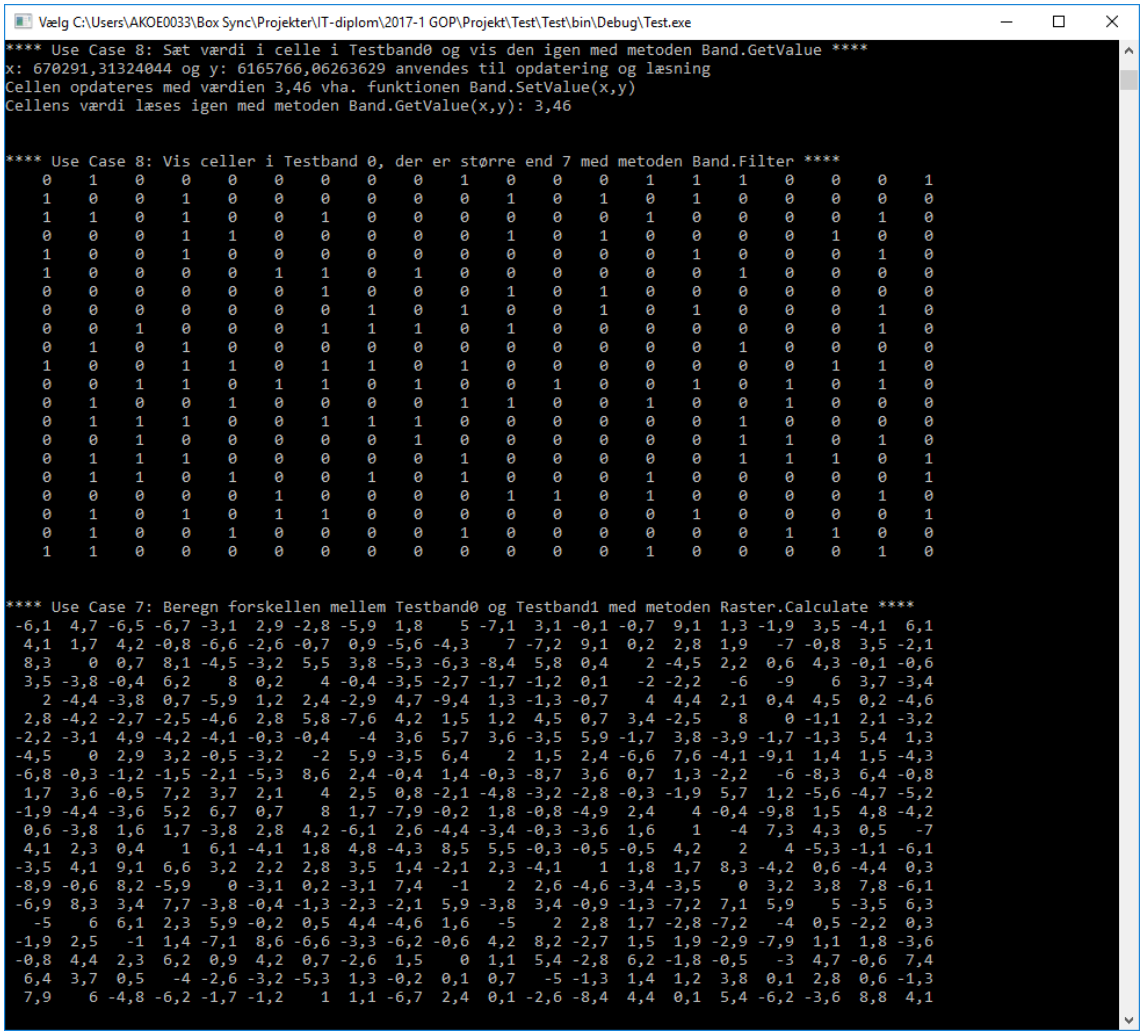
Nedenfor vises screen dumps fra kørsel af funktionstest (3 billeder) samt efterfølgende koden til testprogrammet:



Figur 6: Screen dump af funktionstest



Figur 7: Screen dump af funktionstest fortsat ...



Figur 8: Screen dump af funktionstest fortsat ...

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using RasterLib;
7  using System.IO;
8  using System.Runtime.Serialization.Formatters.Binary;
9
10 namespace Test
11 {
12     class Program
13     {
14         static void Main(string[] args)
15         {
16             Console.WriteLine("Kører test af RasterLib");
17
18             // Use Case 1: Opretter et rasterdatasæt
19             Console.WriteLine("**** USE CASE 1: ****");
20             Console.WriteLine("Opret ny raster med værdierne: 670004.32, 6161030.5, 694990.2,  6181990.4, 1000, -9999");
21             Raster r = new Raster(670004.32, 6161030.5, 689990.2, 6181990.4, 1000, -9999);
22             Console.WriteLine();
23
24             // Use Case 3 og 9: Opret bånd og opdater alle værdier i båndet
25             Console.WriteLine("\n**** USE CASE 3 & 9:  ****");
26             Console.WriteLine("opretter 3 bånd og opdaterer celler i båndene med random værdier");
27             Random rnd = new Random();
28             double x, y, z;
29             string bandname;
30             for (int i = 0; i< 3;i++)
31             {
32                 bandname = "Testband" + i.ToString();
33                 Console.WriteLine("\nAdding band \"" + bandname + "\"");
34                 Console.WriteLine((r.NewBand(bandname, "meter")).ToString());
35                 // Use Case 9: ændre alle værdierne i båndet
```

