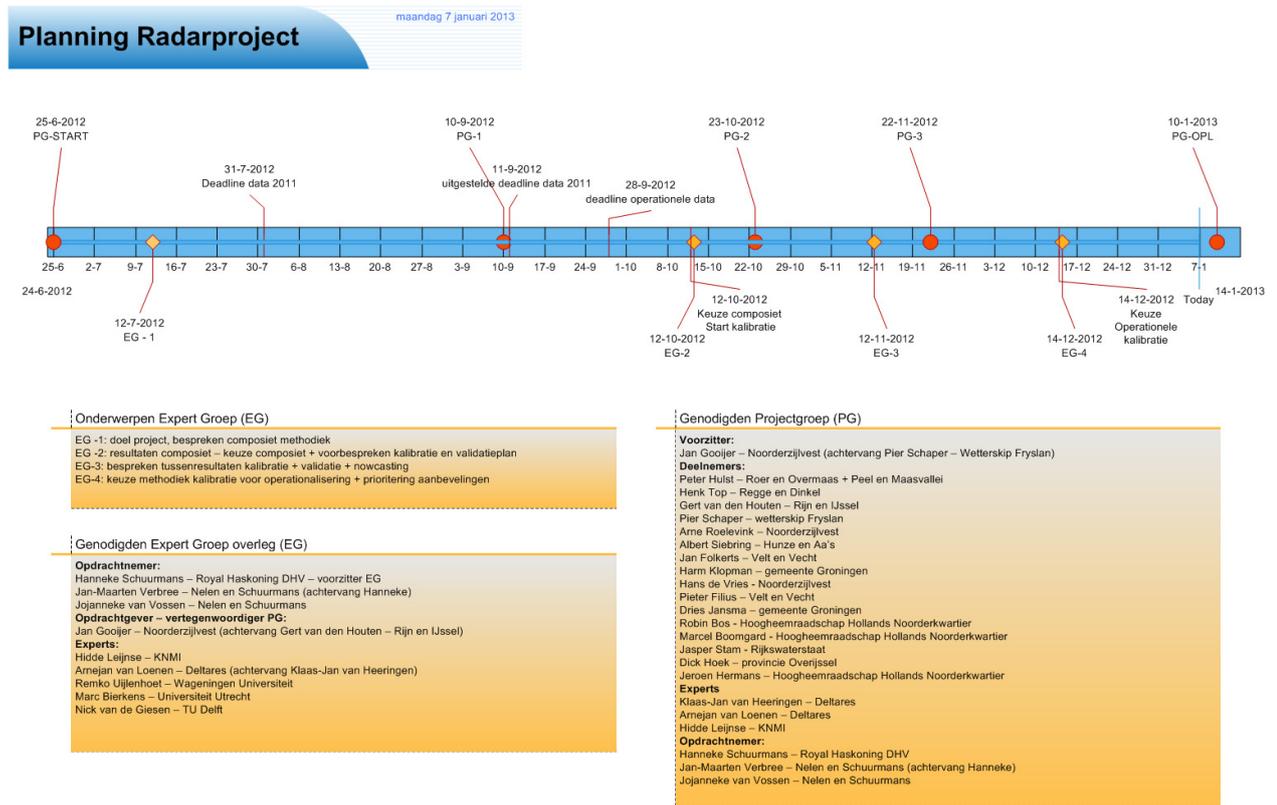


BIJLAGE A Projectorganisatie/verslagen/presentaties

BIJLAGE A.1 Projectorganisatie



BIJLAGE A.2 Overlegverslagen projectgroep

Afspraken startoverleg 25 juni 2012

Hieronder volgt een kort verslag met de actiepunten nav het startoverleg tussen projectgroep en opdrachtnemer. Hetgene voorgesteld is in het gedetailleerd plan van aanpak is goedgekeurd door de projectgroep, met daarbij de volgende aanvullingen/aanpassingen:

Grondstations:

we hebben afgesproken **deze week** de lijst met grondstations die we als eerste 40 aanleveren compleet te maken. **Actie allen.** In de tabel waarin alle metadata van deze stations wordt opgenomen, worden ook de telefoonnummers van de contactpersonen opgenomen zodat de opdrachtnemer direct contact kan opnemen bij vragen of complicaties. Deelnemers die grondstations inbrengen geven zelf een kwaliteitslabel aan hun grondstation. Deze grondstations hoeven nog niet te voldoen aan de eisen van AMO Meteo. We gaan bij de kalibratie obv eigen grondstations streven naar lokale invloed van grondstations op de kalibratie, zodat de partijen vooral ook zelf profiteren wanneer ze goede grondstations inbrengen.

Voor nieuw te plaatsen grondstations gaan wel de eisen van AMO Meteo gelden. Voor de neerslagmeter zelf geldt dat dit een pluviometer of gelijkwaardig moet zijn. Zo krijgen we op lange termijn uniformiteit en een hoge kwaliteit bij de door partijen zelf geplaatste grondstations.

We gaan onderzoeken wie nieuwe grondstations wil plaatsen om zo te bekijken of we de grondstations in bulk kunnen inkopen en daarbij het beheer & onderhoud samen kunnen uitbesteden. Dit kan kosten besparen. **Actie: Arne Roelevink.**

Van de aan te leveren grondstations wordt de data aangeleverd op een ftp site. Het is ook de bedoeling dat historische gegevens van 2011 op deze ftp site komen. Die gegevens moeten in juli geleverd worden. <ftp.lizardssystem.nl> **Actie allen.**

De details voor de aan te leveren gegevens per grondstation worden nog vastgesteld. Een voorbeeldbestand wordt **deze week** rondgestuurd **Actie Jan-Maarten** Van belang is in elk geval dat de data equidistant wordt aangeleverd en dat de tijdzone wordt aangegeven.

Radar:

- de projectgroep heeft een procesmatige invulling en daarin worden alle belangrijke besluiten genomen. Daarnaast is er een expertgroep waarin deskundigen bespreken wat de beste methodes zijn voor het genereren van de afgesproken producten. We besluiten uit de projectgroep 2 mensen af te vaardigen in de expertgroep. Dit zijn Gert van den Houten en Jan Gooijer. Pier Schaper is de vaste vervanger wanneer Gert en Jan niet kunnen. De verslagen van de expertvergaderingen worden gemaakt door de leden van de projectgroep en verspreid in de projectgroep. Bij discussies in de expertgroep is het doel om consensus te krijgen. Lukt dit niet, dan hebben KNMI en Deltares het laatste woord. De expertgroep geeft een eensluidend advies aan de projectgroep. De projectgroep neemt de uiteindelijke beslissing.
- Dorband en Sánchez maken onverkort deel uit van de expertgroep. DHV en N&S zien de 'quick wins' ontstaan door uitgaande van 2D radarbeelden energie te steken in de kalibratie met grondstations en het zo vroeg mogelijk kalibreren. Het verbeteren van de radarbeelden zelf zal moeten worden beschouwd na het toepassen van de kalibratietechnieken en heeft 'een lagere prioriteit. Het verbeteren van de 2D radarbeelden betekent dat er geen gebruik kan worden gemaakt van de aangeleverde bestanden van de meteorologische organisaties (met name DWD). Hierdoor zal de doorlooptijd onder druk komen te staan. De projectgroep geeft

aan hier niet zonder meer mee in te stemmen, maar de discussie in de expertgroep hierover af te wachten.

- We blijven ons inspannen om nieuwe deelnemers te vinden. Vooral gemeenten worden de komende tijd benaderd. Partners mogen ook in het buitenland gezocht worden. Op dit moment is er contact met een partij op het Duitse eiland Borken. Ook waterleidingbedrijven hebben interesse getoond. DHV en N&S sturen partijen die hun benaderen om mee te doen naar ons door zodat het een project blijft.
- DHV zet een sharepoint op waar alle documenten bij elkaar komen te staan.
- De historische data van de radarstations in België en Duitsland voor 2011 waren niet voorzien in de begroting. Voor Duitsland was in de offerteaanvraag aangegeven dat deze gegevens beschikbaar moeten zijn. DHV neemt deze post van 1600 euro op zich. De kosten van de Belgische gegevens (3000 euro) worden betaald uit de pot onvoorzien.
- radarstation Zaventem schijnt de neerslag structureel sterk te onderschatten.
- Radardata ouder dan 1 jaar wordt niet bewaard door DHV en N&S. Dat moeten we ergens anders regelen. Dat kan bijvoorbeeld bij het Waterschapshuis.
- Alle synopsisgegevens van KNMI grondstations worden doorgeleverd op de ftp. Ook wind en verdampinggegevens dus.
- Jeroen Hermans stuurt een mail rond waarin het KNMI zegt dat de kwaliteit van haar handmeetstations beter is dan die van de AW stations (zie andere bijlage van de mail). Dit wordt meegenomen in de expertbijeenkomst.
- De rekening waaruit de kosten van DHV worden betaald, wordt voorlopig toch beheerd door Noorderzijlvest vanwege de traagheid van het Waterschapshuis. We hebben afgesproken dat de eerste twee betalingstermijnen in elkaar worden geschoven en dat dus 50% van de gedekte opdracht direct wordt betaald. Hiervoor is op korte termijn 150.000 euro nodig. NZV stuurt een brief met acceptgiro naar de deelnemers om de toegezegde bedragen te innen. Mocht je specifieke verplichtingnummers willen hebben op die acceptgiro, dan moet je die deze week melden. **Actie allen.**
- Voortgangsoverleg 1 wordt gepland begin september; voortgangsoverleg 2 half oktober; voortgangsoverleg 3 half november en het opleveringsoverleg 2^e week van januari 2013. Jan stuurt datumbriefjes rond. Reageren binnen 1 week is **actie allen.**



BESPREKINGSVERSLAG

Datum: 17 September 2012

Plaats: Amersfoort

Aanwezig: Klaas-Jan van Heeringen (Deltares), Hidde Leijnse (KNMI), Pier Schaper (weterskip Fryslan), Jan Gooijer, Arne Roelevink (ws. Noorderzijlvest) Albert Siebring (ws. Hunze en Aa's), Pieter Filius (ws. Velt en Vecht), Marcel Boomgaard (HHHNK), Harm Klopman, Dries Jansma (gemeente Groningen), Peter Verhulst (ws. Roer en Overmaas), Hanneke Schuurmans (rhdhv), Jan-Maarten Verbree (N&S)

Afwezig: Henk Top (ws. Regge en Dinkel), Gert van den Houten (ws. Rijn en IJssel), Henk Folkerts (ws. Velt en Vecht), Hans de Vries (ws. Noorderzijlvest)

Verslag: Radarproject, Voortgangoverleg 1

1 **Bespreekverslag vorige vergadering**

- De huidige ftp site wordt dusdanig ingericht dat deze te gebruiken is voor de uitwisseling van verslagen/ rapportages/agenda's etc. (ipv de genoemde sharepoint)
- Opmerking Pier: Weliswaar voorlopig geen gezamenlijke belangstelling voor het begeleiden van de inkoop en beheer en onderhoud van de grondstations, maar laten we hier eind van het jaar op terug komen en kijken in hoeverre deze vraag op dat moment meer relevant is.
- De opmerking op blz 1 "Het verbeteren van de radarbeelden zelf zal moeten worden beschouwd na het toepassen van de kalibratietechnieken en heeft 'een lagere prioriteit", is deels achterhaald. Hier is inmiddels aandacht voor geweest in expert groep.

2 **Presentatie voortgang**

Hanneke geeft een presentatie (zie ftp.lizardssystem.nl/ Bespreekverslagen)

Enkele aandachtspunten:

- Voor het maken van het composietbeeld worden 2 opties uitgevoerd. 1) Volumebenadering; gaat uit van onderste elevatie. Het voordeel hiervan is dat over het algemeen dat de onderste elevatie als meest betrouwbare elevatie beschouwd kan worden. Het nadeel is dat de overgang tussen twee radarstation een hele scherpe is, en daarmee niet altijd een realistische overgang kan zijn. 2) CAPPI beelden; maakt gebruik van een parabolische functie. Dit is de methode die op dit moment ook gebruikt wordt door KNMI voor de Nederlandse radar composiet. Beide opties worden onderzocht op basis van 2011-data.
- Duitsland kent standaard maar 1 elevatie die begrensd is op 150 km.
- 2 grote voordelen van dit project worden nogmaals benadrukt:

- Gebruik van buitenlandse radar levert een betere dekking
- Een betere/ uitgebreide dekking van grondstations.

3 **Bespreekpunten: proces + inhoud**

- Jan-Maarten stelt voor om 18 september 2012 als deadline te houden voor het aanleveren van grondstations.
- Jan G. belt Henk Top om er voor te zorgen dat de resterende data van Regge en Dinkel uiterlijk 18 september worden aangeleverd. **Actie Jan**
- Jan-Maarten voegt lijst benodigde operationele data toe aan ftp-site. **Actie Jan-Maarten**
- Jan G. belt met Frank Lantsheer over de nowcast-techniek. Zowel RHDHV als de opdrachtgever hebben eerder begrepen dat het KNMI de software gaat leveren voor de nowcast. Nu doen ze daar moeilijk over. Twee opties: of KNMI levert de software alsnog of ze dragen bij aan de kosten om dit opnieuw te ontwikkelen voor dit project. **Actie Jan**
- Afgesproken wordt om 28 september 2012 als deadline te hanteren voor het aanleveren van de juiste operationele data. **Actie allen**
- Synoptische data voor 2011 is mogelijk niet aanwezig bij KNMI
- Synoptische data is mogelijk beschikbaar via waterschappen/ Deltares. Klaas-Jan geeft aan dat je rekening moet houden met grote gaten. Hij gaat na wat de mogelijkheden zijn. **Actie Klaas-Jan.**
- KMI heeft geen grondstations
- Hidde inventariseert intern in hoeverre KNMI kan helpen bij het ondersteunen bij inkoop grondstations. **Actie Hidde**
- Aandacht behouden voor het benaderen van de gemeenten. Jan G. heeft een presentatie gegeven voor de Verenging van Drentse Gemeente (VDG) en hij gaat morgen met Pier naar gemeente Sneek. Eind oktober/ begin november geeft Arne een presentatie voor Waternetwerk-Noord.
- Marcel laat zien hoe zij bij HHNK communiceren met de gemeenten en hoe zij een gezamenlijke hoofdpst hebben ingericht. **Actie Marcel**
- Voorstel wordt gedaan om een artikel te schrijven, bijvoorbeeld in H2O. Gevraagd wordt wie belangstelling heeft om hier aan mee te schrijven. Nog geen concreet besluit genomen. **Actie allen**
- Voorstel wordt gedaan om een website te maken, beheerd door DHV/ N&S. De inschatting is dat het per waterschap 500 euro kost en het verzoek is om dit niet uit de pot 'onverzien' te halen. Afgesproken wordt om een definitief besluit in de volgende vergadering te nemen maar de kosten van dit soort extra's te verzamelen en in januari nog 1x per een rekening te komen. Anders blijven we bezig met administratie. Het gaat dus nu om een toezegging. **Actie Allen**

4 **Besluitpunten**

Hanneke stelt einddatum ter sprake, nu de aanlevering grondstations is vertraagd. Het streven blijft om het eindproduct op 10 januari te bespreken. Of de eventuele boete al dan niet van toepassing is, wordt schriftelijk + concreet vastgelegd nadat duidelijk is of en hoeveel vertraging er komt. **Actie Jan G.**

5 Rondvraag

-

AKTIEPUNTENLIJST BEHORENDE BIJ HET VERSLAG

Overzicht actiepunten per					
Onderwerp	Genoemd op	Datum afdoening	Actie door	Opmerking	Gereed

BESPREKINGSVERSLAG

Datum: 23 oktober 2012

Plaats: Amersfoort

Aanwezig: ArneJan van Loenen (Deltares), Jan Gooijer, Arne Roelevink (ws. Noorderzijlvest), Gert van den Houten (ws. Rijn & IJssel), Jeroen Hermans (HHNK), Dries Jansma (gemeente Groningen), Peter Hulst (ws. Roer en Overmaas), (rhdhv), Jan-Maarten Verbree (N&S)

Afwezig: Henk Top (ws. Regge en Dinkel), Henk Folkerts (ws. Velt en Vecht), Albert Siebring (ws. H&Aa's), Pieter Filius (ws. V&V), Hans de Vries (ws. Noorderzijlvest), Harm Klopman (gemeente Groningen), Klaas-Jan van Heeringen (Deltares), Hidde Leijnse (KNMI), Pier Schaper (weterskip Fryslan), Hanneke Schuurmans (rhdhv)

Verlag: Radarproject, Voortgangoverleg 2

1 Bespreekverslag vorige vergadering

Peter Verhulst moet zijn Peter Hulst

- KMI heeft wel grondstations, maar dat is voor nu nog niet aan de orde. Dat komt eventueel pas na de oplevering van fase 1 in januari.
- Actiepunt Hidde: KNMI ziet voor zich een rol weggelegd bij het beheer en onderhoud van grondstations. Eerst volgend overleg komt Hidde hierop terug.
Actie Hidde
- Momenteel ligt het project nog op koers wat tijdsplanning betreft. Er zijn dus nog geen afspraken gemaakt over eventuele vertraging.

2 Presentatie voortgang

Jan-Maarten geeft een presentatie (zie <ftp.lizardsystem.nl/> Bespreekverslagen) De belangrijkste punten staan hieronder benoemd.

3 Bespreekpunten: proces + inhoud

Planning

Wat betreft planning hebben we nu 5 maanden gehad, en zijn er nog 2 maanden te gaan. We liggen op schema, maar als de projectgroep bijvoorbeeld extra tijd wil besteden aan het bouwen van het composietbeeld, gaat ten koste van de tijd voor kalibratie. Van belang is te beseffen dat de mogelijkheden om bijvoorbeeld clutter te verwijderen niet uitgeput worden in deze fase van het project. Een deel van de mogelijkheden komen in de aanbevelingen terecht en kunnen na afronding van deze fase in een vervolgproject aangepakt worden.

Composiet

Er worden drie (ipv 2) methode gepresenteerd om een composiet van radarbeelden te maken, namelijk:

1. Weging
2. Onderste elevatie
3. Gewogen elevatie

Het voorstel, ook vanuit de expertgroep, is om te kiezen voor de optie gewogen elevatie. Dit is van de drie voorgestelde, de beste methode. Deze methode betreft een combinatie van de eerste 2 methodes. Daar waar de onderste radarbundel de onderste bundel van ander bundel snijdt, worden beide beelden gebruikt met een weegfactor (blending). Op deze manier is er op de overgang van radarbundels geen harde rand waar te nemen. Het weeggrid wordt een dynamisch grid (ipv een vast grid) zodat op die manier een robuust systeem ontstaat mocht een van de radars (tijdelijk) uitvallen.

Echter, bij deze methode houd je ook harde grenzen. Namelijk, juist daar waar de radarbundels nagenoeg parallel lopen. Dit is over het algemeen op de randen van de radarbundel en buiten Nederland. Je ziet dan wel randen, maar ver weg van Nederland.

Benadrukt wordt dat dit composiet met de voorgestelde methodiek voor het samenstellen van het composiet, duidelijk een ander product oplevert. Zowel KNMI als Hydronet gaat uit van een gewogen composiet met metingen op zo'n 1500m hoogte. Doordat de laagste elevatie wordt gebruikt wordt de neerslag zo dicht mogelijk bij de grond gemeten. Bovendien worden neerslagpieken beter gemeten, omdat extremen in de ene radar niet uitgedoofd worden door menging met een andere radar. Dit betekent dat dit radarproduct ook voor organisaties die relatief dicht bij een radarstation een meerwaarde kan hebben, waardoor echt heel Nederland en de grensgebieden baat hebben bij deelname. Rondom radar is relatief veel clutter aanwezig. Dit wordt deels ondervangen door:

- Rondom radar met een straal van 15 km, data eruit 'stansen' en opvullen met data van omringende radars.
- Middels kalibratie zal de clutter minder kunnen worden (niet verdwijnen).
- In januari wordt ook de radar uit België (Jabbeke) toegevoegd en gebruikt voor het opvullen van de kegels rondom de Bilt en Den Helder.
- Er wordt nog gekeken naar simpele methode om clutter te verwijderen. Complexere technieken zijn mogelijk beschikbaar bij Hyds, het Spaanse meteobedrijf. Gebruik van deze technieken komt eventueel in de aanbeveling terecht.

Grondstations

Het verwerken van de grondstations heeft meer werk opgeleverd dan verwacht. Het meerwerk tot 28 september onder andere in de vorm van communicatie wordt ingeschat op 2500 à 3000 euro door de opdrachtnemer.

De aanlevering van de gemeente Groningen, ws. Velt en Vecht en Wetterskip zou nu goed moeten zijn. Peter streeft er naar om de data voor Roer en Overmaas ook

uiterlijk morgen (24/10) in de juiste format op de ftp te hebben **Actie Peter** (inmiddels afgerond: alle waterschappen leveren nu goede data).

Ws. Rijn en IJssel levert nu aan in UTC +1. Dit blijft zo.

Jan fungeert vanaf nu als tussenpersoon en controleert de data. Op die manier zal er geen extra meerwerk meer zijn op de post grondstations.

De projectgroep gaat akkoord met een meerwerkpost van 3000 euro ex BTW of €3630,= euro inclusief 21% BTW. Deze extra kosten komen ten laste van de post onvoorzien.

Kalibratie en validatie

2 opties worden voorgesteld:

- IDW (Inverse-Distance-Weighted) → snel en simpel
- Krigen

Kalibratie vindt plaats op drie verschillend momenten:

- Elk 5 minuten (real time)
- Elk uur (near real time)
- Na 48 uur.

Afhankelijk hoe snel de kalibratiemethode is, wordt deze toegepast. De verwachting is dat Krigen hoogstwaarschijnlijk lukt voor de near real time kalibratie. De vraag is of dit ook lukt voor de real time kalibratie.

Als een station 1 uur mist, wordt dit station in zijn geheel niet meegenomen in de 24 uur som.

De kalibratie van het 48 uur beeld vindt hoogstwaarschijnlijk plaats op de ruwe data en niet op de al gekalibreerde data. Jan- Maarten vraagt dit na bij Hanneke. **Actie Jan-Maarten.**

De historische data moet de projectgroep zelf opslaan. De data allemaal afzonderlijk opslaan is niet zinvol. Jan gaat nogmaals na wat de mogelijkheden hierin zijn bij het Waterschapshuis. **Actie Jan**

Validatie vindt plaats via:

- Statitiek
- Visueel

Validatie van het kalibratieresultaat vindt plaats via Jack-kniving: Wat is het resultaat als je 1 station weghaalt uit het kalibratiegrid en de radarneerslag vergelijkt met de gemeten neerslag op dat station? Dit gebeurt voor 30 dagen met interessante neerslaggebeurtenissen (geen aaneengesloten periode) neerslag in 2011, waarbij rekening wordt gehouden met de zomer- en winterperiode. Jack-kniving gebeurt voor circa 10 stations, rekeninghoudend met verschillende types grondstations en verspreid in Nederland.

Nowcasting

Er is contact geweest met KNMI tussen Frank Lantsheer en Jan over het leveren van software voor het toepassen van nowcasting. Binnen de planning van dit project is het niet mogelijk om de software van het KNMI te gebruiken. Het KNMI levert in ieder geval voor het eind van dit jaar de beschrijving van de methode. Jan plant een gesprek tussen opdrachtnemer, opdrachtgever en KNMI om over het vervolg te praten. Mocht het KNMI om niet nowcasting data willen leveren uit de catalogus om te compenseren voor het niet nakomen van een afspraak, dan implementeren we deze data. Zo niet, dan wordt nowcasting in deze fase van het project niet toegevoegd aan het eindproduct. De kosten en baten liggen dan te ver uit elkaar. **Actie Jan**

Hanneke geeft opdracht aan KNMI voor het leveren van de operationele data. Levering voor 1 november is hoogstwaarschijnlijk geen probleem. In overleg met Wietze Schuurmans worden de extra kosten als gevolg van het gebruik van volumedata van de radarmasten (in plaats van de CAPPI's) voor de eerste 2 jaar betaald uit onvoorzien (circa 16.000 euro voor twee jaar).

De testdata van KMI is nu binnen bij de opdrachtnemer. KMI is echter op dit moment nog bezig met valideren van de radarmast van Jabbeke. Waarschijnlijk wordt operationele data niet eerder aangeleverd dan januari 2013.

De aanlevering van HIRLAM en ECMWF verzorgt Hanneke. **Actie Hanneke**

De synops database wordt beschikbaar gesteld via ftp. Jan Maarten gaat na of dit via de bestaande of mogelijke nieuwe ftp site gaat. **Actie Jan-Maarten**

Rapportage

In het volgende projectgroep overleg wordt een concept rapportage voorgelegd. Verzoek wordt gedaan om veel plaatjes toe te voegen.

Marketing

Voorstellen voor de website: toevoegen evenementen agenda en bewegende beelden

Peter gaat samen met Joep Rispens (N&S) en ws. Peel en Maasvallei mogelijkheden bespreken met gemeenten.

Peter stuurt eveneens website naar omringende waterbeheerders in België en Duitsland.

De wens voor een website in het Frans en Duits is uitgesproken.

Jan heeft gesprek gehad met Timo Kroon (waterdienst). Waterdienst heeft intentie 60.000 euro bij te dragen aan dit project. Mogelijk dat ze daarvoor in ruil een concreet product willen voor het eind van het jaar.

Arnejan gaat na wat de programmering is van de FEWS dagen en in hoeverre promotie van dit project daarin een rol kan spelen. **Actie Arne Jan.**

Er worden 2 opties gegeven voor participatie:

- Deelname radarproject
- Deelnemers Rainapp.

Wees er van bewust dat de data straks vrij beschikbaar is. Wij moeten er op aansturen als projectgroep dat potentiële deelnemers willen betalen voor de ontwikkeling van het product. Voorstel wordt gedaan voor een lopende rekening namens opdrachtgever, waarmee we er voor zorgen onafhankelijk te zijn van de opdrachtnemer. Dit raakt aan

de gedachtevorming over de voortzetting van de projectgroep na oplevering van het product in januari. **Actie allen: kom met ideeën hoe we de projectgroep na januari voortzetten, met een lopende rekening, onder de vlag van Waterschapshuis?**

4 Besluitpunten

De aanwezige projectleden kiezen voor de methode ‘gewogen elevatie’

Afspraak wordt gemaakt dat in de rapportage een lijst met aanbevelingen wordt opgemaakt.

Een van de aanbevelingen betreft een beschrijving van een techniek om de clutter rondom radarstation (nog meer) te verwijderen.

In de rapportage wordt een voorbeeld van weergave van het ‘weeggrid’ opgenomen. Hierbij wordt wel opgemerkt dat dergelijke plaatjes wel op de juiste manier geïnterpreteerd moeten worden en dat dit vrij complex is.

De aanwezige projectleden zijn akkoord het meerwerk mbt de communicatie rondom het aanleveren van grondstations van 3.000 euro ex BTW te betalen uit de pot onverzien.

Jan checkt of alle operationele data op de juiste wijze op de ftp site staat en laat dit weten aan Frederik (N&S) **Actie Jan**

Een voorbeeld van Jack-kniving wordt uitgewerkt. **Actie Jan-Maarten**

In de aanbeveling wordt opgenomen om de methode van jack-kniving te gebruiken/te onderzoeken voor het bepalen van de kwaliteit van een grondstation. Dus niet het valideren van het radarbeeld uitgaande van een goede meting van het grondstation, maar andersom, als reguliere check van grondstations.

De aanwezige projectleden zijn akkoord om voor nu binnen dit project geen extra investering te doen in nowcasting. De wens voor nowcasting wordt geparkeerd en doorgeschoven naar de lange termijn, tenzij het KNMI gratis nowcasting data wil leveren.

De aanwezige projectleden zijn akkoord met voorstel 500 euro per waterschap bij te dragen, voor het ontwikkelen van de website. De verrekening volgt op het eind van het project.

5 Rondvraag

Gert: In hoeverre valideert de opdrachtnemer de grondstations. Als voorstel doet Gert de case Hupsel. Jan-Maarten gaat na wat hierin de mogelijkheden zijn en komt de eerste volgende projectgroeptoverleg er op terug. **Actie Jan-Maarten.**

Gert: Rijn en Ijssel heeft nog geen rekening ontvangen. Jan gaat dit na, ook de rekening voor de andere deelnemers. **Actie Jan.**

BESPREKINGSVERSLAG

- Datum:** 22 November 2012
- Plaats:** Amersfoort
- Aanwezig:** Arnejan van Loenen, Hidde Leijnse, Pier Schaper, Jan Gooijer, Pieter Filius, Peter Hulst, Hanneke Schuurmans, Jan- Maarten Verbree, Henk Top, Jeroen Hermans
- Afwezig:** Jan Folkerts, Albert Siebring, Hans de Vries, Gert van den Houten, Arne Roelevink, Harm Klopman, Klaas-Jan van Heeringen, Dries Jansma
- Verslag:** Radarproject, Voortgangoverleg 3

1 Bespreekverslag vorige vergadering

- KNMI denkt intern na over beheer grondstations. Jeroen en Pier zijn primair contactpersoon in dit proces vanuit de projectgroep. Kernvragen zijn wat de randvoorwaarden zijn, wat de kosten worden en of het KNMI voldoende ruimte vindt in haar taakbeschrijving om dit te doen. Hidde hierop terug. **Actie Hidde**
- Er is een idee om de historische database van radarbeelden te beleggen bij de STOWA via de meteobase.nl. Dit is nog in een pril stadium, maar zou ideale manier zijn om de STOWA weer bij dit project te trekken.
- Nowcast: we besluiten de beschrijving van de methode af te wachten en de nowcast niet mee te nemen in deze eerste fase van het project.

2 Presentatie voortgang

Hanneke geeft een presentatie (zie <ftp.lizardsystem.nl/> Bespreekverslagen) De belangrijkste punten staan hieronder benoemd.

3 Bespreekpunten: proces + inhoud

Planning

De planning is nog op orde en de einddatum komt in zicht.

Composiet

In de expertgroep is besproken dat de clutter absoluut uit het composietbeeld moet. Hier is inmiddels gevolg aan gegeven. Er worden twee decluttermethoden toegepast: een historische methode op basis van clutter op droge dagen (wordt algoritme dat een keer per maand draait) en een size declutter (1 pixel in verder droge omgeving wordt verwijderd, maar niet aan de rand van buien waardoor grote buien niet kleiner worden gemaakt). Deze decluttering wordt nu op het composiet toegepast. Het is beter dit te doen op de polaire data. Op de plekken waar clutter verwijderd is, wordt data van de andere radars gebruikt.

Grondstations

Door het gebrekkige aantal grondstations dat elke 5-minuten data aanlevert, wordt het 5-minutenbeeld sterk beïnvloed door de wel frequent geleverde data uit Limburg. De aanbeveling is om meer grondstations elke 5-minuten data te laten leveren. Dit zou kunnen bij Hunze & Aa's, Wetterskip Fryslân en de gemeente Groningen. Het KNMI gaat zich inzetten om de data van de AW stations ook per 5 minuten aan te leveren.

Actie WF, H&A, Groningen en KNMI

We stappen af van kwaliteitslabels per grondstation. We gaan ervan uit dat alleen data van goede grondstations wordt aangeleverd. Dit gebeurt in de praktijk ook.

Kalibratie en validatie

Uit de expertgroep is gekomen dat het aan te bevelen is alleen te focussen op kriging als kalibratiemethode. In de praktijk zijn er dus wat problemen met kalibratie bij te weinig grondstations. Daar wordt aan gewerkt.

Rapportage

Het conceptrapport komt in januari beschikbaar. Daarop zijn in één ronde reacties op mogelijk.

Marketing

We gaan als projectgroep 1500 euro aanvullend besteden aan de basisontsluiting van de data via een website. Daarnaast ontwikkelt Nelen & Schuurmans een eenvoudige webviewer die ze gaan aanbieden aan klanten om de data te ontsluiten. In de komende tijd worden alle ontsluitingsvarianten netjes uitgewerkt en gepresenteerd op de website.

Nelen & Schuurmans gaat mede namens de projectgroep brieven versturen naar gemeenten en waterschappen die nog niet meedoen. Nadere afstemming over hoe en wat volgt (en is inmiddels uitgevoerd). Nodig omdat er in december veel contracten worden vernieuwd.

Het KNMI gaat kijken of ze de kostprijs voor volumedata omlaag kunnen brengen, omdat we in de praktijk maar 1 elevatie gebruiken. **Actie Hidde**

4 Besluitpunten

Akkoord met brief mede namens projectgroep verstuurd door Nelen & Schuurmans.

5 Rondvraag

hier is geen gebruik van gemaakt.

BIJLAGE A.3 Overlegverslagen expertgroep

AGENDA

Vergadering : startbijeenkomst expertgroep
Datum vergadering : 12 juli 2012
Dossier :
Tijd : 9.00-12.00
Locatie : Royal HaskoningDHV Amersfoort

Ons kenmerk : BA8186
Datum : 10 juli 2012
Classificatie : Openbaar

Aanwezigen : Hidde Leijnse – KNMI
Arnejan van Loenen - Deltares
Remko Uijlenhoet – Wageningen Universiteit
Gert van den Houten – Rijn en IJssel, vertegenwoordiger projectgroep
Jan-Maarten Verbree – Nelen en Schuurmans
Hanneke Schuurmans – Royal HaskoningDHV

Afwezigen: Nick van de Giesen – TU Delft
Marc Bierkens – Universiteit Utrecht
Jan Gooijer – Noorderzijlvest, vervangen door Gert van den Houten – Rijn en IJssel
Klaas-Jan van Heeringen – Deltares, vervangen door Arne-Jan van Loenen

Bijlagen: Gedetailleerde plan van aanpak.
Verslag startbijeenkomst projectgroep

- 1 Opening - Hanneke
- 2 Voorstel rondje – iedereen
- 3 Procesafspraken:
 - a. Rol binnen expertgroep; hoe zitten we bij elkaar
 - b. Planning overlegmomenten
 - c. Inzet hyds
- 4 Inhoud
 - a. Toelichting op plan van aanpak: doel, tijdsplanning en begroting;
 - b. Toelichting huidige stand van zaken;
 - c. Composiet;
 - d. Kalibratie met stationsdata;
 - e. Validatie;
 - f. Nowcasting.

Ad 1, 2, 3:

- Arne Jan van Loenen is degene die vanuit Deltares in de expert groep zal zitten in plaats van Klaas-Jan van Heeringen
- Verwachting is gedeeld en hetzelfde: de doelgroep zijn de eindgebruikers zijnde waterschappen en gemeenten. Doel is een beter product dan huidige KNMI producten te leveren.
- We streven naar consensus binnen de expertgroep. Als dat niet mogelijk is wordt dit kenbaar gemaakt. De adviezen en keuzes worden voorgelegd aan opdrachtgever/projectgroep.
- Alle input vanuit de experts wordt meegenomen in verslaglegging en technische documentatie. Definitieve keuzes over implementatie hangt af van bewegingsvrijheid binnen het samenhangende geheel van scope – kwaliteit – tijd - financiën. Keuzes en motivatie wordt door opdrachtnemer teruggekoppeld aan expertgroep en opdrachtgever.

Taal van technische documentatie (calibratie en validatie rapportage)

Remko Uijlenhoet heeft per e-mail richting Hanneke de vraag gesteld waarom (technische) documentatie niet in engels plaatsvindt. Hanneke licht tijdens het overleg toe dat zij dit aan Jan Gooijer heeft voorgesteld maar dat deze voor Nederlands heeft gekozen. Toch de moeite om dit punt te laten passeren. Er is geen consensus binnen de expertgroep. Algemeen advies vanuit de expertgroep is zowel engels als Nederlands. Mening verschilt per lid van de expertgroep.

- Voordeel NL (en nadeel van Engels): draagvlak bij deelnemers.
- Voordeel EN (en nadeel van Nederlands): het is een internationaal grensoverschrijdend project met Duitsers en Belgen (waaronder niet-Vlamingen), er zitten buitenlandse experts in de pool of experts die geen input kunnen leveren als ze niet weten wat er precies gebeurd en het sluit beter aan bij wetenschap. Dit project heeft de ambitie state-of-the-art te zijn.
- Actiepunt: overleg noodzakelijk tussen opdrachtgever en opdrachtnemer. Optie is om in principe NL te hanteren met EN voor de technische inhoudelijke stukken die gedeeld moeten worden met buitenlandse partners. Overleg met de opdrachtgever (projectgroep) is nodig om keuze te maken en consequenties te bespreken.

Brondata:

We zijn de stroomschema's doorgelopen. Hidde heeft toelichting gegeven op stroomschema KNMI. Diverse opmerkingen zijn gemaakt over stroomschema, deze worden verwerkt bij update van de stroomschema's. Consensus bestaat dat de brondata die gebruikt worden in dit project beschikbaar moeten zijn als producten bij meteorologische organisaties.

KNMI

In geval van KNMI is dat de KNMI catalogus. Het KNMI kan niet garanderen dat dit product in de catalogus wordt opgenomen. Hiermee vervalt de optie single site cappi's. Twee opties blijven over en zijn besproken:

1. composiet KNMI – mm's
2. volume data per radar- gebruik van onderste of een na-onderste elevatie

Voorkeur gaat uit naar optie 2.

Redenen met name voor de lange termijn: Alle filters / bewerkingen die het KNMI toepast kunnen worden aangepast. Voor de lange termijn zit je dan het best omdat hyds software goed met volume data overweg kan. Bovendien kan met dit product dichtbij de regenradar de resolutie van de griddata worden verhoogd, iets wat voor de lange termijn het overwegen waard is. Hidde geeft aan dat hij verwacht dat rekentijd (voor omzetten naar neerslagvolume) binnen de afgesproken leveringstermijnen kan vallen. Hidde biedt zijn ondersteuning aan voor het leveren van de data 2011 en het projecteren van de data naar cartesisch stelsel.

Actie: Jan-Maarten stuurt naar Hidde de ftp site die voor dit project is aangemaakt en Hidde levert de data voor 2011

Actie: overleg met opdrachtgever noodzakelijk over het verschil in kosten KNMI data t.o.v. uitgangspunt offerte.

DWD

In geval van DWD zijn de 2 opties die bij het KNMI zijn aangegeven hetzelfde.

- RX/RZ: grid van 1 km x 1 km van alle duitse radars – mm's
- DX: 1 km x 1° Azimut – onderste elevatie
- Punt van aandacht is dat DWD een andere Z-R relatie gebruikt dan KNMI

DX data is in RVP6-units. Dit kan worden vertaald naar dBZ. Vervolgens kan dit m.b.v. Z-R relatie worden omgezet naar neerslagvolumes. Experts (Hidde en Remko) raden aan om 1 Z-R relatie te gebruiken. Marshall Palmer ($Z = 200 * R^{1.6}$) wordt door KNMI gebruikt en wordt geadviseerd.

Actie: Hanneke vraagt deze week voor 2011 de DX data bij de DWD aan alsmede de 17 stationsdata van de DWD.

KMI

Op dit moment is uitgegaan van de radar Zaventem. Widemont is vervallen omdat Neuheilenbach dit kan overnemen. Momenteel is KMI bezig met Jabbeke radar.

Ervaring opdrachtnemer: contact met KMI verloopt moeizaam en kost veel meer tijd dan gedacht. Geleverde voorbeelddata was onvolledig. De noodzakelijke metadata ontbreekt.

Hidde verwacht dat Jabbeke in augustus/september operationeel. Hidde en Remko geven aan dat de kwaliteit van de radar van Zaventem te wensen overlaat. Radar is in beheer van luchthaven en niet bij KMI. Het is geen regenradar en ligt er ook af ten toe uit.

Advies: indien er een keuze gemaakt kan worden dan kiezen voor Jarbeke in plaats van Zaventem. Jarbeke is een regenradar, is in beheer bij KMI en levert betere data en dataformat.

Voor deze fase, waarbij methodieken worden getest op een 2011 set van gegevens, (proof of concept) wordt geadviseerd om deze data mogelijk niet of gedeeltelijk (bijvoorbeeld 1 maand aan data) mee te nemen. Dit i.v.m. de hoge datakosten die het KMI rekent en de lage kwaliteit van radar Zaventem. Nadeel van het helemaal niet meenemen van Zaventem is dat de kwaliteitverbetering als gevolg van het meenemen van een Belgische radar niet aangetoond wordt. Dit is wel wenselijk voor het creëren van draagvlak bij nieuwe deelnemers..

Actie 1: overleg met opdrachtgever noodzakelijk om deze keuze af te stemmen.

Actie 2: Hanneke neemt deze week contact op met KMI en vraagt om de mogelijkheid om Jabbeke aan te schaffen voor het operationele product (in het najaar). En ze zal aangeven dat datalevering Zaventem voor dit moment 'on hold' staat i.v.m. voorgaande actie / keuze.

Stationsdata KNMI:

De stationsdata van het KNMI voor 2011 is er en heeft ook voldoende metadata. Kunnen we mee aan de slag.

Punt over betrouwbaarheid van STN versus AWS is behandeld. Geaccumuleerde waarden van AWS leveren andere hoeveelheden dan de eenmalig uitgelezen STN waarden. AWS is nauwkeurig voor hogere temporele resolutie dan 1 dag. Er is immers vanuit STN alleen dagwaarden beschikbaar. Hidde en Remko geven aan dat ze zich kunnen voorstellen dat AWS elke 10 minuten een kleine meetfout heeft en dat dit opgeteld een grotere fout geeft. Punt zal altijd blijven dat als je 2 verschillende soorten regenmeters naast elkaar zet je een verschil ziet. Zowel Hidde als Remko kunnen niet aangeven welke van de twee sets betrouwbaardere waarden geven. Beide zijn goed en mogen 3 sterren krijgen. Pool of experts is ermee akkoord als we AWS als uitgangspunt nemen voor validatie.

Stationsdata DWD:

Moeten nog worden aangeleverd, maar zal naar verwachting deze maand gebeuren. Gert geeft aan dat mogelijk extra stations wenselijk zijn i.v.m. grensoverschrijdende stroomgebieden (Vecht, Oude IJssel, Niers etc). Dit dient te worden aangegeven door de waterschappen Velt en Vecht, Rijn & IJssel, Peel en Maasvallei en Roer & Overmaas. Gert vraagt naar de coördinaten van de 17 DWD stations.

Actie: Hanneke stuurt deze week de coördinaten van de 17 DWD stations naar Gert.

Actie : Projectgroep besluit over aanschaf van extra DWD / MM grondstations.

Stationsdata deelnemers:

Metadata zijn nog niet volledig. Afspraak was een week na startoverleg .

De meetwaarden moeten door de deelnemers in juli worden aangeleverd zoals afgesproken binnen projectgroep. Data die te laat wordt aangeleverd wordt niet meegenomen. Nadeel is dat de meerwaarde van inbreng lokale stations daarmee minder wordt. Expertgroep erover eens dat tijdspad gevolgd moet worden. Hanneke heeft hierover een mail gestuurd deze week naar Jan Gooijer en Arne Roelevink.

