Technische Handleiding

Weer Informatie Waterbeheer (WIWB) API

Het Waterschapshuis

HydroLogic Systems BV
Postbus 2177
3800 CD Amersfoort
033 4753535
hydrologic.nl

P951
V20 September 2023

Inhoud

1 Inleiding

Dit document bevat een handleiding voor gebruik van de Weer Informatie Waterbeheer (WIWB) API. Doelgroep van dit document zijn databeheerders en technisch (ICT) specialisten bij waterschappen en op aangeven van waterschappen leveranciers van ICT diensten/softwarepakketten die aansluiten op de WIWB API.

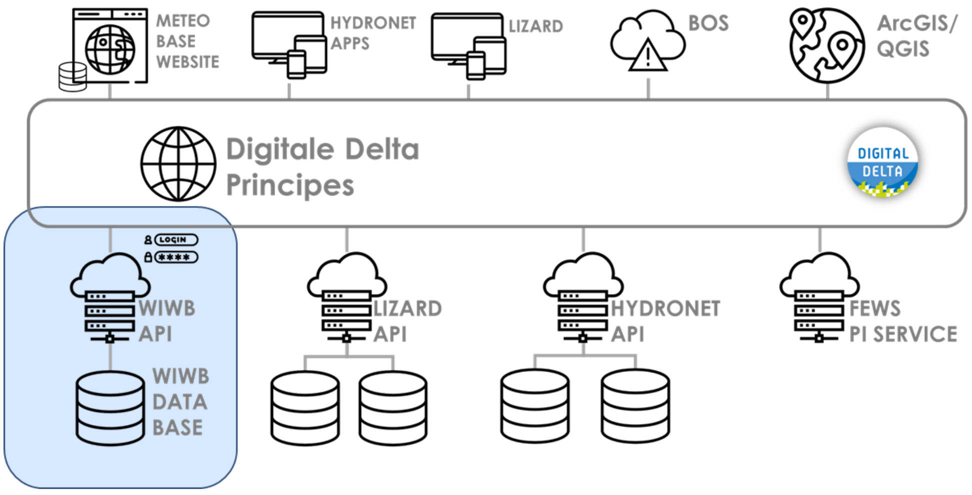
Het tweede hoofdstuk bevat een omschrijving van de WIWB API, met verwijzing naar de API Reference voor ontwikkelaars. Het derde hoofdstuk bevat instructies voor aanmelding en aanvraag voor een account om gebruik te maken van de WIWB API.

2 Omschrijving van de WIWB API

De WIWB API en onderliggende database zijn opgezet met HydroNET technologie en voor WIWB dienstverlening ingericht in opdracht van Het Waterschapshuis (december 2017). Hierop is voor WIWB de Meteobase website aangesloten als gebruikersinterface.

Figuur 1 geeft een overzicht van WIWB, waarbij het blauw gearceerde gedeelte wordt beschreven in dit document.

- **WIWB Database**: opslag van alle databronnen zoals omschreven in paragraaf 2.1. Generieke database voor het opslaan van één of meer dimensionale data structuren en andere formaten zoals tekst en afbeeldingen. Deze structuren worden verder toegelicht in paragraaf 2.1.
- **WIWB API**: application programming interface (API) die door applicaties en externe partijen gebruikt kan worden om data uit de WIWB-database op te vragen. De API kan gebruikt worden voor operationele applicaties of voor het downloaden van grote datasets, bijvoorbeeld via Meteobase.

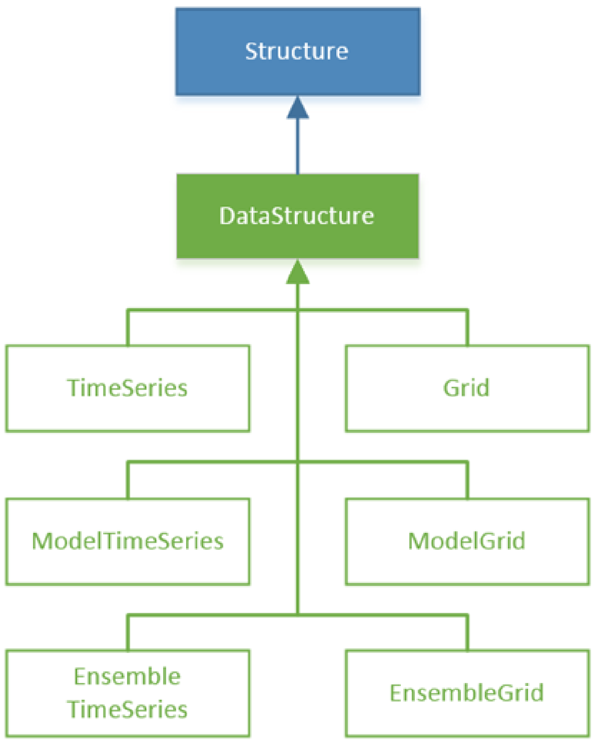


Figuur 1 Topologie en infrastructuur WIWB.

2.1 Operationele WIWB Database

Alle data wordt opgeslagen in de centrale WIWB-database die specifiek is ontworpen voor het opslaan en verwerken van weer- en waterinformatie. Deze is volgens de meest recente technieken opgezet aan de hand van **generieke datastructuren** (zie Figuur 2), voor het gestructureerd opslaan van **Grids** (bijvoorbeeld: neerslagradar), **ModelGrids** (bijvoobeeld: Harmonie, HIRLAM), **EnsembleGrids** (ECMWF), **TimeSeries** (bijvoorbeeld: Synops, IRIS-stations), **ModelTimeseries** (waterstand, modelgegevens) en **EnsembeTimeSeries** (EPS-meteo berekeningen). In de database worden de verschillende datastructuren, tijdsintervallen en correctiestappen separaat opgeslagen.

Een volledig overzicht van de databronnen die worden opgeslagen in de WIWB database is opgenomen in Bijlage A.