```
36                  Console.WriteLine("\nSetting values in \"" + bandname + "\" with Band.SetValue");
37                  foreach (Cell c in r.Bands[bandname].Grid)
38                  {
39                      c.Value = Math.Round(rnd.NextDouble()*10,2);
40                  }
41                  Console.WriteLine("Done!");
42
43              }
44
45              // Use case 5: Vis liste med rasterbånd
46              Console.WriteLine($"\n**** Use case 5: ****");
47              Console.WriteLine("Vis definition og liste med rasterbånd med metoden Raster.Describe");
48              Console.WriteLine(r.Describe());
49              Console.WriteLine();
50
51              // Use Case 11: eksporter alle bånd som ascii-grids til folderen C\Temp\
52              Console.WriteLine($"\n**** Use Case 11: Eksporter alle bånd som ascii-grids til biblioteket c:\\Temp\\ ved kald
                  af metoden Export for rasteren  ****");
53              Console.WriteLine(r.Export("ASCII",@"C:\Temp\"));
54
55              // Use Case 2: Gem raster
56              Console.WriteLine($"\n**** Use Case 2: Gem raster på disk ved kald af metoden Save til biblioteket c:\\Temp\\
                  ****");
57              Console.WriteLine(r.Save("TestRaster", @"C:\Temp\").ToString());
58
59              // Use Case 2: Åben gemt raster
60              Console.WriteLine($"\n**** Use Case 2: Åbn gemt raster til et nyt rasterobjekt ****");
61              Raster r2 = new Raster(0, 0, 1, 1, 1, 0);
62              Stream rasterfil = File.OpenRead(@"C:\Temp\TestRaster.ras");
63              //åbn fil for læsning
64              BinaryFormatter bf = new BinaryFormatter();
65              object rasterobjekt = bf.Deserialize(rasterfil);
66              r2 = (Raster)rasterobjekt; //objekt castes til typen Ansat
67              rasterfil.Close();
68              Console.WriteLine("Test at rasteren er åbnet rigtigt med metoden Describe på ny raster:");
```

```
69              Console.WriteLine(r2.Describe());
70              Console.WriteLine();
71
72              // Use Case 4
73              Console.WriteLine($"\n**** Use case 4: ****");
74              Console.WriteLine("Importer rasterbånd fra ASCII-fil med metoden Raster.ImportASCII");
75              Console.WriteLine("Vis derefter rasterbåndet så importen kan verificeres");
76              r.ImportASCII("ImportBånd", "Meter", @"C:\Temp\Import.asc");
77
78              // Use Case 8: Vis alle værdier i båndet
79              Console.WriteLine("\n**** Use Case 8: ****");
80              Console.WriteLine("Vis værdier i det nye bånd ved kald af metoden ToString() på båndet");
81              Console.WriteLine(r.Bands["ImportBånd"].ToString());
82              Console.WriteLine();
83
84              // Use Case 9: Ændre værdi i enkelt celle ud fra x, y koordinater og vis værdien vha samme koordinater
85              bandname = "Testband0";
86              Console.WriteLine($"\n**** Use Case 8: Sæt værdi i celle i {bandname} og vis den igen med metoden Band.GetValue
                  ****");
87              x = r.Def.Xmin + ((r.Def.Xmax - r.Def.Xmin) * rnd.NextDouble());
88              y = r.Def.Ymin + ((r.Def.Ymax - r.Def.Ymin) * rnd.NextDouble());
89              Console.WriteLine($"x: {x} og y: {y} anvendes til opdatering og læsning");
90              z = Math.Round(rnd.NextDouble() * 10, 2);
91              Console.WriteLine($"Cellen opdateres med værdien {z} vha. funktionen Band.SetValue(x,y)");
92              r2.Bands[bandname].SetValue(x, y, z);
93              Console.WriteLine($"Cellens værdi læses igen med metoden Band.GetValue(x,y): {r2.Bands[bandname].GetValue(x,
                  y)}");
94              Console.WriteLine();
95
96              // Use Case 8: Vis udvalgte værdier i båndet Testband0
97              Console.WriteLine($"\n**** Use Case 8: Vis celler i Testband 0, der er større end 7 med metoden Band.Filter
                  ****");
98              Band resultat = r2.Bands[bandname].Filter(">", 7);
99              Console.WriteLine(resultat.ToString());
100
```