Een juiste tijdsnotatie (door regelmatige synchronisatie en duidelijkheid of tijdzone en zomer/wintertijd) is essentieel voor de waarde van de stationsdata. Het is nu niet duidelijk of bij de deelnemers tijdsynchronisatie plaats vindt.

Actie: Tijdsynchronisatie wordt opgenomen bij metadata. Jan-Maarten past dit aan op de excelsheet van de ftp site.

Software uitwisseling KNMI:

Actie: overleg tussen Jan Gooijer en Frank Lantsheer. Behalve over software uitwisseling zou er bij dit overleg ook het uitwisselen van data moeten worden besproken. DWD geeft suggestie beelden van hun via het KNMI ftp in te winnen. In principe zijn de lijnen kort maar dit is iets dat op beleidsniveau moet worden besproken en besloten.

Composiet:

Er zijn 3 opties aangegeven en besproken, zie hieronder. Verder is ingegaan op de beam-blockage. Hier zal aandacht aan worden besteed in de uitwerking. De resultaten worden teruggekoppeld met de expertgroep.

1. parabool functie: huidige methode KNMI
2. laagste elevatie
3. maximum: nadeel bright band er duidelijk in terug te zien w.s.

Actie: overleg met opdrachtgever over keuze (Hanneke-Jan-Jan-Maarten).

Kalibratie:

Remko geeft aan in eerdere mail dat dit strikt gezien geen kalibratie is maar adjustment. Bij dit overleg niet verder besproken. Opdrachtnemers nemen deze suggestie ter harte en zullen overwegen terminologie over te nemen. Adjustment/aanpassing van radar. Algemeen kan gesteld worden dat scherpe definiëring van termen essentieel is om dezelfde verwachtingen te hebben.

Er is consensus binnen de expertgroep dat het regenradarproduct wordt geoptimaliseerd, niet het regenmeter netwerk.

- Besproken zijn de voorgestelde opties en de mail van Marc Bierkens over geostatistische methoden voor kalibratie. Consensus dat methodiek kan afhangen van product. Real time levering kan mogelijk af met pragmatische methode dan de producten die achteraf worden geleverd.
- Correctieveld (interpolatie Correctie Factor – CF) wordt als ouderwetse methode gezien door de experts, met inachtneming van het soort product
- KED of colocated cokrigening momenteel meer gebruikelijk.
- Historische validatiereeks moet consistent zijn (suggestie van Marc en gebeurt ook huidig bij RainApp en klimatologische set Aart). Je consistentie bepaling (basis waarop alles wordt geschaald) heeft effect op hoeveel effect je statistiek op hogere temporele schaal heeft. Het 24-uursproduct is het belangrijkste qua kwaliteit.
- Punt van aandacht: hoe wordt correctiefactor bepaald. O.b.v. radarpixel of mediaanfilter bijvoorbeeld

- Correctievelde zal multiplicatief moeten zijn. Daarbij een grens hanterend om geen uitschieters te krijgen (delen door bijna nul geeft oneindige waarden). Reden om CF multiplicatief te maken is dat de meeste fouten in radar multiplicatief zijn (uitdoving, verticale profiel, effect van expansie);

Validatie:

Er is consensus dat er bij voorkeur validatie plaatsvindt op 2 vlakken:

1. toegevoegde waarde extra radars
2. toegevoegde waarde extra stations (neerslagmeters).

Bij de stations zal de kwaliteitsverbetering afhangen van de hoeveelheid stations en de kwaliteit hiervan.

Consensus over de 4 scoringscriteria zoals deze in detail plan van aanpak staan (en in ppt behorende bij dit overleg).

Nowcasting

Focus bij dit project ligt op verbeteren van real time radarproduct. Hiermee zal de nowcast verbeteren t.o.v. het huidige KNMI nowcast product. Consensus dat dit met een theoretische onderbouwing voldoet aan kwaliteitseis.

Wel zullen suggesties voor verbetering in de toekomst worden gegeven. Zoals:

- blending met numerical weather prediction models (NWP)
- advectie (huidige methode KNMI)
- advectie en rotatie
- spectrale decompositie

Samenvatting Actiepunten:

Actie: opdrachtgever en opdrachtnemer maken zo snel mogelijk afspraak. Bovenstaande punten zullen behandeld worden alsook de PR.

Actiepunt: overleg noodzakelijk tussen opdrachtgever en opdrachtnemer. Of alles in 2 talen na overleg over meerkosten of keuze met acceptatie van de bijbehorende nadelen.

Actie: Jan-Maarten stuurt naar Hidde de ftp site die voor dit project is aangemaakt.

Actie: overleg met opdrachtgever noodzakelijk over het verschil in kosten KNMI data t.o.v. uitgangspunt offerte.

Actie: Hanneke vraagt voor 2011 de DX data bij de DWD aan alsmede de 17 stationsdata van de DWD.

Actie 1: overleg met opdrachtgever noodzakelijk om deze keuze af te stemmen.

Actie 2: Hanneke neemt deze week contact op met KMI en vraagt om mogelijkheid om Jarbeke aan te schaffen voor operationele product (in het najaar). En aangeven dat datalevering Zaventem voor dit moment 'on hold' staat mogelijk een aantal dagen met interessant weer kiezen.

Actie Hanneke stuurt coördinaten DWD stations door naar Gert

Actie: overleg tussen Jan Gooijer en Frank Lantsheer. Behalve over software uitwisseling zou er bij dit overleg ook het uitwisselen van data moeten worden besproken. DWD geeft suggestie beelden van hun via het KNMI ftp in te winnen. In principe zijn de lijnen kort maar dit is iets dat op beleidsniveau moet worden besproken en besloten.

Actie: overleg met opdrachtgever over keuze composiet (Hanneke-Jan-Jan-Maarten).

AGENDA

Vergadering : tweede expertgroep bijeenkomst
Datum vergadering : 12 okt 2012
Dossier :
Tijd : 13.00-14.30
Locatie : Royal HaskoningDHV Amersfoort

Ons kenmerk : BA8186
Datum : 15 oktober 2012
Classificatie : Openbaar

Aanwezigen : Hidde Leijnse – KNMI
Klaas-Jan van Heeringen - Deltares
Remko Uijlenhoet – Wageningen Universiteit
Nick van de Giesen – TU Delft
Jan Gooijer – Waterschap Noorderzijlvest, vertegenwoordiger projectgroep
Gert van den Houten – Rijn en IJssel, vertegenwoordiger projectgroep
Jan-Maarten Verbree – Nelen en Schuurmans
Jozanneke van Vossen – Nelen en Schuurmans
Hanneke Schuurmans – Royal HaskoningDHV

Afwezig: Marc Bierkens – Universiteit Utrecht

Bijlagen: Gedetailleerde plan van aanpak.
Verslag startbijeenkomst projectgroep

Agenda

1. Opening – Hanneke
2. Stand van zaken
3. Composietmethodiek
4. Resultaat controle
5. Resultaat validatie composietmethodiek
6. Voorstel kalibratie

Presentatie van de expertgroep staat op ftp site. Daar staan ook alle beelden met de 2 composietmethoden voor heel 2011.

[ftp.lizardsystem.nl](ftp:lizardsystem.nl)

user: radar

password: r3g3n

Doel overleg

Tijdens de presentatie zijn we met name ingegaan op de resultaten van de twee composietmethoden. Doel van dit overleg was om de gevolgde methodiek te toetsen en een keuze te maken tussen de twee composietmethoden te weten:

1. lowest elevation
2. weging met parabolische functie

Daarnaast is de werkwijze voor 'kalibratie' (meenemen grondstations) van de radarbeelden besproken. Tijdens het bespreken van de werkwijze en de resultaten zijn diverse acties en aanbevelingen benoemd. Deze staan hieronder opgesomd.

Keuze composietmethode

Aan het eind van het overleg is een keuze gemaakt voor welke methode we gaan gebruiken voor het generen van de radarcomposieten. De voorlopige keuze is gemaakt voor lowest elevation, omdat de 'lowest elevation' theoretisch de beste schatting van de neerslagintensiteit op de grond geeft (radar beam dichtst bij het aardoppervlak). Nadelen zijn dat deze methode:

- visueel onaantrekkelijk is (scherpe randen tussen de radars)
- de Duitse radarsdata weinig worden gebruikt op Nederlands grondgebied

Oprachtnemer maakt een voorbeeld van lowest elevation met 'glad gepoetste randen'. Bij het volgende projectgroepeverleg (23 okt) zal door de projectgroep een definitieve keuze gemaakt worden. De expertgroep krijgt voor de 23-ste de nieuwe resultaten te zien en zal daar nog kort advies op geven.

Actielijst

Algoritme:

- Punten eerst naar RD zetten en dan pas interpoleren. Actie opdrachtnemer (ON)
- Kiezen voor een simpele interpolatie techniek die dichtbij meerdere punten binnen het vakje meeneemt en verder weg interpoleert tussen verder liggende punten (tins). Actie ON
- midden gridcel is op een half kilometer, bij een halve graad. Hidde checkt dit.
- 4/3 aardstraal heeft te maken met de gemiddelde brekingsindex van de atmosfeer, dit is correct.
- Formule voor hoogte berekening heeft Hidde mogelijk al gemaaild naar Frederik: Actie ON
- Hidde: doorgeven hoogte radarstations. Worden meegenomen in de composiet berekeningen

Elevatiemethode:

- Binnenste cirkel van 15 km -> hogere elevatie pakken voor NL in plaats van deze te vervangen voor andere radar. Voor DL geen bewerkingen uitvoeren, alles meenemen, actie ON
- Voor elk radarstation (iig Duitse) een grafiek maken met x,y hoe de radar kijkt. Dus op basis van de elevatie en de afstand. Actie ON
- Graag percentiel plaatjes. Actie ON-Hanneke
- Controle of de grondstations data aannemelijke jaarsommen opleveren. Actie ON-Hanneke
- Dichtheidsplot ipv punten plot bij uitzetten radarbeelden tegen elkaar. Actie ON-Hanneke
- Histogrammen met logaritmische schaal. Actie ON - Hanneke
- Elektronische kalibratie per radar meenemen als aanbeveling

Kalibratie:

- Er worden alleen dagsommen van grondstations bepaald als de data volledig is over die dagen.
- validatie (jack kniving) gefocust op (+/-30) dagen waarbij keuze gemaakt wordt voor aantal dagen in het jaar
- Naast de validatie op dagen, ook ter illustratie een paar uren doorrekenen om te laten zien dat de methode ook werkt voor kalibratie per uur.

AGENDA

Vergadering : derde expertgroep bijeenkomst
Datum vergadering : 12 nov 2012
Dossier :
Tijd : 09.30-12.00
Locatie : Royal HaskoningDHV Amersfoort

Ons kenmerk : BA8186
Datum : 21 november 2012
Classificatie : Openbaar

Aanwezigen : Hidde Leijnse – KNMI
Klaas-Jan van Heeringen - Deltares
Remko Uijlenhoet – Wageningen Universiteit
Marc Bierkens – Universiteit Utrecht
Jan Gooijer – Waterschap Noorderzijlvest, vertegenwoordiger projectgroep
Gert van den Houten – Rijn en IJssel, vertegenwoordiger projectgroep
Jan-Maarten Verbree – Nelen en Schuurmans
Hanneke Schuurmans – Royal HaskoningDHV

Afwezig: Nick van de Giesen – TU Delft

Agenda

1. Opening – Hanneke
2. Stand van zaken
3. Toelichting definitieve keuze composietmethodiek
4. Resultaat composiet
5. Voorstel kalibratie bespreken
6. Keuze datum laatste overleg
7. Afsluiting

Presentatie van de derde expertgroep staat op ftp site. Daar staan ook alle beelden met de 2 composietmethoden voor heel 2011.

[ftp.lizardsystem.nl](ftp:lizardsystem.nl)

user: radar

password: r3g3n

Doel overleg

Tijdens de presentatie zijn we met name ingegaan op de resultaten van de definitieve composietmethode. Doel van dit overleg was om de resultaten van het composiet te bespreken en het voorgestelde plan voor kalibratie te bespreken. Hieronder staan de acties puntsgewijs beschreven.

Stand van zaken

KNMI geeft aan dat zij aan het overwegen zijn om het beheer van grondmeters (van bv. waterschappen) over te nemen. HHNK heeft hierin al interesse getoond en is bezig met het KNMI om dit verder vorm te geven.

Hanneke geeft aan wel contact met het KMI te hebben gehad om testdata te ontvangen maar dit is nog niet geleverd. Ook is nog niet duidelijk wanneer Jabekke operationeel gaat en wat de exacte datakosten zijn.

Er is een overleg geweest tussen KNMI, Jan Gooijer en N&S over Nowcasting. Nowcasting is mogelijk tot maximaal 2 a 3 uur met steeds mindere betrouwbaarheid. De betrouwbaarheid is gemeten door te vergelijken hoeveel op voorspelde *nowcast-neerslag pixels* daadwerkelijk neerslag is gevallen volgens de radar. Er is dus geen validatie op intensiteit geweest. De vraag die voor de projectgroep blijft is of deze informatie interessant is voor de waterbeheerder.

Resultaat composietmethode

De filmpjes van de definitieve keuze van de composietmethode zijn bekeken. Ziet er goed uit, mooie overloop tussen de radars. Wat wel opvalt dat er een aantal plekken zijn aan te wijzen met clutter (Flevoland-windmolens; Harlingen – storing vanuit Den Helder).

Voorstel vanuit de expertgroep:

- deze grote gebieden met clutter dienen te worden verwijderd. Dit is iets dat niet met calibratie is op te lossen. Gevolg is wel dat daarmee minder tijd overblijft om verschillende calibratiemethoden te testen.
- Hidde stelt voor om statistieken van de radars te leveren zodat kan worden nagegaan bij welke radar er outliers worden gesignaleerd
- Suggestie om naast de shape van NL ook de grensstroomgebieden aan te geven. Actie Gert: levering shape file
- Suggestie - Filmpje van nieuwe methode naast de huidige beelden

Calibratie:

Voorstel vanuit de expertgroep:

- IDW is inferieur aan Kriging, voorstel daarop te focussen
- Voorstel om kriging uit te voeren met klimatologische variogrammen
- Als we IDW toepassen, is het voorstel om een decollatie afstand toe te voegen
- Suggestie om eerst een mean-field bias correctie uit te voeren per radar
- We blijven bij 5 min, 1 uur, 24 uur calibratie en voegen geen 10 min. toe. De 10 minuten data wordt dus maar één's per uur meegenomen.
- Bij de calibratie wordt alles opgetild naar een gemiddelde (is te zien bij de IDW test en ook te verwachten bij kriging). De keuze of je de calibratie beperkt tot een bepaalde straal om het grondstation hangt af hoeveel waarde je hecht aan de grondstations en aan het radarbeeld. In principe hechten we meer waarde aan de grondstations en is dit een goede keuze.
- Bij het visualiseren van de calibratie is het verstandig om de correctiefactor logaritmisch te laten zien in de legenda.
- Opvallende clutter moet er sowieso uit.
- Kriging op data en niet op correlatiefactor.
- Hoe omgaan met de verschillende kwaliteitsklassen van regenmeters? Voorstel is om te kijken of we ook die kwaliteitsklassen weg kunnen laten als iedereen toch zijn station op 1 zet.
- Bij colocated kriging krijg je nooit exact je grondstationwaarde terug.
- De kwaliteitsstempel van de neerslagmeters maakt geen onderscheid tussen de kwaliteit van de meter bij droogte en bij neerslag.

Validatie:

Voor de validatie zijn de volgende suggesties gedaan:

- Bij Jack-kniving moet gevalideerd worden op de beste stations (dus met de hoogste kwaliteitsstempel). In geval van kriging kan ook gebruik worden gemaakt van cross validatie
- Visualisatie ook dmv kaart van NL met daarin per station indicatie van de fouten.
- Bij de validatie ook expres bij één grondstation of de radar een fout erin brengen en kijken hoe robuust het systeem de fout eruit haalt of er mee om gaat.
- Bij de visualisatie ook kruisjes waar de meegenomen grondstations er zitten.
- Bij validatie van eindproduct door uit te rekenen met en zonder buitenlandse stations mee te nemen. Zelfde voor het wel of niet meenemen van grondstations (eventueel zelfs op kleiner niveau (wel/niet meenemen deelnemer stations)).
- *Crossvalidatie. Niet alleen op punt maar ook op stroomgebied.*
- Geïnteresseerd in de volgende statistieken (bv. per maand, bijv boxplots):
 - o Bias
 - o Mean absolute error
 - o Standard deviation of error

Grondstations tijdreeks validatie:

- Als er 1 waarde in de tijdreeks ontbreekt wordt station niet meegenomen.
- De maximum grens bij een uur wordt 150 mm, ipv wat voorgesteld was.

Conclusie:

- Focus eerst op Clutter
- Focus vervolgens op Kriging
- Niet tegen KNMI data valideren, maar wel tegen eigen berekening (alleen KNMI stations, ook buitenlandse stations meenemen; geen grondstations, ook grondstations meenemen)

Acties RHDHV/N&S:

- Keuze datum laatste overleg. Deze staat op 14 december. Met datumprikker zal gekeken worden of er nog een beter datum te vinden is voor expertgroep 4
- Verdeling van leveringsfrequentie en robuustheid van de leveringen (voornamelijk van grondstations). Is belangrijk om iets te kunnen zeggen over het nut van calibratie.
- Afspraak maken met Klaas-Jan om de operationele kant te testen en toetsen.

AGENDA

Vergadering : vierde (en laatste) expertgroep bijeenkomst
Datum vergadering : 14 dec 2012
Dossier :
Tijd : 14.00-17.00
Locatie : Royal HaskoningDHV Amersfoort

Ons kenmerk : BA8186
Datum : 21 december 2012
Classificatie : Openbaar

Aanwezigen : Hidde Leijnse – KNMI
Klaas-Jan van Heeringen - Deltares
Remko Uijlenhoet – Wageningen Universiteit
Jan Gooijer – Waterschap Noorderzijlvest, vertegenwoordiger projectgroep
Gert van den Houten – Rijn en IJssel, vertegenwoordiger projectgroep
Jan-Maarten Verbree – Nelen en Schuurmans
Hanneke Schuurmans – Royal HaskoningDHV

Afwezig: Nick van de Giesen – TU Delft
Marc Bierkens – Universiteit Utrecht

Doel overleg

Dit is het laatste expertoverleg binnen het huidige project. Doel van dit overleg is tweeledig.

1. Goedkeuring van de producten
2. prioritering van de aanbevelingen
3. Mogelijkheden voor vervolg.

Tijdens de presentatie hebben we kort alle stappen die we hebben doorlopen in het project de revue laten passeren. Hieronder staan de acties puntsgewijs beschreven die we hebben afgesproken gedurende het overleg.

Agendapunten

- Laatste expertgroep overleg
- Clutter verwijderd
- Kalibratie uitgewerkt
- Operationeel product gevalideerd
- Operationeel klaar voor 9 van de 11 producten!
- Aanbevelingen

Clutter verwijdering

Bij het KNMI wordt een clutter filtering toegepast. Dit heeft echter alleen effect op stilstaande objecten zoals gebouwen. Uit de beelden komt duidelijk naar voren dat er ook clutter ontstaat als gevolg van wateroppervlakten en draaiende objecten als windmolens. Toelichting gegeven op de gevolgde werkwijze van clutterverwijdering – zowel statisch, op basis van historische clutter op regenloze dagen als een operationele clutterverwijdering op basis van

een minimale hoeveelheid pixels. Suggesties vanuit de expertroep:

- threshold van de statische clutter in Z-waarde (dBZ) / fysische betekenis

Kalibratie

In het vorige expertoverleg is uitgekomen dat kriging methode de voorkeur heeft boven andere interpolatiemethodieken. Bij kriging zijn variogrammen belangrijk. In het vorige overleg is dan ook geopperd om gebruik te maken van de klimatologische variogrammen van Van Beek et al. In de afgelopen periode zijn de volgende methoden getest:

- Ordinary Kriging (OK)
- Collocated Cokriging (CCK)
- Kriging with external Drift (KED)
- Inverse Distance Weighting (IDW)

De resultaten hiervan zijn besproken tijdens het overleg. Daaruit komen de volgende punten als belangrijkste naar voren:

- Het meenemen van meer radarstations levert een verbetering op;
- de levering van grondstations voor 5 min en 1 uur laat zeer te wensen over. Hierdoor is het weinig zinvol kriging toe te passen. Een simpelere methodiek op basis van correctiefactor en IDW wordt voorgesteld;
- voor dagwaarden lijkt kriging met external drift de beste methodiek;
- buiten landsgrenzen, oftewel verder van neerslagstations is interpolatie weinig zinvol. Geadviseerd wordt buiten de landsgrenzen de interpolatiewaarden niet te tonen.

Operationele producten

Er wordt een toelichting gegeven op de stand van zaken van de operationale levering alsmede hoe dit wordt vormgegeven. Voor de oplevering is goedkeuring vanuit KNMI en Deltares nodig. Er is op 17 december een afspraak gemaakt met Hidde Leijnse en Klaas-Jan van Heeringen om de code te controleren. Vanuit de expertgroep wordt aangegeven dat dit voldoende is om een goedkeuring te geven aan het eindproduct. Uiteraard zal de expertgroep ook de uiteindelijke rapportage becommentariëren,

Aanbevelingen

De aanbevelingen zijn doorgenomen. Advies vanuit de expertgroep is om de aanbevelingen te splitsen in die voor de afnemers en die voor de ontwikkelaars.

Aanbeveling richting de afnemers is duidelijk de inspanning om meer en betere data van grondstations te leveren. Alleen op deze manier kan de kwaliteit van de calibratie verbeteren

Aanbeveling richting ontwikkelaars is:

- de beam blockage. De suggestie daarbij is om de koppeling te maken met het ongefilterde AHN: wat 'ziet' de radar?
- Combinatie maken van composiet en nowcasting: door de trekrichting mee te nemen en de tussenliggende beelden te interpoleren wordt een 'wasbordpatroon' voorkomen.

Vervolg

Iedereen beaamt dat de vorm van een expertgroep overleg tot zeer positief resultaat heeft geleid. De samenwerking binnen de expertgroep is zeer goed verlopen. De sfeer was altijd zeer positief en open - iedereen was zo op de hoogte van keuzes die gemaakt zijn. Door de strakke planning konden niet alle leden altijd aanwezig zijn maar er was wel degelijk continuïteit in de groep. We hopen deze samenwerking voort te kunnen zetten, juist met de universiteiten en kennispartijen. Als voorbeelden voor financieringsbronnen wordt hetvolgende geopperd:

- KIC Climate
- STW (technologie stichting)
- EFRO: europees fonds voor regionale ontwikkeling

BIJLAGE A.4 presentaties projectgroep


 Nelen & Schuurmans 

NL/DE/BE radarproject

Voortgangsoverleg 1

Kamer 3.06 Het Waterschapshuis Amersfoort

Agenda

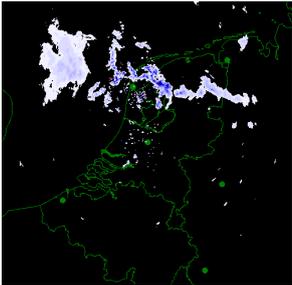
- Besprekverslag vorige vergadering
- Presentatie voortgang (30 min)
- Besprekpunten: proces + inhoud
- Besluitpunten
- Rondvraag

Nelen & Schuurmans 


NL/DE/BE radarproject
Voortgangsoverleg projectgroep 1

Page 2
10 september 2012

KNMI
KNMI + DWD




Nelen & Schuurmans 


NL/DE/BE radarproject
Voortgangsoverleg projectgroep 1

Page 3
10 september 2012

Proces

Planning Radarproject verslag 7 september 2012

Nelen & Schuurmans 


NL/DE/BE radarproject
Voortgangsoverleg projectgroep 1

Page 4
10 september 2012

Inhoudelijke punten (terugkerend)

- Composiet
- Grondstations
- Kalibratie en validatie
- Nowcasting
- Operationalisering
- Rapportage

Nelen & Schuurmans 

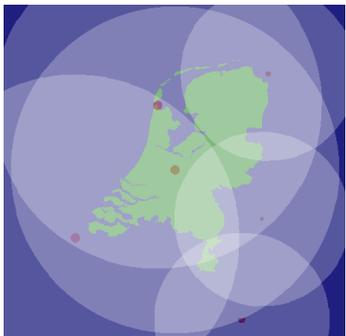

NL/DE/BE radarproject
Voortgangsoverleg projectgroep 1

Page 5
10 september 2012

Hoe komen we tot een composiet?

Dwarsdoorsnede

- Onderste elevatie
- Parabolische functie

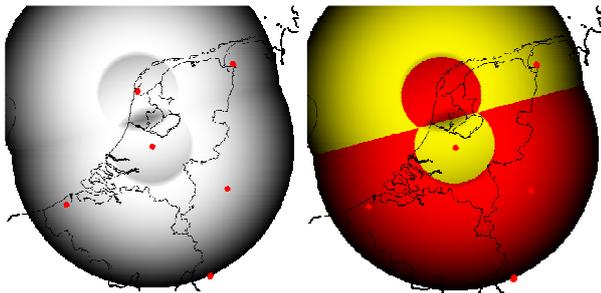


Nelen & Schuurmans 

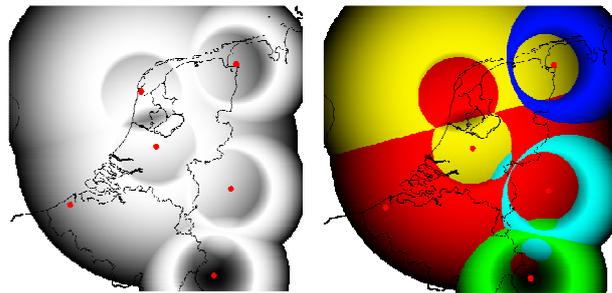

NL/DE/BE radarproject
Voortgangsoverleg projectgroep 1

Page 6
10 september 2012

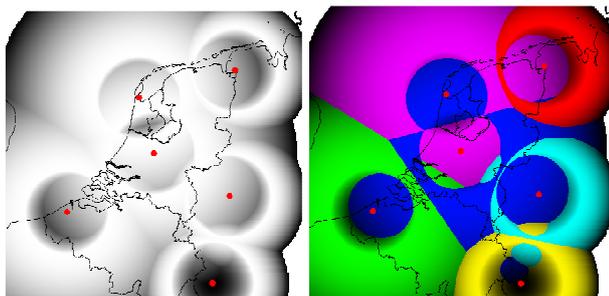
Maximale gewichten: KNMI



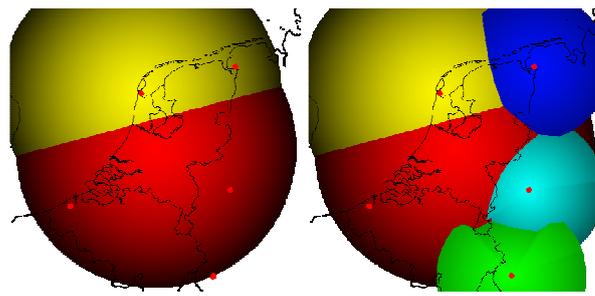
Maximale gewichten: KNMI + DWD



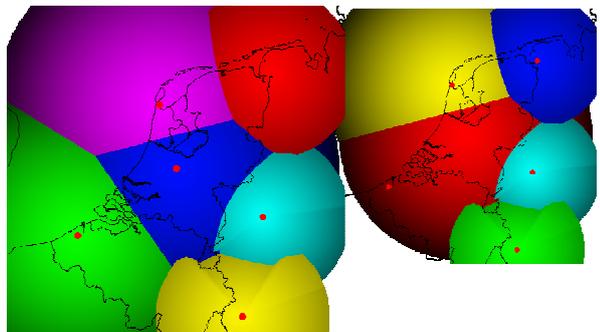
Maximale gewichten: KNMI + DWD + KMI (Jabekke)



Laagste elevatie



Laagste elevatie

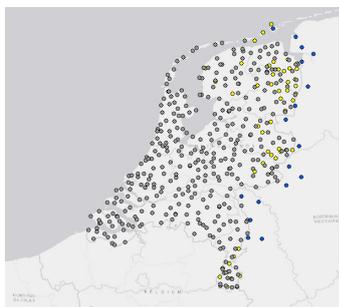


Werkzaamheden

- Inlezen radarbeelden
 - Polaire projectie naar cartetisch stelsel (grid)
 - Calculatie gewichten (parabolische functies)
 - Toevoegen radar
 - Onverhoopt uitvallen radar
 - Inlezen grondstations
- Nog doen:
- Filtering technieken (spikes)
 - Composieten generen: keuze methodiek
 - Calibratie uitwerken
 - Validatie van calibratie methoden: keuze methodiek
 - Nowcasting implementeren

Grondstations

FEWS demo



Grondstations stavaza

Deelnemers status 6 sept 2011

waterschap/gemeente	grondstations	
	Metadata compleet	data 2011 op ftp?
Velt & Vecht	ja	ja
Rijn & IJssel	bijna, x en y coördinaten moeten nog toegevoegd	ja
Regge & Dinkel	nee	nee
Nunze & Aa's	ja	ja
Roer en Overmaas	ja	ja
gemeente Groningen	nee	nee
Wetterskip Fryslân	ja	ja

- DWD
 - 2011: compleet
 - Operationeel: nog niet aangevraagd
- KNMI
 - AWS
 - 2011: uurgegevens, geen 10-min
 - Operationeel: in orde
 - STN
 - 2011: nee
 - Operationeel: in orde, aandachtspunt validatie na een maand

Calibratie

- Methodieken met experts besproken
- Grondstations essentieel

Validatie

- Validatie van composit
- Conclusie expertgroep, validatie van
 - Toegevoegde waarde extra radars
 - Toegevoegde waarde grondstations (databeschikbaarheid)
- Vergelijking met grondstations 2011

Nowcasting

- Conclusie expertgroep: theoretische onderbouwing
- Software uitwisseling KNMI knelpunt: actie opdrachtgever

Operationalisering

- Eerste acties zijn al uitgezet:
 - Koppeling met andere programma's
 - Uitwisseling data
- Aandacht voor stations deelnemers

NL/DE/BE radarproject

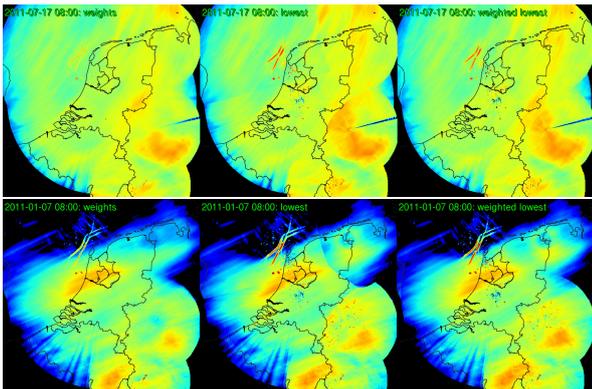
Voortgangsoverleg 2

Het Waterschapshuis Amersfoort

Agenda

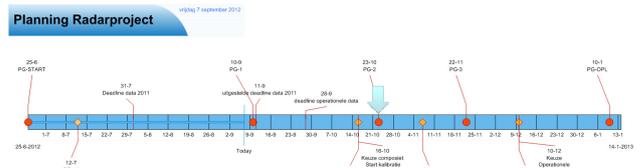
- Opening
- Besprekingsverslag
- Voortgang project
- Marketing
- Besluitpunten
- Rondvraag

Drie methodes



Planning

Planning Radarproject



-5 maanden gehad, nog 2 te gaan

-Goed, **beter**, best

Inhoudelijke punten (terugkerend)

- Composiet
- Grondstations
- Kalibratie en validatie
- Nowcasting
- Operationalisering
- Rapportage

Radarcomposiet

- Toelichting expertoverleg
- Keuze uit 3 methodes:
 - Wegingen
 - Laagste elevatie
 - Gewogen elevatie
- Uitleg gewogen evaluatie



Grondstations stavaza

- Deelnemers status 28 sept 2011 (document met opmerkingen)

Deelnemer	Metadata compleet	data operationeel op ftp?	format data	Interval	Resolutie	Eenheid	UTC	Opmerking
gemeente Groningen	ja	aanwezig, niet operationeel	CSV	5min	0.1 mm	UTC		zie paragraaf
Hunze & Aa's	ja	ja	CSV	5min	1 mm	UTC		
Regge & Dinkel	ja	ja	ixp	20min	0.1 mm cum	UTC		
Rijn & IJssel	ja	ja	zrxp	30min	0.1 mm	UTC+1		
Roeren Overmaas	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf
Velt & Vecht	ja	ja	Pl-XML & CSV	1 uur/10 min	0.1 mm	UTC		zie paragraaf
Waterskip Fryslân	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf

- Welke punten worden operationeel meegenomen?
 - Hunze & Aa's
 - Regge & Dinkel
 - Rijn & IJssel
- Besluitpunt: wat doen met deelnemers die niet volledig zijn?
- Voorstel:
 - binnen projectgroep controle
 - nieuwe invoegronde (na project)
 - kosten rekenen als nieuwe punten

Calibratie

- Voorzet:
 - IDW
 - Krigen
 - Afhankelijk van de uitkomsten keuze voor hogere temporele resolutie
 - Near-realtime calibratie
- Afweging in planning – composiet/calibratie

Validatie

- Validatie van composiet (1^e versie getoond bij expertgroep)
 - Validatie op statistiek
 - Validatie op visueel beeld
- Validatie van calibratie methode
 - Jack-kniving
 - +/- 30 dagen doorrekenen
 - Enkele uren doorrekenen ter illustratie
 - Vergelijking met grondstations 2011

Nowcasting

- Software uitwisseling KNMI knelpunt: actie opdrachtgever

Operationalisering

- Data
 - Radardata
 - KNMI – wordt vanaf xx geleverd (Actie PG)
 - DWD – vanaf november (streven)
 - KMI – vanaf jan/feb operationeel (overleggen gaande)
 - Stationsdata
 - KNMI – wordt vanaf xx geleverd (Actie PG)
 - DWD – vanaf nov (streven)
 - Deelnemers – ronde 1 gereed
 - Voorstellingen
 - Nowcasting – overleg met KNMI gaande (software/data)
 - Hirlam – vanaf xx geleverd (Actie PG)
 - ECMWF – vanaf xx geleverd (Actie PG)

Operationalisering

- Systeem
 - Servers ingericht en data ingelezen
 - Uitlevering via ftp komende maand inrichten
 - Visualisering: in progress

Rondvraag


 Nelen & Schuurmans 
Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject

Voortgangsoverleg 3

Royal HaskoningDHV Amersfoort

Agenda

- Opening
- Besprekingsverslag vorig overleg
- Voortgang project (inhoudelijke punten)
- Operationeel visualiseren
- Marketing
- Besluitpunten
- Rondvraag

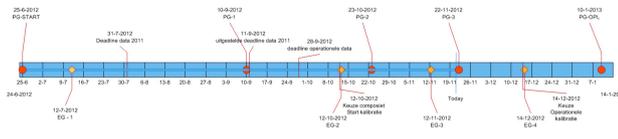
Nelen & Schuurmans 
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3
Page 2
22 november 2012

Voortgang

- Composiet
- Grondstations
- Kalibratie en validatie
- Nowcasting
- Operationalisering
- Rapportage

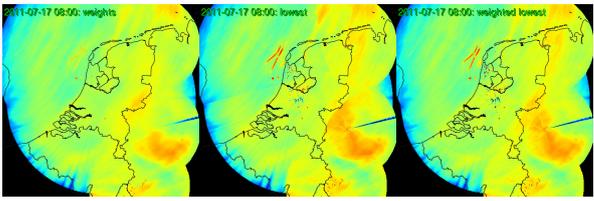
Planning Radarproject vrijdag 21 november 2012



Enhancing Society Together

Composiet

- Keuze gemaakt
- Filmpjes op website en ftp
- Nog doen: clutterverwijdering

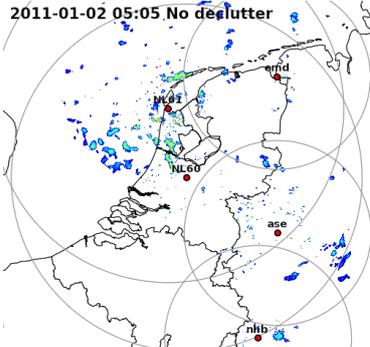


Nelen & Schuurmans 
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3
Page 4
22 november 2012

Composiet: clutter removal

2011-01-02 05:05 No declutter

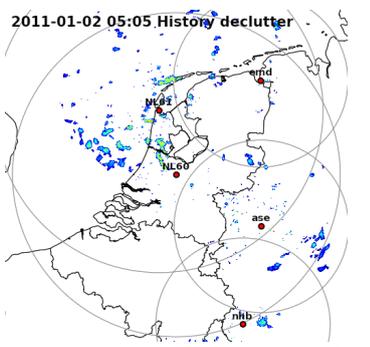


Nelen & Schuurmans 
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3
Page 5
22 november 2012

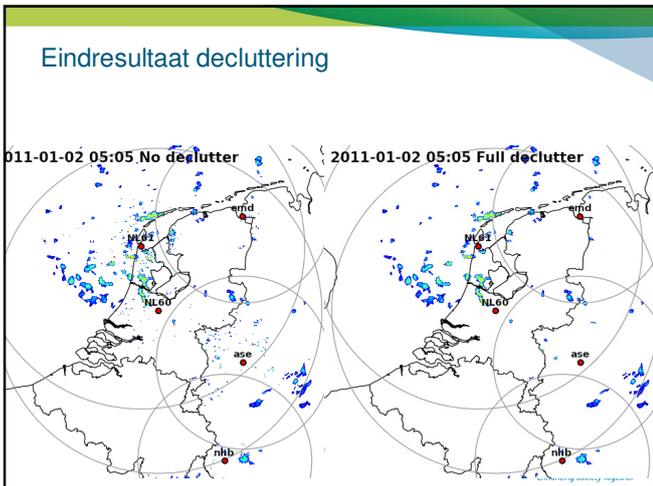
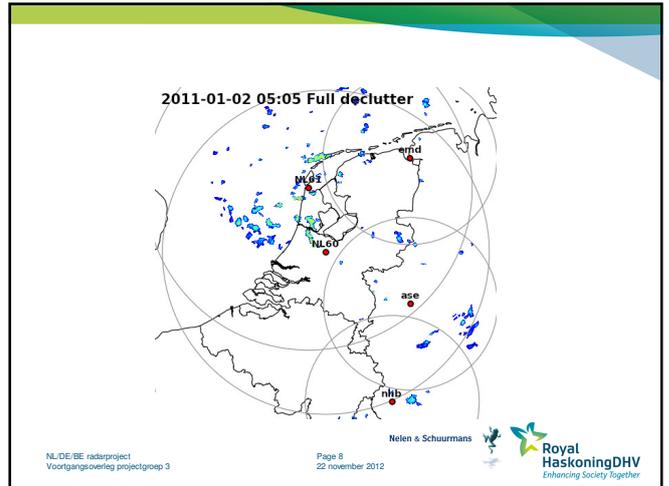
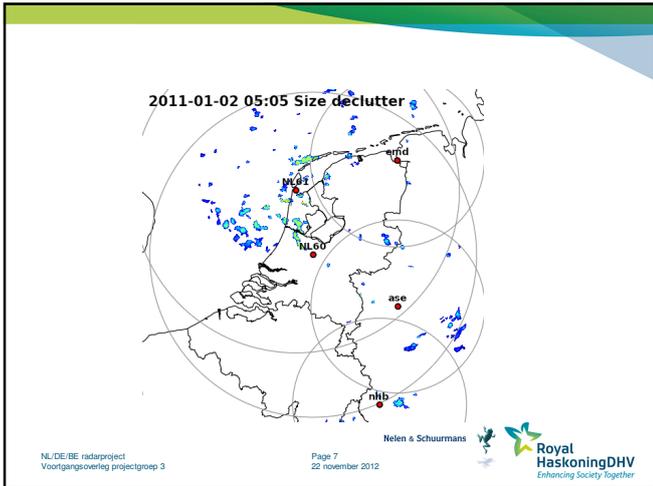
Composiet: clutter removal

2011-01-02 05:05 History declutter



Nelen & Schuurmans 
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3
Page 6
22 november 2012



Grondstations

- 5 minuten: max. 10 (Roer en Overmaas)
- 1 uur: max. 60 (KNMI + DWD + deelnemers)
- 24 uur: max. 60 + 330 KNMI (in real-time veel minder)

Deelnemer:	Metadata compleet	data operationeel op ftp?	format data	Interval	Resolutie	Eenheid	UTC	Opmerking
gemeente Groningen	ja	aanwezig, niet operationeel	CSV	5min	0.1 mm	UTC		zie paragraaf
Huize & Ad's	ja	ja	CSV	5min	1 mm	UTC		
Regge & Dinkel	ja	ja	zrx	20 min	0.1 mm cum	UTC		
Rijn & IJssel	ja	ja	zrxp	30 min	0.1 mm	UTC+1		
Roeren Overmaas	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf
Velt & Vecht	ja	ja	PI-XML & CSV	1 uur/10 min	0.1 mm	UTC		zie paragraaf
Weetterskip Fryslân	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf

- 10 minuten wordt gevoegd bij 1 uur
- Streven naar alleen beste kwaliteit-datapunten

Nelen & Schuurmans
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3

Page 10
 22 november 2012

Kalibratie en validatie

- Focus op kriging
- Resultaat zeer afhankelijk van aanwezigheid grondstations

Type	Subtype	Resolutie	Frequentie	Leveringstermijn
5 minuten beelden:	Realtime	1x1 km	5 minuten	5 minuten
	Near-real-time	1x1 km	5 minuten	60 minuten
	Achteraf	1x1 km	5 minuten	48 uur
Uursommen	Realtime	1x1 km	60 minuten	5 minuten
	Near-real-time	1x1 km	60 minuten	60 minuten
	Achteraf	1x1 km	60 minuten	48 uur
Dagsommen	Realtime	1x1 km	24 uur	5 minuten
	Near-real-time	1x1 km	24 uur	60 minuten
	Achteraf	1x1 km	24 uur	48 uur
Directe prognose (nowcasting)		1x1 km	5 minuten	5 minuten
Middellange termijn prognose 1)		≈ 10x10 km	6 uur	(doorlevering)
ECMWF-EPS		50x50 km	12 uur	(doorlevering)