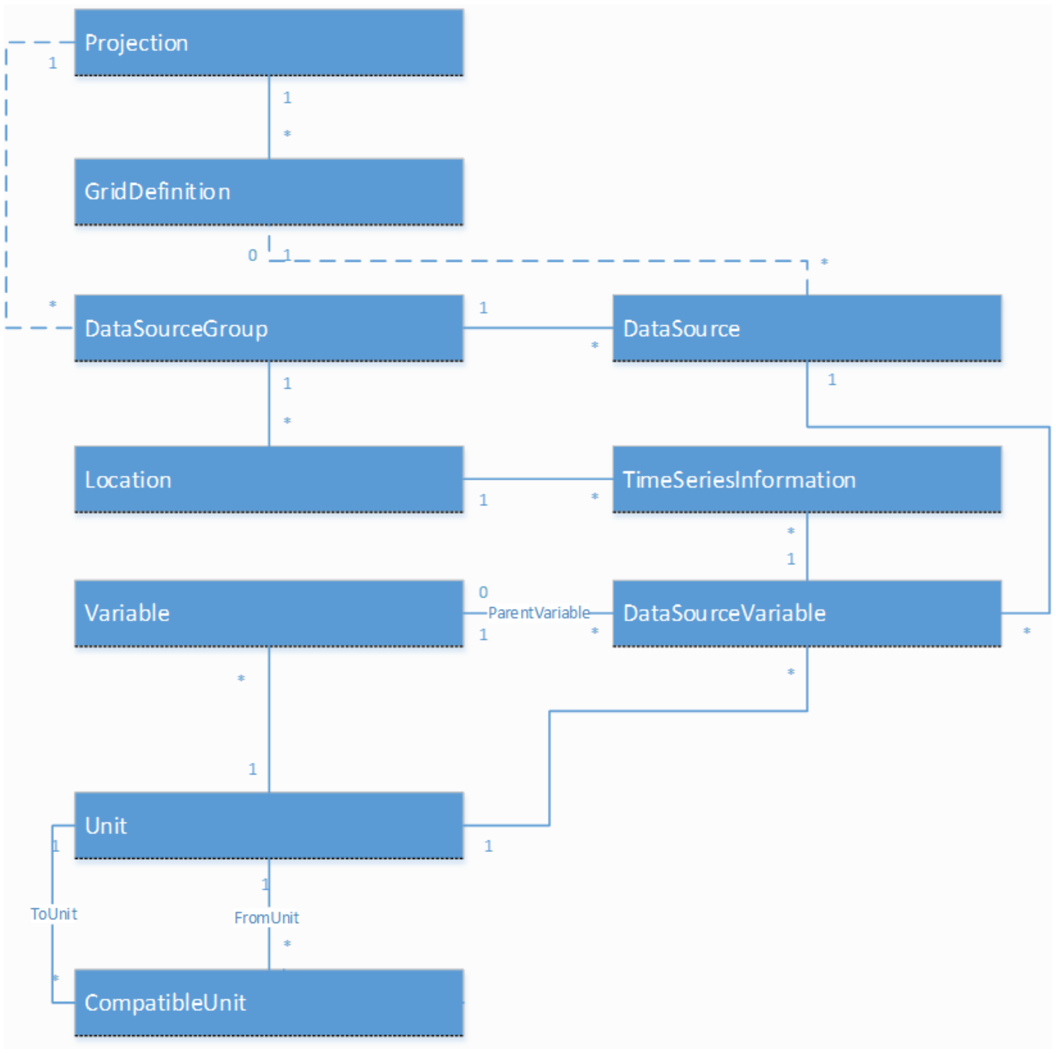
Figuur 2 Generieke datastructuren in de WIWB database

2.2 WIWB API

De data uit de WIWB-database wordt door een web-gebaseerde API opvraagbaar gemaakt Via de API kunnen gebruikers en applicaties door middel van geoptimaliseerde verzoeken informatie opvragen. De API is voorzien van een catalogus met meta-informatie zoals de databeschikbaarheid, de kwaliteit van de data en de vordering van de real-time data-productie. Deze meta-informatie kan via de API eenvoudig worden opgevraagd, voorafgaand aan het feitelijke dataverzoek, aan de hand van het meta-datamodel zoals weergegeven in Figuur 3.

De data die via de API wordt opgevraagd kan standaard worden aangeboden of gedownload in een aantal veelgebruikte bestandsformaten.

Een volledig overzicht van de mogelijke API verzoeken en bestandsformaten voor het opvragen van de data is opgenomen in Bijlage B. Daar wordt ook aanvullende documentatie over de WIWB API gepubliceerd.



Figuur 3. Meta-datamodel van de WIWB API.

2.3 Gebruik van de WIWB API

Voor het gebruik van de API maken wij onderscheid in twee soorten toepassingen:

- **Operationeel gebruik**: toepassing van weerdata in operationele applicaties die gebruik maken van historische, actuele en verwachte weerdata zoals FEWS, Lizard, HydroNET en verschillende BOS-systemen. Dit type dataverzoeken dient beperkt te zijn in omvang, waarbij wordt getoetst op de volgende aspecten:
  - Maximale periode van op te vragen data per verzoek
  - Maximaal op te vragen aantal tijdstappen per verzoek
  - Maximaal op te vragen aantal locaties en parameters per verzoek
  - Maximale absolute omvang van data per verzoek (uitgedrukt in gigabytes)

  De limieten behorende bij deze aspecten worden continu geëvalueerd en kunnen worden bijgesteld. Voor grote dataverzoeken worden download verzoeken geadviseerd.
- **Downloaden grote datasets**: voor analyse, onderzoek en modelering kan het wenselijk zijn om in één keer een grote dataset te downloaden. Via de API kunnen dergelijke dataverzoeken worden ingediend en klaargezet. Omdat snelheid bij dit type gebruik snelheid minder belangrijk is, kan de afhandeling van dergelijke ver verzoeken

afhankelijk van de grootte langer duren om dergelijke verzoeken niet ten koste te laten gaan van het operationeel gebruik. De gebruiker kan de status van het download verzoek opvragen.

De data van WIWB is open. Veiligheid en performance staan voor de waterbeheerders in Nederland evenwel voorop. Het is in gezamenlijk belang van de waterschappen om controle te houden over de toegang tot de database om databeschikbaarheid voor primaire processen voorrang te verlenen, misbruik te voorkomen en om monitoring van het gebruik mogelijk te maken. Daarom wordt **authenticatie** toegepast. Hiermee wordt ervoor gezorgd dat professioneel gebruik door waterbeheerders niet belemmerd wordt door ongeautoriseerde gebruikers buiten de watersector.