```
101             // Use Case 7: Beregn forskellen mellem Testband0 og Testband1 med metoden Raster.Calculate
102             Console.WriteLine($"\n**** Use Case 7: Beregn forskellen mellem Testband0 og Testband1 med metoden
                  Raster.Calculate ****");
103             resultat = r2.Calculate("-", r2.Bands["Testband0"], r2.Bands["Testband1"]);
104             Console.WriteLine(resultat.ToString());
105
106             Console.WriteLine("\nTryk på en tast for at fortsætte!");
107             Console.ReadKey();
108         }
109     }
110 }
111
```

E. Udskrift af koden: RasterLib

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7  using System.Runtime.Serialization.Formatters.Binary;
8
9  namespace RasterLib
10 {
11
12     delegate double? del1(double? v1, double? v2);
13     [Serializable]
14     public class Raster
15     {
16         // Attributter:
17         private Griddef def;
18         private Dictionary<string, Band> bands;
19
20         // Properties:
21         public Griddef Def { get; private set; }
22         public Dictionary<string, Band> Bands { get; private set; }
23
24         // Methods:
25         // Constructor
26         public Raster(double xmin, double ymin, double xmax, double ymax, int cellsize, int nodata_value)
27         {
28             Console.WriteLine("Raster instant. params.: " + xmin.ToString() + " " + ymin.ToString() + " " + xmax.ToString() + ↵
                   " " + ymax.ToString() + " " + cellsize.ToString() + " " + nodata_value.ToString());
29             xmin = Math.Floor(xmin / cellsize) * cellsize;
30             ymin = Math.Floor(ymin / cellsize) * cellsize;
31             xmax = Math.Ceiling(xmax / cellsize) * cellsize;
32             ymax = Math.Ceiling(ymax / cellsize) * cellsize;
33             //Console.WriteLine("Griddef params: " + xmin.ToString() + " " + ymin.ToString() + " " + xmax.ToString() + " " + ↵
                 ymax.ToString() + " " + cellsize.ToString() + " " + nodata_value.ToString());
```

```
34              Def = new Griddef(xmin, ymin, xmax, ymax, cellsize, nodata_value);
35              Bands = new Dictionary<string, Band>();
36          }
37
38          public bool NewBand(string bandname, string unit)
39          {
40              try {Bands.Add(bandname, new Band(this, unit));}
41              catch (ArgumentException e)
42              {
43                  throw new ArgumentException("Båndet eksisterer i forvejen.", e);
44              }
45
46              return true;
47          }
48
49          public bool DeleteBand(string bandname)
50          {
51              if (bands.ContainsKey(bandname))
52              {
53                  bands.Remove(bandname);
54              }
55              else
56              {
57                  throw new ArgumentException("Båndet eksisterer ikke!", "DeleteBand");
58              }
59
60              return true;
61          }
62
63          // Describe
64          public string Describe()
65          {
66              string description=Def.Describe() + "Bands in raster:";
67              string unit, min, max, mean;
68              description = description + "\nBandname\tUnit\tMin\tMax\tMean";
```

```
69              foreach (string bandname in Bands.Keys)
70              {
71                  unit = Bands[bandname].Unit;
72                  if (Bands[bandname].Min()== null)
73                  {
74                      min = "NULL";
75                  } else
76                  {
77                      min = Math.Round(Bands[bandname].Min().Value,2).ToString();
78                  }
79
80                  if (Bands[bandname].Max() == null)
81                  {
82                      max = "NULL";
83                  }
84                  else
85                  {
86                      max = Math.Round(Bands[bandname].Max().Value, 2).ToString();
87                  }
88
89                  if (Bands[bandname].Mean() == null)
90                  {
91                      mean = "NULL";
92                  }
93                  else
94                  {
95                      mean = Math.Round(Bands[bandname].Mean().Value, 2).ToString();
96                  }
97
98                  description = description + "\n" + bandname + ":\t" + unit + "\t" + min + "\t" + max + "\t" + mean;
99              }
100             return description;
101         }
102
103         public bool Save(string filename, string folderpath)
```

```
104         {
105             string filepath;
106             if (Directory.Exists(folderpath))
107             {
108                 filepath = folderpath + filename + ".ras";
109                 Stream fileStreame = File.Create(filepath);
110                 BinaryFormatter bf = new BinaryFormatter();
111                 bf.Serialize(fileStreame, this);
112                 fileStreame.Close();
113                 return true;
114             } else
115             {
116                 throw new ArgumentException("Mappen findes ikke", "Save");
117             }
118 
119         }
120 
121         // Beregner et nyt bånd ud fra 2 andre
122         public Band Calculate(string oprator, Band band1, Band band2)
123         {
124             if (band1.Unit != band2.Unit)
125             {
126                 throw new ArgumentException("Båndene har ikke samme enheder", "Raster.Calculate");
127             }
128 
129             del1 calc;
130             double?[] values1 = band1.Values();
131             double?[] values2 = band2.Values();
132             Band resultat = new Band(this, band1.Unit);
133 
134             // determine whitch Lammda Expression to use
135             switch (oprator)
136             {
137                 case "+":
138                     calc = (val1, val2) => val1 + val2;
```

```
139                     break;
140                 case "-":
141                     calc = (val1, val2) => val1 - val2;
142                     break;
143                 case "*":
144                     calc = (val1, val2) => val1 * val2;
145                     break;
146                 case "/":
147                     calc = (val1, val2) => val1 / val2;
148                     break;
149                 default:
150                     throw new ArgumentException("operatoren er ikke understøttet", "Raster.Calculate");
151             }
152 
153             // looper gennem griddene og skriver den beregnede værdi til det nye bånd
154             for (int i=0; i<resultat.Length; i++)
155             {
156                 //Console.WriteLine($"Calculating {values1[i]} {oprator} {values2[i]} - resultat: {calc(values1[i], values2
                      [i])}");
157                 resultat.Grid[i].Value = calc(values1[i],values2[i]);
158             }
159             return resultat;
160         }
161 
162 
163         // Importerer et ASCII grid til en nyt bånd
164         public bool ImportASCII(string bandname, string unit, string filepath)
165         {
166             // variable til processering af fil