Nelen & Schuurmans
 Royal HaskoningDHV
 Enhancing Society Together

NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3

Page 11
 22 november 2012

Kalibratie

Focus op kriging: deze week af

Nelen & Schuurmans
 Royal HaskoningDHV
 Enhancing Society Together

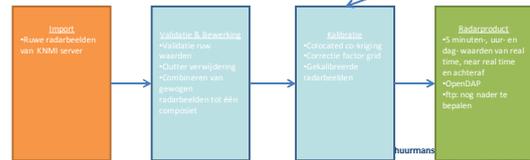
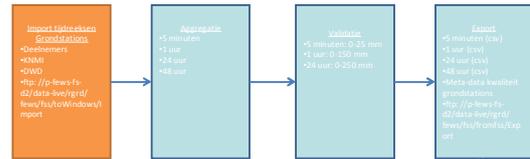
NL/DE/BE radarproject
 Voortgangsoverleg projectgroep 3

Page 12
 22 november 2012

nowcasting

- On hold

operationalisering



rapportage

- Beknopt rapport
- Source code
- Artikel(en)
 - H2O
 - Riolering
 - Het waterschap

INHOUD

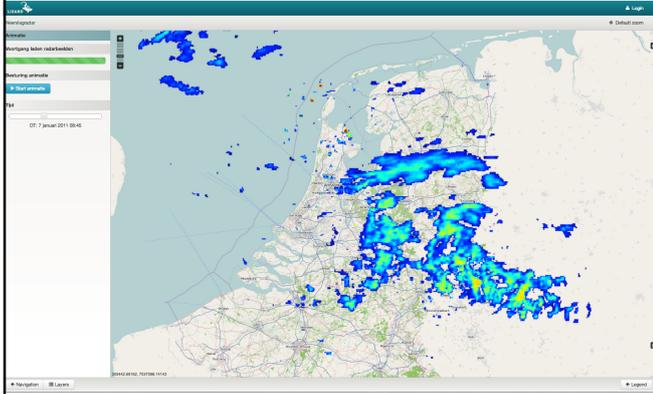
	BLAD
1 INTRODUCTIE	2
1.1 Aanleiding	2
1.2 Doel	2
1.3 Kenmerken aanpak	2
1.4 Leeswijzer	3
2 PRODUCTEN EN WERKWIJZE	4
2.1 Producten	4
2.2 Van regenradar naar neerslagbeeld	4
2.3 Wat is het beste product?	8
3 COMPOSITIE	10
3.1 Inleiding	10
3.2 Transitie	10
3.3 Compositie methoden	13
3.4 Resultaten	13
3.5 Keuze compositiemethode	14
3.6 Aanbevelingen compositie	14
4 CALIBRATIE	15
4.1 Inleiding	15
4.2 Methodiek	15
4.3 Validatie resultaten	15
4.4 Keuze calibratiemethode	16
4.5 Aanbevelingen calibratie	16
5 NOWCASTING	18
6 AANBEVELINGEN	19
7 REFERENTIES	20
8 COLOFON	21

Marketing

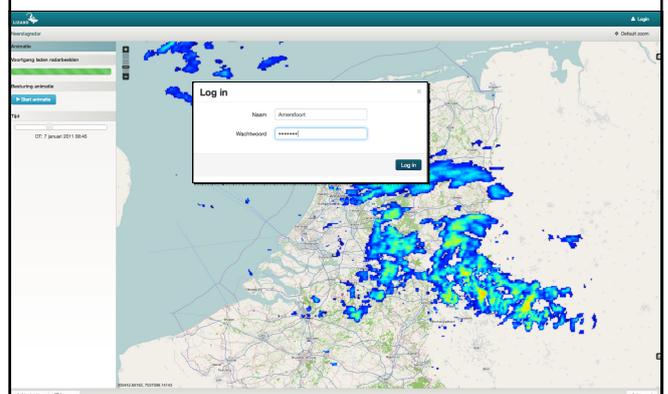
- Risico ontwikkelaars nog niet gedekt door PG
- Datakosten zijn (nog) niet gedekt door PG
- Jaarlijkse data kosten:

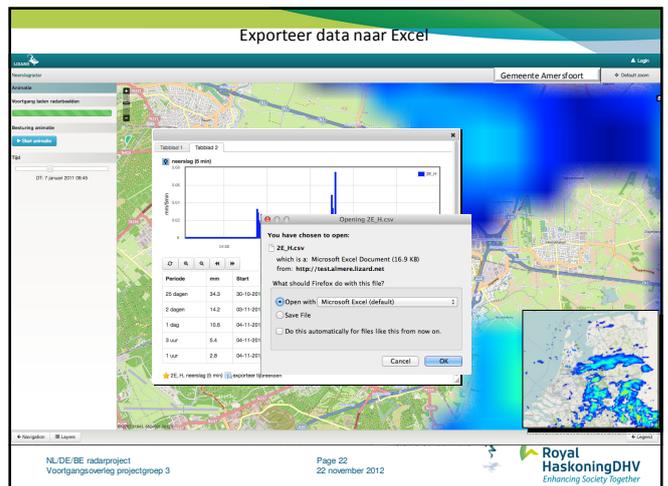
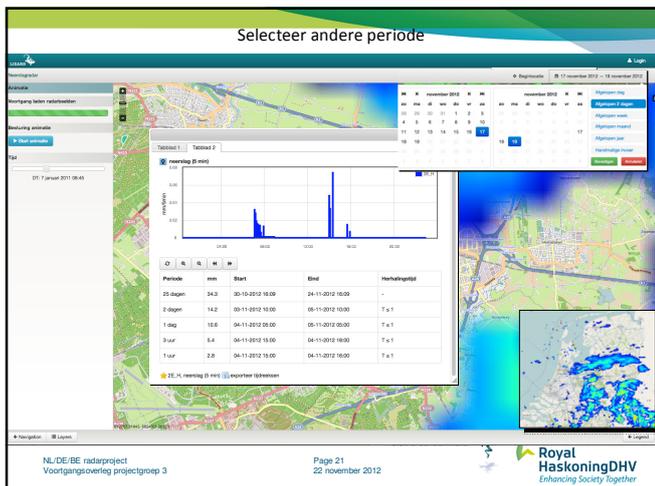
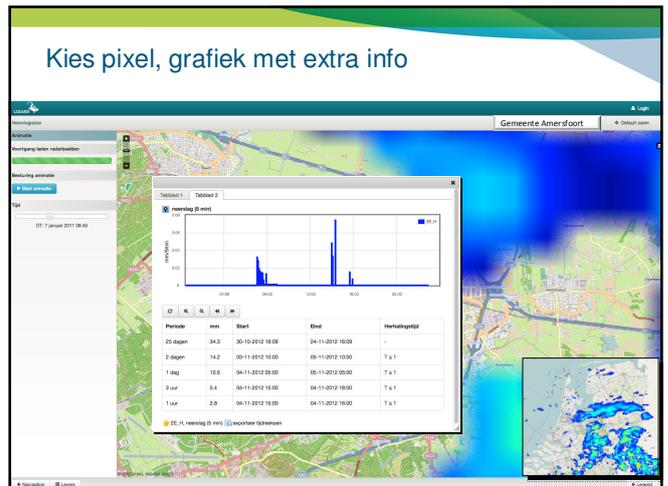
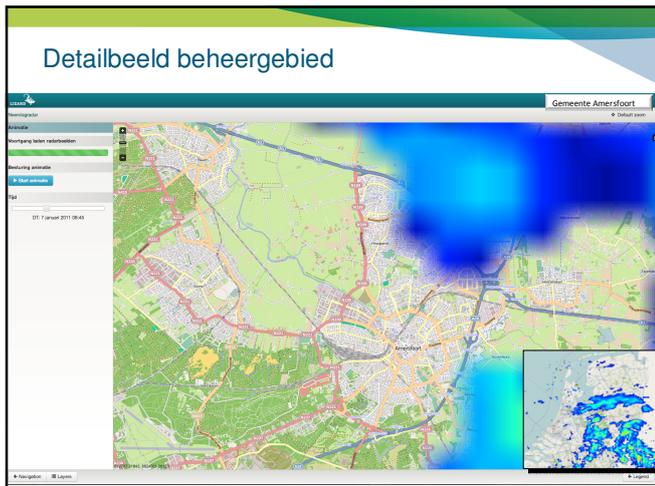
nl-rdr-data-vol			12,022
nl-obs-pcp-insitu-24h	KNMI STN		702
NL-OBS-SURF-DECODED-1H	KNMI AWS		1,058
nl/nwp/lam/grid/p5	KNMI Hirlam		1,858
nl-ecm-eps-ts-surf	KNMI ECMWF		744
nl-rdr-data-frcst-5m	KNMI nowcast		
subtotaal NL data			16,384
DWD data	3 radars + 17 stations		1,856
KMI data	Jabekke		7,200

Webinterface: standaard, animatie afgelopen uren



Extra: inloggen als gebruiker



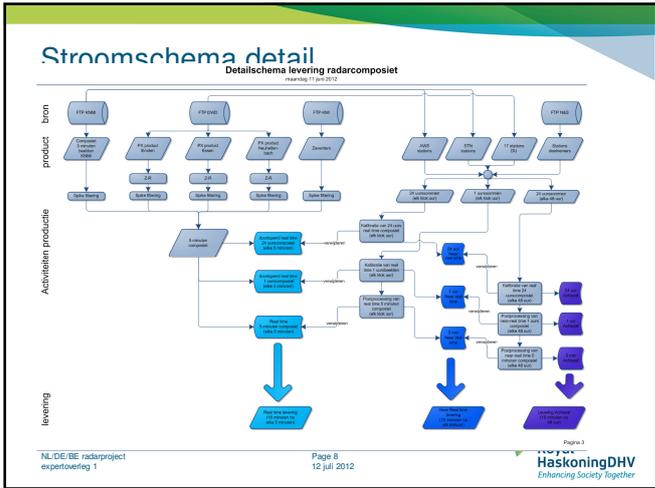
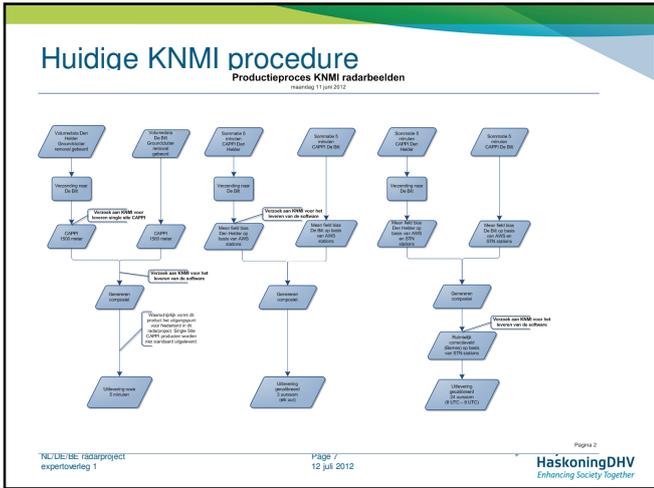


- ### Participatie
- Deelnemers radarproject
 - betalen voor ontwikkeling
 - krijgen 5 jaar datalevering
 - basis website
 - beperkte data opslag
 - Deelnemers RainApp
 - betalen per jaar (met of zonder data abonnement)
 - meer functies (deelgebied, herhalingstijd)
 - data opslag incl. back up

- ### Besluitpunten
- Ontwikkelaars toestemming om namens PG gemeenten aan te schrijven
 - Basis visualisatie

Rondvraag

BIJLAGE A.5 presentaties expertgroep



Input data; stavaza

radar data					
instituut	radars	product	aangeleverd voor 2011?	voldoende info? (metadatum/format)	opmerkingen
KNMI	De Bilt	composit/single site cappi/s	ja	ja	composit voorkeur v.
KNMI	Den Helder				
DWD	Emden	PX/DX/RX/RZ	nee	ja indien keuze RX/RZ	composit voorkeur v.
DWD	Essen				
DWD	Neuheitenbach				
KMI	Zaventem		nee	nee	moelzaam (langzaam)

stations data					
instituut	product	beschikbaar 2011?	voldoende info? (metadatum/format)	opmerkingen	
KNMI	AWS	ja	ja	AWS and STN data is v	
KNMI	STN	ja	ja	Indien data geleverd z	
DWD	17 stations	nee	ja	wordt geleverd na fiat	
deelnemers	40 stations	nee, niet volledig	nee, niet volledig	mail gestuurd naar Jan	

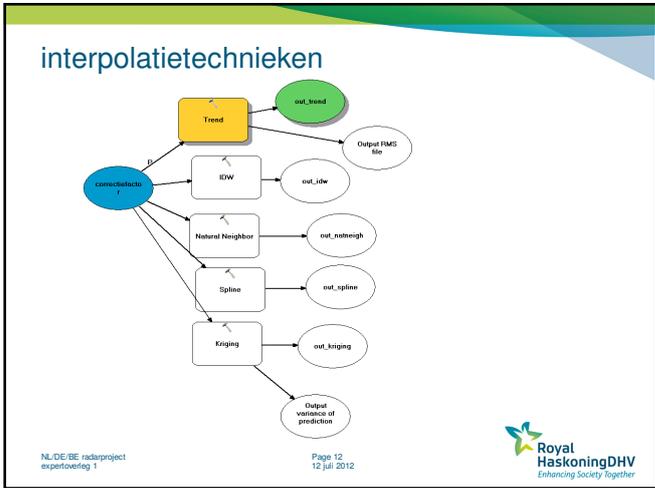
Page 9
12 juli 2012

NL/DE/BE radarproject
expertoverleg 1

HaskoningDHV
Enhancing Society Together

- ### composiet
- Input bespreken
 - Combinatie inputdata bespreken: methode KNMI?
 - Bewerkingslagen:
 - Spike filtering
 - Pixels boven xx mm/5 minuten vervangen door mediaan filter
 - Cluster filtering
 - Bui moet minimaal 4 pixels zijn
 - Mediaan filter
- Page 10
12 juli 2012
- NL/DE/BE radarproject
expertoverleg 1
- Royal HaskoningDHV
Enhancing Society Together

- ### kalibratie
- Radar + grondstations
 - Afhankelijk van product
 - Betrouwbaarheid regenmeters meenemen
 - Klasse 1, 2 of 3
 - Beste waarde veld: gewicht en invloedsveld station variëren
 - Verskil [mm] interpoleren i.p.v. verhouding [-]
 - Interpolatie methodieken
- Page 11
12 juli 2012
- NL/DE/BE radarproject
expertoverleg 1
- HaskoningDHV
Enhancing Society Together



Resultaten

- Random factoren verschil
Metingen en radar

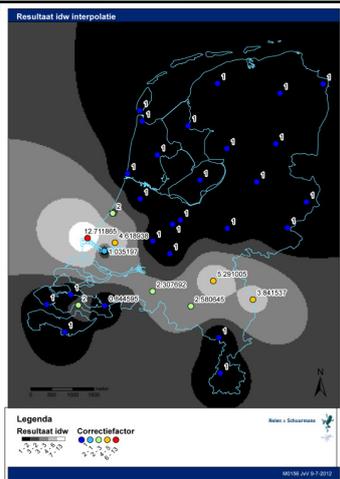
-R=2

Opvallend:

- Makkelijk toepasbaar
- Snel
- Weinig randeffecten
- Gevoelig voor outliers
- Afhankelijk van methode
- Lokale stations beperkt

invloed

NL/DE/BE radarproject
expertoverleg 1



Resultaten

- Random factoren verschil
Metingen en radar

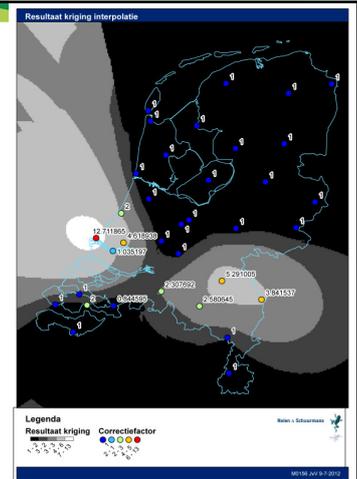
-Variogrammodel:

- Spherical
- Range = 200 km
- Sill = 0.05
- Nugget = 0.01

-Method: ordinary

Opvallend:

- Lastig tunen
- Randeffecten
- Traag
- Gevoelig voor outliers



Resultaten

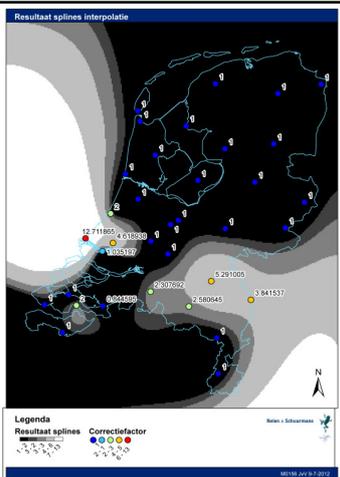
- Random factoren verschil
Metingen en radar

-Tension
-Weight: 0

Opvallend:

- Traag
- Veel randeffecten
- Resultaat afhankelijk van methode

NL/DE/BE radarproject
expertoverleg 1

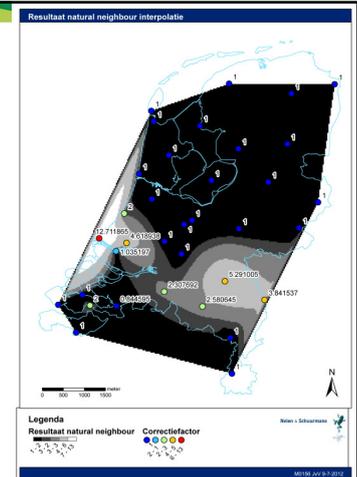


Resultaten

- Random factoren verschil
Metingen en radar

Opvallend:

- Snel
- Wat bij missing values?
- Randeffecten
- Lokale effecten beperkt invloed



Resultaten

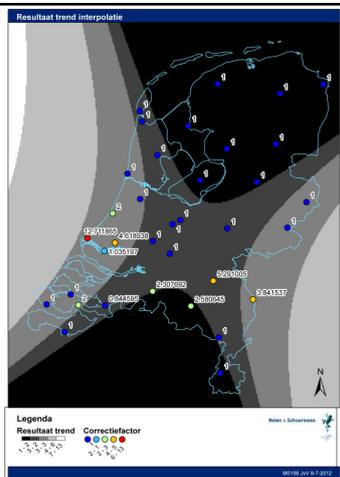
- Random factoren verschil
Metingen en radar

-Order: 2

Opvallend:

-Slechte fit

NL/DE/BE radarproject
expertoverleg 1



Tussenconclusie interpolatie

- IDW, splines en Kriging lijken meest belovend vanuit literatuur
- Niet zozeer de methode als wel de parameters van de methodieken van belang: juiste parameters bij splines en Kriging de uitdaging
- Keuze voor methodiek hangt af van:
 - Precisie
 - Snelheid
 - Reproduceerbaarheid
 - Transparantie
 - Gebruiksgemak/implementatie
 - Omgang met lokale stations
 - Omgang met ontbrekende of slechte meetwaarden

NL/DE/BE radarproject
expertoverleg 1

Page 18
12 juli 2012

Validatie: wat is goed?

4 Scoringscriteria:

- Computational time; this will not be exact number but will be either a 'yes' or 'no' on a fixed limit that is set in advance. Another suggestion is to score the methods from 1 to 3, the fastest methods gets the highest score;
- Accuracy: For each day the root mean squared error will be determined using the jack knifing method, or cross validation or....
- Maintenance: How easy/difficult is it to make changes in the method. For example the input of another radarstation, adding more rainfall stations etc. Objective criteria must be defined.
- Error sensitivity: How sensitive is a certain method for errors in source data? Objective criteria must be defined.

Validatie methoden

- Splitsing toegevoegde waarde composiet en stationsdata?
- Wat is beter?
- T.o.v. huidige KNMI product
- T.o.v. beste waarde veld: jack kniving
- Onderscheid goede methode of goed netwerk (uitbreiding netwerk meters in Groningen geen effect in Limburg)
- Jack kniving ondoenlijk voor alle 5 minuten in 2011....

nowcasting

- theoretische onderbouwing dat betere uitgangssituatie een betere nowcasting levert

procesafspraken

- Uitwisselen informatie
- Planning expertoverleg

NL/DE/BE radarproject

Expertoverleg 2

agenda

- Opening – Hanneke
- Stand van zaken
- Composietmethodiek
- Resultaat controle
- Resultaat validatie composietmethodiek
- Voorstel kalibratie

Stand van zaken

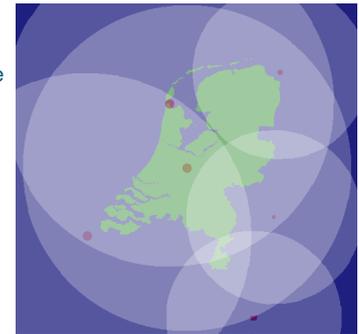
- Grondstations: 2011 en operationeel
- Radar: KNMI en DWD 2011 operationeel?
- Uitwerking composietmethodiek
- Uitwerking kalibratiemethodiek

Planning Radarproject

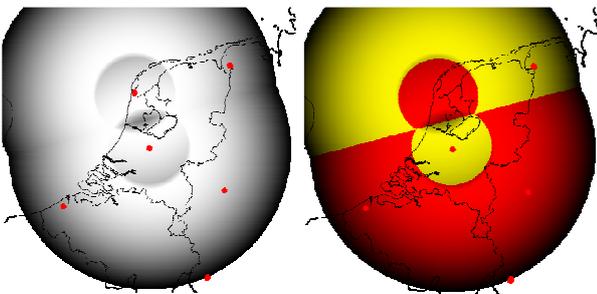
Deelnemer	Metadata compleet	data operationeel op fig?	format data	Interval	Resolutie	Eenheid	UTC	Opmerking
gemeente Groningen	ja	aanwezig, niet operationeel	CSV	5min	0.1 mm	UTC		zie paragraaf
Munro & Auk	ja	ja	CSV	5 min	1 mm	UTC		
Regge & Dinkel	ja	ja	rx	20 min	0.1 mm cum	UTC		
Rijn & Issel	ja	ja	rxp	10 min	0.1 mm	UTC+1		
Water en Overmaas	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf
WRF & Vechte	ja	ja	rx XML & CSV	5 uur/10 min	0.1 mm	UTC		zie paragraaf
Wetterskip Fryslân	ja	ja	CSV	5min	0.1 mm	UTC		zie paragraaf

Composietmethodiek

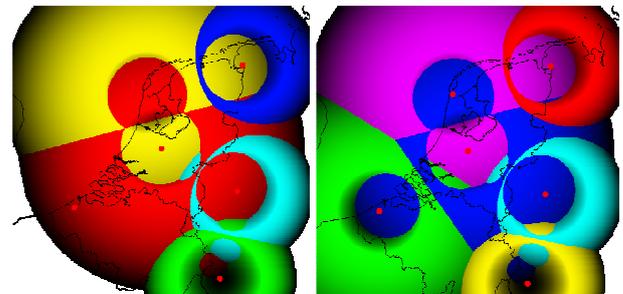
- Weging met parabolische functie
- Laagste elevatie

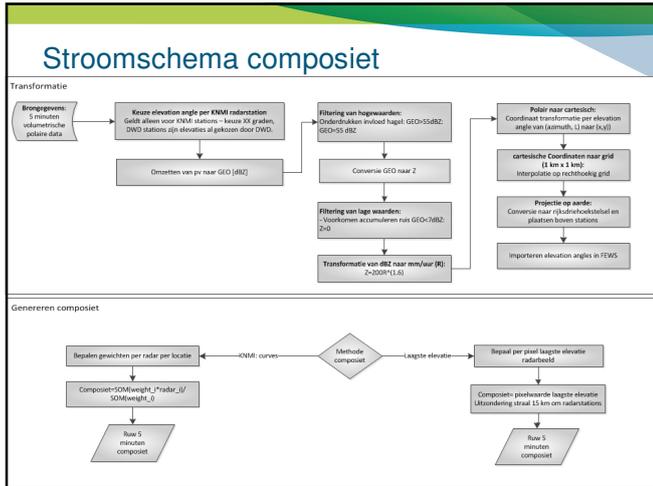


Maximale gewichten (KNMI)



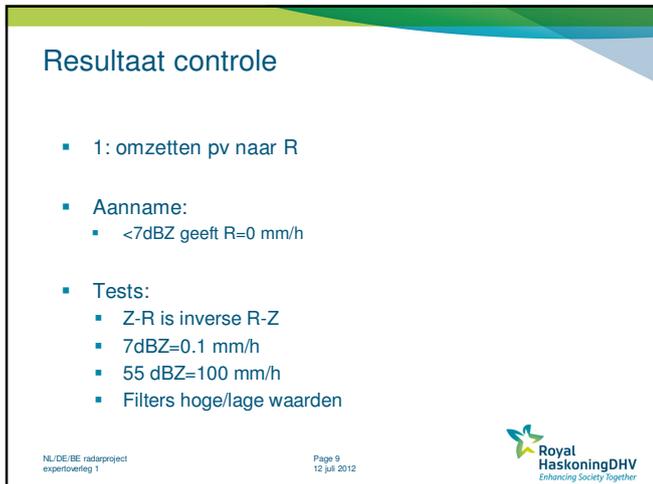
Effect toevoegen DWD en Jabbeke





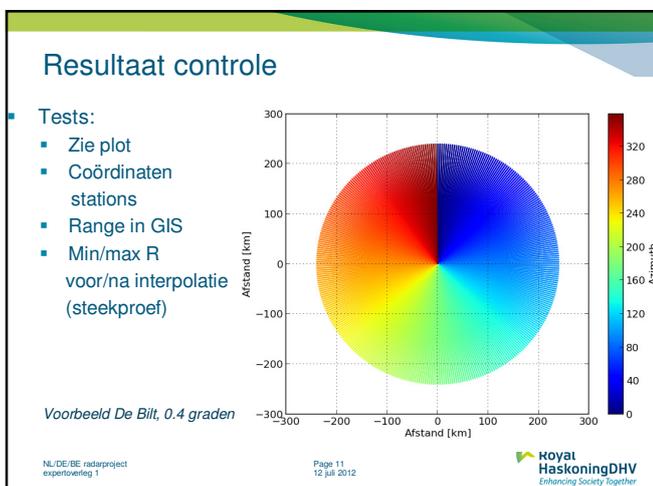
Composiet: stand van zaken

- Radarbeelden ingelezen
- Filters toegepast
- Composiet (2 methoden)
- Controle – Aannames in kaart gebracht
- Analyse resultaten methodieken
- Voorstel methode



Resultaat controle

- 2: omzetten naar RD-stelsel
- Aannames:
 - Een radar-datafile begint bij azimuth=0: Noord
 - Interpolatie naar rechthoekig grid: centrum cel krijgt waarde
 - Interpolatie polair naar rechthoekig: lineair
 - Interpolatie rechthoekig naar RD: cubic splines
- Vraag:
 - Berekening afstand komt overeen met einde volume: tekst suggereert midden: van belang voor kalibratie met grondstations



Resultaat controle

- 3: bouwen composiet
- Aannames:
 - Weging: DWD en KNMI stations hebben andere ranges (120 en 240 km); leidt tot andere wegingsformule
 - Berekening elevatie is benadering; (geeft andere hoogtes dan andere beschikbare benaderingen)
 - Laagste elevatie: als 2 stations exact dezelfde hoogte hebben, dan wordt hoogste R gekozen
 - Hoogte DWD stations meegenomen, KNMI niet
 - Laagste elevatie: afstand < 15km, laagste elevatie van overgebleven stations. Indien geen binnen bereik: no data

Resultaat controle

- Tests
 - Visuele inspectie januari 2011
 - Vergelijking met grondstations

Resultaat validatie composietmethodiek

- Visuele check – plaatjes van 2 methoden naast elkaar
- Zie folder op ftp site

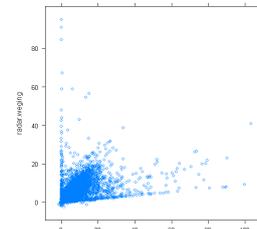
Dagelijks 426 stations

knnl	Stn	327
	Aws	35
Dwd		17
deelnemers	Groningen	1
	Hunze & Aa's	17
	Regge Dinkel	6
	Rijn en IJssel	3
	Roer & Overmaas	10
	Velt & Vecht	4
	Wetterskip Fryslan	6

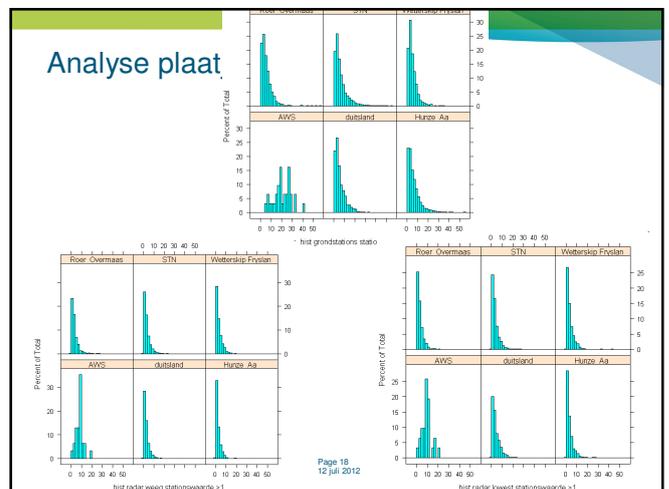
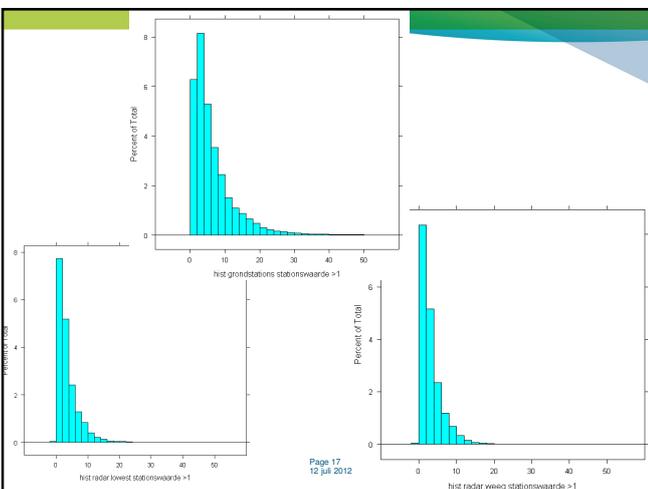
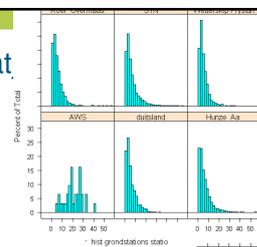
2011: 365 dagen, 426 stations = 155490 datapunten

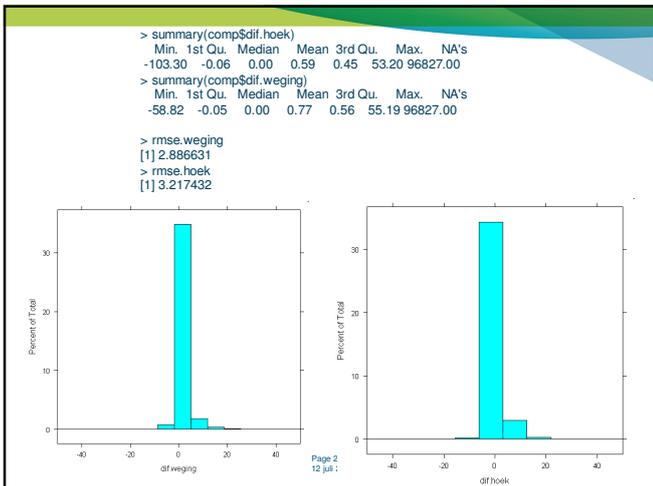
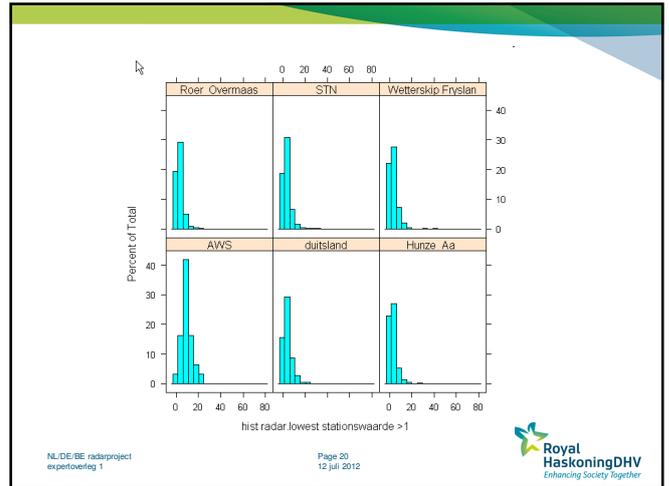
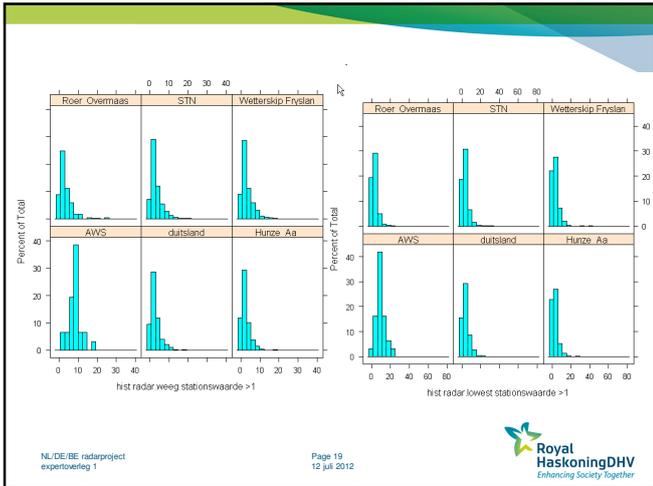
Opm. Dagwaarde NA bij station kan ook 1 ontbrekende uurwaarde zijn

	min	Q1	mediaan	Q3	max	Gem	NA
Station	0	0	0.1	2.6	96.2	2.4	43%
Radar.wegings	-2.41	0.01	0.10	1.15	94.95	1.07	33%
Radar.hoek	-1.39	0.01	0.12	1.23	103.3	1.24	33%



Analyse plaat





Voorstel kalibratiemethodiek

- Alle radarbeelden in een dag aggregeren en kalibreren
- Bepalen 'best value' per station:
 $bv = \text{Gewicht} * S + (1 - \text{gewicht}) * R$

Klasse	Gewicht
I	1.00
II	0.75
III	0.50

Page 22
12 juli 2012

Royal HaskoningDHV
Enhancing Society Together

Voorstel kalibratiemethodiek

- Co-kriging en idw
- Wat te doen met ontbrekende data?
 - Minimum aantal stations (1)
 - Ontbrekende radarbeelden (co-kriging naar ordinary kriging)
 - Hoeveel data mogen ontbreken voor aggregatie ontbreekt? (Nu: station 1 missende waarde, dag ontbreekt)

Page 23
12 juli 2012

Royal HaskoningDHV
Enhancing Society Together

Voorstel kalibratiemethodiek

- Bepalen correctiefactor radar:
 $cf = \text{radar_gecorrigeerd} / \text{radar_origineel}$
- Corrigeren 5 minuten beelden met correctiefactor

Page 24
12 juli 2012

Royal HaskoningDHV
Enhancing Society Together

Validatie

- 4 Scoringscriteria:
 - Computational time; this will not be exact number but will be either a 'yes' or 'no' on a fixed limit that is set in advance. Another suggestion is to score the methods from 1 to 3, the fastest methods gets the highest score;
 - Accuracy: For each day the root mean squared error will be determined using the jack knifing method, or cross validation or....
 - Maintenance: How easy/difficult is it to make changes in the method. For example the input of another radarstation, adding more rainfall stations etc. Objective criteria must be defined.
 - Error sensitivity: How sensitive is a certain method for errors in source data? Objective criteria must be defined.

NL/DE/BE radarproject

Expertoverleg 3
11 november 2012

agenda

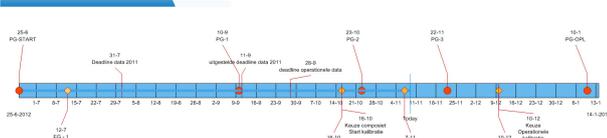
- Opening – Hanneke
- Waar staan we?
- Resultaat composietmethode
- Kalibratie en validatie
- Doorkijk naar laatste overleg (10 dec 2012)

Waar staan we?

- Grondstations: 2011 en operationeel
- Radar: KNMI en DWD op korte termijn operationeel
- Composietmethodiek klaar
- Uitwerking kalibratiemethodiek

Planning Radarproject

10dag 8 november 2012



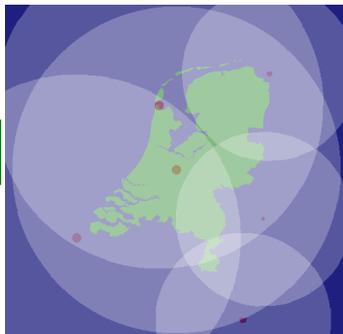
Wat gaan we leveren?

- Onderstaande, met voorlopig nowcasting on hold

Type	Subtype	Resolutie	Frequentie	Leveringstermijn
5 minuten beelden	Realtime	1x1 km	5 minuten	5 minuten
	Near-real-time	1x1 km	5 minuten	60 minuten
	Achteraf	1x1 km	5 minuten	48 uur
Uursonnen	Realtime	1x1 km	60 minuten	5 minuten
	Near-real-time	1x1 km	60 minuten	60 minuten
	Achteraf	1x1 km	60 minuten	48 uur
Dagsommen	Realtime	1x1 km	24 uur	5 minuten
	Near-real-time	1x1 km	24 uur	60 minuten
	Achteraf	1x1 km	24 uur	48 uur
Directe prognose (nowcasting)		1x1 km	5 minuten	5 minuten
Middellange termijn prognose 1)		± 10x10 km	6 uur	(doorlevering)
ECMWF-EPS		50x50 km	12 uur	(doorlevering)

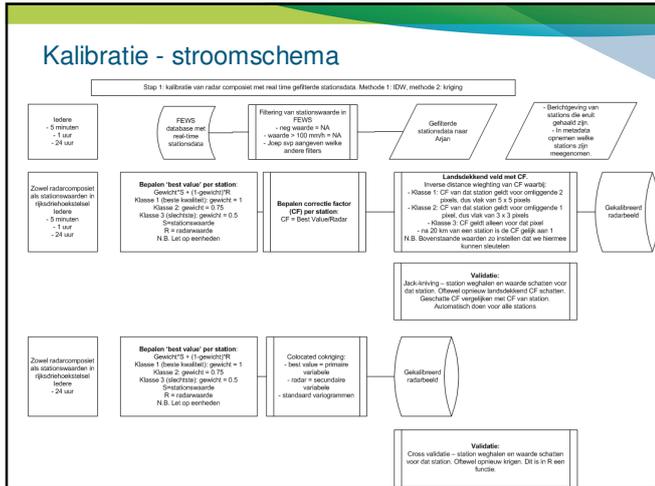
Composietmethodiek

- Weging met parabolische functie
- Laagste elevatie
- Laagste elevatie met weging aan randen



Resultaat composietmethode

- Statistieken komen in rapportage
- Visueel – filmpje januari 2011

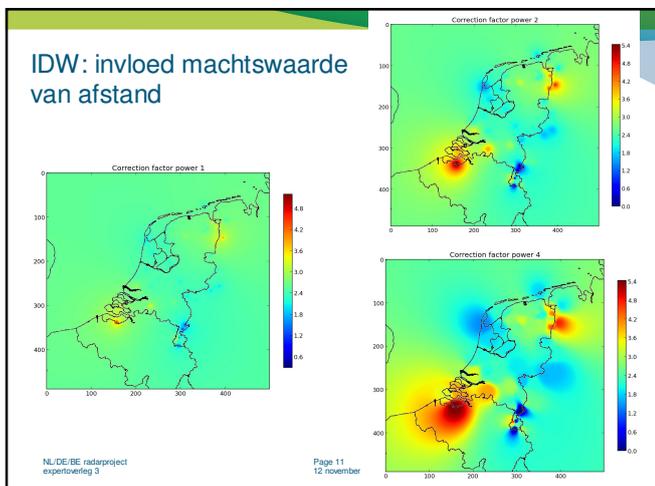
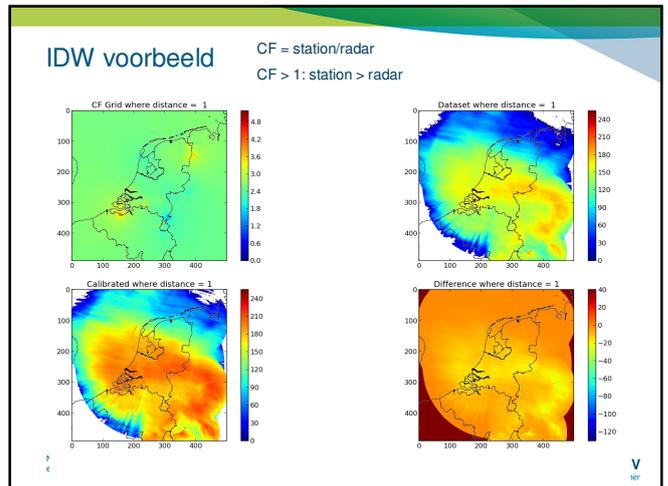
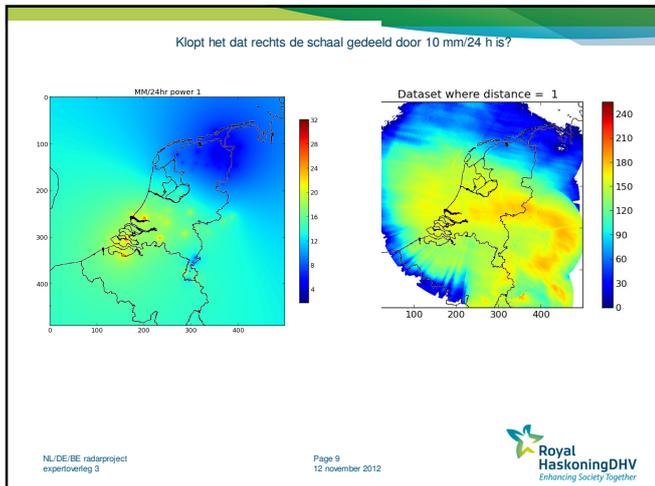


Kalibratie – validatie stations

- Validatie gebeurt op 5 min, 1 uur en 24 uur
- Gaten worden niet opgevuld
- Validatiegrenzen afhankelijk van de meetperiode
- Alle meetwaarden buiten validatiegrenzen worden niet meegenomen, maar krijgen een 'flag' voor (latere) herkenning.

Meetperiode	Ondergrens	Bovengrens
5 minuten	0 mm	25 mm
1 uur	0 mm	100 mm
24 uur	0 mm	250 mm

NLDE/BE radarproject expertoverleg 3 Page 8 12 november 2012



Hoe verder?