Authenticatie vindt plaats op organisatieniveau in afstemming met Het Waterschapshuis. Aanmelding kan volgens de procedure zoals beschreven in hoofdstuk 3.1. Gebruik van de WIWB API wordt gemonitord waarbij onder andere IP adressen en verzoeken worden gelogd. Bij dreigende overbelasting als gevolg van een veelheid aan foutieve en/of overmatig en/of inefficiënte dataverzoeken kan een limiet worden opgelegd, bijvoorbeeld op specifiek IP-adres en gedurende een bepaalde tijdsperiode. Dit zal worden gemeld bij de contactpersoon horende bij het aangemelde IP-adres (zie voor aanmelding paragraaf 3.1).

3 Toegang tot de WIWB API

3.1 Aanmelden

Aanmelden voor het gebruik van de WIWB API doet u door een e-mail bericht te sturen aan Wouter Bakker (w.bakker@hetwaterschapshuis.nl) van Het Waterschapshuis met bijgevoegd een ingevuld Excel aanmeldformulier. Het Excel aanmeldformulier vindt u op de Meteobase website onder Documentatie -> Literatuur. In het Excel aanmeldformulier geeft u de volgende gegevens aan:

- organisatienaam (waterschap)
- contactpersoon met contactgegevens (email en tel.)
- lijst met IP adressen van waaruit toegang is gewenst
- korte toelichting gebruik WIWB API:
  - welke systemen worden gekoppeld
  - welke informatie/data wordt opgevraagd
  - verwachte aantal dataverzoeken per tijdsinterval

Het Waterschapshuis zal deze informatie doorgeven aan de WIWB back office, ingericht door HydroLogic. Uw account wordt voor u beschikbaar gemaakt binnen 5 werkdagen nadat deze door de WIWB back office is ontvangen.

3.2 WIWB API adres

De WIWB API wordt gehost op de volgende URL:
https://wiwb.hydronet.com/api/

3.3 Ondersteuning bij gebruik

Technische support voor de WIWB-API wordt geleverd door de HydroNET helpdesk. De helpdesk is via e-mail bereikbaar tijdens kantooruren (maandag tot en met vrijdag tussen 9:00 en 17:00).

U kunt uw aanvraag voor ondersteuning sturen naar helpdesk@hydrologic.com onder vermelding van:

- Uw naam
- Organisatienaam (waterschap)
- Telefoonnummer waarop u bereikbaar bent
- Duidelijke omschrijving van uw hulpvraag
- Gebruik in het onderwerp van uw bericht ‘WIWB API’

Communicatie via de WIWB vindt plaats via de bij HWH aangemelde contactpersonen (zie 3.1).

Bijlage A WIWB Databronnen

Databron & Code	Databron ID	Type	Oorspronkelijk Interval Vertraging**	Startdatum & einddatum*	Parameters & [Code]
ECMWF 15-day ensemble forecast Link	Ecmwf.Ensemble.15day	Ensemble Grid	1 uur - 6 uur ± 12 uur	*Max. 90 dagen historie beschikbaar*	Precipitation [P] Potential Evaporation [Evaporation] Convective Precipitation [CovectiveP]
ECMWF 42-day extended forecast Link	Ecmwf.42day.Forecast	Model-Grid	7 dagen Komt elke dinsdag beschikbaar	*Max. 365 dagen historie beschikbaar*	Temperature [2t] Temperature Anomaly [2ta] Temperature Propbability [2tp] Mean convective Precipitation Rate [cprate] Mean sunshine duration rate [msdr] Mean sea level pressure [msl] Mean sea level pressure anomaly [msla] Sunshine duration anoumalous rate of accumulation [sundara] Total cloud cover [tcc] Total cloud cover anomaly [tcca] Total precipitation anomalous rate of accumulation [tpara] Total precipitation probability [tpp] Mean total precipitation rate [tprate]
ECMWF Seasonal Forecast Link	Ecmwf.Seasonal.Forecast	ModelGrid	1 maand Komt elke 6e van de maand beschikbaar	*Max. 365 dagen historie beschikbaar*	Temperature [2t] Temperature Anomaly [2ta] Mean convective Precipitation Rate [cprate] Mean convective precipitation rate anomaly [mcpara] Evaporation rate [erate] Evaporation anomalous rate [evara] Mean sunshine duration rate [msdr] Mean sea level pressure [msl] Mean sea level pressure anomaly [msla] Sunshine duration anoumalous rate of accumulation [sundara] Total cloud cover [tcc] Total cloud cover anomaly [tcca] Total precipitation anomalous rate of accumulation [tpara] Mean total precipitation rate [tprate]
KNMI IRC Realtime	Knmi.International.Radar.Composite	Grid	5 min ± 10 minuten	15-10-2019	Precipitation [P]
KNMI IRC Early Reanalysis	Knmi.International.Radar.Composite.Early.Reanalysis	Grid	5 min ± 36 uur	19-07-2019	Precipitation [P]
KNMI IRC Final Reanalysis	Knmi.International.Radar.Composite.Final.Reanalysis	Grid	5 min ± 8 weken	31-12-2018	Precipitation [P]
KNMI IRC Combined***	Knmi.international.Radar.Composite.Combined	Grid	5 min ± 10 minuten	31-12-2018	Precipitation [P]
Ongecorrigeerde Radar (Real time)	Knmi.Radar.Uncorrected	Grid	5 min	01-09-2017	Precipitation [P]