167             string line;
168             string[] words;
169             int i=0;
170             StreamReader sr;
171 
172             // variable til check af griddefinition
```

```
173             string ncols = null, nrows = null, xllcorner=null, yllcorner=null, xllcenter=null, yllcenter=null, cellsize =
                  null, nodata_value ="-9999";
174
175             // variable til opdatering af af grid
176             int idx=0;
177
178
179             // findes båndets navn i forvejen?
180             if (Bands.ContainsKey(bandname))
181             {
182                 throw new ArgumentException("Båndet eksisterer i forvejen","ImportASCII");
183             }
184
185             Bands.Add(bandname, new Band(this, unit));
186
187
188             // prøv at åbne filen
189             try
190             {
191                 FileStream asciifile = new FileStream(filepath, FileMode.Open);
192                 sr = new StreamReader(asciifile);
193             }
194             catch (IOException e)
195             {
196                 Console.WriteLine(e.Message);
197                 return false;
198             }
199
200             // hvis filen kan åbnes gennemgås linier i filen
201             line = sr.ReadLine();
202             char[] separators = new char[] { ' ' };
203             while (line != null)
204             {
205                 words = line.Split(separators, StringSplitOptions.RemoveEmptyEntries);
206                 if (words.Length == 0)
```

```
207                 {
208                     line = sr.ReadLine();
209                     continue;
210                 }
211                 switch (words[0].ToLower())
212                 {
213                     case "nrows":
214                         nrows = words[1];
215                         Console.WriteLine($"nrows set: {nrows}");
216                         break;
217                     case "ncols":
218                         ncols = words[1];
219                         Console.WriteLine($"ncols set: {ncols}");
220                         break;
221                     case "xllcorner":
222                         xllcorner = words[1];
223                         Console.WriteLine($"xllcorner set: {xllcorner}");
224                         break;
225                     case "xllcenter":
226                         xllcenter = words[1];
227                         Console.WriteLine($"xllcenter set: {xllcenter}");
228                         break;
229                     case "yllcorner":
230                         yllcorner = words[1];
231                         Console.WriteLine($"yllcorner set: {yllcorner}");
232                         break;
233                     case "yllcenter":
234                         yllcenter = words[1];
235                         Console.WriteLine($"yllcenter set: {yllcenter}");
236                         break;
237                     case "cellsize":
238                         cellsize = words[1];
239                         Console.WriteLine($"cellsize set: {cellsize}");
240                         break;
241                     case "nodata_value":
```

```
242                         nodata_value = words[1];
243                         Console.WriteLine($"nodata_value set {nodata_value}");
244                         break;
245                     default:
246                         if (i <= 6) //første linje med data - her korrigeres for centerkoordinater og griddefinitionen
                              checkes
247                         {
248                             // korriger hjørnekoordinater hvis header def blev angiver med center koordinater
249                             if (xllcenter != null)
250                             {
251                                 xllcorner = (double.Parse(xllcenter) - (double.Parse(cellsize) / 2)).ToString();
252                                 xllcenter = null;
253                             }
254                             if (yllcenter != null)
255                             {
256                                 yllcorner = (double.Parse(yllcenter) - (double.Parse(cellsize) / 2)).ToString();
257                                 yllcenter = null;
258                             }
259 
260                             // Check om griddefinitionen i filen stemmer overens med rasteren
261                             if (xllcorner != Def.Xmin.ToString() || yllcorner != Def.Ymin.ToString() || nrows !=
                                  Def.Rows.ToString() || ncols != Def.Cols.ToString() || cellsize != Def.Cellsize.ToString())
262                             {
263                                 throw new ArgumentException("Filens griddefenitionen er forskellig fra rasterens. Kan ikke
                                    importerer filen!","Raster.Import");
264                             }
265                         }
266 
267                         // processing gridvalues
268                         foreach (string val in words)
269                         {
270                             if (val != nodata_value)
271                             {
272                                 // Opdater grid hvis val ikke er nodata_value
273                                 //Console.WriteLine($"Updating index {idx} to {val}");
```

```
274                                 Bands[bandname].Grid[idx].Value = double.Parse(val.Replace(".",","));
275                             }
276                             idx++;
277                         }
278                         break;
279                 }
280
281                 line = sr.ReadLine();
282                 i++;
283
284             }
285
286             return true;
287
288
289         }
290
291         // Eksporterer alle bånd til ascii-filer
292         public string Export(string fileformat, string folderpath)
293         {
294             //todo: Check tru/false foraech band.export --> return message
295             string message;
296             switch (fileformat.ToLower())
297             {
298                 case "ascii":
299                     foreach (string bandname in Bands.Keys)
300                     {
301                         Bands[bandname].ExportASCII(bandname, folderpath);
302                     }
303                     message = "Bands exported as *.asc to " + folderpath;
304                     break;
305                 case "xyz":
306                     foreach (string bandname in Bands.Keys)
307                     {
308                         Bands[bandname].ExportXYZ(bandname, folderpath);
```

```
309                     }
310                     message = "Bands exported as *.xyz to " + folderpath;
311                     break;
312                 default:
313                     message = "Unknown fileformat. Cannot export bands.";
314                     break;
315             }
316 
317             return message;
318 
319         }
320 
321 
322 
323     }
324 }
325 
```