- IDW: correctieveld opleggen indien op xx km geen station aanwezig?
- Kriging?
- Clutter removal

NLDE/BE radarproject expertoverleg 3 Page 12 12 november 2012

Laatste overleg

- Resultaten kalibratie en validatie
- Prioriteren van de aanbevelingen

NL/DE/BE radarproject

Expertoverleg 4
14 december 2012

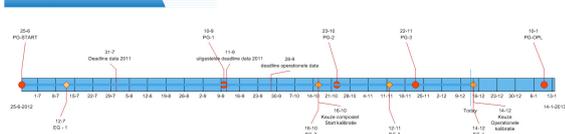
Doel overleg

- Goedkeuring
 - KNMI en Deltares kijken 17-12 naar de implementatie
- Prioritering aanbevelingen

Waar staan we?

- Laatste expertgroep overleg
- Clutter verwijderd
- Kalibratie uitgewerkt
- Operationeel product gevalideerd
- Operationeel klaar voor 9 van de 11 producten!
- Viewer voor neerslagradar ontwikkeld (alpha-release)

Planning Radarproject



Viewer



Wat gaan we leveren?

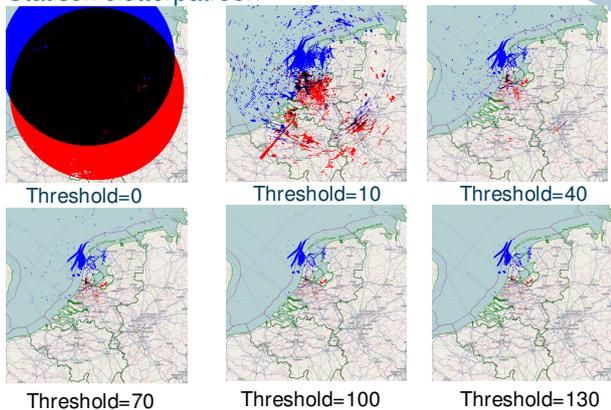
- Onderstaande

Type	Subtype	Resolutie	Frequentie	Leveringstermijn	
5 minuten beelden	Realtime	1x1 km	5 minuten	5 minuten	
	Near-real-time	1x1 km	5 minuten	60 minuten	
Uursommen	Achteraf	1x1 km	5 minuten	5 minuten	Volgende week
	Realtime	1x1 km	60 minuten	5 minuten	
Dagsommen	Near-real-time	1x1 km	60 minuten	60 minuten	Volgende week
	Achteraf	1x1 km	60 minuten	60 minuten	
Directe prognose (nowcasting)	Realtime	1x1 km	24 uur	5 minuten	
	Near-real-time	1x1 km	24 uur	60 minuten	
Middelrange termijn prognose 11	Achteraf	1x1 km	24 uur	48 uur	
	Realtime	1x1 km	5 minuten	5 minuten	
ECMWF-EPS		± 10x10 km	6 uur	(doorlevering)	
		50x50 km	12 uur	(doorlevering)	

Composietmethodiek

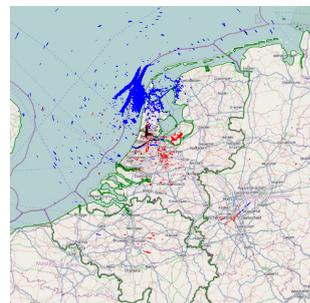
- Laagste elevatie met weging bij overlap
- Clutter verwijdering
 - Statisch: vast patroon om per pixel te kiezen voor radar met minste clutter
 - Dynamisch: per composiet clutter er uit filteren op basis van pixelgrootte

Statisch clutterpatroon

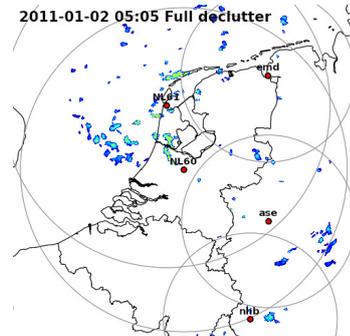
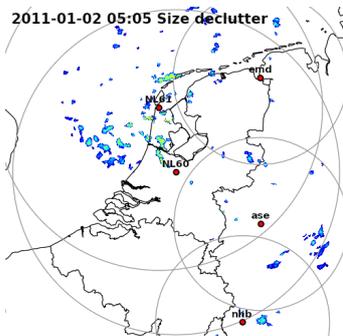
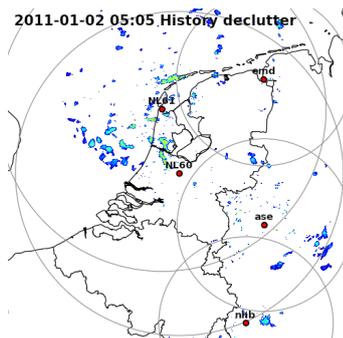
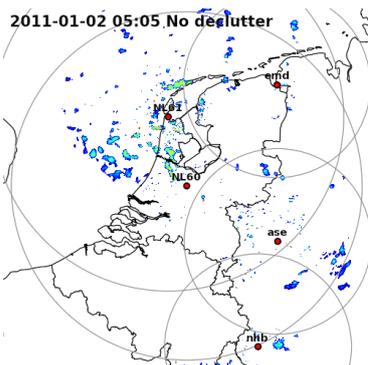


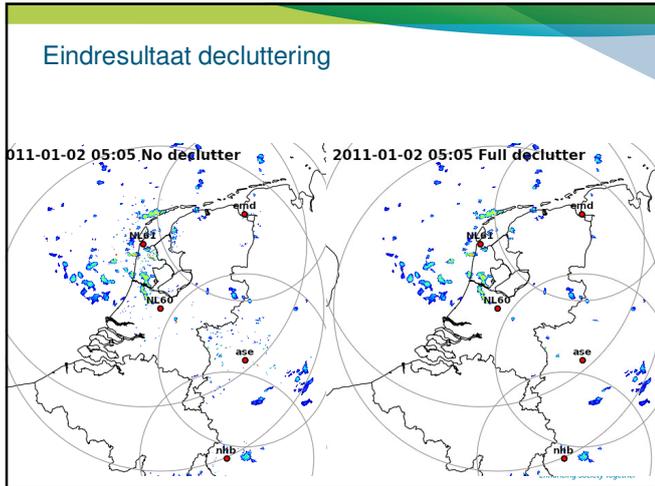
Statisch clutterpatroon

- Threshold=50



Composiet: clutter removal





Kalibratie

- Methodes:
 - Ordinary Kriging (OK)
 - Collocated Cokriging (CCK)
 - Kriging with external Drift (KED)
 - Inverse Distance Weighting (IDW)
- Uitgangspunten
 - Geen onderscheid meer tussen kwaliteit grondstations
 - OK en CCK op basis van klimatologische variogrammen

NL/DE/BE radarproject
expertoverleg 3

Page 14
12 november 2012

Kalibratie - resultaten

- Te weinig grondstations voor realtime en near-realtime, bv:
 - 5 minuten: theoretisch max. 10
 - Hunze en Aas: 5 min. data uur geladen.
 - Roer en Overmaas: de 5 min. data wordt 3 dagen later aangeleverd.
 - Velt en Vecht: Data wordt 4 uur later aangeleverd
- Grondstations wel nuttig voor historische dataset
- Status overzicht

NL/DE/BE radarproject
expertoverleg 3

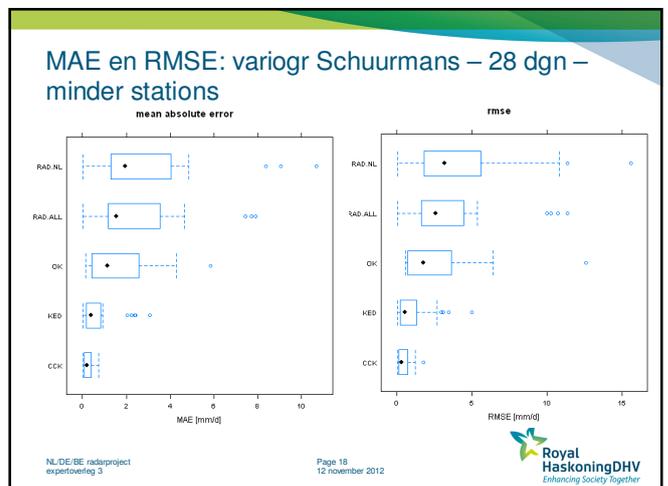
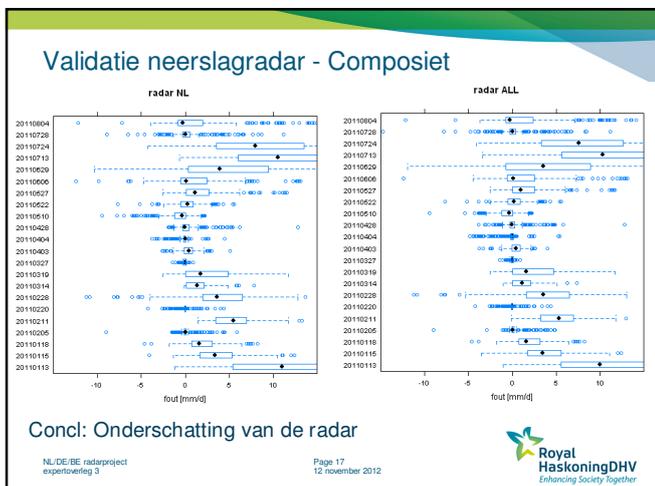
Page 15
12 november 2012

Validatie neerslagradar - Kalibratiemethodes

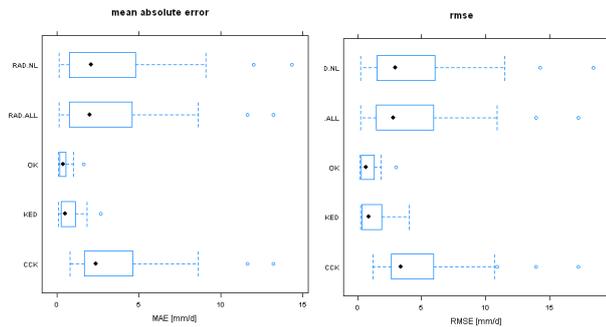
- Hard copy plaatjes

NL/DE/BE radarproject
expertoverleg 3

Page 16
12 november 2012



MAE en RMSE: CCK is niet goed gegaan – 22 dgn – veel stations



Aanbevelingen prioriteren

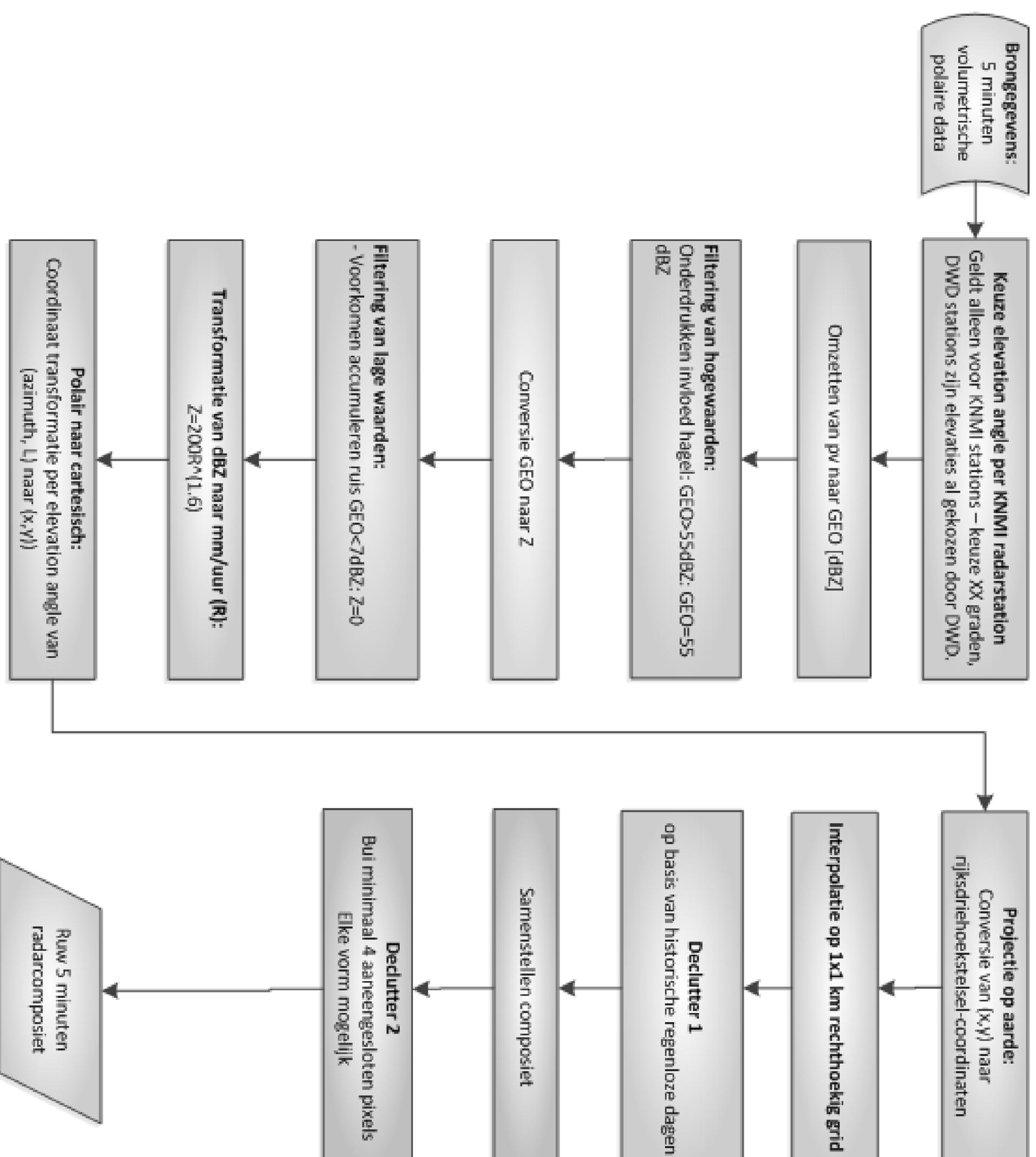
- Composiet maken
 - Beamblockage (verschillende elevaties/ HYDS expertise)
 - Dynamische clutterverwijdering op basis van meerdere frames (buiontwikkeling) ipv vaste omvang per frame
- Kalibratie obv grondstations
 - Meer grondstations, frequenter meten, frequenter opsturen
 - Opstelling grondstations controleren
 - Extra validatie op basis van verschil tussen radar en station
 - Extra validatie op basis van temporele trend
- Voorspellingen
 - Nowcasting produceren op basis van KNMI-software
 - Nowcasting verbeteren met imaging technieken
 - Blending tussen nowcasting en NWP (HYDS)

Laatste overleg

- Afsluitend etentje
- Dank voor de positieve samenwerking!

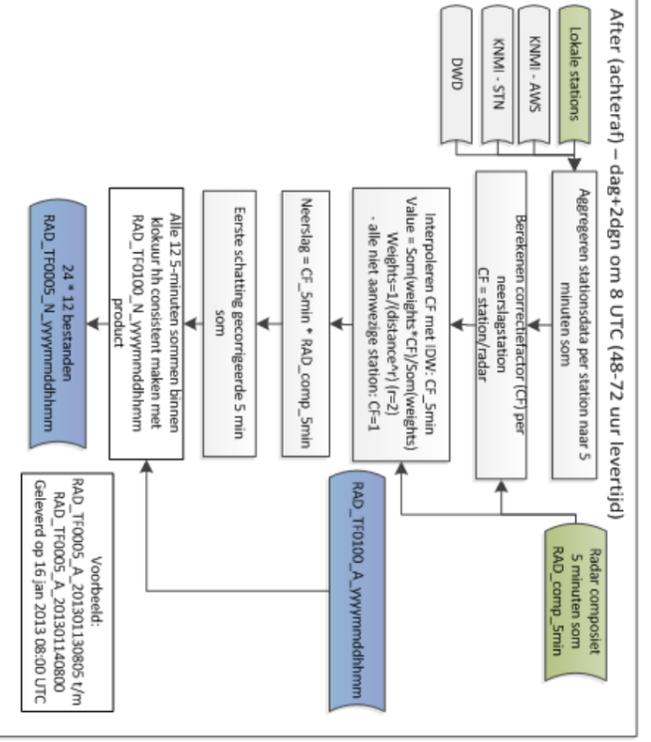
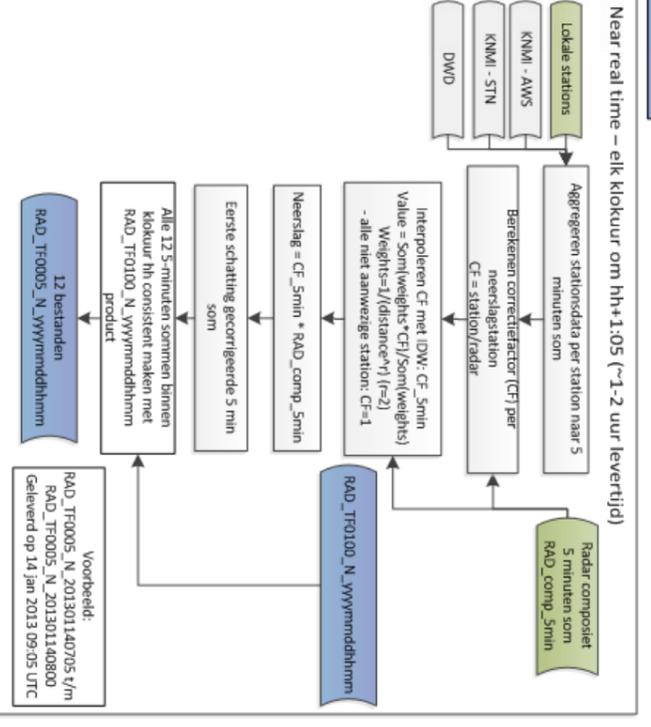
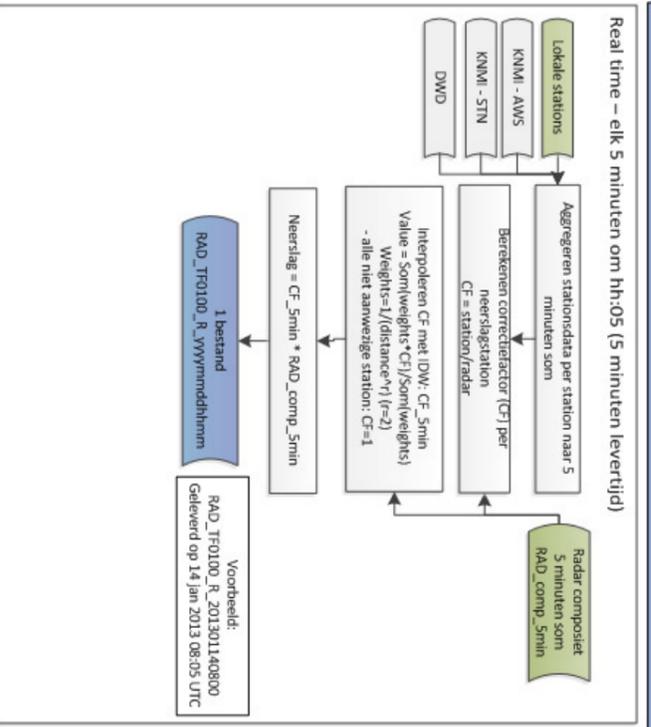
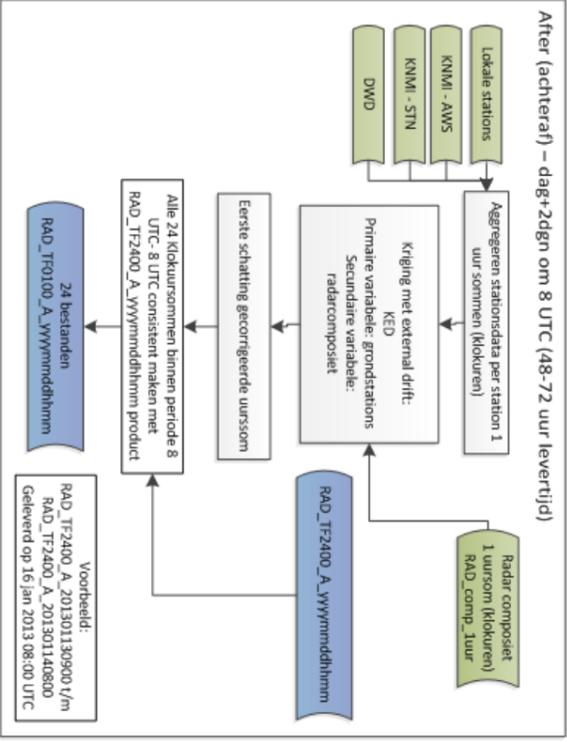
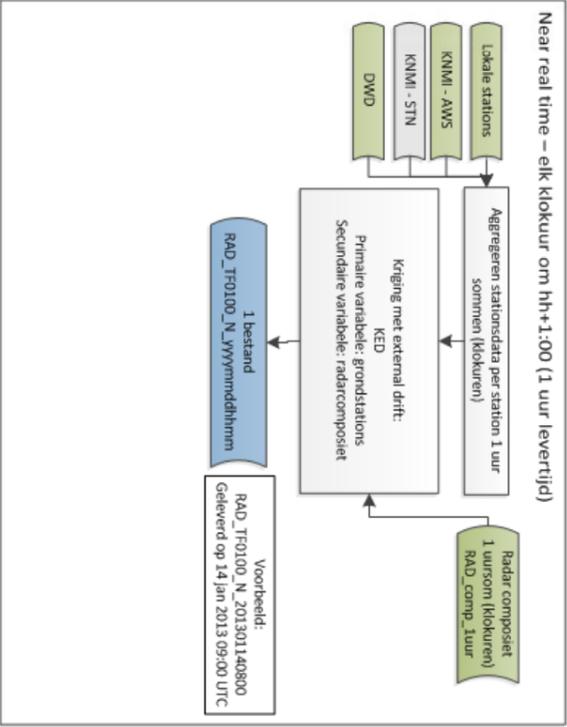
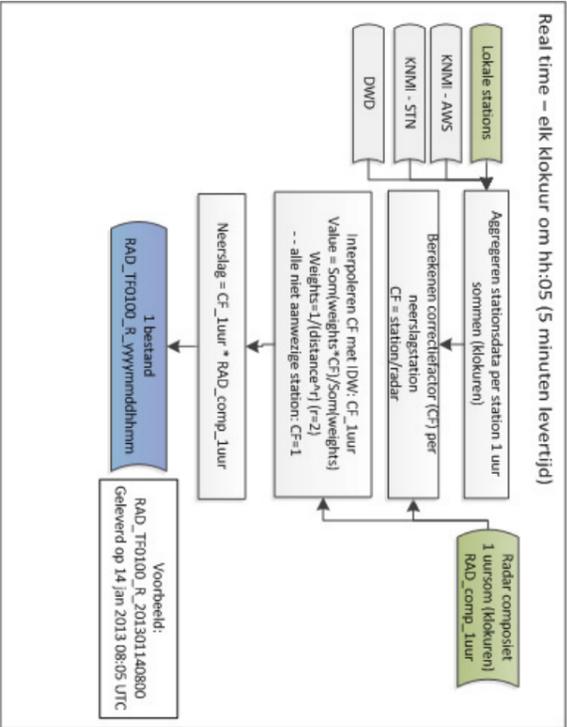
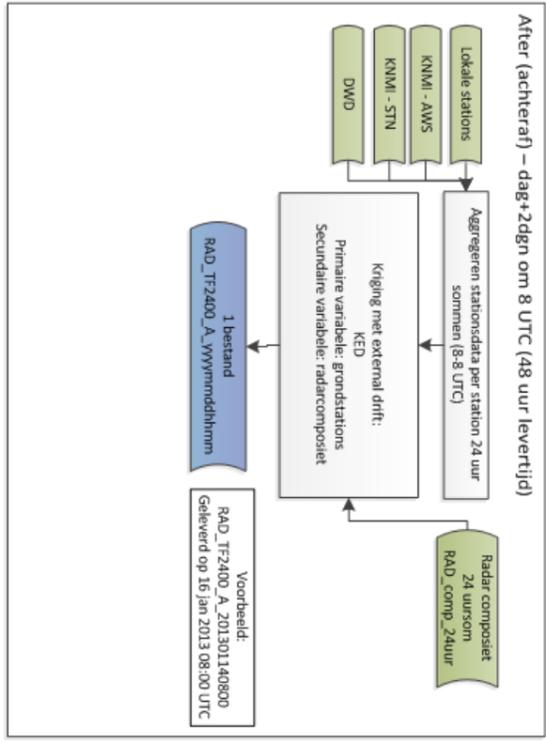
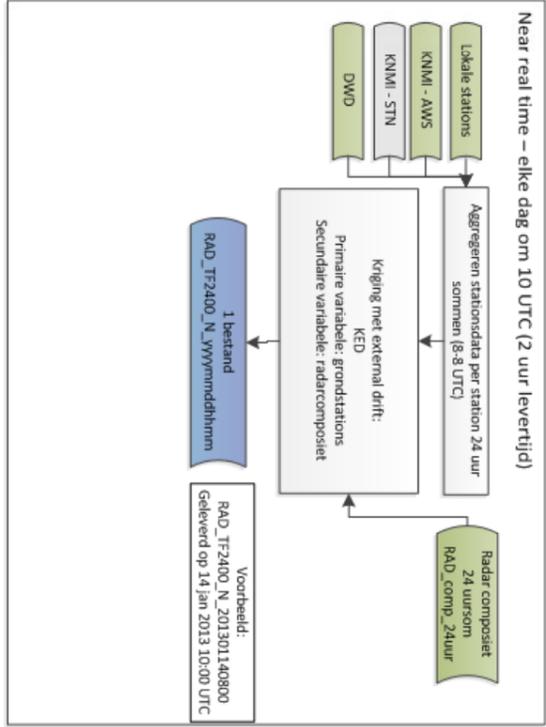
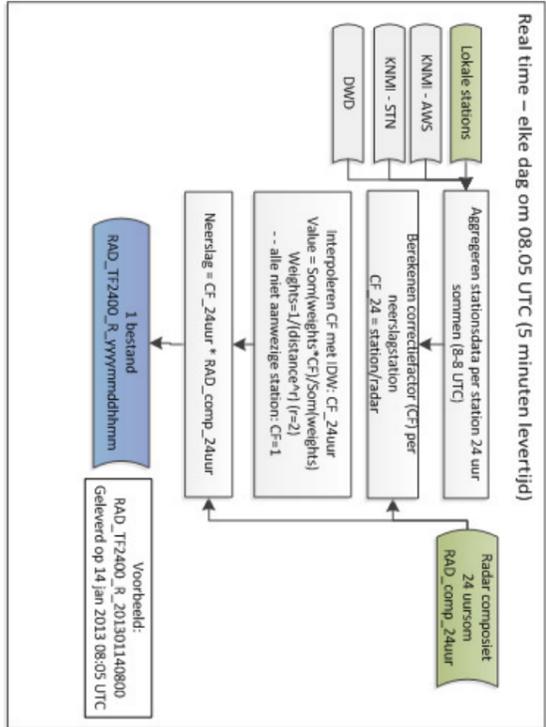
BIJLAGE B Stroomschema's A3 formaat

Genereren composiet

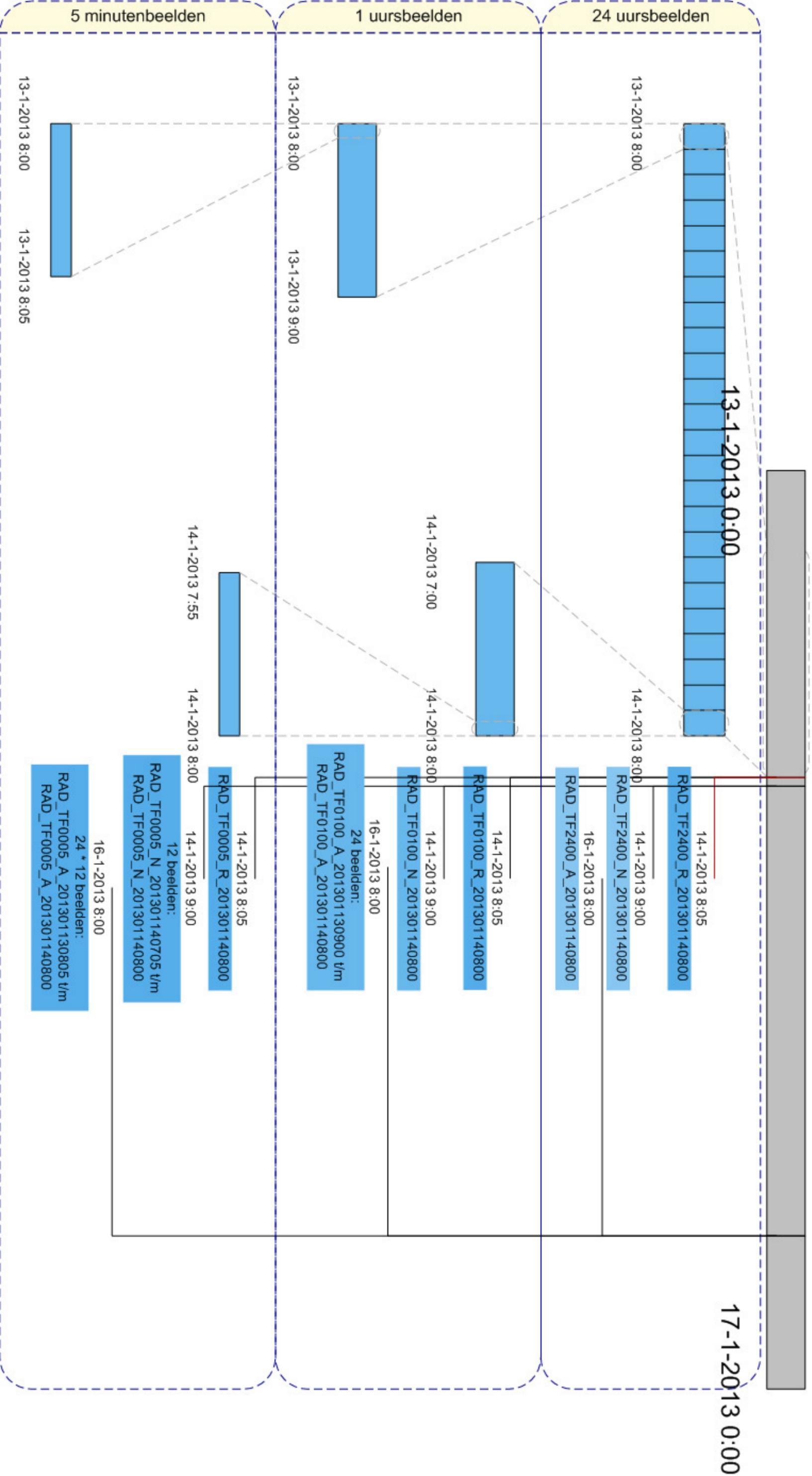


Schematisch overzicht kalibratieproces radarproducten

24 uur producten (dagsom 08 UTC – 08 UTC)



Schematisch overzicht leveringstijd radarproducten



BIJLAGE C Software code

In de volgende bijlagen zijn de volgende deelscripts weergegeven:

- master.py ; stuurt alle volgende subsripts aan
- calc.py
- config.py
- convert.py
- files.py
- gridtools.py
- images.py
- interpolation.py
- io.py
- log.py
- products.py
- scans.py
- utils.py

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans.  GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from radar import config
from radar import log
from radar import scans
from radar import utils
from radar import files
from radar import images
from radar import products

import argparse
import datetime
import logging

def master_single_product(prodcode, timeframe, datetime):
    """ Returns the created products. """
    logging.info('Creating {prodcode}, {timeframe} for {datetime}'.format(
        prodcode=prodcode, timeframe=timeframe, datetime=datetime,
    ))
    products_created = []
    if timeframe not in utils.timeframes(date=datetime, product=prodcode):
        logging.debug('Timeframe not compatible with datetime, skipping.')
        return products_created

    # Calibrated products
    calibrated_product = products.CalibratedProduct(
        product=prodcode,
        timeframe=timeframe,
        date=datetime,
    )
    calibrated_product.make()
    products_created.append(calibrated_product)

    # Geotiff for web viewer
    if timeframe == 'f' and prodcode == 'r':
        images.create_geotiff(datetime)

    # Consistent products, if possible:
    consistent_products_created = []
    for calibrated_product in products_created:
        consistent_products_created.extend(
            products.Consistifier.create_consistent_products(calibrated_pro
            duct)
        )

    # Add to products_created
    products_created += consistent_products_created
    logging.info('Created {} product'.format(len(products_created)))
    return products_created

def master_manual(args):
    """ Manual mode """
    datetimes = utils.MultiDateRange(args['range']).iterdatetimes()
    for datetime in datetimes:
        for prodcode in args['product']:
            for timeframe in args['timeframe']:
                yield dict(
                    timeframe=timeframe,
                    prodcode=prodcode,
                    datetime=datetime,
                )

```

```

def master_auto(args, dt_delivery):
    """
    auto mode; destined to run from cronjob. Makes the products
    that are possible now based on the delivery time.
    """
    delivery_time=dict(
        r=datetime.timedelta(),
        n=datetime.timedelta(hours=1),
        a=datetime.timedelta(days=2),
    )

    for prodcode in args['product']:
        for timeframe in args['timeframe']:
            yield dict(
                timeframe=timeframe,
                prodcode=prodcode,
                datetime=dt_delivery - delivery_time[prodcode]
            )

def master():
    log.setup_logging()
    args = master_args()
    logging.info('Master start')
    try:

        if args['range'] is not None:
            jobs = master_manual(args)
        else:
            dt_delivery = utils.closest_time()
            jobs = master_auto(args, dt_delivery)
            files.wait_for_files(dt_calculation=dt_delivery)

        # Organize
        sourcepath = args['source_dir']
        files.organize_from_path(sourcepath=sourcepath)

        # Create products
        products_created = []
        for job in jobs:
            products_created.extend(master_single_product(**job))

        # Publish to ftp and thredds
        products.publish(products_created)

    except Exception as e:
        logging.exception(e)
        logging.info('Master stop')

def master_args():
    parser = argparse.ArgumentParser(
        description='Get latest <timeframe> image or '
        ' get date range and process it to a calibrated image'
    )
    parser.add_argument(
        '-p', '--product',
        nargs='*',
        choices=['r', 'n', 'a'],
        default=['r', 'n', 'a'],
        help=('Choose product near-realtime, realtime etc.. Options are \n'
            'r = realtime\n'
            'n = near realtime\n'
            'a = afterwards'))
    parser.add_argument(
        '-t', '--timeframe',
        nargs='*',
        choices=['f', 'h', 'd'],
        default=['f', 'h', 'd'],
        help=('Choose time frame of the data options are: \n'
            'f = five minute (e.g. at 15.55) \n'

```

```
        'h = hourly aggregates (on whole hours e.g. 09.00 \n'  
        'd = daily aggregates (24 hours from 08.00 AM to 07.55 AM'))  
parser.add_argument(  
    '-r', '--range',  
    metavar='RANGE',  
    type=str,  
    help='Ranges to use, for example 20110101-20110103,20110105')  
parser.add_argument(  
    '-s', '--source-dir',  
    type=str,  
    default=config.SOURCE_DIR,  
    help=('Path from where all the files'  
          ' are stored that need to be organized'),  
)  
return vars(parser.parse_args())
```

```

# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.
"""
The Z and R functions should be each others inverse:

>>> round(Z(R(10.123456)), 3)
10.123

Rain values must be correct:

>>> Rain(np.array([56,55,54,7,6]).reshape(-1,1)).get()
array([[ 99.85188151],
       [ 99.85188151],
       [ 86.46816701],
       [  0.09985188],
       [  0.1         ]])

"""

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from scipy.ndimage import measurements

import logging
import numpy as np

RADIUS43 = 8495. # Effective earth radius in km.

def R(Z):
    return np.power(Z / 200., 0.625)

def Z(R):
    return 200. * np.power(R, 1.6)

def weight(r, rtop, rmax):
    """ Must this be radar range, rather than surface dist? """
    w = np.where(
        r > rtop,
        1 - np.square((r - rtop) / (rmax - rtop)),
        np.square(r / rtop),
    )
    return w

def calculate_theta(rang, elev, anth):
    """
    Return angle <radar, center-of-earth, datapoint> in radians.

    rang: distance from radar to datapoint in km
    elev: angle between beam and horizontal plane in radians
    anth: the antenna height above the earth surface in km

    horizon_height: height relative to horizon plane
    horizon_dist: distance along horizon plane
    """
    horizon_dist = rang * np.cos(elev)
    horizon_alt = anth + rang * np.sin(elev)

    return np.arctan(horizon_dist / (horizon_alt + RADIUS43))

def calculate_cartesian(theta, azim):
    """
    Return (x,y) in km along earth surface.

    All angles must be in radians.

```

```

    azimuth: clockwise from north
    theta: angle <radar, center-of-earth, datapoint>
    """
    dist = theta * RADIUS43
    return dist * np.sin(azimuth), dist * np.cos(azimuth)

def calculate_height(theta, elev, anth):
    """
    Return height of datapoint above earth surface.