			± 10 minuten		
Radar Corrected B (Near-real time)	Knmi.Radar.CorrectedB	Grid	5 min ± 75 minuten	01-09-2017 28-08-2018	Precipitation [P]
Radar Corrected C2 (After)	Knmi.Radar.CorrectedC2	Grid	5 min ± 36 uur	01-09-2017 26-08-2018	Precipitation [P]
Radar Corrected D2 (Reanalysis)	Knmi.Radar.CorrectedD2	Grid	5 min ± 6 weken	01-09-2017 20-07-2018	Precipitation [P]
EPS meteo parameters **Ordering of ensembles:** *1: Operational run* *2: Control run* *3 to 52: perturbed members*	Knmi.RegionalEps *Deze dataset wordt door het KNMI beschikbaar gesteld, ten behoeve van de Nederlandse waterschappen.*	Ensemble Timeseries	6 uur ± 12 uur	26-08-2002	Precipitation [P] Temperature [TMP] Maximum Temperature [TMAX] Minimum Temperature [TMIN] Total Cloud Cover [TCDC] U-Component of Wind [URGD:10m] V-Component of Wind [VGRD:10m] Dewpoint Temperature [DPT] Snow Depth [SnowDepth] Wind Speed [WindSpeed] Wind Direction [WindDirection]
Hirlam	Knmi.Hirlam	Model Grid	1 uur	01-01-2018 16-05-2022	Temperature [TMP] Total Precipitation [APCP] Total Cloud Cover [TCDC] U-Component of Wind [UGRD:10m] V-Component of Wind [VGRD:10m] Dewpoint Temperature [DPT] Low Cloud Cover [LowCloudCover] Medium Cloud Cover [MediumCloudCover] High Cloud Cover [HighCloudCover] Large Scale Precipitation [LargeScaleP] Water Equivalent of Accumulated Snow Depth [WaterEqSnowDepth] Snow Depth [SnowDepth] Global Radiation Flux [GlobalRadFlux] Net Shortwave Radiation Flux [NetSwRadFlux] Net Longwave Radiation Flux [NetLwRadFlux] Latent Heat Flux [LatHeatFlux] Sensible Heat Flux [SensHeatFlux] Air Pressure [AirPressure]
Harmonie 36	Knmi.Harmonie.36	Model Grid	1 uur	25-10-2017 27-09-2019	Temperature [TMP] Relative Humidity [RH] Rainfall [APCP] Total Cloud Cover [TCDC] U-Component of Wind [UGRD:10m] V-Component of Wind [VGRD:10m] Low Cloud Cover [LowCloudCover] Medium Cloud Cover [MediumCloudCover] High Cloud Cover [HighCloudCover] Large Scale Precipitation [LargeScaleP] Snow Depth [SnowDepth] Global Radiation Flux [GlobalRadFlux] Net Shortwave Radiation Flux [NetSwRadFlux]

					Net Longwave Radiation Flux [NetLwRadFlux] Latent Heat Flux [LatHeatFlux] Sensible Heat Flux [SensHeatFlux] Convective Precipitation [ConvectiveP] Air Pressure [AirPressure] U-Component of max wind gust [U.Max.Wind.Gust] V-Component of max wind gust [V.Max.Wind.Gust]
Harmonie 40	Knmi.Harmonie	Model Grid	1 uur ± 12 uur	25-09-2019	Temperature [TMP] Relative Humidity [RH] Total Precipitation [APCP] Snow Depth [SnowDepth] Graupel [Graupel] Sneeuwval [Sneeuwval] Rainfall, wet part [Rainfall] Total Cloud Cover [TCDC] U-Component of Wind 10m height [UGRD:10m] V-Component of Wind 10m height [VGRD:10m] Dewpoint temperature [DPT] Low Cloud Cover [LowCloudCover] Medium Cloud Cover [MediumCloud-Cover] High Cloud Cover [HighCloudCover] Air pressure [AirPressure] Latent Heatflux [LatentHeatflux] Net Shortwave Radiation [NetSwRad] Net Longwave Radiation [NetLwRad] U-Component max windstoot [U-Component.Max.Windstoot] V-Component max windstoot [V-Component.Max.Windstoot] Global Radiation [GlobalRad] Sensible Heat Flux [SHF] U-Component of Wind 50m height [UGRD:50m] V-Component of Wind 50m height [VGRD:50m] U-Component of Wind 100m height [UGRD:100m] V-Component of Wind 100m height [VGRD:100m] U-Component of Wind 200m height [UGRD:200m] V-Component of Wind 200m height [VGRD:200m] U-Component of Wind 300m height [UGRD:300m] V-Component of Wind 300m height [VGRD:300m]
Iris stations On-gevalideerd	Knmi.IrisUnvalidated	TimeSeries	1 dag ± 36 uur	25-04-2010	Precipitation [P] Snow Depth [SnowDepth]
Iris stations gevalideerd	Knmi.IrisValidated	TimeSeries	1 dag	31-12-1905	Precipitation [P] Snow Depth [SnowDepth]

			± 6 weken		
Referentie gewasver-damping	Knmi.Evaporation	TimeSeries	1 dag ± 36 uur	01-01-1951	Evaporation [Evaporation]
Waqua	Knmi.WaquaTs	Model-TimeSeries	10 min ± 1 dag	03-12-2017 25-10-2022	Water Surge [WaterSurge] Astronomical Tide [AstronomicalTide] Total Water Level [TotalWaterLevel] Observations from WAQC_ODC [WaquaTs.Obs]
KNMI Automatische weerstations 10 Minutes Interval	Knmi.AwsTenMinutes	TimeSeries	10 min ± 1 uur	25-11-2017	Precipitation [P] Precipitation PWS [P_PWS] Temperature [TMP] Relative Humidity [RH] Relative Humidity 1.5m [RH:1.5m] Total Cloud Cover [TCDC] Dewpoint Temperature [DPT] Solar radiation [Radiation.Solar] Wind Speed [WindSpeed] Wind Direction [WindDirection] Maximum Wind Speed [Maxi-mumWindSpeed] Horizontal Visibility [HorizontalVisibil-ity] Precipitation Duration [Precipitation Duration] Air Pressure [AirPressure] Temperature 1.5m [TMP:1.5m]
KNMI Synoptische weerstations Uurlijks interval	Knmi.Synops	TimeSeries	1 uur ± 1 uur	01-01-1951	Neerslag [P] Temperatuur [TMP] Relative Luchtvochtigheid [RH] Bewolkingsgraad [TCDC] Dauwpunt Temperatuur [DPT] Globale Straling [Radiation.Solar] Wind Snelheid [WindSpeed] Wind Richting [WindDirection] Maximum Wind Snelheid [Maximum-WindSpeed] Horizontaal Zicht [HorizontalVisibility] Lucht druk [AirPressure] Minimum Temperatuur [TMIN] Maximum Temperatuur [TMAX] Neerslag Duur [Precipitation Duration]
Meteobase Makkink Verdamping	Meteobase.Evapora-tion.Makkink	Grid	1 dag ± 1 jaar	01-01-1985 01-01-2022	Evaporation [Evaporation]
Meteobase Pennman-Monteith Verdamping	Meteobase.Evapora-tion.PennmanMonteith	Grid	1 dag ± 1 jaar	01-01-1985 01-01-2022	Evaporation [Evaporation]