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RasterLib
{
    [Serializable]
    public class Griddef
    {
        private double xmin, ymin, xmax, ymax, nodata_value;
        private int cellsize, rows, cols;

        // Constructer
        public Griddef(double xmin, double ymin, double xmax, double ymax, int cellsize, double nodata_value)
        {
            Xmin = xmin;
            Ymin = ymin;
            Xmax = xmax;
            Ymax = ymax;
            Cellsize = cellsize;
            Nodata_value = nodata_value;
            Rows = (int)(ymax - ymin) / cellsize;
            Cols = (int)(xmax - xmin) / cellsize;
        }

        // Properties
        public double Xmin { get; private set; }
        public double Ymin { get; private set; }
        public double Xmax { get; private set; }
        public double Ymax { get; private set; }
        public int Rows { get; private set; }
        public int Cols { get; private set; }
        public double Nodata_value { get; private set; }
```

```
36          public int Cellsize { get; private set; }
37
38          // Methods
39          public string GetASCIIHeader()
40          {
41              string header;
42              header = "NCOLS " + Cols.ToString();
43              header = header + "\r\nNROWS " + Rows.ToString();
44              header = header + "\r\nXLLCORNER " + Xmin.ToString();
45              header = header + "\r\nYLLCORNER " + Ymin.ToString();
46              header = header + "\r\nCELLSIZE " + Cellsize.ToString();
47              header = header + "\r\nNODATA_VALUE " + Nodata_value.ToString();
48              return header;
49          }
50
51          public string Describe()
52          {
53              string description;
54              description = "Rows x Cols: " + Rows + " x " + Cols.ToString();
55              description = description + "\nArea: " + ((Xmax-Xmin)*(Ymax-Ymin)).ToString() + " kvm";
56              description = description + "\nLower left corner: " + Xmin.ToString() + ", " + Ymin.ToString();
57              description = description + "\nCellsize: " + Cellsize.ToString() + " m";
58              description = description + "\nnodata_value: " + Nodata_value.ToString() + "\n";
59              return description;
60          }
61
62          public int Coord2Idx(double x, double y)
63          {
64              int ncol, nrow, idx;
65              if (x>Xmin && x<Xmax && y>Ymin && y<Ymax)
66              {
67                  ncol = (int)(x - Xmin) / Cellsize;
68                  //Console.WriteLine("Coord2Idx calculating ncol: ((x)" + x.ToString() + " - (Xmin)" + Xmin.ToString() + ") /   ↵
                      (Cellsize)" + Cellsize.ToString() + " = " + ncol.ToString() );
69                  nrow = (int)(Ymax - y) / Cellsize;
```