    All angles must be in radians.
    anth: the antenna height above the earth surface in km
    elev: angle between beam and horizontal plane
    theta: angle <radar, center-of-earth, datapoint>
    alpha: angle <datapoint, radar, center-of-earth>
    beta: angle <center-of-earth, datapoint, radar>

    a and b are the lengths of the triangle sides opposite to angles
    alpha and beta respectively.
    """
    alpha = elev + np.pi / 2
    beta = np.pi - theta - alpha
    b = RADIUS43 + anth
    a = b * np.sin(alpha) / np.sin(beta) # Law of sines used here.

    return a - RADIUS43

class Rain(object):
    """ Do calculations from dBZ to rain """
    def __init__(self, dBZ):
        self._dBZ = dBZ.copy()

    def _clip_hail(self):
        self._dBZ[np.greater(self._dBZ, 55)] = 55

    def _make_Z(self):
        self._Z = np.power(10, self._dBZ / 10)

    def _remove_noise(self):
        self._Z[np.less(self._dBZ, 7)] = 0

    def get(self):
        self._clip_hail()
        self._make_Z()
        self._remove_noise()
        return R(self._Z)

def declutter_by_area(array, area):
    """
    Remove clusters with area less or equal to area.
    """

    # Create array
    if isinstance(array, np.ma.MaskedArray):
        nonzero = np.greater(array.filled(fill_value=0), 0.1)
    else:
        nonzero = np.greater(array, 0.1)

    logging.debug('Starting size declutter.')
    labels, count1 = measurements.label(nonzero)
    logging.debug('Found {} clusters.'.format(count1))
    areas = measurements.sum(nonzero, labels, labels)
    index = np.less_equal(areas, area)
    nonzero[index] = False
    labels, count2 = measurements.label(nonzero)
    logging.debug(
        'Removed {} clusters with area <= {}'.format(
            count1 - count2, area,

```

```
    ),  
    )  
  
    if isinstance(array, np.ma.MaskedArray):  
        array.data[index] = 0  
    else:  
        array[index] = 0
```

```

# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

import datetime
import os
import re

# Debug
DEBUG = True

# Directories
BUILDOUT_DIR = os.path.join(
    os.path.dirname(os.path.realpath(__file__)),
    '..',
)

AGGREGATE_DIR = os.path.join(BUILDOUT_DIR, 'var', 'aggregate')
CALIBRATE_DIR = os.path.join(BUILDOUT_DIR, 'var', 'calibrate')
CONSISTENT_DIR = os.path.join(BUILDOUT_DIR, 'var', 'consistent')
GROUND_DIR = os.path.join(BUILDOUT_DIR, 'var', 'ground')
IMG_DIR = os.path.join(BUILDOUT_DIR, 'var', 'img')
LOG_DIR = os.path.join(BUILDOUT_DIR, 'var', 'log')
MULTISCAN_DIR = os.path.join(BUILDOUT_DIR, 'var', 'multiscan')
RADAR_DIR = os.path.join(BUILDOUT_DIR, 'var', 'radar')
COMPOSITE_DIR = os.path.join(BUILDOUT_DIR, 'var', 'composite')
SOURCE_DIR = os.path.join(BUILDOUT_DIR, 'var', 'source')
THREDDS_DIR = os.path.join(BUILDOUT_DIR, 'var', 'thredds')

MISC_DIR = os.path.join(BUILDOUT_DIR, 'misc')
SHAPE_DIR = MISC_DIR # Backwards compatibility

JSON_DIR = os.path.join(BUILDOUT_DIR, 'json')

# Default nodatavalue
NODATAVALUE = -9999

# Declutter defaults
DECLUTTER_HISTORY = 50
DECLUTTER_SIZE = 4

# Radar codes
DWD_RADARS_2011 = ('ase', 'nhb', 'emd')
DWD_RADARS = ('ess', 'nhb', 'emd')
KNMI_RADARS = ('NL60', 'NL61')
ALL_RADARS = DWD_RADARS + KNMI_RADARS

# New DWD files have an id that corresponds to the radar code
RADAR_ID = {
    'emd': '10204',
    'ess': '10410',
    'nhb': '10605',
}

# Regex patterns
RADAR_PATTERNS = [
    # KNMI
    re.compile(
        'RAD_(?P<code>.*)_VOL_NA_(?P<timestamp>[0-9]{12})\.h5',
    ),
    # DWD 2011
    re.compile(
        'raa00-dx_(?P<code>.*)-(?P<timestamp>[0-9]{10})-dwd---bin',
    ),
    # DWD
    re.compile(
        'raa00-dx_(?P<id>.*)-(?P<timestamp>[0-9]{10})-(?P<code>.*)-bin',
    ),
]

```

```

]
GROUND_PATTERN = re.compile(
    '(?P<timestamp>[0-9]{14})_Grondstations_(?P<code>.*)\.csv',
)
CALIBRATION_PATTERN = re.compile(
    'GEO *= *(?P<a>[-.0-9]+) *\ *PV *\ * (?P<b>[-.0-9]+)',
)

# Templates that reveal datetime format when code and id are substituted
TEMPLATE_KNMI = 'RAD_{code}_VOL_NA_%Y%m%d%H%M.h5'
TEMPLATE_DWD_2011 = 'raa00-dx_{code}-%y%m%d%H%M-dwd---bin'
TEMPLATE_DWD = 'raa00-dx_{id}-%y%m%d%H%M-{code}---bin'
TEMPLATE_GROUND = '%Y%m%d%H%M%S_Grondstations_{code}.csv'

# Format for all-digit timestamp
TIMESTAMP_FORMAT = '%Y%m%d%H%M%S'

# Gridproperties for resulting composite (left, right, top, bottom)
COMPOSITE_EXTENT = (-110000, 390000, 700000, 210000)
COMPOSITE_CELLSIZE = (1000, 1000)

# DWD coordinates using standard transformation from EPSG:4314 to EPSG:4326
DWD_COORDINATES = dict(
    ase=(51.405659776445475, 6.967144448033989),
    emd=(53.33871596412482, 7.023769628293414),
    ess=(51.405659776445475, 6.967144448033989),
    nhb=(50.1097523464156, 6.548542364687092),
)

# Radar altitudes
# To be merged and stored in km.
ANTENNA_HEIGHT = dict(
    ase=185.10,
    emd=58.00,
    ess=185.10,
    nhb=585.15,
    NL60=44,
    NL61=51,
    JABBEKE=50, # Dummy!
)

# KNMI scan selection
KNMI_SCAN_NUMBER = 2
KNMI_SCAN_TYPE = 'Z'

# Naming of products and files
MULTISCAN_CODE = 'multiscan'
GROUND_CODE = {'f': '5min', 'h': 'uur', 'd': '24uur'}
TIMEFRAME_DELTA = {
    'f': datetime.timedelta(minutes=5),
    'h': datetime.timedelta(hours=1),
    'd': datetime.timedelta(days=1),
}
FRAMESTAMP = dict(f='0005', h='0100', d='2400')
PRODUCT_CODE = {t: {p: 'TF{}_{}'.format(FRAMESTAMP[t], p.upper())
                    for p in 'rna'}
                 for t in 'fhd'}
PRODUCT_TEMPLATE = 'RAD_{code}_{timestamp}.h5'

# Delays for waiting-for-files
WAIT_SLEEP_TIME = 10 # Seconds
WAIT_EXPIRE_DELTA = datetime.timedelta(minutes=3)

# Productcopy settings, for fews import, for example.
COPY_TARGET_DIRS = [
]

# Import local settings
try:
    from radar.localconfig import *
except ImportError:
    pass

```



```

#!/usr/bin/python
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans.  GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from radar import config
from radar import log
from radar import utils
from radar import gridtools

from osgeo import gdal
from osgeo import gdalconst

from scipy import interpolate

import csv
import datetime
import logging
import numpy as np
import os

log.setup_logging()

class Aggregator(object):

    KEYS = 'X', 'Y', 'NAAM', 'REGIO'

    METHODS = {
        1: 'radar weging [mm/dag]',
        2: 'radar laagste hoek [mm/dag]',
        3: 'radar gewogen hoek [mm/dag]',
    }

    CODES = {
        1: 'p.radar.m1',
        2: 'p.radar.m2',
        3: 'm3',
    }

    def __init__(self, datapath, coordspath, outputpath):

        coordsdict = {}
        with open(coordspath) as coords:
            coordsreader = csv.DictReader(coords)
            for d in coordsreader:
                coordsdict[d['ID']] = {k.lower(): d[k] for k in self.KEYS}

        self.stations = coordsdict
        self.datapath = datapath
        self.outputpath = outputpath

        self.ids, self.coords = zip(*[(k, (int(v['x']), int(v['y'])))
                                     for k, v in self.stations.items()])

    def _aggregate_radar(self, aggregatedatetime, method):

        template = 'aggregate.m{method}_{%Y%m%d%H%M%S}.{extension}'
        path = os.path.join(
            config.AGGREGATE_DIR,
            aggregatedatetime.strftime(template).format(
                method=method, extension='tif',
            ),
        )
        if os.path.exists(path):
            logging.debug('We have this one in cache: {}'.format(
                os.path.basename(path),
            ))

```

```

    ))
    return gdal.Open(path)

logging.info('doing {}'.format(aggregatedatetime))
phkwargs = dict(
    basedir=config.COMPOSITE_DIR,
    template='{code}_{timestamp}.tif',
)
ph = utils.PathHelper(code=self.CODES[method], **phkwargs)

datetime_start = aggregatedatetime - datetime.timedelta(days=1)
datetime_stop = aggregatedatetime - datetime.timedelta(minutes=5)

text = '{}-{}'.format(
    datetime_start.strftime('%Y%m%d%H%M'),
    datetime_stop.strftime('%Y%m%d%H%M'),
)

scandatetimes = utils.DateRange(text).iterdatetimes()

dataset = gdal.GetDriverByName(b'mem').CreateCopy(
    b'', gdal.Open(ph.path(
        datetime.datetime(2011,1,1),
    )),
)

rain = np.ma.zeros(gridtools.BaseGrid(dataset).get_shape())
count = np.zeros(gridtools.BaseGrid(dataset).get_shape())

for scandatetime in scandatetimes:
    logging.debug('adding {}'.format(scandatetime))
    composite = gdal.Open(ph.path(scandatetime))
    if composite is None:
        logging.warn('No composite found for method {} at
            {}'.format(
                method, scandatetime,
            ))
        continue
    ma = gridtools.ds2ma(composite)
    count += ~ma.mask # Where no mask, we count rain
    rain += ma.filled(0)

rain /= 12 # Composite unit is mm/hr, but we add every 5 minutes.

rain.mask = np.less(count, 1)

dataset.GetRasterBand(1).WriteArray(rain.filled(config.NODATAVALUE)
)

gdal.GetDriverByName(b'GTiff').CreateCopy(
    path, dataset, 0, ['COMPRESS=DEFLATE']
)

gridtools.RasterLayer(dataset,
**utils.rain_kwargs(name='jet')).save(
    path.replace('.tif', '.png'),
)

# Adding the counts as tif
count_dataset = gdal.GetDriverByName(b'gtiff').Create(
    path.replace('.tif', '_count.tif'),
    dataset.RasterXSize, dataset.RasterYSize, 1,
    gdalconst.GDT_UInt16,
)
count_dataset.GetRasterBand(1).WriteArray(count)

return dataset

def _interpolate(self, dataset):
    x_in, y_in = gridtools.BaseGrid(dataset).get_grid()

```

```

values = gridtools.ds2ma(dataset)
x_out, y_out = np.array(self.coords).transpose()

return interpolate.griddata(
    (x_in.reshape(-1), y_in.reshape(-1)),
    values.reshape(-1),
    (x_out, y_out),
    method='linear',
    fill_value=config.NODATAVALUE,
)

```

```
def main(self):
```

```

data = open(self.datapath)
output = open(self.outputpath, 'w')

outputwriter = csv.DictWriter(
    output,
    (
        'station',
        'type',
        'waarde station [mm/dag]',
        self.METHODS[1],
        self.METHODS[2],
        self.METHODS[3],
        'datum',
    )
)
outputwriter.writeheader()

datareader = csv.DictReader(data)
datareader.next() # Skip the value type row

for d in datareader:
    result = {}
    aggregatedatetime = datetime.datetime.strptime(
        d[''], '%Y-%m-%d %H:%M:%S',
    )
    logging.debug(aggregatedatetime)
    for method in self.METHODS:
        aggregate = self._aggregate_radar(
            aggregatedatetime=aggregatedatetime, method=method,
        )
        result[method] = self._interpolate(aggregate)

    for i in range(len(self.stations)):
        id = self.ids[i]
        try:
            outputwriter.writerow({
                'station': self.stations[id]['naam'],
                'type': self.stations[id]['regio']
                    .replace('"', '')
                    .replace('&', 'en'),
                'waarde station [mm/dag]': d[id],
                self.METHODS[1]: result[1][i],
                self.METHODS[2]: result[2][i],
                self.METHODS[3]: result[3][i],
                'datum': aggregatedatetime.strftime('%Y%m%d')
            })
        except KeyError as e:
            logging.error(e)

data.close()
output.close()

```

```

# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans.  GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from radar import config
from radar import scans

import datetime
import logging
import os
import shutil
import time

def organize_from_path(sourcepath):
    """ Walk basepath and move every scan in there to it's desired location
    """
    logging.info('Starting organize from {}'.format(
        sourcepath,
    ))
    count = 0

    for path, dirs, names in os.walk(sourcepath):
        for name in names:

            # Is it radar?
            try:
                scan_signature = scans.ScanSignature(scanname=name)
            except ValueError:
                scan_signature = None

            # Is it ground?
            try:
                ground_data = scans.GroundData(dataname=name)
            except ValueError:
                ground_data = None

            if scan_signature:
                target_path = scan_signature.get_scanpath()
            elif ground_data:
                target_path = ground_data.get_datapath()
            else:
                logging.debug(
                    'Could not determine target path for {}'.format(name),
                )
                continue

            source_path = os.path.join(path, name)
            if not os.path.exists(os.path.dirname(target_path)):
                os.makedirs(os.path.dirname(target_path))
            shutil.move(source_path, target_path)
            count += 1
    logging.info('Moved {} files'.format(count))

def wait_for_files(dt_calculation, td_wait=None, sleep=10):
    """
    Return if files are present or utcnow > dt_files + td_wait

    Waiting for config.ALL_RADARS and ground 5min file.
    """
    if td_wait is None:
        td_wait = config.WAIT_EXPIRE_DELTA

    logging.info('Waiting for files until {}'.format(
        dt_calculation + td_wait,
    ))

```

```
    ))

    dt_radar = dt_calculation - datetime.timedelta(minutes=5)
    dt_ground = dt_calculation

    set_expected = set()

    # Add radars to expected files.
    for radar in config.ALL_RADARS:
        set_expected.add(scans.ScanSignature(
            scancode=radar, scandatetime=dt_radar,
        ).get_scancode())
    set_expected.add(scans.GroundData(
        datacode='5min', datadatetime=dt_ground,
    ).get_dataname())

    logging.debug('looking for {}'.format(', '.join(set_expected)))

    # keep walking the source dir until all
    # files are found or the timeout expires.

    while True:
        set_arrived = set()
        for path, dirs, names in os.walk(config.SOURCE_DIR):
            set_names = set(names)
            set_arrived |= (set_names & set_expected)

            if set_arrived:
                set_expected -= set_arrived
                logging.debug('Found: {}'.format(', '.join(set_arrived)))
                if not set_expected:
                    logging.info('All required files have arrived.')
                    return True
                logging.debug('Awaiting: {}'.format(
                    ', '.join(set_expected),
                ))

            if datetime.datetime.utcnow() > dt_calculation + td_wait:
                break

            time.sleep(config.WAIT_SLEEP_TIME)

    logging.info('Timeout expired, but {} not found.'.format(
        ', '.join(set_expected),
    ))

    return False
```

```

# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from osgeo import gdal
from osgeo import gdalconst
from osgeo import ogr

from matplotlib.backends import backend_agg
from matplotlib import figure
from matplotlib import colors
from matplotlib import cm
from matplotlib import patches

from PIL import Image

import numpy as np

def ds2ma(dataset, bandnumber=1):
    """
    Return np masked array from band in gdal dataset.
    """
    band = dataset.GetRasterBand(bandnumber)
    fill_value = band.GetNoDataValue()
    array = band.ReadAsArray()
    mask = np.equal(array, fill_value)
    masked_array = np.ma.array(array, mask=mask, fill_value=fill_value)
    return masked_array

def h5ds2ma(dataset):
    """
    Return np masked array dataset.

    Expects an attribute fillvalue set on dataset.
    """
    fill_value = dataset.attrs['fill_value']
    array = dataset[:]
    mask = np.equal(array, fill_value)
    masked_array = np.ma.array(array, mask=mask, fill_value=fill_value)
    return masked_array

def default_normalize(array):
    normalize = colors.Normalize()
    return normalize(array)

class BaseGrid(object):
    """
    A grid is defined by its size, extent and projection.

    Extent is (left, right, top, bottom); cellsize is (width, height);
    projection is a wkt string.
    """
    def __init__(self, dataset=None, extent=None, size=None, projection=None):
        """
        Either use a dataset, or an extent, a cellsize and optionally
        a projection
        """
        if dataset and not (extent or size or projection):
            self._init_from_dataset(dataset)
        elif (size is not None and extent is not None) and not dataset:
            for s in size:
                if not isinstance(s, int):
                    raise TypeError('Size elements must be of type int.')

```

```

        self.size = size
        self.extent = extent
        self.projection = projection
    else:
        raise NotImplementedError('Incompatible arguments')

def _init_from_dataset(self, dataset):
    self.size = dataset.RasterXSize, dataset.RasterYSize
    self.projection = dataset.GetProjection()

    x, a, b, y, c, d = dataset.GetGeoTransform()
    self.extent = x, x + a * self.size[0], y, y + d * self.size[1]

def get_geotransform(self):
    left, right, top, bottom = self.extent
    cellwidth = (right - left) / self.size[0]
    cellheight = (top - bottom) / self.size[1]
    return left, cellwidth, 0, top, 0, -cellheight

def get_center(self):
    left, right, top, bottom = self.extent
    return (right - left) / 2, (top - bottom) / 2

def get_cellsize(self):
    left, right, top, bottom = self.extent
    cellwidth = (right - left) / self.size[0]
    cellheight = (top - bottom) / self.size[1]
    return cellwidth, cellheight

def get_shape(self):
    return self.size[::-1]

def get_grid(self):
    """
    Return x and y coordinates of cell centers.
    """
    cellwidth, cellheight = self.get_cellsize()
    left, right, top, bottom = self.extent

    xcount, ycount = self.size
    xmin = left + cellwidth / 2
    xmax = right - cellwidth / 2
    ymin = bottom + cellheight / 2
    ymax = top - cellheight / 2

    y, x = np.mgrid[
        ymax:ymin:ycount * 1j, xmin:xmax:xcount * 1j]
    return x, y

def create_dataset(self, bands=1,
                  nodatavalue=-9999, datatype=gdalconst.GDT_Float64):
    """
    Return empty in-memory dataset.

    It has our size, extent and projection.
    """
    dataset = gdal.GetDriverByName(b'MEM').Create(
        b'', self.size[0], self.size[1], bands, datatype,
    )
    dataset.SetGeoTransform(self.get_geotransform())
    dataset.SetProjection(self.projection)

    rasterbands = [dataset.GetRasterBand(i + 1) for i in range(bands)]
    for band in rasterbands:
        band.SetNoDataValue(nodatavalue)
        band.Fill(nodatavalue)

    return dataset

def create_imagelayer(self, image):
    pass

```

```

def create_vectorlayer(self):
    """ Create and return VectorLayer. """
    return VectorLayer(self)

class AbstractLayer(BaseGrid):
    """ Add imaging methods """

    def _rgba():
        raise NotImplementedError

    def _save_img(self, filepath):
        self.image().save(filepath)

    def _save_tif(self, filepath, rgba=True):
        dataset = self._rgba_dataset() if rgba else
        self._single_band_dataset()
        gdal.GetDriverByName(b'GTiff').CreateCopy(
            str(filepath), dataset, 0, ['COMPRESS=DEFLATE'],
        )

    def _save_asc(self, filepath):
        """ Save as asc file. """
        dataset = self._single_band_dataset()
        gdal.GetDriverByName(b'AAIGrid').CreateCopy(filepath, dataset)

    def _rgba_dataset(self):
        dataset = self.create_dataset(bands=4, datatype=gdalconst.GDT_Byte)
        bands = [dataset.GetRasterBand(i + 1) for i in range(4)]
        data = self._rgba().transpose(2, 0, 1)
        for band, array in zip(bands, data):
            band.WriteArray(array)
        return dataset

    def _single_band_dataset(self):
        dataset = self.create_dataset()
        band = dataset.GetRasterBand(1)
        band.WriteArray(self.ma.filled())
        band.SetNoDataValue(self.ma.fill_value)
        return dataset

    def _checker_image(self):
        pattern = (np.indices(
            self.get_shape(),
        ) // 8).sum(0) % 2
        return Image.fromarray(cm.gray_r(pattern / 2., bytes=True))

    def show(self):
        """
        Show after adding checker pattern for transparent pixels.
        """
        image = self.image()
        checker = self._checker_image()
        checker.paste(image, None, image)
        checker.show()

    def image(self):
        return Image.fromarray(self._rgba())

    def save(self, filepath, **kwargs):
        """
        Save as image file.
        """
        if filepath.endswith('.tif') or filepath.endswith('.tiff'):
            self._save_tif(filepath, **kwargs)
        elif filepath.endswith('.asc'):
            self._save_asc(filepath)
        else:
            self._save_img(filepath)

class RasterLayer(AbstractLayer):

```

```

"""
Layer containing grid data.
"""
def __init__(self, dataset=None, band=1, colormap=None, normalize=None,
              array=None, extent=None, projection=None):
    if dataset and array is None and extent is None and projection is
    None:
        rasterband = dataset.GetRasterBand(band)
        self.__init__from_dataset(dataset=dataset)
        self.__ma__from_rasterband(rasterband=rasterband)
    elif array is not None and dataset is None:
        self.size = array.shape[::-1]
        if extent is None:
            self.extent = [0, self.size[0], 0, self.size[1]]
        else:
            self.extent = extent
        self.projection = projection
        self.__ma__from_array(array=array)
    else:
        raise NotImplementedError('Incompatible arguments')

    self.normalize = normalize or default_normalize
    self.colormap = colormap or cm.gray

def __ma__from_rasterband(self, rasterband):
    """
    Store masked array and gridproperties
    """
    fill_value = rasterband.GetNoDataValue()
    array = rasterband.ReadAsArray()
    mask = np.equal(array, fill_value)
    self.ma = np.ma.array(array, mask=mask, fill_value=fill_value)

def __ma__from_array(self, array):
    self.ma = np.ma.array(
        array,
        mask=array.mask if hasattr(array, 'mask') else False,
    )

def _rgba(self):
    return self.colormap(self.normalize(self.ma), bytes=True)

class VectorLayer(AbstractLayer):
    def __init__(self, basegrid):
        """
        """
        self.projection = basegrid.projection
        self.extent = basegrid.extent
        self.size = basegrid.size
        self.__add_axes()

    def __add_axes(self):
        """
        Add matplotlib axes with coordinates setup according to geo.
        """
        dpi = 72
        figsize = tuple(c / dpi for c in self.size)
        fig = figure.Figure(figsize, dpi, facecolor='g')
        fig.patch.set_alpha(0)
        backend_agg.FigureCanvasAgg(fig)

        rect, axis = self._mpl_config()
        axes = fig.add_axes(rect, axisbg='y')
        axes.axis(axis)
        axes.autoscale(False)

        axes.patch.set_alpha(0)
        axes.get_xaxis().set_visible(False)
        axes.get_yaxis().set_visible(False)

```

```

        self.axes = axes

def _mpl_config(self):
    """
    Return rect, axis.

    To get the matplotlib axes to match exactly the geotransform
    coordinates, an appropriate combination of the axes rect and
    the axis limits is required.

    Moreover, a factor is applied to make the axes a little larger
    than the figure, because otherwise some edge artifacts may
    be visible.
    """
    factor = 0.1
    rect = (-factor, -factor, 1 + 2 * factor, 1 + 2 * factor)

    left, right, top, bottom = self.extent
    width = right - left
    height = top - bottom
    cellwidth, cellheight = self.get_cellsize()

    # For some reason, 2 pixels have to be added to
    axis = (
        left - width * factor + cellwidth * 0,
        right + width * factor + cellwidth * 1,
        bottom - height * factor + cellheight * 0,
        top + height * factor + cellheight * 1,
    )

    return rect, axis

def _rgba(self):
    canvas = self.axes.get_figure().canvas
    buf, shape = canvas.print_to_buffer()
    rgba = np.fromstring(buf, dtype=np.uint8).reshape(
        *(self.get_shape() + tuple([4]))
    )
    return rgba

def add_image(self, image_path):
    """ Add a raster image, assuming extent matches ours. """
    image = Image.open(image_path)
    self.axes.imshow(image, extent=self.extent)

def add_line(self, shapepath, *plotargs, **plotkwargs):
    """ Plot shape as matplotlib line """
    axes = self.axes
    dataset = ogr.Open(str(shapepath))
    for layer in dataset:
        for feature in layer:
            x, y = np.array(feature.geometry().GetPoints()).transpose()
            self.axes.plot(x, y, *plotargs, **plotkwargs)

def add_patch(self, shapepath, *plotargs, **plotkwargs):
    """ Plot shape as matplotlib line """
    axes = self.axes
    dataset = ogr.Open(shapepath)
    for layer in dataset:
        for feature in layer:
            xy = np.array(feature.geometry().GetBoundary().GetPoints())
            self.axes.add_patch(
                patches.Polygon(xy, *plotargs, **plotkwargs)
            )

def add_multipolygon(self, shapepath, *plotargs, **plotkwargs):
    """ Plot shape as matplotlib line """
    axes = self.axes
    dataset = ogr.Open(shapepath)
    for layer in dataset:
        for feature in layer:
            count = feature.geometry().GetGeometryCount()

```

```
polygons = [feature.geometry().GetGeometryRef(i)
             for i in range(count)]
for polygon in polygons:
    xy = np.array(polygon.GetBoundary().GetPoints())
    self.axes.add_patch(
        patches.Polygon(xy, *plotargs, **plotkwargs)
    )

#class GeoImage(AbstractLayer):
#    def __init__(self, layers):
#        self.layers = layers

#    def _merge(self, rgba1, rgba2):
#        """ Return rgba for rgba1 pasted on rgba2. """
#        img1 = Image.fromarray(rgba1)
#        img2 = Image.fromarray(rgba2)
#        img2.paste(img1, None, img1)
#        return np.array(img2)

#    def _iterrgba(self):
#        for layer in self.layers:
#            yield layer._rgba()

#    def _rgba(self):
#        return reduce(self._merge, self._iterrgba())
```

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans.  GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from radar import config
from radar import utils
from radar import gridtools
from radar import scans

from osgeo import gdal
from PIL import Image

import h5py
import logging
import numpy as np
import os
import shlex
import subprocess

def data_image(masked_array, max_rain=20):
    """ """
    basegrid = scans.BASEGRID
    rasterlayerkwargs = utils.rain_kwargs(name='jet', max_rain=max_rain)
    return gridtools.RasterLayer(
        array=masked_array,
        extent=basegrid.extent,
        projection=basegrid.projection,
        **rasterlayerkwargs).image()

def shape_image():
    """ Return rgba image with shape of country. """
    basegrid = scans.BASEGRID
    shape_layer = basegrid.create_vectorlayer()
    shape_layer.add_line(
        os.path.join(config.SHAPE_DIR, 'west_europa_lijn.shp'),
        color='k',
        linewidth=1,
    )
    return shape_layer.image()

def osm_image():
    """ Return rgba image with osm background. """
    ds_osm_rd = gdal.Open(os.path.join(config.SHAPE_DIR, 'osm-rd.tif'))
    osm_rgba = np.ones(
        (ds_osm_rd.RasterYSize, ds_osm_rd.RasterXSize, 4),
        dtype=np.uint8,
    ) * 255
    osm_rgba[:, :, 0:3] = ds_osm_rd.ReadAsArray().transpose(1, 2, 0)
    return Image.fromarray(osm_rgba)

def white_image():
    """ Return white rgba image. """
    basegrid = scans.BASEGRID
    white_rgba = np.ones(
        basegrid.size + (4,),
        dtype=np.uint8,
    ) * 255
    return Image.fromarray(white_rgba)

def create_geotiff(dt_aggregate, code='5min'):
    pathhelper = utils.PathHelper(

```

```

        basedir=config.AGGREGATE_DIR,
        code=code,
        template='{code}_{timestamp}.h5'
    )

    rasterlayerkwargs = utils.rain_kwargs(name='jet', max_rain=2)

    aggregatepath = pathhelper.path(dt_aggregate)

    tifpath_rd = os.path.join(
        config.IMG_DIR, 'geotiff', 'rd',
        dt_aggregate.strftime('%Y-%m-%d-%H-%M.tiff')
    )
    tifpath_google = os.path.join(
        config.IMG_DIR, 'geotiff', 'google',
        dt_aggregate.strftime('%Y-%m-%d-%H-%M.tiff')
    )

    with h5py.File(aggregatepath, 'r') as h5:
        array = h5['precipitation']
        mask = np.equal(array, h5.attrs['fill_value'])
        masked_array = np.ma.array(array, mask=mask)

        # Create the rd tiff.
        utils.mkdir(os.path.dirname(tifpath_rd))
        gridtools.RasterLayer(array=masked_array,
                               extent=h5.attrs['grid_extent'],
                               projection=h5.attrs['grid_projection'],
                               **rasterlayerkwargs).save(tifpath_rd, rgba=
                               True)

    logging.info('saved {}'.format(tifpath_rd))

    # Warp to google.
    utils.mkdir(os.path.dirname(tifpath_google))
    warptopts = '-t_srs EPSG:900913 -r bilinear -overwrite -co "COMPRESS=
    DEFLATE" -q'
    command = 'gdalwarp {} {} {}'.format(
        warptopts, tifpath_rd, tifpath_google,
    )
    subprocess.call(shlex.split(command))

    logging.debug('saved {}'.format(tifpath_google))

```

```

# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.
# no utf-8 encoding stuff yet. CSV library becomes a nuisance
from __future__ import print_function
from __future__ import absolute_import
from __future__ import division

import argparse
import csv
import datetime
import logging
import os
import sys

import numpy
from osgeo import gdal

from radar import config
from radar import gridtools
from radar import log
from radar import utils
from radar import scans

log.setup_logging()

class RainStation(object):
    """
    Combine information of a weather station (lat,lon and sationid), with
    a bunch of measurements.
    """
    def __init__(self, station_id, lat, lon, measurement, klasse):
        self.station_id = station_id
        self.lat = float(lat)
        self.lon = float(lon)
        self.klasse = int(klasse)
        self.measurement = measurement

class DataLoader(object):
    """
    Dataloader accepts and processes the ground station data as well as the
    radar datasets.
    The input format of the ground station data is in csv. The Dataloader
    serves as a prerequisite for the Interpolator.
    """
    def __init__(self, datafile=None, metafile=None, date=None,
                 delta=datetime.timedelta(minutes=5), scs=config.ALL_RADARS):
        try:
            self.raindata = self.read_csv(datafile.encode('utf-8'))
        except:
            logging.warn('Groundstations have not delivered data')
        self.stationsdata = self.read_csv(metafile.encode('utf-8'))
        td_aggregate = delta
        self.delta = delta
        aggregate = scans.Aggregate(
            dt_aggregate=date,
            td_aggregate=td_aggregate,
            scancodes=scs,
            declutter=None)
        aggregate.make()
        self.dataset = aggregate.get()
        sizex, sizey = self.dataset.attrs['grid_size']
        self.basegrid = gridtools.BaseGrid(
            extent=self.dataset.attrs['grid_extent'],
            projection=self.dataset.attrs['grid_projection'],
            size=(sizex, sizey)
        )
        self.date = date

    @classmethod
    def read_csv(cls, filename):
        """

```

```

    This is quite a generic method. But because csv library does not
    read
    unicode it is specifically in this interpolation.py
    """
    file_open = open(filename, 'r')
    csvdata = csv.reader(file_open, delimiter=',', quotechar='"')
    data = [i for i in csvdata]
    return data

def processdata(self, skip=None, klasse=1):
    """
    Takes apart the csv and creates instances of RainStation that can
    be
    used for the data analysis.
    """
    data = self.stationsdata
    xcol = data[0].index('X')
    ycol = data[0].index('Y')
    klassecol = data[0].index('KWALITEIT')
    idcol = data[0].index('ID')
    self.rainstations = []
    for line in data[1:]:
        measurement = self.processrain(line[0])
        if measurement != 'nothere':
            self.rainstations.append(RainStation(line[idcol],
            line[xcol], line[ycol], measurement, line[klassecol]))
    logging.info('{} gauge stations are available'.format(
        len(self.rainstations),
    ))

def processrain(self, station_id):
    """
    Masking out NaN values and giving back measurement and timestamp
    """
    timestring = '%Y-%m-%d %H:%M:%S'
    try:
        select_id = self.raindata[0].index(station_id)
        timestamps = numpy.array(self.raindata).T[0].tolist()
        rownumber =
        timestamps.index(datetime.datetime.strptime(self.date,
            timestring))
        data = float(numpy.array(self.raindata).T[select_id]
            [rownumber])
    except:
        data = 'nothere'
    return data

def stations_dummies(self):
    data = self.stationsdata
    xcol = data[0].index('X')
    ycol = data[0].index('Y')
    klassecol = data[0].index('KWALITEIT')
    idcol = data[0].index('ID')
    self.stations = []
    for line in data[1:]:
        self.stations.append(RainStation(line[idcol],
            line[xcol], line[ycol], 1.0, line[klassecol]))

class Interpolator:
    """
    This where a few Interpolation techniques can be compared.
    Takes the data from the data handler and processes the different csv's
    to an interpolated grid.

    Interpolation techniques:
    * Inverse Distance Weighting
    * Kriging (colocated cokriging, kriging external drift, and ordinary)
    * Linear Rbf
    """
    def __init__(self, dataloader):
        self.dataloader = dataloader
        precipitation = self.dataloader.dataset['precipitation'][:]
```

```

        self.precipitation = numpy.array(precipitation)

def get_rain_and_location(self):
    """
    Separates the measurements location and data. Also requests the
    classes
    of each weather stations. These are now by default set to 1.
    """
    xy = numpy.array([[i.lat,i.lon] for i in
self.dataloader.rainstations])
    z = numpy.array([i.measurement for i in
self.dataloader.rainstations])
    klasse = numpy.array([i.klasse for i in
self.dataloader.rainstations])
    mask = numpy.equal(z, -999.0)
    self.mask = mask
    z = numpy.ma.array(z, mask=mask)
    x,y = xy.T[0], xy.T[1]
    if len(x[~mask]) == 0:
        logging.debug('All stations give back -999.0 measurements'
            ' are now discarded')
    return x[~mask],y[~mask],z.data[~mask], klasse[~mask]

def get_dummies(self):
    """
    For correction field calculation dummies are needed to fill the
    gaps.
    """
    self.dataloader.stations_dummies()
    xyz = numpy.ma.array([[i.lat,i.lon,
        i.measurement] for i in self.dataloader.stations])
    return xyz.T

def get_id(self):
    """
    Get only the ids
    """
    station_id = numpy.array(
        [i.station_id for i in self.dataloader.rainstations])
    return station_id[~self.mask]

def get_correction_factor(self):
    """
    Make a correction factor grid based on the input defined in
    the radar pixels and the weatherstations.
    Returns a numpy array with Correction Factors.
    """
    countrymask = utils.get_countrymask()
    x,y,z, klasse = self.get_rain_and_location()
    radar = self.get_radar_for_locations()
    correction_factor = z / radar
    # correction_factor cannot be infinite, but this can occur by
    # zero division
    correction_factor[correction_factor==numpy.inf] = 1.0
    # correction_factor should not be larger than 10 or negative
    correction_factor[correction_factor > 10] = 1.0
    correction_factor[correction_factor < 0.0] = 1.0
    stationsx,stationsy,stationsz = self.get_dummies()
    for i in range(len(z)):
        stationsz[stationsx==x[i]] = correction_factor[i]
    self.create_interpolation_grid()
    correction_factor_grid =
self.get_idwgrid(stationsx,stationsy,stationsz)
    return correction_factor_grid

def get_calibrated(self):
    """
    Make a correction factor grid based on the input defined in
    the radar pixels and the weatherstations.
    Returns a numpy array with Correction Factors.
    """
    radar_pixels = numpy.array(self.get_radar_for_locations())

```

```

    x,y,z, klasse = self.get_rain_and_location()
    self.create_interpolation_grid()
    krige_grid = self.ked(x,y,z)
    return krige_grid

def get_radar_for_locations(self, rasterdata=None, size=2):
    """
    Radar "pixel" values for location closest to weather station.
    Returns those pixels that are closest to the rain stations
    """
    basegrid = self.dataloader.basegrid
    if rasterdata != None:
        rasterband = rasterdata.T
    else:
        rasterband = self.precipitation.T
    xy = numpy.array(self.get_rain_and_location()[0:2]).T
    geotransform = basegrid.get_geotransform()
    origx = geotransform[0]
    origy = geotransform[3]
    pixelwidth = geotransform[1]
    pixelheight = geotransform[5]
    radar_pixels = []
    for i in range(len(xy)):
        xoff = int((xy[i][0] - origx) / pixelwidth)
        yoff = int((xy[i][1] - origy) / pixelheight)
        size = size #how many surrounding pixels you want to include
        data = rasterband[xoff:xoff + size, yoff:yoff + size]
        radar_pixels.append(numpy.median(data))
    return radar_pixels

def create_interpolation_grid(self):
    """
    Run this to get some basic stuff, like an empty grid to
    interpolated
    the data to, The extent and size of the tifs.
    """
    basegrid = self.dataloader.basegrid
    self.nx, self.ny = basegrid.size
    self.xi, self.yi = [numpy.float32(array).flatten()
                        for array in basegrid.get_grid()]

def get_idwgrid(self, x,y, z, p=2):
    """
    This function returns a idwgrid with inputs x,y location and z
    as the to be interpolated value
    """
    nx,ny = self.nx, self.ny
    xi,yi = self.xi, self.yi
    grid = self.simple_idw(x,y,z,xi,yi, p)
    grid = grid.reshape((ny, nx))
    return grid

def get_linear_rbf(self, x,y, z):
    """
    This function returns a idwgrid with inputs x,y location and z
    as the to be interpolated value
    """
    nx,ny = self.nx, self.ny
    xi,yi = self.xi, self.yi
    grid = self.linear_rbf(x,y,z,xi,yi)
    grid = grid.reshape((ny, nx))
    return grid

def simple_idw(self, x, y, z, xi, yi, p):
    """
    Simple idw function
    """
    dist = self.distance_matrix(x,y, xi,yi)
    # In IDW, weights are 1 / distance
    weights = 1.0 / dist**(p)
    # Make weights sum to one

```

```

        weights /= weights.sum(axis=0)
        # Multiply the weights for each interpolated point by all observed
        Z-values
        zi = numpy.ma.dot(weights.T, z)
        return zi

def linear_rbf(self, x, y, z, xi, yi):
    """
    Linear Rbf interpolation.
    http://en.wikipedia.org/wiki/Radial_basis_function
    """
    dist = self.distance_matrix(x,y, xi,yi)
    # Mutual pairwise distances between observations
    internal_dist = self.distance_matrix(x,y, x,y)
    # Now solve for the weights such that mistfit at the observations
    is minimized
    weights = numpy.linalg.solve(internal_dist, z)
    # Multiply the weights for each interpolated point by the distances
    zi = numpy.ma.dot(dist.T, weights)
    return zi

def correction(self,ab):
    inhours = self.dataloader.delta.total_seconds()/60/60
    a,b=ab
    return a * inhours ** b

def kriging_in_r(self, x, y, z):
    """
    Cokriging (and ordinary kriging) is quite fast in R.
    This would anyway be more pragmatic than rewriting/porting it to
    Python.
    For the moment this will be the 'best' way as R makes it very easy
    to
    use kriging without fitting a variogram model, but using a standard
    variogram.
    """
    self.create_interpolation_grid()
    import rpy2
    import rpy2.robjects as robj
    robj.r.library('gstat')
    self.create_interpolation_grid()
    xi, yi = robj.FloatVector(self.xi.tolist()),
    robj.FloatVector(self.yi.tolist())
    dataset = self.precipitation.flatten()
    mask = numpy.equal(dataset, -9999)
    rxi = robj.FloatVector(dataset.tolist())
    radar = self.get_radar_for_locations()
    radar = robj.FloatVector(radar)
    x,y,z = robj.FloatVector(x), robj.FloatVector(y),
    robj.FloatVector(z)
    rain_frame = robj.DataFrame({'x': x, 'y': y, 'z':z})
    radar_frame = robj.DataFrame({'x': xi, 'y': yi, 'radar': rxi})
    target_frame = robj.DataFrame({'x':xi, 'y':yi})
    doy = self.dataloader.date.timetuple().tm_yday
    x_sill = [0.84, -0.25]
    a_sill = [0.20, -0.37]
    t0sill = [162, -0.03]
    x_range = [15.51, 0.09]
    a_range = [2.06, -0.12]
    t0range = [7.37, 0.22]
    sill_ = (self.correction(x_sill)+self.correction(a_sill)) *
    numpy.cos(
        2 * numpy.pi * 1/365*(doy-self.correction(t0sill)))**4
    range_ = (self.correction(x_range)+self.correction(a_range)) *
    numpy.cos(
        2 * numpy.pi * 1/365*(doy-self.correction(t0range)))
        **4
    vgm_args = {
        'nugget':0,
        'model_type': 'Exp',
        'sill':sill_,
        'range': range_,
    }

```

```

    }
    v = robj.r.vgm(vgm_args['sill'], vgm_args['model_type'],
    vgm_args['range'],
    vgm_args['nugget'])
    krige = robj.r('NULL')
    krige = robj.r.gstat(krige, "rain", robj.r('z ~ 1'), robj.r('~ x +
    y'),
    data=rain_frame, model=v, nmax=40)
    result = robj.r.predict(krige, target_frame)
    kriged_est = numpy.array(result[2])
    kriged_est = kriged_est.reshape((self.ny, self.nx))
    return kriged_est

def cokriging_in_r(self, x, y, z):
    """
    Cokriging (and ordinary kriging) is quite fast in R.
    This would anyway be more pragmatic than rewriting/porting it to
    Python.
    For the moment this will be the 'best' way as R makes it very easy
    to
    use kriging without fitting a variogram model, but using a standard
    variogram.
    """
    import rpy2
    import rpy2.robjects as robj
    robj.r.library('gstat')
    self.create_interpolation_grid()
    xi, yi = robj.FloatVector(self.xi.tolist()),
    robj.FloatVector(self.yi.tolist())
    dataset = self.precipitation.flatten()
    mask = numpy.equal(dataset, -9999)
    rxi = robj.FloatVector(dataset.tolist())
    radar = self.get_radar_for_locations()
    radar = robj.FloatVector(radar)
    x,y,z = robj.FloatVector(x), robj.FloatVector(y),
    robj.FloatVector(z)
    rain_frame = robj.DataFrame({'x': x, 'y': y, 'z':z})
    radar_frame = robj.DataFrame({'x': xi, 'y': yi, 'radar': rxi})
    target_frame = robj.DataFrame({'x':xi, 'y':yi})
    radar_variance = dataset[~mask].var()
    doy = self.dataloader.date.timetuple().tm_yday
    x_sill = [0.84, -0.25]
    a_sill = [0.20, -0.37]
    t0sill = [162, -0.03]
    x_range = [15.51, 0.09]
    a_range = [2.06, -0.12]
    t0range = [7.37, 0.22]
    sill_ = (self.correction(x_sill)+self.correction(a_sill) *
    numpy.cos(
        2 * numpy.pi * 1/365*(doy-self.correction(t0sill))))**4
    range_ = (self.correction(x_range)+self.correction(a_range) *
    numpy.cos(
        2 * numpy.pi * 1/365*(doy-self.correction(t0range))))
    **4
    vgm_args = {
        'nugget':0,
        'model_type': 'Exp',
        'sill':sill_,
        'range': range_,
    }
    v = robj.r.vgm(vgm_args['sill'], vgm_args['model_type'],
    vgm_args['range'],
    vgm_args['nugget'])
    rain_variance = robj.r.var(z)[0]
    correlation_radar_rain = numpy.abs(robj.r.cor(z, radar))
    if str(correlation_radar_rain[0]) != 'nan':
        variance_correction = numpy.sqrt(rain_variance *
        radar_variance)
        # The cross coefficient is used in the cross variogram
        (crossgram)
        #
        cross_coef = (correlation_radar_rain * variance_correction)[0]

```

```

    # change it back to rpy strict

    # The variogram is combined. This is a bit awkward in Rpy.
    # So one way is change the args manually (see below)
    # or load variables in R before hand.
    variogram_radar = v
    variogram_rain = v
    cck = robj.r('NULL')
    cck = robj.r.gstat(cck, "rain", robj.r('z ~ 1'), robj.r('~ x +
y'),
        data=rain_frame, model=variogram_rain, nmax=40)
    cck = robj.r.gstat(cck, "radar", robj.r('radar~ 1'), robj.r('~
x + y'),
        data=radar_frame, model=variogram_radar,
        merge=robj.r("c('rain','radar')"), nmax=1)
    cck = robj.r.gstat(cck, robj.r('c("rain", "radar")'), model=v,
nmax=40)
    result = robj.r.predict(cck, target_frame)
    self.crossval_cck = robj.r('gstat.cv')(cck)
    rain_est = numpy.array(result[2])
    rain_est = rain_est.reshape((self.ny, self.nx))
else:
    rain_est = dataset.reshape((self.ny, self.nx))
return rain_est

def ked(self, x, y, z):
    """
    Kriging External Drift (or universal kriging).
    Inputs should be equally long.

    TODO: take out general R stuff that CCK and OK also use.
    """
    import rpy2
    import rpy2.robjects as robj
    robj.r.library('gstat')
    self.create_interpolation_grid()
    xi, yi = robj.FloatVector(self.xi.tolist()),
    robj.FloatVector(self.yi.tolist())
    dataset = self.precipitation.flatten()
    mask = numpy.equal(dataset, -9999)
    rxi = robj.FloatVector(dataset.tolist())
    radar = self.get_radar_for_locations()
    radar = robj.FloatVector(radar)
    x,y,z = robj.FloatVector(x), robj.FloatVector(y),
    robj.FloatVector(z)
    rain_radar_frame = robj.DataFrame({'x': x, 'y': y, 'z':z, 'radar':
radar})
    radar_frame = robj.DataFrame({'x': xi, 'y': yi, 'radar': rxi})
    target_frame = robj.DataFrame({'x':xi, 'y':yi})
    vgm_args = {'model_type': 'Sph', 'range1': 20000, 'range2':1040000}
    try:
        vgm = robj.r.variogram(robj.r("z~radar"), robj.r('~ x + y'),
            data=rain_radar_frame, cutoff=50000, width=5000)
        residual = robj.r('fit.variogram')(vgm, robj.r.vgm(1,'Sph',
25000, 1))
        ked = robj.r('NULL')
        ked = robj.r.gstat(ked, 'raingauge', robj.r("z~radar"),
            robj.r('~ x + y'),
            data=rain_radar_frame,
            model=residual, nmax=40)
        result = robj.r.predict(ked, radar_frame, nsim=0)
        rain_est = numpy.array(result[2])
        #self.crossval_ked = robj.r('gstat.cv')(ked)
    except:
        rain_est = dataset
    rain_est = rain_est.reshape((self.ny, self.nx))
    return rain_est

def plot_stations(self):
    bg = self.dataloader.basegrid
    vl = bg.create_vectorlayer()
    x,y = self.get_rain_and_location()[0:2]

```

```
stations = numpy.vstack((x,y)).T
station_ids = self.get_id()
for i in range(len(stations)):
    vl.axes.add_artist(patches.Circle(
        stations[i], 2000,
        facecolor='r', edgecolor='k', linewidth=0.4,
    ))
return vl

def distance_matrix(self, x0, y0, x1, y1):
    """
    Calculate the distance matrix
    """
    obs = numpy.float32(numpy.vstack((x0, y0))).T
    interp = numpy.float32(numpy.vstack((x1, y1))).T

    # Make a distance matrix between pairwise observations
    # Note: from <http://stackoverflow.com/questions/1871536>
    # (Yay for ufuncs!)
    d0 = numpy.subtract.outer(obs[:,0], interp[:,0])
    d1 = numpy.subtract.outer(obs[:,1], interp[:,1])

    return numpy.hypot(d0, d1)
```

```

"""
Copyright (C) 2011 Maik Heistermann, Stephan Jacobi and Thomas Pfaff 2011

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the "Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE SOFTWARE.
"""

```

```

# Below is a section copied from wradlib/io.py

```

```

#-----
#-----
# Name:          clutter
# Purpose:
#
# Authors:       Maik Heistermann, Stephan Jacobi and Thomas Pfaff
#
# Created:       26.10.2011
# Copyright:     (c) Maik Heistermann, Stephan Jacobi and Thomas Pfaff 2011
# Licence:       The MIT License
#-----
#-----

```

```

#!/usr/bin/env python

```

```

"""

```

```

Raw Data I/O
^^^^^^^^^^^^^^

```

```

Please have a look at the tutorial :doc:`tutorial_supported_formats` for an
introduction
on how to deal with different file formats.