Meteobase Neerslag	Meteobase.Precipitation	Grid	1 uur	01-01-1990 01-01-2019	Precipitation [P]
SATDATA3.0 Evapotranspiration	Satdata.Evapotranspiration	Model-Grid	1 dag ± 12 uur	*Max. 90 dagen historie beschikbaar*	Actual evapotranspiration [EvapotranspirationActual] Evapotranspiration shortage [EvapotranspirationShortage] Quality Flag [QualityFlag] Uncertainty [Uncertainty]
SATDATA3.0 Evapotranspiration Reanalysis Versie 2.0	Satdata.Evapotranspiration.Reanalysis.V2	Grid	1 dag ± 70 dagen	24-07-2012	Actual evapotranspiration [EvapotranspirationActual] Evapotranspiration shortage [EvapotranspirationShortage] Quality Flag [QualityFlag] Uncertainty [Uncertainty]

** Einddatum alleen weergegeven indien de databron niet (meer) operationeel is.*
*** De maximale vertraging is een schatting van hoeveel tijd er maximaal is verstreken is voordat de data in WIWB beschikbaar komt. Er kunnen geen rechten aan worden ontleend.*
**** Deze databron is een automatische combinatie van de drie KNMI IRC bronnen, waarbij automatisch de best beschikbare kwaliteit wordt geselecteerd.*

Bijlage B Authenticatie van de WIWB API

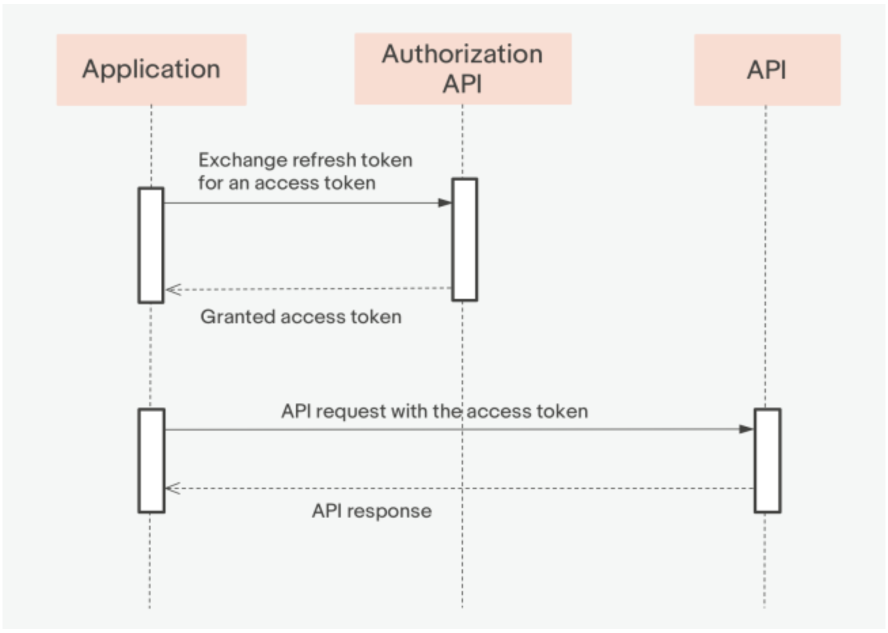
Om zowel de veiligheid als de performance van de WIWB API te verbeteren is in januari 2023 een update doorgevoerd. Sinds de januari 2023 update wordt er voor de authenticatie gebruikt gemaakt gaat van het Open ID Connect authenticatie framework. OpenID Connect is een internationale standaard en wordt bijvoorbeeld ook toegepast in de Digitale Delta maar ook door de grote Tech bedrijven zoals Google en Microsoft.

Met de implementatie van het OpenID Connect framework is de eerder benodigde IP-whitelisting komen te vervallen. Dit legt de WIWB gebruikers minder restricties op vanaf welke locatie zij de WIWB API gebruiken, maar er zal wel gebruik worden gemaakt van een veilige en professionele manier van authenticatie.

Sinds de januari 2023 update ontvangen gebruikers van de WIWB API een set Client Credentials (een ID en een Secret). Met deze Client ID en Secret kan een gebruiker geautomatiseerd een access token aanvragen. Dit proces staat ook wel bekend als de Client Credential Flow in de OpenID Connect standaard. Elke aangevraagde access token heeft slechts een beperkte geldigheid van één uur. Met een access token kunnen vervolgens verzoeken naar de WIWB API worden gedaan. Door de beperkte geldigheid van de access tokens zal periodiek een nieuw token moeten worden aangevraagd.

De workflow voor WIWB gebruikers is als volgt:

1) Opvragen van een geldig access token
2) Gebruik van het access token voor WIWB API verzoeken
3) Opvragen van een nieuw access token zodra het oude token (bijna) verloopt


Application
Authorization API
API
Exchange refresh token for an access token
Granted access token
API request with the access token
API response

Figuur 4 Schematische weergave van de manier van authentiseren bij de WIWB API.

Om een nieuw access token aan te vragen moet een verzoek worden gestuurd aan de authenticatie API (zie Figuur 4). Dit POST verzoek moet als volgt worden vormgegeven.

URL	https://login.hydronet.com/auth/realms/hydronet/protocol/openid-connect/token
Method	POST
Content-Type	application/x-www-form-urlencoded
Form-Data	client_id=*<received from Helpdesk>*
	client_secret=*<received from Helpdesk>*
	grant_type= client_credentials

Een CURL voorbeeld, met daarin de volgende ClientId / Client Secret combinatie ziet er als volgt uit

Client ID:	api-wiwb-demo
Client Secret:	895269fb-ca52-429e-af0b-96a9f0266e71

```
curl --request POST \
  --url https://login.hydronet.com/auth/realms/hydronet/protocol/openid-connect/token \
  --header 'Content-Type: application/x-www-form-urlencoded' \
  --data grant_type=client_credentials \
  --data client_id=api-wiwb-demo \
  --data client_secret=895269fb-ca52-429e-af0b-96a9f0266e71
```

De authenticatie API zal vervolgens de client ID en Secret controleren. Als deze geldig zijn wordt een nieuw access token teruggeven. De output van de API ziet er als volgt uit :