```
70                  //Console.WriteLine("Coord2Idx calculating nrow: (Ymax)" + Ymax.ToString() + " - (y)" + y.ToString() + ") /
                      (Cellsize)" + Cellsize.ToString() + " = " + nrow.ToString());
71                  idx = nrow * Cols + ncol;
72                  //Console.WriteLine("Coord2Ids calculating idx: (nrow)" + nrow.ToString() + " * (Cols)" + Cols.ToString() + "
                      + (ncol)" + ncol.ToString() + " = " + idx.ToString());
73                  return idx;
74              } else
75              {
76                  throw new ArgumentOutOfRangeException("Koordinater ligger uden for rasterens griddefenition", "Coor2Idx");
77              }
78
79          }
80
81
82
83
84      }
85  }
86
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6  using System.IO;
 7
 8  namespace RasterLib
 9  {
10      delegate bool del2(double? v, double f);
11      [Serializable]
12      public class Band
13      {
14          // Attributes
15          private Cell[] grid;
16          private string unit;
17          private Raster rast;
18          private Griddef def;
19          private int length;
20
21          // Properties
22          public Cell[] Grid { get; private set; }
23          public string Unit { get; private set; }
24          public Raster Rast { get; private set; }
25          public Griddef Def { get; private set; }
26          public int Length { get; private set; }
27
28          // Constructor
29          public Band (Raster raster, string unit)
30          {
31              Rast = raster;
32              Def = raster.Def;
33              Length = Def.Rows * Def.Cols;
34              Grid = new Cell[Length];
35              Unit = unit;
```

```
36
37              // finder centerpunktet på øverste højre celle:
38              double centerX = Def.Xmin + (Def.Cellsize / 2);
39              double centerY = Def.Ymax - (Def.Cellsize / 2);
40              int i=0;
41              for (int irow = 0; irow < Def.Rows; irow++) // itererer over rækker og beregner cellernes y (center) værdi
42              {
43                  centerX = Def.Xmin + (Def.Cellsize / 2);
44                  centerY = centerY - Def.Cellsize;
45                  for (int icol = 0; icol < Def.Cols; icol++)
46                  {
47                      i = irow * Def.Cols + icol;
48                      centerX = centerX + Def.Cellsize;
49                      Grid[i] = new Cell(centerX, centerY, null);
50                  }
51              }
52          }
53
54          public double?[] Values()
55          {
56              double? [] values = new double?[Length];
57              int i = 0;
58              foreach (Cell c in Grid)
59              {
60                  values[i] = Grid[i].Value;
61                  i++;
62              }
63              return values;
64          }
65
66          public double? GetValue(double x, double y)
67          {
68              int idx;
69              //Console.WriteLine("GetValue parameters x & y: " + x.ToString() + " " + y.ToString());
70              try
```

```
71              {
72                  idx = Def.Coord2Idx(x, y);
73                  //Console.WriteLine("Getting value at idx: " + idx.ToString());
74              }
75              catch (ArgumentOutOfRangeException e)
76              {
77                  return null;
78              }
79              return Grid[idx].Value;
80
81          }
82
83          public bool SetValue(double x, double y, double? value)
84          {
85              int idx;
86              //Console.WriteLine("SetValue parameters x & y: " + x.ToString() + " " + y.ToString());
87              try
88              {
89                  idx = Def.Coord2Idx(x, y);
90                  //Console.WriteLine("Setting value at idx: " + idx.ToString());
91              }
92              catch (ArgumentOutOfRangeException e)
93              {
94                  return false;
95              }
96              Grid[idx].Value = value;
97              return true;
98
99
100         }
101
102         public override string ToString()
103         {
104             string strRows = "";
105             int i = 0;
```

```
106             double?[] values = Values();
107             for (int r = 0; r < Def.Rows; r++)
108             {
109                 for (int c = 0; c < Def.Cols; c++)
110                 {
111                     i = r * Def.Cols + c;
112                     strRows = strRows + Math.Round(values[i].Value,1).ToString().PadLeft(5);
113                 }
114                 strRows = strRows + "\n";
115             }
116 
117             return strRows;
118         }
119 
120         public Band Filter(string oprator, double filtervalue)
121         {
122             del2 filter;
123             Band resultat = new Band(Rast,"Ja/Nej");
124             double?[] values = Values();
125             double? value = null;
126             // determine whitch Lammda Expression to use
127             switch (oprator)
128             {
129                 case "==":
130                     filter = (val, fil) => val == fil;
131                     break;
132                 case "!=":
133                     filter = (val, fil) => val != fil;
134                     break;
135                 case "<":
136                     filter = (val, fil) => val < fil;
137                     break;
138                 case ">":
139                     filter = (val, fil) => val > fil;
140                     break;
```

```
141                 case ">=":
142                     filter = (val, fil) => val >= fil;
143                     break;
144                 case "<=":
145                     filter = (val, fil) => val <= fil;
146                     break;
147                 default:
148                     throw new ArgumentException("Operatoren er ikke understøttet", "Band.Filter");
149             }
150 
151             // Looper gennem båndet og anvener filter til at skrive 0 eller 1 til nyt bånd
152             for (int i=0;i<Length;i++)
153             {
154                 value = values[i];
155                 if (filter(value, filtervalue))
156                 {
157                     resultat.Grid[i].Value = 1.0;
158                 }
159                 else
160                 {
161                     resultat.Grid[i].Value = 0.0;
162                 }
163                 //Console.WriteLine($"Value: {value} inserted in Grid[{i}]: {resultat.Grid[i].Value}");
164 
165             }
166 
167             return resultat;
168         }
169 
170         public double? Min()
171         {
172             return Values().Min();
173         }
174 
175         public double? Max()
```