```

```

.. autosummary::
   :nosignatures:
   :toctree: generated/

```

```

    readDX

```

```

"""

```

```

import sys
import re
import numpy as np

```

```

def unpackDX(raw):
    """function removes DWD-DX-product bit-13 zero packing"""
    # data is encoded in the first 12 bits
    data = 4095
    # the zero compression flag is bit 13
    flag = 4096

    beam = []

    ##      # naive version
    ##      # 49193 function calls in 0.772 CPU seconds
    ##      # 20234 function calls in 0.581 CPU seconds
    ##      for item in raw:
    ##          if item & flag:

```

```

##             beam.extend([0]* (item & data))
##         else:
##             beam.append(item & data)

# performance version - hopefully
# 6204 function calls in 0.149 CPU seconds

# get all compression cases
flagged = np.where(raw & flag)[0]

# if there is no zero in the whole data, we can return raw as it is
if flagged.size == 0:
    assert raw.size == 128
    return raw

# everything until the first flag is normal data
beam.extend(raw[0:flagged[0]])

# iterate over all flags except the last one
for this, next in zip(flagged[:-1],flagged[1:]):
    # create as many zeros as there are given within the flagged
    # byte's data part
    beam.extend([0]* (raw[this] & data))
    # append the data until the next flag
    beam.extend(raw[this+1:next])

# process the last flag
# add zeroes
beam.extend([0]* (raw[flagged[-1]] & data))

# add remaining data
beam.extend(raw[flagged[-1]+1:])

# return the data
return np.array(beam)

def readDX(filename):
r"""Data reader for German Weather Service DX raw radar data files
developed by Thomas Pfaff.

The algorithm basically unpacks the zeroes and returns a regular array
of
360 x 128 data values.

Parameters
-----
filename : binary file of DX raw data

Returns
-----
data : numpy array of image data [dBZ]; shape (360,128)

attributes : dictionary of attributes - currently implemented keys:

    - 'azim' - azimuths np.array of shape (360,)
    - 'elev' - elevations (1 per azimuth); np.array of shape (360,)
    - 'clutter' - clutter mask; boolean array of same shape as `data`;
      corresponds to bit 15 set in each dataset.
"""

azimuthbitmask = 2**(14-1)
databitmask = 2**(13-1) - 1
clutterflag = 2**15
dataflag = 2**13 - 1
# open the DX file in binary mode for reading
if type(filename) == file:
    f = filename
else:
    f = open(filename, 'rb')

# the static part of the DX Header is 68 bytes long

```

```

# after that a variable message part is appended, which apparently can
# become quite long. Therefore we do it the dynamic way.
staticheadlen = 68
statichead = f.read(staticheadlen)

# find MS and extract following number
msre = re.compile('MS([ 0-9]{3})')
mslen = int(msre.search(statichead).group(1))
# add to headlength and read that
headlen = staticheadlen + mslen + 1

# this is now our first header length guess
# however, some files have an additional 0x03 byte after the first one
# (older files or those from the Uni Hannover don't, newer have it, if
# the header would end after an uneven number of bytes)
#headlen = headend
f.seek(headlen)
# so we read one more byte
void = f.read(1)
# and check if this is also a 0x03 character
if void == chr(3):
    headlen = headlen + 1

# rewind the file
f.seek(0)

# read the actual header
header = f.read(headlen)

# we can interpret the rest directly as a 1-D array of 16 bit unsigned
ints
raw = np.fromfile(f, dtype='uint16')

# reading finished, close file.
f.close()

# a new ray/beam starts with bit 14 set
# careful! where always returns its results in a tuple, so in order to
get
# the indices we have to retrieve element 0 of this tuple
newazimuths = np.where( raw == azimuthbitmask )[0] ###Thomas
kontaktieren!!!!!!!!!!!!!!!!!!!!!!

# for the following calculations it is necessary to have the end of the
data
# as the last index
newazimuths = np.append(newazimuths, len(raw))

# initialize our list of rays/beams
beams = []
# initialize our list of elevations
elevs = []
# initialize our list of azimuths
azims = []

# iterate over all beams
for i in range(newazimuths.size-1):
    # unpack zeros
    beam = unpackDX(raw[newazimuths[i]+3:newazimuths[i+1]])
    # the beam may regularly only contain 128 bins, so we
    # explicitly cut that here to get a rectangular data array
    beams.append(beam[0:128])
    elevs.append((raw[newazimuths[i]+2] & databitmask)/10.)
    azims.append((raw[newazimuths[i]+1] & databitmask)/10.)

beams = np.array(beams)

attrs = {}
attrs['elev'] = np.array(elevs)
attrs['azim'] = np.array(azims)
attrs['clutter'] = (beams & clutterflag) != 0

```

```
# converting the DWD rvp6-format into dBZ data and return as numpy
array together with attributes
return (beams & dataflag) * 0.5 - 32.5, attrs
```

```

# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

import copy
import logging.config
import logging
import os

from radar import config

CONSOLE_LEVEL = 'DEBUG' if config.DEBUG else 'INFO'
LOGFILE = os.path.join(config.LOG_DIR, 'radar.log')

LOGGING = {
    'disable_existing_loggers': True,
    'version': 1,
    'formatters': {
        'verbose': {
            'format': '%(levelname)s %(asctime)s %(module)s %(message)s'
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'level': CONSOLE_LEVEL,
            'formatter': 'simple',
        },
        'file': {
            'class': 'logging.handlers.RotatingFileHandler',
            'level': 'INFO',
            'formatter': 'verbose',
            'filename': LOGFILE,
            'mode': 'a',
            'maxBytes': 10485760,
            'backupCount': 5,
        },
    },
    'loggers': {
        'root': {
            'level': 'DEBUG',
            'handlers': ['console', 'file'],
        },
        '': {
            'level': 'DEBUG',
            'handlers': ['console', 'file'],
        },
    },
}

def setup_logging(logfile=None):
    if logfile is None:
        logging_dict = LOGGING
    else:
        logging_dict = copy.deepcopy(LOGGING)
        logging_dict['handlers']['file']['filename'] = logfile

    try:
        os.makedirs(os.path.dirname(
            logging_dict['handlers']['file']['filename'],
        ))
    except OSError:

```

```
    pass # Already exists
logging.config.dictConfig(logging_dict)
```

```

#!/usr/bin/python
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans.  GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from osgeo import gdal

import argparse
import calendar
import datetime
import ftplib
import h5py
import logging
import numpy as np
import os
import shutil
import tempfile

from radar import config
from radar import log
from radar import utils
from radar import scans
from radar.interpolation import DataLoader, Interpolator

log.setup_logging()

class ThreddsFile(object):
    """
    Contains a number of products in h5 format.
    """
    REPLACE_KWARGS = dict(
        f=dict(minute=0),
        h=dict(hour=0),
        d=dict(day=1),
    )
    def __init__(self, datetime, product, timeframe, consistent=False):
        """
        Raise ValueError if date does not match with product and timeframe
        """
        # Some inits.
        if consistent:
            self.SOURCE_DIR = config.CONSISTENT_DIR
        else:
            self.SOURCE_DIR = config.CALIBRATE_DIR

        self.timeframe = timeframe
        self.product = product
        self.datetime = datetime
        self.timedelta = config.TIMEFRAME_DELTA[self.timeframe]

        self.path = utils.PathHelper(
            basedir=config.THREDDS_DIR,
            code=config.PRODUCT_CODE[self.timeframe][self.product],
            template=config.PRODUCT_TEMPLATE,
        ).path(datetime)

        self.steps = dict(
            f=12,
            h=24,
            d=calendar.monthrange(
                self.datetime.year, self.datetime.month,
            )[0],
        )[self.timeframe]

        # Check if supplied date is valid at all.
        if datetime.microsecond == 0 and datetime.second == 0:
            if datetime.minute == (datetime.minute // 5) * 5:

```

```

        if timeframe == 'f':
            return
        if datetime.minute == 0:
            if timeframe == 'h':
                return
            if datetime.hour == 8:
                if timeframe == 'd':
                    return

    raise ValueError('Datetime incompatible with timeframe.')

@classmethod
def get(cls, product):
    """
    Return instance of threddsfile where this product belongs in.
    """
    replace_kwargs = cls.REPLACE_KWARGS[product.timeframe]
    return cls(
        datetime=product.date.replace(**replace_kwargs),
        product=product.product,
        timeframe=product.timeframe,
        consistent=isinstance(product, ConsistentProduct)
    )

def _sources(self):
    """
    Return a list of paths of products which should exist to create
    this. Later, if interfaces have improved, this could as well
    become a list of product objects.
    """

    path_helper = utils.PathHelper(
        basedir=self.SOURCE_DIR,
        code=config.PRODUCT_CODE[self.timeframe][self.product],
        template=config.PRODUCT_TEMPLATE,
    )

    for i in range(self.steps):
        source_datetime = self.datetime + i * self.timedelta
        yield source_datetime, path_helper.path(source_datetime)

def _data(self):
    """ Return precipitation, available, timestamp. """
    shape = scans.BASEGRID.get_shape() + (self.steps,)
    precipitation = np.ma.array(
        np.empty(shape), mask=True, dtype=np.float32,
        fill_value=config.NODATAVALUE,
    )
    available = []
    datetimes = []

    for i, (datetime, path) in enumerate(self._sources()):
        datetimes.append(datetime)
        try:
            with h5py.File(path, 'r') as h5:
                precipitation[..., i] = h5['precipitation'][...]
                available.append(True)
                logging.debug('Read {} for ThreddsFile'.format(path))
        except IOError:
            logging.debug('{} not available.'.format(path))
            available.append(False)
    return precipitation, available, datetimes

def update(self):
    """ Create a temporary new h5 file and move it in place. """
    precipitation, available, datetimes = self._data()

    h5_temp_path = tempfile.mkstemp()[1]
    with h5py.File(h5_temp_path) as h5:

        # East
        east = scans.BASEGRID.get_grid()[0][0]

```

```

dataset = h5.create_dataset(
    'east', east.shape, east.dtype,
    compression='gzip', shuffle=True,
)
dataset[...] = east

# North
north = scans.BASEGRID.get_grid()[1][:, 0]
dataset = h5.create_dataset(
    'north', north.shape, north.dtype,
    compression='gzip', shuffle=True,
)
dataset[...] = north

# Time
reference_datetime = self.datetime.replace(
    hour=0, minute=0, second=0, microsecond=0
)
dataset = h5.create_dataset(
    'time', [len(datetimes)], np.uint32,
    compression='gzip', shuffle=True,
)
dataset.attrs['standard_name'] = b'time'
dataset.attrs['long_name'] = b'time'
dataset.attrs['calendar'] = b'gregorian'
dataset.attrs['unit'] = reference_datetime.strftime(
    'seconds since %Y-%m-%d'
)
dataset[...] = [round((d - reference_datetime).total_seconds())
                 for d in datetimes]

# Precipitation
dataset = h5.create_dataset(
    'precipitation', precipitation.shape, precipitation.dtype,
    compression='gzip', shuffle=True,
)
dataset[...] = precipitation

# Availability
dataset = h5.create_dataset(
    'available', [len(available)], np.uint8,
    compression='gzip', shuffle=True,
)
dataset[...] = available

# Dimensions
h5['precipitation'].dims.create_scale(h5['north'])
h5['precipitation'].dims.create_scale(h5['east'])
h5['precipitation'].dims.create_scale(h5['time'])

h5['precipitation'].dims[0].attach_scale(h5['north'])
h5['precipitation'].dims[1].attach_scale(h5['east'])
h5['precipitation'].dims[2].attach_scale(h5['time'])

h5['available'].dims.create_scale(h5['time'])
h5['available'].dims[0].attach_scale(h5['time'])

utils.makedirs(os.path.dirname(self.path))
shutil.move(h5_temp_path, self.path)
logging.info('Updated ThreddsFile {}'.format(self.path))

"""
Updates threddsfile with available data.
- Always create a complete file, which then replaces the old one.
- datasets: precipitation, available, timestamp.
-

- Append new data
- If there are holes in the data
"""

```

```

class CalibratedProduct(object):
    """
    Depending on the product requested the produce method will create:
    E.g. At 2012-12-18-09:05 the products that can be made at that moment
    depending on the product are:
        real-time           => 2012-12-18-09:05
        near-real-time      => 2012-12-18-08:05
        afterwards          => 2012-12-16-09:05
    """

    def __init__(self, product, timeframe, date):
        self.prodcode = product
        self.product = self.prodcode # Backwards compatible
        self.timeframe = timeframe
        self.groundfile_datetime = utils.get_groundfile_datetime(
            prodcode=product, date=date,
        )
        self.groundpath = scans.GroundData(
            datacode=config.GROUND_CODE[self.timeframe],
            datadatetime=self.groundfile_datetime,
        ).get_datapath()
        self.path = utils.PathHelper(
            basedir=config.CALIBRATE_DIR,
            code=config.PRODUCT_CODE[self.timeframe][self.product],
            template=config.PRODUCT_TEMPLATE,
        ).path(date)
        self.calibratepath = self.path # Backwards compatible
        self.date = date

    def make(self):
        td_aggregate = config.TIMEFRAME_DELTA[self.timeframe]
        if self.groundfile_datetime.year < 2012:
            groundpath = os.path.join(config.SHAPE_DIR, 'tests/2011.csv')
        dataloader = DataLoader(
            metafile=os.path.join(config.SHAPE_DIR, 'grondstations.csv'),
            datafile=self.groundpath,
            date=self.date,
            delta=td_aggregate)
        try:
            dataloader.processdata()
            stations_count = len(dataloader.rainstations)
        except:
            logging.debug("Can't process data; there is none for:
                {}".format(
                    self.groundfile_datetime))
            stations_count = 'None'
        interpolater = Interpolator(dataloader)
        try:
            mask = utils.get_countrymask()
            calibrated_radar = interpolater.get_calibrated()
            result = (mask * calibrated_radar + (1 - mask) *
                interpolater.precipitation)
            self.calibrated = result
        except:
            self.calibrated = interpolater.precipitation
            logging.warn("Taking the original radar data for date:
                {}".format(
                    self.date))
        self.metadata = dict(dataloader.dataset.attrs)
        dataloader.dataset.close()
        self.metadata.update({'stations_count': stations_count})
        calibrated_ma = np.ma.array(
            self.calibrated,
            mask=np.equal(self.calibrated, config.NODATAVALUE),
        )
        calibrate = utils.save_dataset(
            dict(precipitation=calibrated_ma),
            self.metadata,
            self.calibratepath,

```

```

    )
    logging.info('Created calibrated {}'.format(calibrate))

def get(self):
    try:
        return h5py.File(self.calibratepath, 'r')
    except IOError:
        logging.warn(
            'Creating calibrated product {}, because it did not'
            ' exist'.format(self.calibratepath))
        self.make()
    return h5py.File(self.calibratepath, 'r')

def make_cfgrid(self):
    td_aggregate = config.TIMEFRAME_DELTA[self.timeframe]
    dataloader = DataLoader(
        metafile=os.path.join(config.SHAPE_DIR, 'grondstations.csv'),
        datafile=self.groundpath,
        date=self.datadatetime,
        delta=td_aggregate)
    dataloader.processdata()
    stations_count = len(dataloader.rainstations)
    interpolator = Interpolator(dataloader)
    interpolator.get_correction_factor()

class Consistifier(object):
    """
    The products that are updated afterwards with new gaugestations need to
    be consistent with the aggregates of the same date.
    E.g. In the hour 12.00 there are 12 * 5 minute products and 1 one hour
    product. The 12 seperate 5 minutes need add up to the same amount of
    precipitation as the hourly aggregate.

    The consistent products are only necessary for 3 products:
    - 5 minute near-realtime
    - 5 minute afterwards
    - hour near-realtime

    To make the products one can initiate class and run make_consistent:
    cproduct = ConsistentProduct('n', 'f', 201212180700)
    cproduct.make_consistent()
    Products are written in config.CONSISTENT_DIR
    """
    """ Is a class purily for encapsulation purposes."""
    SUB_TIMEFRAME = {'d': 'h', 'h': 'f'}

    @classmethod
    def _reliable(cls, product):
        """
        Return if product enables consistification of other products.
        """
        prodcode, timeframe = product.prodcode, product.timeframe
        if prodcode == 'a':
            if timeframe == 'd':
                return True
            if timeframe == 'h' and isinstance(product, ConsistentProduct):
                return True
        if prodcode == 'n' and timeframe == 'h':
            return True
        return False

    @classmethod
    def _subproduct_datetimes(cls, product):
        """ Return datetimes for subproducts of product. """
        amount_of_subproducts = dict(h=12, d=24)[product.timeframe]
        sub_timeframe = cls.SUB_TIMEFRAME[product.timeframe]
        sub_timedelta = config.TIMEFRAME_DELTA[sub_timeframe]
        for i in range(amount_of_subproducts):
            yield product.date + sub_timedelta * (i -
                amount_of_subproducts)

```

```

@classmethod
def _subproducts(cls, product):
    """ Return the CalibratedProducts to be consistified with product
    """
    sub_timeframe = cls.SUB_TIMEFRAME[product.timeframe]
    for datetime in cls._subproduct_datetimes(product):
        yield CalibratedProduct(date=datetime,
                                product=product.product,
                                timeframe=sub_timeframe)

@classmethod
def _precipitation_from_product(cls, product):
    """ Return precipitation as masked array. """
    with product.get() as h5:
        data = h5['precipitation']
        mask = np.equal(data, config.NODATAVALUE)
        precipitation = np.ma.array(data, mask=mask)
    return precipitation

@classmethod
def create_consistent_products(cls, product):
    """ Returns a list of consistent products that were created. """
    # TODO Remove confusion between calibrated and consistent below
    consistified_products = []
    if cls._reliable(product):
        # Calculate sum of subproducts
        subproduct_sum = np.ma.zeros(scans.BASEGRID.get_shape())
        for subproduct in cls._subproducts(product):
            subproduct_sum = np.ma.sum([
                subproduct_sum,
                cls._precipitation_from_product(subproduct),
            ], 0)

        # Calculate factor
        factor = cls._precipitation_from_product(product) /
            subproduct_sum

        # Create consistent products
        for subproduct in cls._subproducts(product):
            consistified_products.append(
                ConsistentProduct.create(
                    product=subproduct,
                    factor=factor,
                    consistent_with = os.path.basename(subproduct.path)
                )
            )

        # Run this method on those products as well, since some
        # consistified products allow for consistent products
        # themselves,
        # For example a.d consistifies a.h which in turn consistifies
        # a.f.
        more_consistified_products = []
        for consistified_product in consistified_products:
            more_consistified_products.extend(
                cls.create_consistent_products(consistified_product)
            )
        consistified_products.extend(more_consistified_products)
    return consistified_products

class ConsistentProduct(object):
    """ Consistified products are usually created by the consistifier. """

    def __init__(self, datetime, prodcode, timeframe):
        self.datetime = datetime
        self.date = datetime # Backwards compatible
        self.prodcode = prodcode
        self.product = prodcode # Backwards compatible
        self.timeframe = timeframe

```

```

        self.path = utils.PathHelper(
            basedir=config.CONSISTENT_DIR,
            code=config.PRODUCT_CODE[self.timeframe][self.prodcode],
            template=config.PRODUCT_TEMPLATE,
        ).path(datetime)

def get():
    """
    Return h5 dataset opened in read mode.

    Crashes when the file does not exist. This should be caught by
    caller.
    """
    return h5py.File(self.path, 'r')

@classmethod
def create(cls, product, factor, consistent_with):
    """
    Return ConsistentProduct.

    Creates a ConsistentProduct from product with data multiplied
    by factor and adds consistent_with to the metadata.
    """
    # Create the consistent product object
    consistent_product = cls(
        datetime=product.date,
        prodcode=product.product,
        timeframe=product.timeframe,
    )

    # Create the h5 datafile for it
    with product.get() as h5:
        data = h5['precipitation']
        mask = np.equal(data, config.NODATAVALUE)
        data = dict(precipitation=np.ma.array(data, mask=mask) *
            factor)
        meta = dict(h5.attrs)
        meta.update(consistent_with=consistent_with)
    utils.save_dataset(
        data=data,
        meta=meta,
        path=consistent_product.path
    )

    # get() will now work, so return the object.
    filepath = consistent_product.path
    filename = os.path.basename(filepath)
    logging.info('Created consistent product {}'.format(filename))
    logging.debug('Created consistent product {}'.format(filepath))
    return consistent_product

def publish(products):
    """
    Each product is published via the ThreddsFile and the ftpserver.
    """
    # Prepare dirs
    for target_dir in config.COPY_TARGET_DIRS:
        utils.makedirs(target_dir)

    # Login to ftp
    if hasattr(config, 'FTP_HOST'):
        ftp = ftplib.FTP(config.FTP_HOST, config.FTP_USER,
            config.FTP_PASSWORD)
    else:
        ftp = utils.FakeFtp()
        logging.warning('FTP not configured.')

    for product in products:
        # Skip calibrated products if consistent products are expected.
        if isinstance(product, CalibratedProduct):
            if utils.consistent_product_expected(product=product.product,

```

```

                                                                    timeframe=
                                                                    product.timeframe):
        continue

    # Update thredds
    ThreddsFile.get(product=product).update()

    # Store to ftp
    with open(product.path, 'rb') as product_file:
        response = ftp.storbinary(
            'STOR {}'.format(os.path.basename(product.path)),
            product_file
        )
    logging.debug('ftp response: {}'.format(response))

    # Copy to copy dirs
    logging.debug('Copying product to {} dirs.'.format(
        len(config.COPY_TARGET_DIRS),
    ))
    for target_dir in config.COPY_TARGET_DIRS:
        shutil.copy(product.path, target_dir)

ftp.quit()
```

```
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.

from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from radar import calc
from radar import config
from radar import io
from radar import utils
from radar import gridtools

from scipy import interpolate

import datetime
import h5py
import json
import logging
import multiprocessing
import numpy as np
import os
import re

BAND_RAIN = 1
BAND_RANG = 2
BAND_ELEV = 3

BAND_META = {
    BAND_RAIN: dict(name='rain'),
    BAND_RANG: dict(name='range'),
    BAND_ELEV: dict(name='elevation'),
}

def create_basegrid():
    left, right, top, bottom = config.COMPOSITE_EXTENT
    cellwidth, cellheight = config.COMPOSITE_CELL_SIZE
    width, height = right - left, top - bottom

    extent = [left, right, top, bottom]
    size = int(width / cellwidth), int(height / cellheight)
    projection = utils.projection(utils.RD)

    return gridtools.BaseGrid(
        extent=extent,
        size=size,
        projection=projection,
    )

BASEGRID = create_basegrid()

class GroundData():
    """
    Hold datetime and code for groundstationfile

    Import from filename or from code and datetime.
    Get filepath.

    TODO:
    - Unify interface with that of scansignature
    - Better name... This is not data.
    """
    def __init__(self, dataname=None, datacode=None, datadatetime=None):
        if datacode and datadatetime and (dataname is None):
            self._code = datacode
            self._datetime = datadatetime
        elif dataname and (datacode is None) and (datadatetime is None):
            self._from_dataname(dataname)
        else:
            raise ValueError(
```

```

        'Specify either dataname or datadatetime and datacode.'
    )

def _from_dataname(self, dataname):
    datacode = self._code_from_dataname(dataname)
    datetime_format = config.TEMPLATE_GROUND.format(code=datacode)
    datadatetime = datetime.datetime.strptime(dataname,
    datetime_format)

    self._code = datacode
    self._datetime = datadatetime

def _code_from_dataname(self, dataname):
    match = re.match(config.GROUND_PATTERN, dataname)
    if match:
        return match.group('code')
    raise ValueError(
        "Currently no ground pattern matching '{}'.format(dataname),
    )

def _format_for_code(self, code):
    return config.TEMPLATE_GROUND.format(code=code)
    raise ValueError(
        "Currently no ground format matching '{}'.format(code),
    )

def get_dataname(self):
    return datetime.datetime.strftime(
        self._datetime, config.TEMPLATE_GROUND.format(code=self._code),
    )

def get_datapath(self):
    return os.path.join(
        config.GROUND_DIR,
        self._code,
        self._datetime.strftime('%Y'),
        self._datetime.strftime('%m'),
        self._datetime.strftime('%d'),
        self.get_dataname(),
    )

class ScanSignature(object):
    """
    Hold datetime and code for a scan.

    Import from scanname or from code and datetime.
    Export to scanname or to code and datetime.
    Get timestamp or scanpath.
    """
    def __init__(self, scanname=None, scancode=None, scandatetime=None):
        if scancode and scandatetime and (scanname is None):
            self._datetime = scandatetime

            # Before 2012 DWD radar 'ess' was called 'ase'
            if self._datetime.year < 2012 and scancode == 'ess':
                self._code = 'ase'
            else:
                self._code = scancode

            # After 2012 DWD files have an additional id.
            if self._datetime.year >= 2012:
                self._id = config.RADAR_ID.get(self._code, '')
            else:
                self._id = ''

        elif scanname and (scancode is None) and (scandatetime is None):
            self._from_scanname(scanname)
        else:
            raise ValueError(
                'Specify either scanname or scandatetime and scancode.'
            )

```

```

    )

def _from_scanname(self, scanname):
    radar_dict = self._radar_dict_from_scanname(scanname)
    datetime_format = self._get_datetime_format(radar_dict)
    scandatetime = datetime.datetime.strptime(scanname,
    datetime_format)

    self._datetime = scandatetime
    self._code = radar_dict['code']
    self._id = radar_dict['id']

def _radar_dict_from_scanname(self, scanname):
    for pattern in config.RADAR_PATTERNS:
        match = re.match(pattern, scanname)
        if match:
            radar_code = match.group('code')
            try:
                radar_id = match.group('id')
            except:
                radar_id = ''
            return {'id': radar_id, 'code': radar_code}
    raise ValueError("Currently no pattern matching
    '{}'.format(scanname))

def _get_datetime_format(self, radar_dict):
    """
    Return the datetime format string that corresponds to current scan.
    """
    radar_code = radar_dict['code']
    radar_id = radar_dict['id']

    if radar_code in config.KNMI_RADARS:
        return config.TEMPLATE_KNMI.format(**radar_dict)
    elif radar_code in config.DWD_RADARS:
        if radar_id:
            return config.TEMPLATE_DWD.format(**radar_dict)
            return config.TEMPLATE_DWD_2011.format(**radar_dict)
        elif radar_code in config.DWD_RADARS_2011:
            return config.TEMPLATE_DWD_2011.format(**radar_dict)
    raise ValueError("There is no format for {}".format(radar_dict))

def get_scanname(self):
    return datetime.datetime.strftime(
        self._datetime, self._get_datetime_format(
            radar_dict={
                'code': self._code, 'id': self._id,
            },
        ),
    )

def get_scanpath(self):
    return os.path.join(
        config.RADAR_DIR,
        self._code,
        self._datetime.strftime('%Y'),
        self._datetime.strftime('%m'),
        self._datetime.strftime('%d'),
        self.get_scanname(),
    )

def get_datetime(self):
    return self._datetime

def get_timestamp(self):
    return self._datetime.strftime(config.TIMESTAMP_FORMAT)

def get_code(self):
    if self._code == 'ase':
        return 'ess'
    return self._code

```

```

def get_scan(self):
    if not os.path.exists(self.get_scanpath()):
        logging.warn(
            '{} not found.'.format(self.get_scanpath()),
        )
        return None
    elif self._code in config.DWD_RADARS:
        return ScanDWD(self)
    elif self._code in config.KNMI_RADARS:
        return ScanKNMI(self)
    elif self._code in config.DWD_RADARS_2011:
        return ScanDWD(self)
    logging.error(
        "Currently no scan class matching '{}".format(self._code),
    )
    return None

def __repr__(self):
    return '<ScanSignature: {} [{}]>'.format(self._code,
        self._datetime)

def __hash__(self):
    """ Bitwise or is a good method for combining hashes. """
    return hash(self._datetime) ^ hash(self._code)

```

```

class GenericScan(object):
    """
    Procedures for manipulating scan data.
    """
    def __init__(self, scansignature):
        self.signature = scansignature

    def _interpolate(self, points, values, grid):
        """
        Return numpy masked array.
        """
        # Interpolate
        data_out = interpolate.griddata(
            np.array([point.flatten() for point in points]).transpose(),
            np.array([value.flatten() for value in values]).transpose(),
            grid,
            method='linear',
            fill_value=config.NODATAVALUE,
        ).transpose(2, 0, 1)

        # Create masked array
        ma_out = np.ma.array(
            data_out,
            mask=np.equal(data_out, config.NODATAVALUE),
            fill_value=config.NODATAVALUE,
        )

        return ma_out

    def get(self):
        try:
            data = self.data()
        except IOError as error:
            logging.error('{}: {}'.format(error.strerror, error.filename))
            return None
        rang, azimuth, elev = data['polar']
        rain = data['rain']
        latlon = data['latlon']
        anth = data['ant_alt']
        # vvv For mocking JABBEKE. Can be removed when JABBEKE is really
        # vvv in production.
        #if self.signature.get_code() == 'JABBEKE':
        #    latlon = (51.179558, 3.09363)
        #    print('adjusted location for JABBEKE')

        theta = calc.calculate_theta(

```

```

        rang=rang,
        elev=np.radians(elev),
        anth=anth / 1000,
    )

    points_aeqd = calc.calculate_cartesian(
        theta=theta,
        azim=np.radians(azim),
    )

    projections = (
        utils.projection_aeqd(*latlon),
        utils.projection(utils.RD),
    )

    points_rd = utils.coordinate_transformer.transform(
        points=points_aeqd,
        projections=projections,
    )

    # Interpolate
    grid_rain, grid_rang, grid_elev = self._interpolate(
        points=points_rd,
        values=(
            rain,
            rang * np.ones(rain.shape),
            elev * np.ones(rain.shape),
        ),
        grid=BASEGRID.get_grid(),
    )

    location = utils.transform((0, 0), projections)

    ds_rd = BASEGRID.create_dataset(bands=3)
    ds_rd.SetMetadata(dict(
        source=self.signature.get_scanname(),
        timestamp=self.signature.get_timestamp(),
        station=self.signature.get_code(),
        location=json.dumps(location),
        antenna_height=json.dumps(anth),
        max_elevation=json.dumps(elev.max()),
        min_elevation=json.dumps(elev.min()),
        max_range=json.dumps(rang.max()),
    ))

    banddata = {
        BAND_RAIN: grid_rain,
        BAND_RANG: grid_rang,
        BAND_ELEV: grid_elev,
    }

    for i in range(1, ds_rd.RasterCount + 1):
        band = ds_rd.GetRasterBand(i)
        band.SetNoDataValue(banddata[i].fill_value)
        band.WriteArray(banddata[i].filled())
        band.SetMetadata(BAND_META[i])

    return ds_rd

```

```
class ScanKNMI(GenericScan):
```

```

    def data(self):
        """ Return data dict for further processing. """
        dataset = h5py.File(self.signature.get_scanpath(), 'r')
        scan_key = 'scan{}'.format(config.KNMI_SCAN_NUMBER)

        latlon = self._latlon(dataset)
        rain = self._rain(dataset[scan_key])
        polar = self._polar(dataset[scan_key])
        ant_alt = config.ANTENNA_HEIGHT[self.signature.get_code()]

```

```

dataset.close()

return dict(
    latlon=latlon,
    rain=rain,
    polar=polar,
    ant_alt=ant_alt,
)

def _latlon(self, dataset):
    """ Return (lat, lon) in WGS84. """
    lon, lat = dataset['radar1'].attrs['radar_location']
    return float(lat), float(lon)

def _rain(self, dataset):
    """ Convert and return rain values. """
    type_key = 'scan_{}_data'.format(config.KNMI_SCAN_TYPE)
    PV = np.float64(dataset[type_key])

    # Get calibration from attribute
    calibration_formula = (
        dataset['calibration'].attrs['calibration_Z_formulas'][0]
    )
    calibration_formula_match = re.match(
        config.CALIBRATION_PATTERN, calibration_formula
    )
    a = float(calibration_formula_match.group('a'))
    b = float(calibration_formula_match.group('b'))
    logging.debug('From "{}": a={}, b={} in dBZ = a * PV + b'.format(
        calibration_formula, a, b,
    ))

    dBZ = a * PV + b
    return calc.Rain(dBZ).get()

def _polar(self, dataset):
    """ Return (rang, azim, elev). """

    range_size = dataset.attrs['scan_number_range']
    range_bin = dataset.attrs['scan_range_bin']
    rang = np.arange(
        0.5 * range_bin,
        (range_size + 0.5) * range_bin,
        range_bin,
    ).reshape(1, -1)

    azim_step = dataset.attrs['scan_azim_bin']
    azim = np.arange(
        azim_step / 2,
        360 + azim_step / 2,
        azim_step,
    ).reshape(-1, 1)

    elev = dataset.attrs['scan_elevation'] * np.ones(azim.shape)
    return rang, azim, elev

class ScanDWD(GenericScan):

    def data(self):
        """ Return data dict for further processing. """
        dBZ, meta = io.readDX(self.signature.get_scanpath())

        return dict(
            latlon=self._latlon(),
            rain=self._rain(dBZ),
            polar=self._polar(meta),
            ant_alt=config.ANTENNA_HEIGHT[self.signature.get_code()]
        )

    def _polar(self, meta):
        rang = np.arange(meta['clutter'].shape[1]).reshape(1, -1) + 0.5

```

```

    azim = meta['azim'].reshape(-1, 1) + 180 / meta['azim'].size
    elev = meta['elev'].reshape(-1, 1)
    return rang, azim, elev

def _rain(self, dBZ):
    """ wradlib did conversion to dBZ. """
    return calc.Rain(dBZ).get()

def _latlon(self):
    latlon = config.DWD_COORDINATES[self.signature.get_code()]
    return latlon

class MultiScan(object):

    def __init__(self, multiscandatetime, scancodes):
        self.scancodes = scancodes
        self.multiscandatetime = multiscandatetime

        self.path = self.pathhelper = utils.PathHelper(
            basedir=config.MULTISCAN_DIR,
            code=config.MULTISCAN_CODE,
            template='{code}_{timestamp}.h5'
        ).path(multiscandatetime)

    def _add(self, dataset, scan):
        """
        Add scan to dataset, if it is available
        """
        source = scan.get()

        radar = scan.signature.get_code()
        target = dataset.create_group(radar)

        # Add general metadata
        target.attrs['projection'] = source.GetProjection()
        target.attrs['geotransform'] = source.GetGeoTransform()
        for k, v in source.GetMetadata().items():
            target.attrs[k] = v

        # Add bands with their metadata
        for source_band in [source.GetRasterBand(i + 1)
                           for i in range(source.RasterCount)]:
            # Data
            data = source_band.ReadAsArray()
            target_band = target.create_dataset(
                source_band.GetMetadataItem(b'name'),
                data.shape,
                dtype='f4',
                compression='gzip',
                shuffle=True,
            )
            target_band[:] = data

            # Metadata
            target_band.attrs['fill_value'] = source_band.GetNoDataValue()
            for k, v in source_band.GetMetadata().items():
                if k == 'name':
                    continue # This will be used as the name of the
                               dataset.
                target_band.attrs[k] = v

        logging.debug('{} added to multiscan.'.format(radar))

    def get(self):
        """
        Return a readonly hdf5 dataset for requested scans.

        If the dataset is not available, it is created.
        If it is available, but lacks some scan requested, it is appended.
        """
        utils.makedirs(os.path.dirname(self.path))

```

```

try:
    # Improperly closed h5 files cannot be opened.
    dataset = h5py.File(self.path, 'a')
except IOError:
    dataset = h5py.File(self.path, 'w')

if len(dataset):
    logging.debug(
        'Multiscan file already has {}'.format(', '.join(dataset))
    )
else:
    logging.debug('Starting with empty multiscan file.')

for scancode in self.scancodes:
    scan = ScanSignature(
        scandatetime=self.multiscandatetime, scancode=scancode,
    ).get_scan()
    if scan is None or scancode in dataset:
        continue
    self._add(dataset=dataset, scan=scan)

dataset.close()
return h5py.File(self.path, 'r')

```

```
class Composite(object):
```

```

    def __init__(self, compositedatetime, scancodes, declutter):

        self.scancodes = scancodes
        self.declutter = declutter

        if None in declutter.values():
            raise ValueError('Declutter may not contain None values.')

        self.multiscan = MultiScan(
            multiscandatetime=compositedatetime,
            scancodes=scancodes,
        )

    def _calculate(self, datasets):
        """
        Return a composite dataset based on the
        weighted lowest altitudes method.
        """
        if not datasets:
            return np.ma.array(
                np.zeros(BASEGRID.get_shape()),
                mask=True,
                fill_value=config.NODATAVALUE,
            )

        # Read datasets
        metadata = [dict(ds.attrs) for ds in datasets]

        stations = [m['station'] for m in metadata]

        anth = np.array(
            [json.loads(m['antenna_height']) for m in metadata],
        ).reshape((-1, 1, 1)) / 1000

        rain = np.ma.array(
            [gridtools.h5ds2ma(ds['rain']) for ds in datasets],
            fill_value=config.NODATAVALUE,
        )

        rang = np.ma.array(
            [gridtools.h5ds2ma(ds['range']) for ds in datasets],
            fill_value=config.NODATAVALUE,
        )

```

```

elev = np.ma.array(
    [gridtools.h5ds2ma(ds['elevation']) for ds in datasets],
    fill_value=config.NODATAVALUE,
)

# Extend mask around NL stations
if 'NL60' in stations and 'NL61' in stations:
    for i in map(stations.index, ['NL60', 'NL61']):
        rain.mask[i][np.less(rang[i], 15)] = True

# Do history declutter
if self.declutter['history']: # None or 0 disables history declutter
    if 'NL60' in stations and 'NL61' in stations:
        logging.debug('Starting history declutter.')
        h5 = h5py.File(
            os.path.join(config.SHAPE_DIR, 'clutter.h5'), 'r',
        )

        logging.debug('Using clutter threshold of {}'.format(
            self.declutter['history'],
        ))
        logging.debug('Clutterstats:')
        logging.debug(
            '    count: {}'.format(h5.attrs['cluttercount']),
        )
        logging.debug('    range: {}'.format(h5.attrs['range']))

        # Initialize clutter array of same dimensions as rain array
        clutter = np.zeros(rain.shape, rain.dtype)
        for i, radar in enumerate(stations):
            if radar in h5:
                clutter[i] = h5[radar]

        # Determine the mask:
        cluttermask = np.logical_and(
            np.equal(clutter, clutter.max(0)),
            np.greater(clutter, self.declutter['history']),
        )

        for i, radar in enumerate(stations):
            logging.debug('Masking {} pixels for {}'.format(
                cluttermask[i].sum(), radar,
            ))

        # Extend rain mask with cluttermask.
        rain.mask[cluttermask] = True
        h5.close()
    else:
        logging.debug('Need both NL60 and NL61 for history declutter.')

rang.mask = rain.mask
elev.mask = rain.mask

# Calculate heights
theta = calc.calculate_theta(
    rang=rang, elev=np.radians(elev), anth=anth,
)

alt_min = calc.calculate_height(
    theta=theta, elev=np.radians(elev - 0.5), anth=anth,
)
alt_max = calc.calculate_height(
    theta=theta, elev=np.radians(elev + 0.5), anth=anth,
)

lowest_alt_max = np.ma.where(
    np.equal(alt_min, alt_min.min(0)),
    alt_max,
    np.ma.masked,
).min(0)

```

```

bandwidth = alt_max - alt_min

vertdist1 = (lowest_alt_max - alt_min)
vertdist1[np.ma.less(vertdist1, 0)] = 0

vertdist2 = (lowest_alt_max - alt_max)
vertdist2[np.ma.less(vertdist2, 0)] = 0

overlap = vertdist1 - vertdist2

weight = overlap / bandwidth

composite = (rain * weight).sum(0) / weight.sum(0)
composite.fill_value = config.NODATAVALUE

return composite

def _metadata(self, radars, datasets):
    """ Return metadata dict. """
    requested_stations = json.dumps(self.scancodes)
    timestamp = self.multiscan.multiscandatetime.strftime(
        config.TIMESTAMP_FORMAT,
    )
    metadatas = [dict(ds.attrs) for ds in datasets]
    stations = json.dumps(radars)
    locations = json.dumps([json.loads(metadata['location'])
                            for metadata in metadatas])
    ranges = json.dumps([json.loads(metadata['max_range'])
                        for metadata in metadatas])
    method = 'weighted lowest altitudes'
    declutter_size = str(self.declutter['size'])
    declutter_history = str(self.declutter['history'])

    return dict(
        requested_stations=requested_stations,
        stations=stations,
        locations=locations,
        ranges=ranges,
        method=method,
        timestamp=timestamp,
        declutter_size=declutter_size,
        declutter_history=declutter_history,
    )

def _dataset(self, ma, md):
    """ Return dataset with composite. """
    dataset = BASEGRID.create_dataset()
    dataset.SetMetadata(md)
    band = dataset.GetRasterBand(1)
    band.WriteArray(ma.filled())
    band.SetNoDataValue(ma.fill_value)
    return dataset

def get(self):
    """
    Return a composite dataset based on the
    weighted lowest altitudes method.
    """
    multiscan_dataset = self.multiscan.get()
    if len(multiscan_dataset):
        (
            radars,
            datasets,
        ) = zip(*[(code, dataset)
                 for code, dataset in multiscan_dataset.items()
                 if code in self.scancodes])
    else:
        radars, datasets = [], []

    logging.debug('Calculating composite.')