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJkUFdiVFBrNVhCNzdSaVpPSGhSSEFOM-
nFyY2p0eVAyVzhJXzQ4VUV2M0U0In0.eyJleHAiOjE2MzY3NDY5ODgsImlhdCI6MTYzNjczMjU4OCwianRpIjoiOWJhNWYzYzg-
tOGExMC00NTRlLTlkNzItM2E4MzFmZDkxNjViIiwiaXNzIjoiaHR0cHM6Ly90ZXN0LWxvZ2luLmh5ZHJvbmV0LmNvbS9hdXRoL3JlY-
Wxtcy9oeWRyb25ldCIsImF1ZCI6ImFjY291bnQiLCJzdWIiOiJhODE5MTA5OS05YTIwLTQ1N2ItOWM1Mi1iOWNlMmEzOG-
MxZGUiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOiJkZW1vIiwiYWNyIjoiMSIsInJlY-
WxtX2FjY2VzcyI6eyJyb2xlcyI6WyJvZmZsaW5lX2FjY2VzcyIsImRlZmF1bHQtcm9sZXMtaHlkcm9uZXQiLCJ1bWF-
fYXV0aG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIjp7ImFjY291bnQiOnsicm9sZXMiOlsibWFuYWdlL-
WFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsInZpZXctcHJvZmlsZSJdfX0sInNjb3BlIjoicHJvZmlsZSBlbWFpbCIsImNsaWVu-
dEhvc3QiOiIxNzIuMTYuMjQxLjE1MCIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwiY2xpZW50SWQiOiJkZW1vIiwicHJl-
ZmVycmVkX3VzZXJuYW1lIjoic2VydmljZS1hY2NvdW50LWRlbW8iLCJjbGllbnRBZGRyZXN-
zIjoiMTcyLjE2LjI0MS4xNTAifQ.LfmXjywQpN5qd-f6BsqH5HvbRrTV8ZR5DKYBf0_3bbfu7DPkbZcRkQe3h78uNzRbWj-ixaF-
VqIicPJq0yTtHsgxfmTAxTXGl0Ql0DmcNL_PHPHVI3Jnc3D4FgMZNq8J7xike4lLrf4jAO-cx3BmC-jgO78rbDGhoRUUibwieD8YBQyNaL-
cRpo-V6Cl_nEUdRkqZhPfJC76L0b0As8P2dZy2HoOzqATqEz2RL1wn5nDFQDjgW-Pl4-
iB7ZBQXslry8QUw0gP5SO6u1APPMSCxY7VwoU5CkOyXZQNOKllrlTvl3kg_UatzFG8bcsAqUDKL3OI0j-i2miLAVXf2smOh2w",
    "expires_in": 3600,
    "refresh_expires_in": 0,
    "token_type": "Bearer",
    "not-before-policy": 0,
    "scope": "profile email"
}
```

De output van de API is in JSON formaat, met daarin aantal velden. De twee relevante velden zijn:

- access_token
  Bevat een nieuw en geldig access token. Een zogeheten JSON Web Token (JWT)
- expires_in
  Geeft de geldigheid van een access token aan, in secondes. Een waarde van 3600 betekent dus een geldigheid van 1 uur

Het access_token kan vervolgens worden gebruikt in verzoeken naar de WIWB API. Het access token dient te worden opgenomen in de header van het verzoek, waarbij voor het type van de authorization voor Bearer moet worden gekozen.

Elke token (JWT) ziet er uit als een string van willekeurige karakters, maar deze string kan worden gedecodeerd om de achterliggende informatie zichtbaar te maken. Bijvoorbeeld op de website https://www.jwt.io/ kan men het token invullen, om vervolgens de achterliggende data te zien. De gedecodeerde output van het token bevat informatie over wanneer een token is uitgegeven (iat) en wanneer een token verloopt (exp).

Een voorbeeld Python script inclusief deze manier van authentiseren is weergeven in Bijlage C.

Bijlage C Voorbeeld Python script

```
#!/usr/bin/env python
# coding: utf-8


# Vereiste python modules
import datetime
import requests
import json
import base64
import jwt         # pip/conda install pyjwt


# WIWB API URL voor authenticatie
api_url_auth = "https://login.hydronet.com/auth/realms/hydronet/proto-
col/openid-connect/token"
# WIWB API URL voor data verzoeken
api_url_wiwb = "https://wiwb.hydronet.com/api"

# Client ID and secret
# let op, dit
client_id = "api-wiwb-voorbeeld"
client_secret = "8945269fb-ca52-42b9e-af0b-96a9af0266e71"


# Functie voor het ophalen van een WIWB access token
def get_wiwb_token(client_id, client_secret):
    authorization = base64.b64encode(bytes(client_id + ":" + client_se-
cret, "ISO-8859-1")).decode("ascii")
    headers = {
        "Authorization": f"Basic {authorization}",
        "Content-Type": "application/x-www-form-urlencoded"
    }
    body = { "grant_type": "client_credentials" }
    response = requests.post(api_url_auth, data = body, headers = headers)
    wiwb_token = response.json().get("access_token")
    return(wiwb_token)

# Functie om geldigheid WIWB token te controleren
def token_expired(token):
    token_decoded = jwt.decode(token, options={"verify_signature": False})
    token_exp_datetime = datetime.datetime.utcfromtimestamp(token_de-
coded['exp'])
    current_datetime = datetime.datetime.utcnow() - datetime.time-
delta(minutes=1)
    if current_datetime > token_exp_datetime:
        return True
    else:
        return False

# Functie om token verversen, indien nodig
def get_or_refresh_token(token):
    if token_expired(token):
        return(get_wiwb_token(client_id, client_secret))
    else:
        return(token)
```

```
# Functie op HTTP header te genereren inclusief authenticatie
def create_http_header_with_auth(token):
    # controleer geldigheid token, vernieuw indien noodzakelijk
    access_token = get_or_refresh_token(token)
    # HTTP header verzoek opstellen inclusief het token
    # deze header kan nu met alle WIWB data verzoeken worden meegestuurd
    api_header = {"content-type": "application/json", "Authorization":
"Bearer " + access_token}
    return(api_header)


# Initieel ophalen van een token
token = get_wiwb_token(client_id, client_secret)

# Voorbeeld verzoek, ophalen van alle locaties van de Knmi.AwsTenMinutes
databron
json_request_locations = {
     "DataSourceCodes": ["Knmi.AwsTenMinutes"]
}