```
176         {
177             return Values().Max();
178         }
179
180         public double? Mean()
181         {
182             int i = 0;
183             double? sum = 0;
184             double? mean = null;
185             foreach(double? v in Values())
186             {
187                 if (v != null)
188                 {
189                     i++;
190                     sum = sum + v;
191                 }
192             }
193             if (i > 0)
194             {
195                 mean = sum / i;
196             }
197
198             return mean;
199         }
200
201
202         public bool ExportASCII(string filename, string folderpath)
203         {
204             StreamWriter sw;
205             string filepath = folderpath + filename + ".asc";
206             try
207             {
208                 FileStream asciifile = new FileStream(filepath, FileMode.OpenOrCreate);
209                 sw = new StreamWriter(asciifile);
210             }
```

```
211             catch (IOException e)
212             {
213                 Console.WriteLine("An IO exception has been thrown!");
214                 Console.WriteLine(e.ToString());
215                 Console.WriteLine();
216                 return false;
217             }
218             sw.WriteLine(Def.GetASCIIHeader());
219             double?[] values = new double?[Length];
220             values = Values();
221             string val;
222             int i;
223             for (int r = 0; r < Def.Rows; r++)
224             {
225                 string strrows = "";
226                 for (int c = 0; c < Def.Cols; c++)
227                 {
228                     i = r * Def.Cols + c;
229                     if (values[i]==null)
230                     {
231                         val = "-9999";
232                     } else
233                     {
234                         val = values[i].ToString().Replace(',','.');
235                     }
236                     strrows = strrows + " " + val;
237                 }
238                 strrows = strrows.Substring(1, strrows.Length - 1);
239                 sw.WriteLine("\n" + strrows);
240             }
241             sw.Close();
242             return true;
243 
244         }
245 
```

```
246         public bool ExportXYZ(string filename, string folderpath)
247         {
248             string val;
249             StreamWriter sw;
250             string filepath = folderpath + filename + ".xyz";
251             try
252             {
253                 FileStream asciifile = new FileStream(filepath + ".xyz", FileMode.OpenOrCreate);
254                 sw = new StreamWriter(asciifile);
255             } catch (IOException e)
256             {
257                 Console.WriteLine("An IO exception has been thrown!");
258                 Console.WriteLine(e.ToString());
259                 Console.WriteLine();
260                 return false;
261             }
262             sw.WriteLine("X;Y;Z");
263             foreach (Cell c in Grid)
264             {
265                 if (c.Value == null)
266                 {
267                     val = "";
268                 }
269                 else
270                 {
271                     val = c.Value.ToString();
272                 }
273                 sw.WriteLine("\n" + c.CenterX.ToString() + ";" + c.CenterY.ToString() + ";" + val);
274             }
275             sw.Close();
276             return true;
277 
278         }
279 
280 
```