```

```

    if len(datasets) == 0:
        logging.warn(
            'no composite created for {}: None of {} found.'.format(
                self.multiscan.multiscandatettime,
                ', '.join(self.multiscan.scancodes),
            ),
        )
        pass
        # return None

    ma = self._calculate(datasets)
    md = self._metadata(radars=radars, datasets=datasets)

    multiscan_dataset.close()

    if self.declutter['size']: # None or 0 disables history declutter
        calc.declutter_by_area(array=ma, area=self.declutter['size'])

    dataset = self._dataset(ma=ma, md=md)

    logging.info('Created composite {}: {}'.format(
        self.multiscan.multiscandatettime.strftime('%Y-%m-%d %H:%M:%S'),
        ', '.join(json.loads(dataset.GetMetadataItem(b'stations'))),
    ))

    return dataset

def make_aggregate(aggregate):
    aggregate.make()
    return aggregate

class Aggregate(object):
    """
    """
    CODE = {
        datetime.timedelta(minutes=5): '5min',
        datetime.timedelta(hours=1): 'uur',
        datetime.timedelta(days=1): '24uur',
        datetime.timedelta(days=2): '48uur',
    }

    TD = {v: k for k, v in CODE.items()}

    SUB_CODE = {
        '48uur': '24uur',
        '24uur': 'uur',
        'uur': '5min',
    }

    def __init__(self, dt_aggregate, td_aggregate, scancodes, declutter):
        self.dt_aggregate = dt_aggregate
        self.td_aggregate = td_aggregate
        self.scancodes = scancodes
        self.code = self.CODE[td_aggregate]

        # Coerce self.declutter to requirement for Composite object
        if declutter is None or declutter['size'] is None:
            declutter_size = config.DECLUTTER_SIZE
        else:
            declutter_size = declutter['size']

        if declutter is None or declutter['history'] is None:
            declutter_history = config.DECLUTTER_HISTORY
        else:
            declutter_history = declutter['history']

        self.declutter = dict(size=declutter_size, history=
            declutter_history)
        # Prevent illegal combinations of dt and dd, can be nicer...

```

```

if self.code == '48uur':
    if dt_aggregate != dt_aggregate.replace(hour=8, minute=0,
                                              second=0, microsecond=
                                              0):
        raise ValueError
if self.code == '24uur':
    if dt_aggregate != dt_aggregate.replace(hour=8, minute=0,
                                              second=0, microsecond=
                                              0):
        raise ValueError
if self.code == 'uur':
    if dt_aggregate != dt_aggregate.replace(minute=0,
                                              second=0, microsecond=
                                              0):
        raise ValueError
if self.code == '5min':
    if dt_aggregate != dt_aggregate.replace(
        minute=(dt_aggregate.minute // 5) * 5,
        second=0,
        microsecond=0,
    ):
        raise ValueError

def _sub_datetimes(self):
    """
    Return datetime generator of datetimes of the next aggregate
    resolution.

    Note that it ends at this aggregate's datetime.
    """
    step = self.TD[self.SUB_CODE[self.code]]
    end = self.dt_aggregate
    start = self.dt_aggregate - self.td_aggregate + step

    current = start
    while current <= end:
        yield current
        current += step

def get_path(self):
    """ Return the file path where this aggregate should be stored. """
    path_helper = utils.PathHelper(basedir=config.AGGREGATE_DIR,
                                   code=self.code,
                                   template='{code}_{timestamp}.h5')
    return path_helper.path(self.dt_aggregate)

def _check(self, h5):
    """ Return if h5 is consistent with requested parameters. """
    if set(h5.attrs['radars']) != set(self.scancodes):
        # Aggregate was made for a different set of radars
        raise ValueError('Unmatched radarset in existing aggregate.')
    if h5.attrs['declutter_history'] != self.declutter['history']:
        raise ValueError('Other history declutter setting in
        aggregate')
    if h5.attrs['declutter_size'] != self.declutter['size']:
        raise ValueError('Other size declutter setting in aggregate')
    if not h5.attrs['available'].all():
        if self.code != '5min':
            raise ValueError('Missing radars in existing aggregate')
        # Check if it is useful to create a new composite
        dt_composite = self.dt_aggregate - self.TD[self.code]
        files_available = []
        for radar in sorted(self.scancodes):
            files_available.append(os.path.exists(ScanSignature(
                scancode=radar, scandatetime=dt_composite
            ).get_scanpath()))
        if not (h5.attrs['available'] == files_available).all():
            raise ValueError('Missing radars, but scanfiles exist.')
    else:
        logging.debug('Missing radars caused by missing
        scanfiles.')
    return

```

```

def _create(self):
    """
    Create a new dataset from the composite.
    """
    # Create the composite
    dt_composite = self.dt_aggregate - self.TD[self.code]

    composite = Composite(compositedatettime=dt_composite,
                          scancodes=self.scancodes,
                          declutter=self.declutter).get()

    # Composite unit is mm/hr and covers 5 minutes. It must be in mm.
    fill_value = config.NODATAVALUE
    array = composite.GetRasterBand(1).ReadAsArray()
    mask = np.equal(array, fill_value)
    masked_array = np.ma.array(
        array, mask=mask, fill_value=fill_value,
    ) / 12

    composite_meta = composite.GetMetadata()

    # Create the data for the h5
    radars = sorted(
        [str(radar)
         for radar in
          json.loads(composite_meta['requested_stations'])],
    )
    radar_list = zip(json.loads(composite_meta['stations']),
                    json.loads(composite_meta['ranges']),
                    json.loads(composite_meta['locations']))
    locations_dict = {rad: loc for rad, rng, loc in radar_list}
    ranges_dict = {rad: rng for rad, rng, loc in radar_list}

    available = np.array([radar in ranges_dict for radar in radars])
    ranges = [ranges_dict.get(radar, fill_value) for radar in radars]
    locations = np.array([locations_dict.get(radar, 2 * [fill_value])
                          for radar in radars])

    h5_meta = dict(
        grid_projection=BASEGRID.projection,
        grid_extent=BASEGRID.extent,
        grid_size=BASEGRID.size,
        radars=radars,
        ranges=ranges,
        locations=locations,
        available=available,
        method=composite_meta['method'],
        declutter_history=float(composite_meta['declutter_history']),
        declutter_size=float(composite_meta['declutter_size']),
        timestamp_first_composite=composite_meta['timestamp'],
        timestamp_last_composite=composite_meta['timestamp'],
        composite_count=1,
        fill_value=fill_value,
    )

    h5_data = dict(
        precipitation=masked_array,
    )

    path = self.get_path()
    utils.save_dataset(h5_data, h5_meta, path)
    logging.info('Created aggregate {} ({}).format(
        self.dt_aggregate, self.code
    ))

def _merge(self, aggregates):
    """
    Return h5_dataset by merging iterable of aggregate objects.
    """
    if self.code == 'uurl1':
        # Make the iterables in parallel

```

```

        pool = multiprocessing.Pool()
        iterable = iter(pool.map(make_aggregate, aggregates))
        pool.close()
    else:
        # Make the iterables the usual way
        iterable = iter(map(make_aggregate, aggregates))

    aggregate = iterable.next()
    h5 = aggregate.get()

    meta = dict(h5.attrs)
    available = [meta['available']]

    array = h5['precipitation']
    fill_value = config.NODATAVALUE
    mask = np.equal(array, fill_value)
    masked_array = np.ma.array(array, mask=mask, fill_value=fill_value)

    h5.close()

    for aggregate in iterable:

        h5 = aggregate.get()

        array = h5['precipitation']
        fill_value = config.NODATAVALUE
        mask = np.equal(array, fill_value)
        masked_array = np.ma.array([
            masked_array,
            np.ma.array(array, mask=mask)
        ]).sum(0)

        for i in range(len(meta['radars'])):
            if meta['ranges'][i] == config.NODATAVALUE:
                meta['ranges'][i] = h5.attrs['ranges'][i]
            if (meta['locations'][i] == 2 *
                [config.NODATAVALUE]).all():
                meta['locations'][i] = h5.attrs['locations'][i]

        available.append(meta['available'])

        meta['composite_count'] += h5.attrs.get(
            'composite_count',
        )
        meta['timestamp_last_composite'] = h5.attrs.get(
            'timestamp_last_composite',
        )

        h5.close()

    meta['available'] = np.vstack(available)
    data = dict(precipitation=masked_array)
    path = self.get_path()

    utils.save_dataset(data, meta, path)
    logging.info('Created aggregate {} ({}).format(
        self.dt_aggregate, self.code
    ))

def make(self):

    """ Creates the hdf5 file corresponding to this objects. """
    logging.debug('Creating aggregate {} ({}).format(
        self.dt_aggregate, self.code,
    ))
    path = self.get_path()

    # If there is already a good one, return it
    if os.path.exists(path):
        logging.debug('Checking if existing aggregate can be reused.')
        try:
            with h5py.File(path, 'r') as h5:

```

```

        self._check(h5)
        logging.debug('Check ok.')
        return
    except KeyError as error:
        logging.debug('Check failed: {}'.format(error))
    except ValueError as error:
        logging.debug('Check failed: {}'.format(error))

# So, now we really need to create an aggregate.
sub_code = self.SUB_CODE.get(self.code)

if sub_code is None:
    return self._create()

# If there is a sub_code, return corresponding aggregates merged.
td_aggregate = self.TD[sub_code]
dt_aggregate = self._sub_datetimes()
sub_aggregates = (Aggregate(dt_aggregate=dt_aggregate,
                            td_aggregate=td_aggregate,
                            scancodes=self.scancodes,
                            declutter=self.declutter)
                  for dt_aggregate in self._sub_datetimes())

return self._merge(aggregates=sub_aggregates)

def get(self):
    """ Return opened h5 dataset in read mode. """
    path = self.get_path()
    try:
        return h5py.File(path, 'r')
    except IOError:
        logging.warn(
            'Creating aggregate {} ({}), because it did not
            exist'.format(
                self.dt_aggregate, self.code,
            ),
        )
        self.make()
    return h5py.File(path, 'r')

```

```
# -*- coding: utf-8 -*-
# (c) Nelen & Schuurmans. GPL licensed, see LICENSE.rst.

from __future__ import print_function
from __future__ import unicode_literals
from __future__ import absolute_import
from __future__ import division

from osgeo import osr

from radar import config

from matplotlib import colors
from matplotlib import cm

from PIL import Image

import codecs
import cStringIO
import csv
import datetime
import h5py
import logging
import numpy as np
import os

# Some projections
UTM = 3405
DHDN = 4314
WGS84 = 4326
GOOGLE = 900913

AEQD_PROJ4 = ('+proj=aeqd +a=6378.137 +b=6356.752 +R_A'
              '+lat_0={lat} +lon_0={lon} +x_0=0 +y_0=0')

# Copied from lizard_map/coordinates.py
RD = ("+proj=sterea +lat_0=52.156160555555555 +lon_0=5.387638888888889 "
      "+k=0.999908 +x_0=155000 +y_0=463000 +ellps=bessel "
      "+towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812 "
      "+units=m +no_defs")

# Projections and transformations
def projection_aeqd(lat=None, lon=None):
    sr = osr.SpatialReference()
    sr.ImportFromProj4(str(AEQD_PROJ4.format(lat=lat, lon=lon)))
    return sr.ExportToWkt()

def projection(desc, export='wkt'):
    sr = osr.SpatialReference()
    if isinstance(desc, int):
        sr.ImportFromEPSG(desc)
    elif isinstance(desc, (str, unicode)):
        if desc.startswith('+proj='):
            sr.ImportFromProj4(str(desc))
        else:
            sr.ImportFromWkt(str(desc))
    if export == 'wkt':
        return sr.ExportToWkt()
    if export == 'proj4':
        return sr.ExportToProj4()
    return sr

def transform(point, desc):
    srs = [projection(e, export=None) for e in desc]
    ct = osr.CoordinateTransformation(*srs)
    return ct.TransformPoint(*point)[0:2]
```

```

class CoordinateTransformer(object):
    """
    Transform coordinates or give cached coordinates back.

    TODO Make this a simple function, caching is not trivial here.
    """

    def __init__(self):
        self.cache = {}

    def transform(self, points, projections):
        """
        Transform arrays of points from one projection to another.
        """
        shape = np.array([p.shape for p in points]).max(0)

        points_in = np.array([
            points[0].flatten(),
            points[1].flatten(),
        ]).transpose()

        ct = osr.CoordinateTransformation(
            projection(projections[0], export=None),
            projection(projections[1], export=None),
        )

        points_out = np.array(ct.TransformPoints(points_in))[:, 0:2]

        result = points_out.reshape(shape[0], shape[1], 2).transpose(2, 0,
1)
        return result

# Singleton
coordinate_transformer = CoordinateTransformer()

# Dateranges and date helpers
def timestamp2datetime(ts, fmt='%Y%m%d%H%M%S'):
    return datetime.datetime.strptime(ts, fmt)

def datetime2timestamp(dt, fmt='%Y%m%d%H%M%S'):
    return dt.strftime(fmt)

class DateRange(object):

    FORMAT = {
        4: '%Y',
        6: '%Y%m',
        8: '%Y%m%d',
        10: '%Y%m%d%H',
        12: '%Y%m%d%H%M',
    }
    STEP = datetime.timedelta(minutes=5)

    def __init__(self, text, step=STEP):
        self._step = step
        self._start, self._stop = self._start_stop_from_text(text)

    def _start_stop_from_text(self, text):
        """
        Ste
        """
        if '-' in text:
            text1, text2 = text.split('-')
            start = self._single_from_text(text=text1)
            stop = self._single_from_text(text=text2, last=True)
        else:
            start = self._single_from_text(text=text)

```

```

        stop = self._single_from_text(text=text)
        return start, stop

def _single_from_text(self, text, last=False):
    """
    Return datetime that matches text.

    If last, return the last possible step for text.
    """
    first = datetime.datetime.strptime(
        text,
        self.FORMAT[len(text)],
    )
    if not last:
        return first

    td_kwargs = {
        4: {'years': 1},
        6: {'months': 1},
        8: {'days': 1},
        10: {'hours': 1},
        12: {'minutes': 5}, # Makes a range of minutes possible.
    }[len(text)]
    return first - self._step + datetime.timedelta(**td_kwargs)

def iterdatetimes(self):
    dt = self._start
    while dt <= self._stop:
        yield dt
        dt += self._step

class MultiDateRange(object):
    """ Hold a series of datetime ranges and generate it. """

    def __init__(self, text, step=5):
        step = datetime.timedelta(minutes=step)
        self._dateranges = [DateRange(subtext, step)
                             for subtext in text.strip(',').split(',')]

    def iterdatetimes(self):
        for dr in self._dateranges:
            for dt in dr.iterdatetimes():
                yield dt

# Path helpers and organizers
class PathHelper(object):
    """
    >>> import datetime
    >>> dt = datetime.datetime(2011, 03, 05, 14, 15, 00)
    >>> ph = PathHelper('a', 'c', '{code}:{timestamp}')
    >>> ph.path(dt)
    u'a/c/2011/03/05/14/c:20110305141500'
    """

    TIMESTAMP_FORMAT = config.TIMESTAMP_FORMAT

    def __init__(self, basedir, code, template):
        """
        Filetemplate must be something like '{code}_{timestamp}.'
        """
        self._basedir = os.path.join(basedir, code)
        self._code = code
        self._template = template

    def path(self, dt):
        return os.path.join(
            self._basedir,
            dt.strftime('%Y'),
            dt.strftime('%m'),
            dt.strftime('%d'),

```

```

        self._template.format(
            code=self._code,
            timestamp=dt.strftime(self.TIMESTAMP_FORMAT)
        )
    )

def path_with_hour(self, dt):
    return os.path.join(
        self._basedir,
        dt.strftime('%Y'),
        dt.strftime('%m'),
        dt.strftime('%d'),
        dt.strftime('%H'),
        self._template.format(
            code=self._code,
            timestamp=dt.strftime(self.TIMESTAMP_FORMAT)
        )
    )

# Timing
def closest_time(timeframe='f', dt_close=None):
    """
    Get corresponding datetime based on the timeframe.
    e.g. with dt_close = 2012-04-13 12:27
    timeframe = 'f' returns 2012-04-13 12:25
    timeframe = 'h' returns 2012-04-13 12:00
    timeframe = 'd' returns 2012-04-12 08:00
    """
    if dt_close is not None:
        now = dt_close
    else:
        now = datetime.datetime.utcnow()
    if timeframe == 'h':
        closesttime = now.replace(minute=0, second=0, microsecond=0)
    elif timeframe == 'd':
        closesttime = now.replace(hour=8, minute=0, second=0, microsecond=0)
        if closesttime > now:
            closesttime = closesttime - datetime.timedelta(days=1)
    else:
        closesttime = now.replace(
            minute=now.minute - (now.minute % 5), second=0, microsecond=0,
        )
    return closesttime

def timeframes(date, product):
    """ Return a list of timeframe codes corresponding to a date. """
    result = []
    if date.second == 0 and date.microsecond == 0:
        if date.minute == (date.minute // 5) * 5:
            result.append('f')
        if date.minute == 0:
            result.append('h')
            if ((date.hour == 8 and product != 'n') or
                (date.hour == 9 and product == 'n')):
                result.append('d')
    return result

def consistent_product_expected(product, timeframe):
    return (timeframe == 'f' and (product == 'n' or product == 'a')
            or (timeframe == 'h' and product == 'n'))

def get_groundfile_datetime(prodcode, date):
    """
    Return datetime for groundfile for a given product code and datetime

    For ground data to be more complete, the datetime of the grounddata
    must a certain amount later than the radar datetime. So for a

```

```

product at 2012-12-18 09:05 the groundfile datetimes must be:
    real-time      => 2012-12-18-09:05
    near-real-time => 2012-12-18-10:05
    afterwards     => 2012-12-20-09:05
'''
delta = dict(
    r=datetime.timedelta(minutes=0),
    n=datetime.timedelta(hours=1),
    a=datetime.timedelta(days=2),
)
return date + delta[prodcode]

# Visualization
def rain_kwargs(max_rain=120, name='buienradar', threshold=0.1):
    """ Return colormap and normalizer suitable for rain. """
    if name == 'buienradar':
        rain_colors = {
            'red': (
                (0, 240, 240),
                (2, 158, 110),
                (5, 88, 0),
                (10, 0, 255),
                (100, 131, 192),
                (max_rain, 192, 192),
            ),
            'green': (
                (0, 240, 240),
                (2, 158, 110),
                (5, 88, 0),
                (10, 0, 0),
                (100, 0, 0),
                (max_rain, 0, 0),
            ),
            'blue': (
                (0, 255, 255),
                (2, 255, 255),
                (5, 255, 200),
                (10, 110, 0),
                (100, 0, 192),
                (max_rain, 192, 192),
            ),
        }

        cdict = {}
        for key, value in rain_colors.items():
            cdict[key] = []
            for element in value:
                cdict[key].append((
                    element[0] / max_rain,
                    element[1] / 255,
                    element[2] / 255,
                ))

        colormap = colors.LinearSegmentedColormap('rain', cdict)
        normalize = colors.Normalize(vmin=0, vmax=max_rain)

        return dict(colormap=colormap, normalize=normalize)

    if name == 'jet':
        colormap = cm.jet

        def normalize(data):
            ma = np.ma.array(data)
            ma[np.less(ma, threshold)] = np.ma.masked
            return colors.LogNorm(vmin=0.1, vmax=max_rain)(ma)

        return dict(colormap=colormap, normalize=normalize)

def merge(images):
    """

```

```

Return a pil image.

Merge a list of pil images with equal sizes top down based on
the alpha channel.
"""

def paste(image1, image2):
    image = image2.copy()
    mask = Image.fromarray(np.array(image1)[:, :, 3])
    rgb = Image.fromarray(np.array(image1)[:, :, 0:3])
    image.paste(rgb, None, mask)
    return image

return reduce(paste, images)

def makedirs(dirname):
    """ Return True if directory was created, else False. """
    try:
        os.makedirs(dirname)
        return True
    except OSError:
        return False

class UnicodeWriter:
    """
    A CSV writer which will write rows to CSV file "f",
    which is encoded in the given encoding.
    """

    def __init__(self, f, dialect=csv.excel, encoding="utf-8", **kwds):
        # Redirect output to a queue
        self.queue = cStringIO.StringIO()
        self.writer = csv.writer(self.queue, dialect=dialect, **kwds)
        self.stream = f
        self.encoder = codecs.getincrementalencoder(encoding)()

    def writerow(self, row):
        self.writer.writerow([str(s).encode("utf-8") for s in row])
        # Fetch UTF-8 output from the queue ...
        data = self.queue.getvalue()
        data = data.decode("utf-8")
        # ... and reencode it into the target encoding
        data = self.encoder.encode(data)
        # write to the target stream
        self.stream.write(data)
        # empty queue
        self.queue.truncate(0)

    def writerows(self, rows):
        for row in rows:
            self.writerow(row)

class UTF8Recoder:
    """
    Iterator that reads an encoded stream and reencodes the input to UTF-8
    """
    def __init__(self, f, encoding):
        self.reader = codecs.getreader(encoding)(f)

    def __iter__(self):
        return self

    def next(self):
        return self.reader.next().encode("utf-8")

def write_csv(outfilename, output):
    csvwriter = UnicodeWriter(
        outfilename,

```

```

        delimiter=','.encode('utf-8'),
        quotechar='\"'.encode('utf-8'),
    )
    csvwriter.writerows(output)

def save_attrs(h5, attrs):
    for key, value in attrs.items():
        if isinstance(value, dict):
            group = h5.create_group(key)
            save_attrs(h5=group, attrs=value)
            continue
        h5.attrs[key] = np.array([value])

def save_dataset(data, meta, path):
    """
    Accepts an array jam-packed with data, a metadata file and a path
    to produce a fresh h5 file.
    """
    logging.debug('Saving hdf5 dataset: {}'.format(path))
    mkdir(os.path.dirname(path))
    h5 = h5py.File(path, 'w')

    # Geographic group
    geographic = dict(
        geo_par_pixel=b'X,Y',
        geo_dim_pixel=b'KM,KM',
        geo_pixel_def=b'CENTRE',
        geo_number_columns=500,
        geo_number_rows=490,
        geo_pixel_size_x=1.000,
        geo_pixel_size_y=-1.000,
        geo_product_corners=[-110000, 210000,
                              -110000, 700000,
                              390000, 700000,
                              390000, 210000],
        map_projection=dict(
            projection_indication=b'Y',
            projection_name=b'STEROGRAPHIC',
            projection_proj4_params=projection(RD, export='proj4'),
        ),
    )

    # Overview group
    availables = meta['available']
    availables_any = availables.any(0) if availables.ndim == 2 else availables
    products_missing = str(', ').join(
        [radar
         for radar, available in zip(meta['radars'], availables_any)
         if not available],
    )

    product_datetime_start = (timestamp2datetime(
        meta['timestamp_last_composite'],
    ) +
    config.TIMEFRAME_DELTA['f']).strftime('%d-%b-%Y;%H:%M:%S.%f').upper()
    product_datetime_end = product_datetime_start

    overview = dict(
        hdftag_version_number=b'3.5',
        number_image_groups=1,
        number_radar_groups=0,
        number_satellite_groups=0,
        number_station_groups=0,
        product_datetime_end=product_datetime_end,
        product_datetime_start=product_datetime_start,
        product_group_name=str(os.path.splitext(os.path.basename(path))
                               [0]),
        products_missing=products_missing,
        #product_group_doc=b'http://nationaleregenradar.nl',
        #dataset_raster_type=b'Composited interpolated rectangular radar

```

```

        data',
    )

    # Image group
    calibration=dict(
        calibration_flag=b'Y',
        calibration_formulas=b'GEO = 0.010000 * PV + 0.000000',
        calibration_missing_data=0,
        calibration_out_of_image=65535,
    )

    image1 = dict(
        calibration=calibration,
        image_bytes_per_pixel=2,
        image_geo_parameter=b'PRECIP_[MM]',
        image_product_name=overview['product_group_name'],
        image_size=data['precipitation'].size,
    )

    groups = dict(
        overview=overview,
        geographic=geographic,
        image1=image1,
    )

    save_attrs(h5, groups)
    dataset = h5.create_dataset('image1/image_data',
        data['precipitation'].shape,
        dtype='u2', compression='gzip', shuffle=
            True)

    image_data = dict(
        CLASS=b'IMAGE',
        VERSION=b'1.2',
    )

    save_attrs(dataset, image_data)

    # Creating the pixel values
    dataset[...] = np.uint16(data['precipitation'] * 100).filled(
        calibration['calibration_out_of_image']
    )

    # Keep the old way for compatibility with various products
    for name, value in data.items():
        dataset = h5.create_dataset(name, value.shape,
            dtype='f4', compression='gzip', shuffle=True)
        dataset[...] = value.filled(config.NODATAVALUE)

    for name, value in meta.items():
        h5.attrs[name] = value

    h5.close()

    return path

def get_countrymask():
    """
    Get a prefabricated country mask, 1 everywhere in the country and 0
    50 km outside the country. If extent and cellsize are not as in
    config.py, this breaks.
    """
    h5 = h5py.File(os.path.join(config.MISC_DIR, 'countrymask.h5'))
    mask = h5['mask'][...]
    h5.close()
    return mask

class FakeFtp(object):

```

```
def __init__(self, *args, **kwargs):  
    pass  
  
def storbinary(self, *args, **kwargs):  
    return 'Fake FTP in use!'  
  
def quit(self):  
    pass
```

BIJLAGE D Beschrijving metadata

- Voorbeeld van metadata radarbestand
- Achtergrond documentatie KNMI

Metadata in

var/calibrate/TF0005_R/2012/01/06/RAD_TF0005_R_20120106080000.h5:

=====
/geographic/

```

geo_par_pixel: X,Y
geo_number_columns: 500
geo_pixel_size_y: -1.0
geo_pixel_size_x: 1.0
geo_pixel_def: CENTRE
geo_dim_pixel: KM,KM
geo_number_rows: 490
geo_product_corners: [-110000 210000 -110000 700000 390000 700000
390000 210000]

```

=====
/geographic/map_projection/

```

projection_name: STEREOGRAPHIC
projection_proj4_params: +proj=sterea +lat_0=52.15616055555555 +lon_0=
5.387638888888889 +k=0.999908 +x_0=155000 +y_0=463000 +ellps=bessel
+towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812 +units=
m +no_defs
projection_indication: Y

```

=====
/image1/

```

image_bytes_per_pixel: 2
image_product_name: RAD_TF0005_R_20120106080000
image_geo_parameter: PRECIP_[MM]
image_size: 245000

```

=====
/image1/calibration/

```

calibration_out_of_image: 65535
calibration_missing_data: 0
calibration_formulas: GEO = 0.010000 * PV + 0.000000
calibration_flag: Y

```

=====
/image1/image_data/
-----=====
/overview/

```

number_satellite_groups: 0
product_group_name: RAD_TF0005_R_20120106080000
hdftag_version_number: 3.5
number_radar_groups: 0
products_missing: NL60, NL61, emd, ess, nhb
number_image_groups: 1
number_station_groups: 0
product_datetime_start: 06-JAN-2012;08:00:00.000000
product_datetime_end: 06-JAN-2012;08:00:00.000000

```

Metadata die nu nog in het bestand zit, maar die waarschijnlijk gaat verdwijnen:

=====
/

```

available: False
ranges: -9999
timestamp_last_composite: 2
grid_size: 500
grid_projection: P
radars: NL60
declutter_size: 4.0

```

meta.txt

1/8/2013

```
fill_value: -9999
locations: [-9999 -9999]
grid_extent: -110000
declutter_history: 50.0
composite_count: 1
stations_count: 0
timestamp_first_composite: 2
method: w
```



KNMI HDF5 Data Format Specification, v3.5

Hans Roozkrans and Iwan Holleman

Internal report, November 2003

1 Contents

1	Contents.....	2
2	Change notes.....	4
2.1	Change notes for version 3.5	4
2.2	Change notes for version 3.4.....	4
3	KNMI HDF 5 implementation	6
3.1	What is HDF ?.....	6
3.2	Versions	6
3.3	Tag naming.....	6
3.4	Purpose of KNMI HDF 5 Data Format	6
4	Overview	7
4.1	Mandatory groups	7
4.2	Repeated groups	7
5	Description of KNMI HDF 5 Data Format	9
5.1	Compression of datasets.....	9
5.2	Data types	9
5.3	Mandatory and optional fields.....	9
5.4	Overview group	10
5.5	Geographic group.....	11
5.6	Image group	13
5.7	Profile group.....	17
5.8	Discharge group	17
5.9	Visualisation group	18
5.10	Satellite group.....	18
5.11	Radar group	20
5.12	Station group	20
5.13	Classification group	21
6	Data types	22
6.1	Table.....	22
7	Conventions.....	23
7.1	Timestamps	23
7.2	HDF 5 file naming convention.....	23

7.3 Product_group_name convention.....24
7.4 Product_name convention.....28

2 Change notes

2.1 Change notes for version 3.5

The following changes are introduced with respect to KNMI HDF5 version 3.4:

The main change in this version is the extension of the KNMI HDF5 Image Format Specification beyond image data format only. The datamodel is extended to be able to store wind profile data, lightning localization data, and polar scan data from radar. The title of document has therefore been changed to "KNMI HDF5 Data Format Specification". In addition, a few minor changes have been applied to the datamodel which will not affect the current systems.

- The repeated **image group is no longer mandatory**, it will off course always be present for HDF5 files containing image data.
- Two new repeated groups have been added: **profile group and discharge group**, intended for storage of wind profile and lightning localizations, respectively.
- The name of the **lightning group is changed to station group**, and the lightning subgroup in the image group has been removed. The naming of the tags in the station group has been changed and two tags for storage of availability information have been added.
- The **grid, point, and vector groups** have been deleted as they are not used.

- A "**image_datetime_valid**" tag has been added in the image group(s) for specifying the date and time that a forecast product is valid.
- The tags for the new repeated groups have been added in the overview group: **number_profile_groups** and **number_discharge_groups**. In addition, the **number_lightning_groups** tag has been renamed to **number_station_groups**. Finally, **number_grid_groups**, **number_point_groups**, and **number_vector_groups** have been removed from the overview group.
- A tag "**geo_par_pixel**" describing the geophysical parameters of the image coordinates has been added to the geographical group.
- The "**DISPLAY_ORIGIN**" subtags to the HDF5 images in the overview, image, classification groups has been changed from mandatory to only mandatory for images with an unusual orientation.
- A few tags describing the characteristics of a polar radar scan have been added to the **radar subgroup** of the image group.
- A **profile group** with contents has been added which is intended for storage of profile data from weather radar, sodar, and profiler. It can be extended for profiles from radiosonde and models.
- A **discharge group** with contents has been added which is intended for storage of timeseries of localizations from lightning detection systems.
- Two tags, one for storage of limited availability

information and one for adjustment information, have been added to the **radar group**.

2.2 Change notes for version 3.4

The following changes are introduced with respect to KNMI HDF5 version 3.2:

- **The naming convention of repeated groups.** The change is that potential repeated groups **always** get a number at the end of the groupname. In contrary to the definition in version 3.2, now the groupname ends with 1 in case there is only one group included in the file (e.g. image1 instead of image). This is only valid for groups that are allowed to be repeated (image, visualisation, satellite, radar, lightning, classification, grid, point, and vector).
- **The implementation of calibration tables.** If the values in a calibration table are linear calibrated then not all pixel values need to be included in the table. The minimum and maximum values of linear range(s) may included only and then the values in between can be calculated by interpolation. This convention may save a lot of table space (e.g. in case of 16-bit pixel values).
An example:

Old table:

Integer value	Calibrated value
0.0	-50.0
1.0	-49.5
2.0	-49.0
...	...

251.0	75.5
252.0	76.0
253.0	76.5
254.0	77.0

New table:

Integer value	Calibrated value
0.0	-50.0
255.0	77.5

- **Definition of order of geo_product_corners** (in Geographic group). The first corner is always the southwest corner . Then northwest, northeast and southeast. In earlier versions it was defined as lowerleft, upperleft, etc.
- **HDF5 file naming convention.** The timestamp convention was not defined in earlier versions (the convention that is specified in chapter 7 is not applicable in file names). In chapter 8 the timestamp convention as used within KNMI is described.

- **Tables in HDF5 can only contain elements of the same data type.** This means that annotation/classification tables may contain only strings. So, also the pixel values must be written as strings.
- **The names of two attributes in the Overview group have been changed:**
Old name (version 3.2):
New name (version 3.3):
 KNMIhdf5_version_number
 hdf5tag_version_number
 KNMI_hdf_url
 hdf5tag_url

The following changes are introduced with respect to KNMI HDF5 **version 3.3**:

- **In the Overview group:** "number_XXX_groups" tags are only mandatory in case the value is larger than 0. In previous version these groups were mandatory in any case.

3 KNMI HDF5 implementation

In this document the HDF5 Data format as implemented within KNMI is described. Some more specific background information is given.

3.1 What is HDF ?

HDF (Hierarchical Data Format) is a multi-object file format for sharing scientific data in a distributed environment. HDF is developed by the National Center for Supercomputing Applications (NCSA) in the USA. An important feature of HDF files is that the files are self-describing. The aim of HDF is to make files, containing e.g. image data, self-contained. By grouping related pieces of information together and forcing a certain structure of the file, it will be easy for other users to make use of the data in the files. Self-contained also means that all meta-data associated with the dataset and/or needed for further processing of the dataset is included in the file. Another attractive feature of HDF is the "free" availability of software libraries to handle HDF files. Libraries are developed by NCSA for different operating systems (Unix, Linux, Windows). More general information on HDF can be found on: <http://hdf.ncsa.uiuc.edu>.

3.2 Versions

The implementation of the file format described in this document is based on HDF version 5.0 (HDF5). KNMI has based its definition of HDF5 on the MEOS-HDF format of Kongsberg Spacetec. MEOS (Multi-mission Earth Observation System) is Kongsberg Spacetec's line of systems for earth observation satellites ground stations. Spacetec is

the builder of KNMI's Omnivoor/Beelden image database system.

For users of the KNMI HDF5 files the following HDF5 utilities are most useful: "h5view" and "h5dump". These utilities (and many others) can be downloaded from: <http://hdf.ncsa.uiuc.edu/hdf5>.

3.3 Tag naming

All information placed within an HDF5 file can be stored and retrieved by a named tag. Tag names should be unambiguous and self-explanatory. In general, tags should be lower case with the underscore character used as the word spacer. Use of abbreviations is common when naming tags. However, KNMI and Spacetec suggest that abbreviations only are used when it can be used unambiguously and when it's quite obvious what it stands for. E.g. "clong" could be used to represent "centre longitude", but we suggest that "center_longitude" is used instead.

3.4 Purpose of KNMI HDF5 Data Format

The purpose of the KNMI HDF5 Data Format is to store the (image) data in a platform independent and self-contained way. The HDF5 Data Format as described in this document is implemented in the image data infrastructure of KNMI (BIK) for all EUMETSAT and NOAA satellite data, weather radar data, and lightning image data. This document is distributed to users of KNMI HDF5 data files to enable them to process the data.

4 Overview

The layout of the HDF5 files is similar to a directory structure in a file system. A folder, or divider, similar to a directory name, is used as a placeholder for information that is logically grouped together. The presence or absence of fields gives information on what can be done with the file. A minimum required set of tags and groups must be present in each KNMI HDF5 file.

Groups	Subgroups
overview (M)	
geographic (M)	
	map_projection
image (R)	
	calibration
	statistics
	satellite
	radar
profile (R)	
discharge (R)	

visualisation (R)	
satellite (R)	
	sensor
radar (R)	
station (R)	
classification (R)	
	calibration

M These groups are **mandatory**

R These groups can be **repeated**

4.1 Mandatory groups

The following groups are the minimal required content of the KNMI HDF5 files.

Overview group

This group provides an overview of the dataset. It contains references, identifiers, a quicklook and fields describing the content of the file.

Geographic group

All information about the geographic reference of the dataset is placed in this group. This includes map projection and parameters used in this operation.

4.2 Repeated groups

All other groups are optional. This implies these groups will be only included in case it is relevant or applicable to the type of data. These optional groups are called repeated groups, where a repetition of zero indicates the absence of a group. The **Overview** group contains fields stating the number of repeated groups present in the HDF5 file. The names of the groups will be numbered as follows: **groupnamen** where **groupname** is the name of the group, and **n** is the group number. Numbering starts at 1 and runs until the number of groups as stated in the overview group.

Image group

This group forms the core of a HDF5 image file. It contains most relevant image information. The image data (pixel values) itself and metadata about the image are placed in this group. There can be more than one image group in one HDF5 file. In the KNMI HDF5 implementation the following rules for the storage of multiple image-layers are defined:

- One HDF5 file can only contain multiple images (2D arrays) in case these images all have the same **geographical** properties (spatial resolution, area coverage and map projection). This implies that the images, stored in **one** HDF5 file, **all** have an **equal** number of pixel columns and rows.
- One Image group can only contain multiple images (2D arrays) in case these images all have the same **product_name** (so, the subgroups, calibration, etc. applies to all images in this image group). This only makes sense for animations (timed sequences of images).

For the moment KNMI does not have plans to

store time series of images in one HDF5 file and this technical note does not describe the HDF5 implementation for the storage of time series.

The KNMI implementation implies that in case more than one image-layer is stored in one HDF5 file (e.g. all AVHRR channel images), each image-layer is stored in a separate (repeated) Image group. The storage of image-layers in the image_data tag is implemented following the NCSA standards as described in the document "HDF5 Image and Palette Specification" (see: <http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html>).

Each Image group contains a Calibration subgroup that contains all relevant metadata in relation to the calibration of the pixelvalues of the image(s), stored in the group. The transformation of (binary) pixel values to real geophysical values is described here.

Profile group

This group is intended for storage of (wind) profile data from weather radars, profilers, sodars, and models.

Discharge group

This group is intended for storage of localization data from a lightning detection system.

Visualisation group

In this group a colour palette is stored (following the NCSA specs; see: <http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html>).

There can be more Visualisation groups in one HDF5 file.

Satellite group

In this group, all relevant information regarding the satellite is placed. This includes the position of the satellite at the time of data capturing. Orbit prediction data are included here. Also descriptions of the on-board instruments/sensors can be placed here (e.g. spectral bands related to channels, etc.). If data of more than one satellite is included in the HDF5 file then more Satellite groups will be included in the file.

Radar group

Idem as satellite group for (weather) radar systems.

Station group

Idem as satellite group for general observation systems, e.g., lightning detection stations.

Classification group

If automatic classification has been done, results from this process can be placed here. E.g. land/sea/cloud masks. There can be more Classification groups in one HDF5 file. This group is currently used for storage of geographical overlays for satellite images.

5 Description of KNMI HDF5 Data Format

In the HDF5 structure, besides groups, two other data types can be recognised: datasets and attributes. Datasets are meant for storage of the real image data and metadata entities larger than 16Kb. Attributes belong to a group or a dataset and are meant for storage of small entities of metadata. In the following tag description it is defined for each tag whether it is a dataset or an attribute (*in the D/A column*).

5.1 Compression of datasets

HDF5 offers the possibility to perform a lossless compression on all datasets. The compression is performed using the "zlib" library. The zlib library is a free, general-purpose lossless data-compression library for use on virtually any computer hardware and operating system (for information see: <http://www.gzip.org/zlib/>). The compression level is specified by an integer between 1 and 9 (inclusive) where 1 results in the fastest compression while 9 results in the best compression ratio. The default value is 6 being an optimum between speed and compression ratio. Upon extraction of datasets from a HDF5 file, the decompression is handled automatically by the HDF5 library. Compression should be performed to all datasets in the KNMI HDF5 format.

5.2 Data types

The Type column defines the data type of the field. This is one of Int, Long, Float, Double, String, Table

and Image for integer, long integer, floating point number, floating point number with double precision, text, tables and images, respectively. The physical size occupied on storage by the given field is not a concern for this document. The underlying software hides this fact from the user. When reading a float value from the HDF file, the retrieved value will be in the native float format on the target computer, regardless of where the file was originally created.

5.3 Mandatory and optional fields

The issue of mandatory and optional fields in the file is not straightforward. A general HDF browser will be able to read data from the file no matter which fields are required (mandatory) and which are optional. This will also be the case for browsers (viewers) which understand the structure of KNMI HDF5 files. For example, if an image is map projected, the map projection group must be present and contains a set of fields, which are required in that context. If the image is not map projected, the map projection group does not have to be present.

Explanation of M/O column:

M= Mandatory field. This flag is used to indicate whether this field is either required for further processing using HDF5 libraries or required by the Omnivoor/Beelden database and processing system. All fields, having an M attached, should be included in the KNMI HDF5 tag. If not, the field is not applicable to the type of image data set stored in the file or the group (in which the field is included) is optional and not present in the HDF5 file.

O= Optional field. This flag indicates fields that are included in the tag for the purpose of the user, to provide background information or to help the user.

5.4 Overview group

This group provides an overview of the dataset. It contains references, identifiers, a representative quicklook of the image data and fields describing the content of the file.

Tag Name	D/A	Type	O/M	Description	Convention/Example
product_group_name	A	String	M	Name of the productgroup; used as dataset identifier in the OMBE system	See chapter 7; e.g. "METEOSAT_7_MVIRI_ECMWF"
products_missing	A	String	M	Product_name of products included in the productgroup but missing in the HDF file. In case of more than one missing product, names are comma separated.	See chapter 7; e.g. METEOSAT_7_MVIRI_VIS_ECMWF, METEOSAT_7_MVIRI_VIS_ECMWF
product_datetime_start	A	String	M	Start date and time stamp of product(s) in HDF5 file	See chapter 7; e.g. "01-JAN-2002;12:34:40.120"
product_datetime_end	A	String	M	End date and time stamp of product(s) in HDF5 file (maybe the same as start date) ; Start and end date together define the period for which all products (observations and/or forecasts) included in the HDF5 file are valid; e.g. in case of a time serie the start date/time defines the date/time of the first image in the time serie and the end date/time defines the date/time of the last image.	See chapter 7; e.g. "01-JAN-2002;12:49:34.880"
abbttitle	A	String	O	Dataset identification: abbreviated title for the dataset	
product_group_title	A	String	O	Dataset identification: the explicit name of the geographic dataset, to sufficiently identify it by the users	
product_group_doc	A	String	O	Documentation reference	
hdftag_version_number	A	String	M	Version number of KNMI HDF-tag	="3.5"
hdf5_url	A	String	O	Internet reference (for the documentation) of HDF5	="http://hdf.ncsa.uiuc.edu/hdf5"
hdftag_url	A	String	O	Internet reference of KNMI HDF5 tag	="http://www.knmi.nl/onderzk/imag efomat/bik_web/HDF5_info/HDF5 _intro.htm"
dataset_summary	A	String	O	dataset overview: a brief textual description of the geographic dataset which summarises the content of the geographic dataset	
dataset_org_descry	A	String	O	dataset overview: description of the organisation responsible for commissioning production (together with a statement of status if appropriate)	
dataset_raster_type	A	String	O	dataset overview: type of raster data (spatial data, aerial data, semantic data or grid data, etc.)	
dataset_raster_descry	A	String	O	dataset overview: description of raster data, depending on the type of raster data	

dataset_sample	D	Image	M ¹	Sample of image_data as stored in the Image group. The sample is meant to show an overview of the dataset. This can be a RGB color composite of image layers stored in the HDF file.	The NCSA specs are implemented here; see: http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html	
CLASS	CLASS	A	String	M	Identifier to interpret dataset as image	= "IMAGE"
	IMAGE_SUBCLASS	A	String	O	Indicator of type of palette that should be used	= "IMAGE_TRUECOLOR" or "IMAGE_INDEXED"
	IMAGE_COLORMODEL	A	String	O	Indicator of color model of palette that should be used	= "RGB"
	IMAGE_VERSION	A	String	M	Version number of image specification	= "1.2"
	DISPLAY_ORIGIN	A	String	M ³	Indicator at which corner pixel (0,0) should be viewed	= "UL" or "LL" or "UR" or "LR"
	PALETTE	A	REF_OBJ	O	Object reference pointer to a color palette in "Visualisation" group	
	INTERLACE_MODE	A	String	O	Indicator of storage layout of image data	= "INTERLACE_PIXEL" or "INTERLACE_PLANE"
dataset_sample_descr	A	String	O	Description of dataset_sample, channels used, ...	e.g. "R: channel_1,G: channel_2,B: channel_4"	
dataset_meta_language	A	String	O	metadata language: language used in the textual statements	e.g. "CEN"	
number_image_groups	A	Int	M ²	Number of image groups present in file	1 or 2 etc.	
number_profile_groups	A	Int	M ²	Number of profile groups present in file		
number_discharge_groups	A	Int	M ²	Number of discharge groups present in file		
number_visualisation_groups	A	Int	M ²	Number of visualisation groups present in file		
number_satellite_groups	A	Int	M ²	Number of satellite groups present in file		
number_radar_groups	A	Int	M ²	Number of radar groups present in file		
number_station_groups	A	Int	M ²	Number of station groups present in file		
number_classification_groups	A	Int	M ²	Number of classification groups		

¹ : Mandatory, only in case the size of the images, stored in the HDF5 file, makes the inclusion of a quicklook useful.