# Verzoek versturen naar de WIWB API
# Token wordt automatisch ververst indien noodzakelijk
resp = requests.post(api_url_wiwb + "/entity/locations/get",
                     headers=create_http_header_with_auth(token),
                     data=json.dumps(json_request_locations))


# Als het verzoek succesvol is, print de output
if resp.status_code == 200:
    knmi_locations = resp.json()
    print(knmi_locations)
```

Bijlage D WIWB API Reference

D.1 General

This document contains the API reference of HydroNET 4 Server, which is used for the 'Weer Informatie Waterbeheer (WIWB)' API. This document is a very short technical introduction to the API and gives examples of how to use the API.

In chapter 2 the entities (or meta data model) is presented and examples are given for requests on entities and filtering. Chapter 3 explains data structures and how to retrieve data from the API.

All requests are done with POST rather than GET. There is only one exception:

- Request to download a file. Handling an attachment coming from a POST response is very difficult in clients, so this is done with GET.

The POST body is given in JSON. The API does not support XML requests. Because of this, `Content-Type: application/json` should be given in the header of the request. It is recommended to use a free tool like Insomnia[1] to do requests.

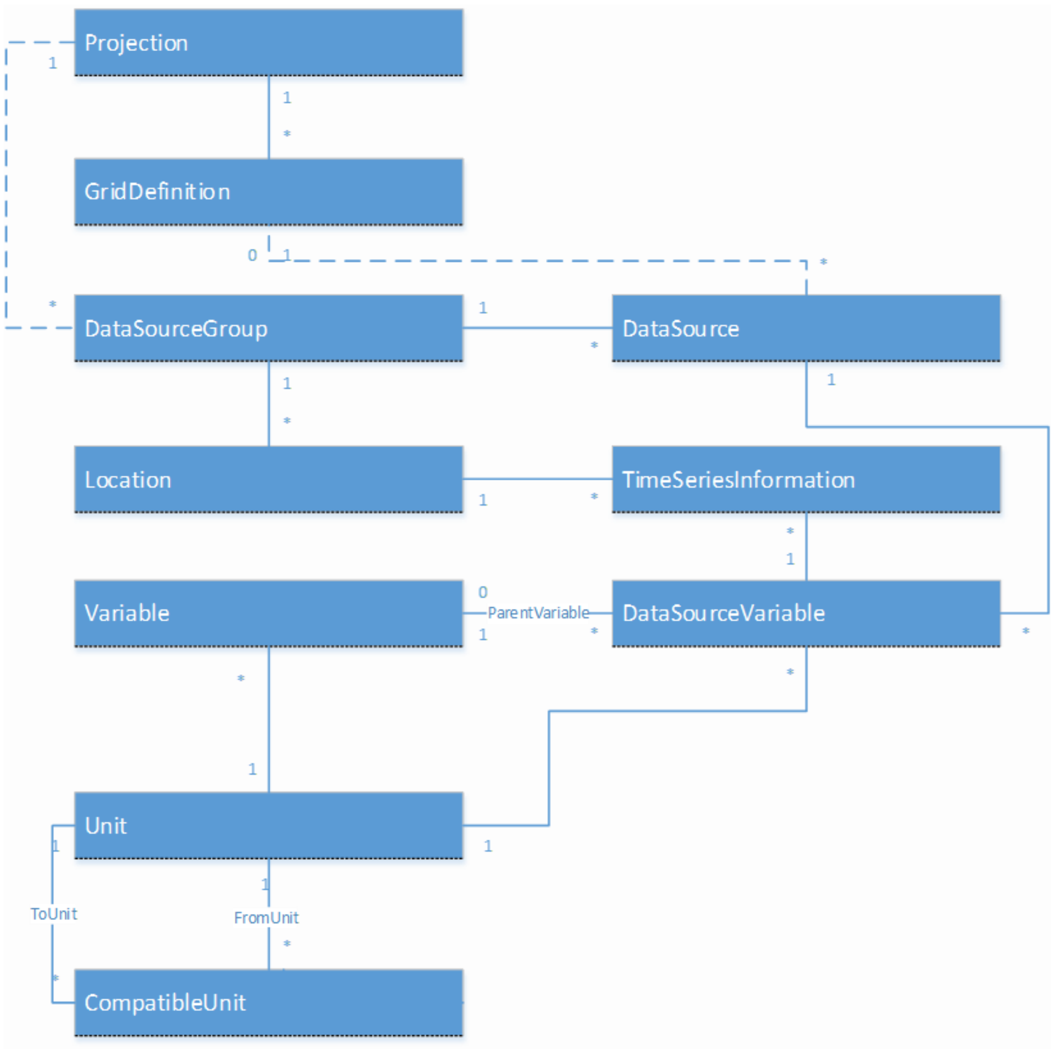
The following chapters describe the most important requests that can be done to the API. A distinction is made between entity and data requests.

Please note, by default the datetimestamp returned by the API always specified the end datetime of an interval.

[1] https://insomnia.rest/download

D.2 Entities (meta-data)

The following paragraphs give examples of several entity requests. Entities are objects that are in the HydroNET database. Examples are data sources, variables and locations. Entities are used to retrieve information about the state of entities. The first paragraph explains the basics of entity requests.



D.2.1 Entity request and response basics

Most entity requests support filtering on properties and relations of the entity type. There are a number of basic filters that can be applied to entities. All filters are optional, but some entity types require that you give at least one filter in order to avoid retrieving all entities in a table.

Request filters and parameters

The following filters are applicable to most entities:

- **DataSourceCodes**: a list of unique data source codes to filter by (case insensitive string).
- **DataSourceGroupCodes**: a list of unique data source group codes to filter by (case insensitive string).

- **VariableCodes**: a list of unique variable codes to filter by (case insensitive string).
- **LocationIdentifiers:** a list of location identifiers unique to the whole system (case insensitive string). A location identifier is a combination of a data source group code and a location code.

One date format is supported: `yyyyMMddHHmmss` (e.g. 20150201123000 for 01-02-2015 12:30:00).

Filters and implemented as AND filters. If multiple filters are applied, entities must fit both filters to be returned (as opposed to an OR filter, where only a single filter would have to fit the entity).

The body of an example request filter on data sources and variables looks as given below. Note that in JavaScript the double quotes around the property names should be omitted.