```
281
282         }
283     }
284
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace RasterLib
8  {
9      [Serializable]
10     public class Cell
11     {
12         //Atributes:
13         private double centerX;
14         private double centerY;
15         private double? value;
16
17         // Properties:
18         public double CenterX { get; private set; }
19         public double CenterY { get; private set; }
20         public double? Value { get; set; }
21
22         // Methods:
23         // Constructor
24         public Cell(double centerX, double centerY, double? value)
25         {
26             CenterX = centerX;
27             CenterY = centerY;
28             Value = value;
29         }
30     }
31 }
32
```

F. Udskrift af koden: PSRaster

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6  using System.Management.Automation;
 7  using RasterLib;
 8
 9  namespace psRaster
10  {
11      [Cmdlet("New","Raster")]
12      public class NewRaster: Cmdlet
13      {
14          private Raster raster;
15          private double xmin;
16          private double ymin;
17          private double xmax;
18          private double ymax;
19          private int cellsize;
20          private int nodata_value;
21          private bool checkParams = true;
22
23          // Properties
24          [Parameter(Position = 0, Mandatory = true, ValueFromPipelineByPropertyName = true)]
25          public string Xmin
26          {
27              get { return xmin.ToString(); }
28              set
29              {
30                  if (double.TryParse(value, out xmin))
31                  {
32                      Console.WriteLine("xmin ok");
33                  }
34                  else
35                  {
```

```
36                      Console.WriteLine("xmin er ikke en double");
37                      checkParams = false;
38                  }
39              }
40          }
41
42          [Parameter(Position = 1, Mandatory = true, ValueFromPipelineByPropertyName = true)]
43          public string Ymin
44          {
45              get { return ymin.ToString(); }
46              set
47              {
48                  if (double.TryParse(value, out ymin))
49                      Console.WriteLine("Ymin ok");
50                  else
51                  {
52                      Console.WriteLine("Ymin er ikke et tal");
53                      checkParams = false;
54                  }
55              }
56          }
57
58          [Parameter(Position = 2, Mandatory = true, ValueFromPipelineByPropertyName = true)]
59          public string Xmax
60          {
61              get { return xmax.ToString(); }
62              set
63              {
64                  if (double.TryParse(value, out xmax))
65                      Console.WriteLine("xmax ok");
66                  else
67                  {
68                      Console.WriteLine("xmax er ikke en tal");
69                      checkParams = false;
70                  }
```

```
71              }
72          }
73
74          [Parameter(Position = 3, Mandatory = true, ValueFromPipelineByPropertyName = true)]
75          public string Ymax
76          {
77              get { return ymax.ToString(); }
78              set
79              {
80                  if (double.TryParse(value, out ymax))
81                      Console.WriteLine("Ymin ok");
82                  else
83                  {
84                      Console.WriteLine("Ymax er ikke et tal");
85                      checkParams = false;
86                  }
87              }
88          }
89
90          [Parameter(Position = 4, Mandatory = true, ValueFromPipelineByPropertyName = true)]
91          public string Cellsize
92          {
93              get { return Cellsize.ToString(); }
94              set
95              {
96                  if (int.TryParse(value, out cellsize))
97                      Console.WriteLine("Cellsize ok");
98                  else
99                  {
100                     Console.WriteLine("Cellsize er ikke et tal");
101                     checkParams = false;
102                 }
103             }
104         }
105
```

```
106         [Parameter(Position = 5, Mandatory = true, ValueFromPipelineByPropertyName = true)]
107         public string Nodata_value
108         {
109             get { return nodata_value.ToString(); }
110             set
111             {
112                 if (int.TryParse(value, out nodata_value))
113                     Console.WriteLine("Nodata_value ok");
114                 else
115                 {
116                     Console.WriteLine("Nodata_value er ikke et tal");
117                     checkParams = false;
118                 }
119             }
120         }
121 
122         // Metoder
123         protected override void BeginProcessing()
124         {
125             WriteVerbose("BeginProcessing started");
126             base.BeginProcessing();
127             WriteVerbose("BeginProcessing completed");
128         }
129 
130         protected override void ProcessRecord()
131         {
132             WriteVerbose("ProsessingRecord started");
133             //base.ProcessRecord();
134             //WriteObject("Prosessing " + Text);
135             if (checkParams)
136             {
137                 //WriteObject("Processing started: instantiating raster object");
138                 try
139                 {
140                     raster = new Raster(xmin, ymin, xmax, ymax, cellsize, nodata_value);
```

```
141                 }
142                 catch (Exception ex)
143                 {
144                     WriteError(new ErrorRecord(
145                        ex,
146                        "UnableToInstantiateRaster",
147                        ErrorCategory.InvalidOperation, raster));
148                     return;
149                 }
150 
151             }
152             WriteVerbose("ProsessingRecord completed");
153         }
154 
155         protected override void EndProcessing()
156         {
157             WriteVerbose("EndProsessing started");
158             base.EndProcessing();
159             WriteObject(raster);
160             WriteVerbose("EndProsessing complete");
161         }
162 
163     }
164 }
165 
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7  using System.Runtime.Serialization.Formatters.Binary;
8  using System.Management.Automation;
9  using RasterLib;
10
11 namespace psRaster
12 {
13     [Cmdlet("Open", "Raster")]
14     public class OpenRaster: Cmdlet
15     {
16         // Attributter
17         Raster raster;
18         string filepath;
19         Stream rasterfile;
20
21         // Properties
22         [Parameter(Position = 0, Mandatory = true, ValueFromPipelineByPropertyName = true)]
23         public string Filepath
24         {
25             get { return filepath; }
26             set { filepath = value;  }
27         }
28
29         // Metoder
30         protected override void BeginProcessing()
31         {
32             WriteVerbose("Beginning prosessing");
33
34
35         }
```

```
36
37          protected override void ProcessRecord()
38          {
39
40              WriteVerbose("Prosessing " + filepath);
41              raster = new Raster(0, 0, 1, 1, 1, 0);
42              try
43              {
44                  rasterfile = File.OpenRead(filepath);
45                  BinaryFormatter bf = new BinaryFormatter();
46                  object rasterobjekt = bf.Deserialize(rasterfile);
47                  raster = (Raster)rasterobjekt;
48              }
49              catch (Exception ex)
50              {
51                  WriteError(new ErrorRecord(
52                         ex,
53                         "UnableToInstantiateRaster",
54                         ErrorCategory.InvalidOperation, raster));
55                  return;
56              }
57
58          }
59
60          protected override void EndProcessing()
61          {
62              base.EndProcessing();
63              WriteVerbose("Prosessing complete");
64              WriteObject(raster);
65          }
66
67
68
69      }
70
```

```
71  }
72
```