² : Mandatory, only in case the relevant group is represented in the file (so, in case the value of the tag is zero than the tag is not mandatory).

³ : Mandatory, only in case the (0,0) pixel should **not** be displayed in the upper-left corner.

5.5 Geographic group

All information about the geographic reference of the dataset is placed in this group. This includes map projection and parameters used in this operation. By definition all image data included in one HDF5 dataset have the same georeferences. So, the metadata included in this group applies to all images, point, grid, vector and classification data that are stored in the HDF5 file (only **one** Geographic group is included in an HDF5 file).

Name	D/A	Type	O/M	Description	Comment
geo_number_columns	A	Int	M	Number of pixels per line in images (x dimension)	e.g. 256
geo_number_rows	A	Int	M	Number of pixel lines in images (y dimension)	e.g. 256

geo_pixel_size_x	A	Float	M	Horizontal pixel size in projection plane (x-axis positive towards east)	e.g. 2.5
geo_pixel_size_y	A	Float	M	Vertical pixel size in projection plane (y-axis positive towards north)	e.g. -2.5
geo_par_pixel	A	String	M ³	Geophysical parameters of image coordinates (horizontal,vertical)	e.g. "X,Y"
geo_dim_pixel	A	String	M	Dimensions of image pixel size (horizontal,vertical)	e.g. "KM,KM"
geo_column_offset	A	Float	M	Geographic column offset of pixel (0,0) from origin of projection plane	e.g. 200
geo_row_offset	A	Float	M	Geographic row offset of pixel (0,0) from origin of projection plane	e.g. 100
geo_pixel_def	A	String	M	Definition of lat/lon position inside the image pixels	e.g. "CENTRE" or "LU" etc.
geo_product_center	A	Table of Float	M ¹	Latitude and longitude at the centre point of the image	Tab(Lon, Lat)
geo_product_corners	A	Table of Float	M ¹	Latitude and longitude of each of the four product corners (starting with southwest corner and then clockwise)	Tab(Lon1, Lat1, Lon2, Lat2, Lon3, Lat3, Lon4, Lat4)
geo_ref_tiepoints	D	Table of Float	M ²	List of points mapping pixel co-ordinates (x,y) to (lon,lat). The purpose of this table is to enable a geo navigation in the image by users.	Tab(X1,Y1,LON1,LAT1,X2,Y2,LON2,LAT2,)
geo_navigation_accuracy	A	Int	O	Estimated navigation error (number or pixels)	e.g. 1

¹ : One of these tags is mandatory

² : Mandatory, only in case of geographical image data without a valid projection

³ : Mandatory, only in case of non-geographical image data

Map projection subgroup

Name	D/A	Type	O/M	Description	Convention/Example
projection_indication	A	String	M	Projection indication	"Y" or "N"
projection_name	A	String	M	Name of the projection according to strict naming convention	One of: "STEREOGRAPHIC", "MERCATOR", "SATELLITE_VIEW"
projection_descr	A	String	O	Description of the projection (formulas) or reference to URL	" http://www..... "
projection_proj4_params	A	String	M ¹	List of space separated tag=value pairs to be used with the PROJ4 projection library. Each tag is preceded with a +. The URL of the PROJ4 document and library is referenced in "projection_descr" (in this case http://www.remotesensing.org/proj/)	e.g. "+proj=stere +a=6378.4 +b=6356.9 +lat_0=90.0 +lon_0=0.0 +lat_ts=60.0"
projection_semi_major_axis	A	Float	O ²	Semi major axis (earth radius at Equator) in "geo_dim_pixel" units	6378.4
projection_semi_minor_axis	A	Float	O ²	Semi minor axis (earth radius at pole) in "geo_dim_pixel" units	6356.9
projection_fplat	A	Float	O ²	Fundamental point in decimal latitude degrees (-90.0 - 90.0)	e.g. 90.0
projection_fplon	A	Float	O ²	Fundamental point in decimal longitude degrees (-180.0 - 360.0)	e.g. 0.0
projection_lat_true_scale	A	Float	O ²	Latitude of true scale in decimal degrees (-90.0 – 90.0)	e.g. 60.0
projection_def_v1	A	Float	O ²	deflection of vertical 1	
projection_def_v2	A	Float	O ²	deflection of vertical 2	

projection_def_v3	A	Float	○ 2	deflection of vertical 3	
projection_std_meridian_1	A	Float	○ 2	standard meridian 1	
projection_std_meridian_2	A	Float	○ 2	standard meridian 2	
projection_std_meridian_3	A	Float	○ 2	standard meridian 3	
projection_std_par_1	A	Float	○ 2	standard parallel 1	
projection_std_par_2	A	Float	○ 2	standard parallel 2	
projection_std_par_3	A	Float	○ 2	standard parallel 3	
projection_scale_factor	A	Float	○ 2	Scale factor at the central meridian	
projection_zone	A	String	○ 2	UTM zone number	
projection_height	A	Float	○ 2	Height of satellite platform in "geo_dim_pixel" units	e.g. 35785.831

¹ and ²: The **projection_proj4_params** tag is only mandatory in case the image data is projected. The tag contains all projection information needed to perform a projection using the PROJ4 library. If the **projection_proj4_params** tag is not available, then the other tags containing projection parameters must be filled.

5.6 Image group

All data related to the image data and the displaying of such must be placed in this group. There can be more than one Image group included in the HDF5 file.

Name	D/A	Type	O/M	Description	Convention/Example
image_product_name	A	String	M	Name of product stored in the Image group	See chapter 9; e.g. "METEOSAT_7_MVIRI_VIS_ECMWF"
image_source_ref	A	Table of REF_OBJ	O	Reference to "source" metadata as stored in Satellite, Radar or Lightning groups (see paragraphs 5.8-5.10). In case of "fused" products this tag may contain more than one reference to different groups.	
image_data	D	Image	M	Image, stored in 2-dimensional array of "Nrows x Ncolumns"	The NCSA specs are implemented here; see: http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html
CLASS	A	String	M	Identifier to interpret dataset as image	= "IMAGE"
IMAGE_SUBCLASS	A	String	O	Indicator of type of palette that should be used	= "IMAGE_TRUECOLOR" or "IMAGE_INDEXED" or "IMAGE_BITMAP"
IMAGE_COLORMODEL	A	String	O	Indicator of color model of palette that should be used	= "RGB"
IMAGE_WHITE_IS_ZERO	A	Int	O	Used in case of "IMAGE_BITMAP" to define bit	0 = false, 1 = true
IMAGE_VERSION	A	String	M	Version number of image specification	= "1.2"
DISPLAY_ORIGIN	A	String	M ⁴	Indicator of at which corner pixel (0,0) should be viewed	= "UL" or "LL" or "UR" or "LR"
PALETTE	A	REF_OBJ	O	Object reference pointer to colour palette in "Visualisation" group	
image_size	A	Long	M	The total size of the image data in bytes	e.g. 1123078
image_bytes_per_pixel	A	Int	M	Number of bytes used to represent a pixel value	e.g. =1, for one-byte-per-pixel data

image_geo_parameter	A	String	M	Geophysical parameter or quantity with unit represented in image	e.g. ="TEMPERATURE_[K]"	
image_preview	D	Image	M ¹	Thumbnail of image_data (for quicklook purposes). E.g. including every 10th pixel column and row of original image.	The NCSA specs are implemented here; see: http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html	
	CLASS	A	String	M	Identifier to interpret dataset as image	= "IMAGE"
	IMAGE_SUBCLASS	A	String	O	Indicator of type of palette that should be used	= "IMAGE_TRUECOLOR" or "IMAGE_INDEXED" or "IMAGE_BITMAP"
	IMAGE_COLORMODEL	A	String	O	Indicator of color model of palette that should be used	= "RGB"
	IMAGE_WHITE_IS_ZERO	A	Int	O	Used in case of "IMAGE_BITMAP" to define bit	0 = false, 1 = true
	IMAGE_VERSION	A	String	M	Version number of image specification	= "1.2"
	DISPLAY_ORIGIN	A	String	M ⁴	Indicator of at which corner pixel (0,0) should be viewed	= "UL" or "LL" or "UR" or "LR"
	PALETTE	A	REF_OBJ	O	Object reference pointer to colour palette in "Visualisation" group	
image_start_obs	A	String	M ²	Date and time when image observation is started	See chapter 7; e.g. "01-JAN-2002;12:34:40.120"	
image_end_obs	A	String	M ²	Date and time when image observation is ended	See chapter 7; e.g. "01-JAN-2002;12:49:34.880"	
image_datetime_valid	A	String	M ⁵	Date and time for which the forecast product being stored in this image group is valid.	See chapter 7; e.g. "01-JAN-2002;12:49:34.880"	
image_number_image_obs	A	Int	M ³	Number of image observations included in composites.	e.g. 10	
image_obs_timestamp	A	Table of String	M ³	Timestamp for each image included in the composite. UTC time stamps of each image, included in composite, are listed. Timestamps are comma separated	For time stamps convention see chapter 7. E.g. Tab(01-JAN-2002;12:34:40.120,02-JAN-2002;12:24:32.130,...)	

¹ : Mandatory, only in case the size of the image, stored in the Image group, makes the inclusion of a preview useful (e.g. > 256 x 256 pixels in an original image).

² : Mandatory, only in case the observation times of the image deviate from the observation times included in the Overview group

³ : Mandatory, only in case of a composite image.

⁴ : Mandatory, only in case the (0,0) pixel should **not** be displayed in the upper-left corner.

⁵ : Mandatory, only in case of a forecast image

Calibration subgroup

Name	D/A	Type	O/M	Description	Convention/Example
calibration_flag	A	String	M	Is image data calibrated? Is there a fixed relation between the pixel bytes and a geophysical parameter?	"Y" = data is calibrated "N" = data is uncalibrated (raw)
calibration_level	A	String	O	Level of calibration? E.g. the NASA definition: 0 = raw, 1 = sensor calibrated, 2 = atmospheric corrected, etc.	e.g. "NASA level 0"
calibration_reference	A	String	O	Reference to calibration data used (e.g. NOAA tables); e.g. NOAA URL	" http://www..... "
calibration_formulas	A	String	M ¹	Text representation of formulas used for conversion of pixel values (PV) to geophysical parameter or quantity (GEO). The layout is fixed.	e.g. "GEO=0.933*PV+1.444"

calibration_table	D	Table of float	M ¹	Table containing mapping between pixel values and calibrated geophysical parameter or quantity	e.g. see chapter 6, table 1.1 and 1.2
calibration_missing_data	A	Int	M	Pixel value representing missing data (e.g. bad or missed lines)	e.g. 255
calibration_out_of_image	A	Int	M	Pixel value representing "out of image" or "out of range"	e.g. 255
calibration_annotation_tables	D	Table of string	O	Table containing mappings from pixel values to some textual values (for the purpose of annotation).	e.g. see chapter 6, table 1.3

¹ Either a formula or table is needed in this subgroup. In case of a simple formula (e.g. linear cases) a formula is preferred, otherwise a table.

Statistics subgroup

Name	D/A	Type	O/M	Description	Convention/Example
stat_min_value	A	Float	M	Minimum geophysical pixel value present in image (one value for each image)	e.g. 260.5
stat_max_value	A	Float	M	Maximum geophysical pixel value present in image (one value for each image)	e.g. 315.6
stat_min_value_5	A	Float	O	Minimum geophysical pixel value after 5% of the pixels are clipped from the low end of the histogram (one value for each image)	e.g. 261.5
stat_max_value_5	A	Float	O	Maximum geophysical pixel value after 5% of the pixels are clipped from the high end of the histogram (one value for each image)	e.g. 299.7
stat_histogram	A	Table of Long	O	Histogram for image in the image group	e.g. Tab(0,0,25,50,300,...)
stat_bin_count	A	Integer	O	Number of bins used in the histogram	e.g. 15
stat_bin_size	A	Integer	O	Size of each bin (in counts)	e.g. 17
stat_std_dev	A	Float	O	Standard deviation of pixel values in image	e.g. 80.22
stat_mean	A	Float	O	Mean of pixel values in image	e.g. 155.5

Satellite subgroup

All information related to the quality and to the processing of the image/VAP is placed in this group. This includes level of processing, e.g. level1 (pan corrected and orthorectified), level2 (map projected). Other information that can be placed here is a reference to the algorithms used by the processor. Reference to external processing procedures, e.g. NOAA. There can be more than one Satellite group included in the HDF5 file.

Name	D/A	Type	O/M	Description	Convention/Example
satellite_product	A	String	O	Description of satellite product contained in image	e.g. "AVHRR_CHANNEL_1"
image_acquisition_time	A	String	O	UTC when the input data for this image dataset was received on site (e.g. at KNMI)	For UTC convention see chapter 7
image_generation_time	A	String	O	UTC when this image dataset was produced	For UTC convention see chapter 7
image_processing_level	A	String	O	Level of processing	Any convention can be used here

image_processing_software	A	String	O	Processing software used: reference to algorithms used	Can be a URL (where a document is located)
image_processing_hist	A	String	O	Processing history: listing of processing steps	e.g. "CALIBRATED, NAVIGATED, PROJECTED"
navigation_processing_hist	A	String	O	Navigation processing history: description of navigation steps	e.g. "LANDMARK_CORRECTION"
image_accuracy_indication	A	String	O	Accuracy indication of geophysical pixel value: e.g. for SST image (static information)	e.g. "0.5 C"
image_quality_indication	A	String	O	Indication if quality metadata available	"Y" = quality fields included "N" = quality fields not included
image_quality_ingest	A	String	O	Ingest quality parameters	
image_quality_ber	A	Double	O	Bit Error Rate	
image_nbr_missing_lines	A	Int	O	Number of detected missing lines	
image_missing_line_numbers	A	Table of Int	O	The line numbers of the missing lines	Tab(1, 103, ...)
missing_lines_correction_procedures	A	String	O	Procedure used to correct for missing or bad lines.	e.g. "DUPLICATE_LAST" or "IGNORE" or "BLEND" or "NONE"
lineage_id	A	Long	O	image quality: unique id for the process history of the data set	
lineage_method	A	String	O	image quality: method used for creating or processing data set	
lineage_organisation	A	String	O	image quality: organisation applying the method on the dataset	
lineage_start_date	A	String	O	image quality: startdate of process	
lineage_end_date	A	String	O	image quality: enddate of process	
lineage_purpose	A	String	O	image quality: purpose	
lineage_confidence	A	String	O	image quality, metaquality: confidence	
lineage_reliability	A	String	O	image quality, metaquality: reliability	
lineage_methodology	A	String	O	image quality, metaquality: methodology	
lineage_abstraction	A	String	O	image quality, metaquality: abstraction effect	

Radarsubgroup

Name	D/A	Type	O/M	Description	Convention/Example
radar_product	A	String	O	Description of radar product contained in image	e.g. "PCAPPI"
radar_product_parameter	A	String	O	Parameter specifying radar product, e.g. height of CAPPI	e.g. "0.8KM"
radar_method	A	String	O	Description of method used to derive radar product	e.g. "Interpolation"
radar_quality	A	String	O	Indication of quality	e.g. "The_best"
radar_elevation	A	Float	O	Elevation of radar beam in degrees	e.g. 0.3

radar_rotation	A	Float	O	Azimuthal speed of radar antenna in degrees/sec.	e.g. 18
radar_prf_low	A	Float	O	Low PRF used during acquisition of radar data in Hz.	e.g. 750
radar_prf_high	A	Float	O	High PRF used during acquisition of radar data in Hz.	e.g. 1000

5.7 Profile group

This group is intended for storage of vertical profiles from e.g. weather radars, sodars, or profilers, and it can be extended from radiosonde observations of model profiles. The values of a variable at the different levels/heights in a profile are stored in a datasets. Different datasets are proposed for several variables and additional datasets can be included. The only mandatory dataset is the one described the geometric height of the vertical levels in the profile.

Name	D/A	Type	O/M	Description	Convention/Example
profile_number_levels	A	Int	M	Number of vertical levels in profile, i.e., length of profile datasets	e.g. 15
profile_missing_data	A	Float	M	Float value indicating "missing data" in profile datasets	e.g. -9999.0
profile_height	D	Table of Float	M	Dataset with geometric height of vertical levels in profile in meter	e.g. Tab(100,300, ...)
profile_u_wind	D	Table of Float	O	Dataset with east-west component of wind in m/s	e.g. Tab(-5.0,13.0,...)
profile_v_wind	D	Table of Float	O	Dataset with north-south component of wind in m/s	e.g., Tab(20.0, 15.4,...)
profile_w_wind	D	Table of Float	O	Dataset with vertical component of wind in m/s	e.g. Tab(0.5, -1.0,...)
profile_radial_stddev	D	Table of Float	O	Dataset with standard deviation of radial velocity as deduced from the wind model fit used in profile extraction in m/s	e.g. Tab(1.0,1.5, ...)
profile_reflectivity	D	Table of Float	O	Dataset with reflectivity factor at levels in profile in dBZ	e.g. Tab(-31.5, -20.5, ...)
profile_number	D	Table of Int	O	Dataset with number of points used to compile values at each level	e.g. Tab(1000,400, ...)

5.8 Discharge group

This group is intended for storage of the timeseries of the localizations observed by a lightning detection system. The timeseries of each parameter detected by the system is stored as an array in a dataset. The data and time of each discharge event (localization) is given as a time offset in seconds against a reference data and time which is stored in this group as well. The time offset is stored in a dataset of doubles to allow for storage of a high accuracy time offset (microsec.) over a daily period. A number of parameters describing the characteristics of a discharge have been defined, but addition of other datasets is possible.

Name	D/A	Type	O/M	Description	Convention/Example
number_discharges	A	Int	M	Number of discharges in timeseries, i.e., length of discharge datasets	e.g. 100

reference_datetime	A	String	M	Date and time stamp against which discharges are referenced	See chapter 7; e.g. "01-JAN-2002;12:34:40.120"
time_offset	D	Table of Double	M	Dataset with time offsets of discharges with respect to reference date and time in sec., double allows for microsec accuracy	e.g., Tab(0.001,0.0011, ...)
longitude	D	Table of Float	M	Dataset with geographical longitudes of discharges in decimal degrees	e.g., Tab(4.78,5.17,...)
latitude	D	Table of Float	M	Dataset with geographical latitudes of discharges in decimal degrees	e.g. Tab(51.1,52.9, ...)
event_type	D	Table of Char	O	Dataset with types of observed discharges: "0" single-point, "1" start of CC, "2" CC discharge, "3" end of CC, "4" CG stroke, "5" CG return stroke	e.g. Tab(1,2,2,3,...)
position_error	D	Table of Float	O	Dataset with position errors of CG stroke localizations as deduced by detection system in meter	e.g. Tab(500,1500,..)
rise_time	D	Table of Float	O	Dataset with rise times of induced current for detected CG strokes in sec.	e.g. Tab(0.000010,0.000020, ...)
decay_time	D	Table of Float	O	Dataset with decay times of induced current for detected CG strokes in sec.	e.g. Tab(0.000125,0.0005,...)
current	D	Table of Float	O	Dataset with estimated currents of CG strokes in ampere	e.g., Tab(100000,250000,)

5.9 Visualisation group

In this group a color_palette is stored. Documentation on how the use of color_palettes is implemented in the HDF5 tag is described in chapter 8. There can be more than one Visualisation group stored in one HDF5 file.

Name	D/A	Type	O/M	Description	Convention/Example
color_palette	D	Table	M	Color palette in RGB format: two-dimensional array of "Ncolors x 3"	The NCSA specs are implemented here; see: http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html
CLASS	A	String	M	Identifier to interpret dataset as color palette	= "PALETTE"
PAL_VERSION	A	String	M	Version number of palette specification	= "1.2"
PAL_TYPE	A	String	M	Definition of type of palette, usually a directly indexed array	= "STANDARD8"
PAL_COLORMODEL	A	String	M	Definition of the color model used in palette, usually RGB	= "RGB"

5.10 Satellite group

In this group, mostly static information regarding the satellite is placed. This includes the position of the satellite at the time of data capturing. Also descriptions of the on-board instruments can be placed here (e.g. spectral bands related to channels). Moreover, in case of polar orbiting satellites actual orbit information is stored.

Name	D/A	Type	O/M	Description	Convention/Example
satellite_name	A	String	M	Name of satellite (e.g. NOAA)	e.g. "NOAA" or "METEOSAT", etc.
satellite_id	A	String	M	ID of satellite (e.g. 16)	e.g. 16 or 7
satellite_description	A	String	O	Textual description of satellite or reference to URL	e.g. http://www.eumetsat.de/
satellite_agency	A	String	O	Owner/agency of satellite (e.g. RADARSAT International)	e.g. "EUMETSAT"
satellite_platform_type	A	String	O	Polar orbiting, geostationary, ground based	e.g. "GEOSTATIONARY"
satellite_platform_position	A	String	M	Position relative to ground of the satellite (or instrument in case of weather radar)	e.g. "AT CROSSING POINT OF EQUATOR AND MERIDIAN"
satellite_launch_date	A	String	O	Start Date of the source identifier (can be launch date of satellite or first date of first operational data delivery).	e.g. "17 MARCH 1996"
satellite_acquisition_station	A	String	O	Groundstation where the data is gathered from the satellite	e.g. "KNMI in De Bilt"
satellite_asc_desc_flag	A	String	M ¹	Satellite pass is ascending or descending	"A" = ascending; "D" = descending;
satellite_subtrack	A	Table of Float	M ¹	Satellite sub track (at nadir) ; Two pairs of Lat/Lon are included: the position of the satellite at start_obs and at end_obs.	Tab(Lat(start), Lon(start), Lat(end), Lon(end))
satellite_orbit_number	A	Int	O ¹	Orbit number for pass	e.g. 12388
satellite_orbit_prop_ind	A	String	O ¹	Indicator which orbit proposition data is present in group	"TBUS" = TBUS data is present "TLE" = Two line elements is present "SV" = State vector is present "NONE" = no orbit prop data is present
satellite_orbit_prop	A	String	O ¹	State vector, or TBUS message or two line elements used in the navigation processing. For info see: http://noaasis.noaa.gov/NOAASIS/ml/navigation.html	The TBUS, state vector or two-line elements in the form of a text string
satellite_orbit_prop_age	A	String	O ¹	Age of state vector (or TBUS or two-line elements). Date of the TBUS message, state vector or two-line elements.	20-JAN-2002

¹ These fields are included only in case of polar orbiting satellites.

Sensor subgroup

Name	D/A	Type	O/M	Description	Convention/Example
sensor_name	A	String	M	Name of instrument/sensor (of which data is included in dataset)	e.g. "MVIRI" or "AVHRR"
sensor_type	A	String	O	Type of sensor	e.g. "IMAGER" or "SOUNDER"
sensor_descr	A	String	O	Short description sensor or instrument	e.g. "ADVANCED VERY HIGH RESOLUTION RADIOMETER"
sensor_descr_long	A	String	O	Long description sensor or instrument or reference to URL	e.g. http://www.eumetsat.de/
sensor_number_channels	A	Int	O	Number of channels	e.g. 5

sensor_desc_channels	A	String	O	Description of channels(waveband, signal/noise ratio)	See chapter 7
sensor_radiometric_res	A	String	O	Radiometric resolution of sensor (Number of bits)	e.g. "10 BITS"
sensor_spatial_res	A	String	O	Spatial resolution (NADIR in km)	e.g. "1.1 KM"
sensor_temporal_res	A	String	O	Temporal resolution or repetition frequency	e.g. "9 DAYS" or "30 MINUTES"

5.11 Radar group

Name	D/A	Type	O/M	Description	Convention/Example
radar_id	A	String	M ¹	ID of radar station or WMO station number	e.g. "NL50"
radar_name	A	String	M ¹	Name of radar station	e.g. "DE_BILT"
radar_location	A	Table of float	M	Longitude and latitude of radar station in decimal degrees	e.g. Tab(5.17, 52.10)
radar_height	A	Float	O	Height above mean-sea level of radar antenna feed in meters	e.g. 48.0
radar_num_contrib	A	Int	O	Number of scans that this radar has contributed to image data	e.g. 96
radar_adjustment	A	String	O	String describing adjustment applied to radar reflectivity observations	e.g. "F=0.0+0.01*range"
radar_system	A	String	O	Description of radar hardware and manufacturer	e.g. "METEOR_360AC"
radar_software	A	String	O	Description of radar processing software	e.g. "RAINBOW"
radar_wavelength	A	Float	O	Wavelength of radar in cm	e.g. 5.2
radar_beamwidth	A	Float	O	3dB width of radar beam in degree	e.g. 1.0
radar_angles	A	Table of float	O	Elevations of radar antenna used in degree	e.g. Tab(0.3, 1.1, 2.0, 3.0)

¹ One of these tags is required.

5.12 Station group

Name	D/A	Type	O/M	Description	Convention/Example
station_id	A	String	M ¹	ID of detection station or WMO station number	e.g. "NL06"
station_name	A	String	M ¹	Name of detection station	e.g. "HOOGVEEN"
station_location	A	Table of float	M	Longitude and latitude of detection station in decimal degrees	e.g. Tab(5.17, 52.10)
station_height	A	Float	O	Height above mean-sea level of detection station in meters	e.g. 48.0
station_availability	D	Table of Char	O	Array with availability reports in 1-minute intervals. For each 1-minute interval the number of seconds with a properly functioning station is reported.	e.g. Tab(60,60,...)

station_number_availability	A	Int	O	Number of 1-minute availability reports in "station_availability" array	e.g. 5
station_system	A	String	O	Description of detection hardware and manufacturer	"SAFIR"

¹ One of these tags is required.

5.13 Classification group

This group is meant to store "classified" image data. E.g. a land/sea/cloud mask that can be used for presentation purposes or to enable the user to do calculations only for certain surfaces (e.g. SST's only for cloudfree sea pixels). The classification group contains one image, where the pixel values are encoded to represent different features in the image.

Name	/A	Type	/M	Description	Convention/Example
classification_flag	A	String	M	YES means that the image has been classified, NO otherwise	"Y" or "N"
classification_type	A	String	M	Type of classification	e.g. "LANDSEACLOUDMASK"
classification_image	D	Image	M	An image mask with encoded values (X/Y array)	The NCSA specs are implemented here; see: http://hdf.ncsa.uiuc.edu/HDF5/doc/ADGuide/ImageSpec.html
CLASS	A	String	M	Identifier to interpret dataset as image	= "IMAGE"
IMAGE_SUBCLASS	A	String	O	Indicator of type of palette that should be used	= "IMAGE_TRUECOLOR" or "IMAGE_INDEXED" or "IMAGE_BITMAP"
IMAGE_COLORMODEL	A	String	O	Indicator of color model of palette that should be used	= "RGB"
IMAGE_WHITE_IS_ZERO	A	Int	O	Used in case of "IMAGE_BITMAP" to define bit color	0 = false, 1 = true
IMAGE_VERSION	A	String	M	Version number of image specification	= "1.2"
DISPLAY_ORIGIN	A	String	M ¹	Indicator of at which corner pixel (0,0) should be viewed	= "UL" or "LL" or "UR" or "LR"
PALETTE	A	REF_OBJ	O	Object reference pointer to colour palette in "Visualisation" group	
classification_table	D	Table of string	M	A table with information on how to interpret the encoded values in the image mask	e.g. see chapter 6, table 1.3

¹ : Mandatory, only in case the (0,0) pixel should **not** be displayed in the upper-left corner.

Processing subgroup

Name	D/A	Type	/M	Description	Convention/Example
processing_software	A	String	O	Reference to the processing software. E.g. a URL	" http://www... "
processing_algorithm	A	String	O	Reference to algorithm or method used. E.g. a URL	" http://www... "

6 Data types

In this section some of the data types found in KNMI HDF5 Data Format Specification are described.

Data types	Description
Int	Integer value
Long	Long integer value
Float	Float value
Double	Double value
String	Character-array
Image	Two-dimensional array of 1,2, 4 byte values
Table	One/two dimensional array of data values

6.1 Table

A table is a one or two dimensional array of data elements.

Calibration/annotation/classification tables are two dimensional arrays. These tables contain two columns. The first column contains the pixel values of a product. The second column contains the related calibrated value or a text representation. In HDF5 all elements of a table need to be of the same data

type. So, if text is included in the second column then the elements in the first column (the pixel values) need to be also string typed.

Not all pixel values need to be represented in the table. E.g. in case of linear calibrations it is allowed to exclude one or pixel values in the first column (the idea behind this is that tables become a lot shorter). The excluded pixel values can then be calculated by the user by interpolation between the neighbouring pixel values that are included in the table.

Table 1.1 Table showing calibration of pixel values in channel 4 to °C

Integer value	Calibrated value
0.0	-50.0
1.0	-49.5
2.0	-49.0
...	...
251.0	75.5
252.0	76.0
253.0	76.5
254.0	77.0
255.0	77.5

Table 1.2 Same example as table 1.1, but now only the first and last pixel value are included.

The values in between can be calculated by interpolation.

Integer value	Calibrated value
0.0	-50.0
255.0	77.5

Table 1.3 Table showing annotation of values in a classification image. Table showing annotation of values in a classification image.

Data value	Classification
"0"	"Unclassified"
"1"	"Sea"
"2"	"Land"
"3"	"Clouds"
"4"	"Snow"

7 Conventions

7.1 Timestamps

When timestamps are represented as text they shall have the following convention: **DD-MON-YYYY;HH:MM:SS.sss**.

DD	two digit number representing the day of the month, e.g 05.
MON	three character normal english abbreviated month name, e.g. JAN, FEB, MAR, and so on.
YYYY	four digit year number, e.g. 2000.
HH	two digit number representing the hour of the day, e.g. 08,
MM	two digit number representing the minute of the hour, e.g 58.
SS	two digit number representing the seconds of the minute, e.g 23.
sss	three digit number representing the milliseconds of the second, e.g 549

Full example: 05-JAN-2000;08:58:23.549

7.2 HDF5 file naming convention

Within the image data infrastructure of KNMI (BIK), files are generated, stored and dispatched. Within BIK there is a convention for the names of the generated HDF5 files:

`<product_group_name>[subsystem][processing_param]<timestamp>.H5`

The fields within <..> are mandatory while the one within [...] are optional. Note that there can be more than one processing_param, they will then be separated by '_':

Explanation of fields:

- product_group_name = Product group name as defined within OMNIVOOR-beelden (see chapter 9)
- subsystem =
 - automatic_production -> AP,
 - interactive_production (via Web) -> USERNAME
 - rt server -> RT,
- processing param (when file is generated in production shell of Omnivoor/Beelden)=
 - IMAGE_WARP, SUBSET,
 - COMPOSITION,
 - PIXEL_CONVERT
 - LAYER_EXTRACT
- timestamp = time when file was generated in BIK. Convention: YYYYDDMMHHMM (year-month-day-hour-minute)

7.3 Product_group_name convention

Product_group_names and product_names are to be generated by the frontend systems and then to be included in the HDF5 files. The convention for the names is defined in the Omnivoor/Beelden database project and the frontends need to confirm to this convention. The product_group_name is primarily used to link the dynamic metadata in the HDF5-file to the Omnivoor metadata database. Each productgroup contains a unique set of one or more products as defined by the administrator/ manager of BIK. The product_group_name will be a unique static identifier to classify an image-productgroup and will be included in the "overview" group of the HDF5 tag.

The following **general** convention rules are implemented in BIK:

- The product_group_name is uniquely linked to a **defined set of products**.
- The product_group_name is a string that has **variable length** but a **maximum length of 50** characters.
- The product_group_name always contains **three** "underscores".
- In case it is impossible to fill in something, NA (not applicable) has to be filled in.
- The product_group_name is always written in **CAPITALS**.

The naming convention and examples for three data types (satellite, radar and lightning) is described.

I. Satellite Data

The product_group_name is divided in four parts, describing:

1. the **source type** (e.g. NOAA, METEOSAT, MSG)
2. the mission **ID** of satellite (e.g. 16, 7, 1)
3. type of **sensor(s)** (e.g. AVHRR, MVIRI, SEVIRI)
4. a **miscellaneous** field for geography or other information (e.g. FULL PASS, GLOBE, ECMWF)

Examples:

Sourcetype	_ID	_sensor	_miscellaneous	product_group_name
METEOSAT	7	MVIRI	GLOBE	METEOSAT_7_MVIRI_GLOBE
MSG	1	SEVIRI	ECMWF	MSG_1_SEVIRI_ECMWF
MSG	1	SEVIRI	HRVEUROPE	MSG_1_SEVIRI_HRVEUROPE
NOAA	16	AVHRR	FULL PASS	NOAA_16_AVHRR_FULLPASS

II. Radar Data

The product_group_name is divided in four parts, describing:

- The **Source type**. For radar data it is always equal to: **RAD**
- The **Radar ID**. The radar ID consists of four characters: **AAII**. The first two letters indicate the country from which the radar data is originating, and the last two numbers indicate the specific radar or composite content. Country abbreviations and number are taken from OPERA (see below).
- The **Product type**. The type of radar product, e.g. pseudoCAPPI, echotops, is indicated by a three character string in agreement with current CRIS definition (see below)

- A **Miscellaneous field**. The miscellaneous field is used for instance to indicate accumulation time.

Table Radar I. *The AA subfield of the ID is used to define from which country the radar product is originating. Abbreviations are taken from OPERA.*

AA	Country
NL	Netherlands
DL	Germany
UK	United Kingdom
BX	Belgium
FR	France
DN	Denmark

Table Radar II. *The II subfield of the ID is used to define which radar site or composite image is contained by datafile. Definition is in agreement with OPERA standards.*

II	Content
00	Not used.
01-19	Not used for radar data (normally used for 'global distribution')
20-39	Used to identify national and regional composites.
40-89	Used to identify radar site data.
90-99	Reserved but frequently used for test bulletins

Table Radar III. *The Product type field (PPP) is used to specify the type of radar product in accordance with definition in current CRIS/Rainbow.*

PPP	Product
PPZ	Plan-Position Indicator of reflectivity Z, or similarly for velocity V and spectral width W
PCP	PseudoCAPPI of reflectivity

CLT	Cluttermap corresponding to pseudoCAPPI
ETH	Echotop product (general)
ETW	Echotop for Hail Warning (45dBZ)
HAW	Hail Warning Product
VP2	Wind Profile
RAU	Rainfall Accumulation (uncorrected)
RAC	Rainfall Accumulation (corrected)
CAL	Calibration Data
BIT	BITE messages
LOG	LOG messages

Examples of radar product_group_names:

Source	_ID	_Product	_miscellaneous	product_group_name
RAD	NL50	ETH	NA	RAD_NL50_ETH_NA
RAD	NL21	PCP	NA	RAD_NL21_PCP_NA
RAD	NL51	RAU	24H	RAD_NL51_RAU_24H
RAD	NL50	CAL	DBZ	RAD_NL50_CAL_NA
RAD	FR21	PCP	NA	RAD_FR21_PCP_NA
RAD	EU20	PCP	FIRST	RAD_EU20_PCP_FIRST

III. Lightning Data

The product_group_name is divided in four parts, describing:

- The **Source type**. For lightning data it is always equal to: **LGT**

- The **Lightning ID**. The Lightning ID consists of four characters: **AAII**. The first two letters indicate the country from which the lightning data is originating, and the last two numbers indicate the specific content. Lightning data are considered to be a composite, thus it will be in the range 21..40. Country abbreviations and number are taken from OPERA (see below table L.I and L.II).
- The **Product type**. The type of lightning product, e.g. Localization files, Density maps or Hazardous weather forecast, is indicated by a three character string, meaning is stated below in table L.III.
- A **Miscellaneous field**. The miscellaneous field is used for instance to indicate accumulation time.

Table Lightning I. *The AA subfield of the ID is used to define from which country the lightning product is originating. Abbreviations are taken from OPERA. only countries of interest for the Dutch lightning detection system are named here.*

AA	Country
NL	Netherlands
DL	Germany
BX	Belgium
FR	France

Table Lightning II. *The II subfield of the ID is used to define which lightning composite image is contained by datafile. Definition is in agreement with OPERA standards.*

II	Content
20-39	Used to identify national and regional composites.
90-99	Reserved but frequently used for test bulletins

Table Lightning III. *The Product type field (PPP) is used to specify the type of lightning product in accordance with definitions in the current SAFIR system,*

PPP	Product
LAP	Lightning Accumulated Positions
LOG	LOG messages

Examples of lightning product_group_names:

Source	_ID	_Product	_miscellaneous	product_group_name
--------	-----	----------	----------------	--------------------

LGT	NL21	LAP	15M	LGT_NL21_LAP_15M
LGT	NL21	LAP	24H	LGT_NL21_LAP_24H

7.4 Product_name convention

The product_name will be a unique static identifier to classify an image-product and will be included in the "Image" group of the HDF tag . The following **general** convention rules are implemented in BIK:

- The product_name is a string that has **variable length** but a maximum length of **50** characters.
- The Product_name always contains **four** "underscores".
- In case it is impossible to fill in something, NA (not applicable) has to be filled in.
- The product_name is always written in **capitals**.

I. Satellite Data

The product_name is divided in five parts, describing:

1. the **source type** (e.g. NOAA, METEOSAT, MSG)
2. the mission **ID** of satellite (e.g. 16, 7, 1)
3. type of **sensor(s)** (e.g. AVHRR, MVIRI, SEVIRI)
4. description of **product type** (e.g. VIS, IR, CH1, CH5)
5. a **miscellaneous** field for geography or other information (e.g. GLOBE, EUROPE, FULL PASS)

Examples:

Source type	_ID	_sensor	_type	_miscellaneous	Product_name
METEOSAT	7	MVIRI	VIS	GLOBE	METEOSAT_7_MVIRI_VIS_GLOBE
MSG	1	SEVIRI	VIS0.6	ECMWF	MSG_1_SEVIRI_VIS0.6_ECMWF
MSG	1	SEVIRI	HRV	EUROPE	MSG_1_SEVIRI_HRV_EUROPE
NOAA	16	AVHRR	CH1	FULLPASS	NOAA_16_AVHRR_VIS_FULLPASS

II. Radar Data

The product_name is divided in five parts, describing:

1. The **Source type**. For radar data it is always equal to: **RAD**
2. The **Radar ID**. The radar ID consists of four characters: **AAII**. The first two letters indicate the country from which the radar data is originating, and the last two numbers indicate the specific radar or composite content. Country abbreviations and number are taken from OPERA (see Tables R.I and R.II).
3. The **Product type**. The type of radar product, e.g. pseudoCAPPI, echotops, is indicated by a three character string in agreement with current CRIS definition (see Table R.III)
4. The **Product parameter**. Parameter describing the elevation of a PPI, the height of a pseudoCAPPI, threshold of an echotop product, etc...

5. A **Miscellaneous field**. The miscellaneous field is used for instance to indicate accumulation time. Four parts of the **product_name** are equal to the **product_group_name**, and only the product parameter is different. The convention for the product parameter is given below.

Table Radar IV. *The Product Parameter field (PARM) is used to detail the product information. It always consists of one letter and a float (AF.F). The letter indicates the type of parameter and the float its value.*

PPP	PARM	Description
PPZ	E0.3	PPI taken at elevation of 0.3 degrees
PCP	H0.8	CAPPI taken at altitude of 0.8 km
ETH	Z7.0	Echotops using reflectivity threshold of 7.0 dBZ

Examples:

Typesource	_ID	_Product	_Param	_miscellaneous	Product_name
RAD	NL50	PPZ	E0.5	NA	RAD_NL50_PPZ_E0.5_NA
RAD	NL21	ETH	Z7.0	NA	RAD_NL21_ETH_Z7.0_NA
RAD	BX21	PCP	H1.0	NA	RAD_BX21_PCP_H1.0_NA

III. Lightning Data

The product_name is divided in five parts, describing:

- The **Source type**. For lightning data it is always equal to: **LGT**
- The **lightningID**. The lightning ID consists of four characters: **AAll**. The first two letters indicate the country from which the lightning data is originating, and the last two numbers indicate the specific radar or composite content. Country abbreviations and number are taken from OPERA (see Tables L.I and L.II).
- The **Product type**. The type of lightning product, (see Table L.III)
- The **Product parameter**. Parameter describing the type of strokes included in the product (see Table L.IV).
- A **Miscellaneous field**. The miscellaneous field is used for instance to indicate accumulation time.

Four parts of the **product_name** are equal to the **product_group_name**, and only the product parameter is different. The convention for the product parameter is given below in the examples

Table Lightning IV. *The Product Parameter field (_Param) is used to describe the stroke types included in the product.*

_Param	Description
ALL	All kind of strokes are included
CC	Only cloud-cloud strokes are included
CG	Only cloud-ground strokes are included

NA	Not applicable
----	----------------

Examples:

Typesource	_ID	_Product	_Param	_miscellaneous	Product_name
LGT	NL21	LAP	ALL	05M	LGT_NL21_LAP_ALL_05M
LGT	NL21	LAP	CG	24H	LGT_NL21_LAP_CG_24H