```
{
    "DataSourceCodes": [
        "HydroEstimator",
        "ECMWF.EPS.MexicoCity"
    ],
    "VariableCodes": [
        "P"
    ]
}
```

Response

The response of an entity request is always in JSON. The exact structure of the response of depends on the entity type that is requested. There is however a convention for the response regarding structure. Each entity collection is returned with as key its own name (always plural).

The pseudo-example below shows what a pseudo response looks like for DataSources. If relations are included, these are grouped by entity type in separate properties.

```
{
    "DataSources": {
        ...
    },
    "RelationsX": {
        ...
    },
    "RelationsY": {
        ...
    }
}
```

An example response of a variables request is given below. The Variables property contains the variableswhere the unique identifier is used as key (VariableCode with value P and IRRATE). It depends on the type of entity which property is used as key. In some cases the primary key of the table is used, in other case the unique code.

The Unit of each variable is returned as well and stored in a separate Units property. Each variable has a property UnitCode pointing to the unique code of the unit. Again the units are stored with the code as property name.

```
{
    "Variables": {
        "P": {
            "Name": "Precipitation",
            "Code": "P",
            "Description": "Precipitation",
            "UnitCode": "MM",
            "State": 1
        },
        "IRRATE": {
            "Name": "Instantaneous Rain Rate",
            "Code": "IRRATE",
            "UnitCode": "None",
            "State": 1
        }
    },
    "Units": {
        "MM": {
            "Name": "mm",
            "Code": "MM",
            "ParentUnit": null
        },
        "None": {
            "Name": "None",
            "Code": "None",
            "ParentUnit": null
        }
    }
}
```

Exception handling

To some extent exception information is returned to the client. This is done in JSON. The level of detail depends on the type of exception. Sensitive information is not disclosed. The HTTP status code also depends on the type of exception.

```
{
    "ClientType": "Server",
    "Exception": {
        "Name": "ArgumentException",
        "Message": "At least one filter should be supplied when retrieving
DataSourceGroups (DataSourceGroupCodes, DataSourceCodes, VariableCodes,
ThemeIds or LocationIdentifiers)."
    },
    "RequestId": 5118728
}
```

D.2.2 Data sources

An application usually starts with retrieving information about a data source or a set of data sources. E.g. the start and end date of a data source are relevant to the time range that is shown to a user, or to show the latest data. Usually an application knows for which data sources it is configured, so it does a request to get information for those specific data sources.

Request

`URL: %api_url%/entity/datasources/get`

The data sources request has the same parameters as the base entity request (paragraph 0).

The filters are applied as follows:

- **DataSourceCodes**: when applied, only data sources with the given code(s) are re-turned.
- **DataSourceGroupCodes**: when applied, only data sources that are in the a data source group with one of the given codes are returned.

Request example with DataSourceCodes filter:

```
{
    "DataSourceCodes": ["HydroEstimator"]
}
```

Response

Partial response example:

```
{
    "DataSources": {
        "HydroEstimator": {
            "Name": "Noaa HydroEstimator",
            "Code": "HydroEstimator",
            "Settings": {
                "Time": {
                    "Interval": {
                        "Type": "Hours",
                        "Value": 1
                    }
                },
                "Grid": {
                    "GridDefinitionId": 7
                }
            },
            "StartDate": "20150131230000",
            "EndDate": "20171031080000",
            "IsEnabled": true,
            "TimeZoneOffset": "+0000",
            "PrimaryStructureType": "Grid",
            "DataSourceGroupCode": "Noaa"
        }
    }
}
```

D.2.3 Data source groups

Data source groups are used to group data sources that have similar properties. The most important aspect is locations: it is possible that multiple data sources use the same locations. An example is the stations data of the KNMI, the Dutch weather institute; the same stations codes are used for synoptic data, evaporation measurements and the regional EPS. Locations are connected to data source groups to avoid having to replicate locations used in multiple data sources. All locations in a data source group have the same projection.

Request

`URL: %api_url%/entity/datasourcegroups/get`

The data source groups request can apply most of the parameters in the base entity request (paragraph 0). It is not compulsory to give a filter; if no filter is given, all data source groups are returned.

The filters are applied as follows:

- **DataSourceCodes**: when applied, only data source groups that have a data source with one of the given codes are returned.
- **DataSourceGroupCodes**: when applied, only data sources groups that have one of the given codes are returned.

Request example with DataSourceCodes filter:

```
{
    "DataSourceCodes": ["HydroEstimator", "GrapeCompass.Fields"]
}
```

Response

Response example:

```
{
    "DataSourceGroups": {
        "HydroNet": {
            "Code": "HydroNet",
            "OwnerOrganisationId": 1,
            "Name": "HydroNET System",
            "Description": null,
            "ProjectionId": 3,
            "Settings": {
                "LocationsSource": "HydroNet"
            }
        },
        "Noaa": {
            "Code": "Noaa",
            "OwnerOrganisationId": 1,
            "Name": "Noaa",
            "Description": "National Oceanic and Atmospheric Administration",
            "ProjectionId": 3,
            "Settings": {
                "LocationsSource": "HydroNet"
            }
        }
    },
    "Projections": {
        "3": {
            "ProjectionId": 3,
            "Name": "WGS84",
            "Epsg": 4326,
            "ProjectionString": "+proj=longlat +ellps=WGS84 +datum=WGS84
+no_defs "
        }
    }
}
```

D.2.4 Variables

Request

`URL: %api_url%/entity/variables/get`

The variables request has the same parameters as the base entity request (paragraph 0).

The filters are applied as follows:

- **DataSourceCodes**: when applied, only variables connected to the given data sources are returned (connection through DataSourceVariables).

- **DataSourceGroupCodes**: when applied, only variables connected to data sources that are in the a data source group with one of the given codes are returned.
- **VariableCodes**: when applied, only variables with the given codes are returned).

Request example with VariableCodes filter:

```
{
    "VariableCodes": ["P", "IRRATE"]
}
```

Response

Response example:

```
{
    "Variables": {
        "P": {
            "Name": "Precipitation",
            "Code": "P",
            "Description": "Precipitation",
            "UnitCode": "MM",
            "State": 1
        },
        "IRRATE": {
            "Name": "Instantaneous Rain Rate",
            "Code": "IRRATE",
            "UnitCode": "None",
            "State": 1
        }
    },
    "Units": {
        "MM": {
            "Name": "mm",
            "Code": "MM",
            "ParentUnit": null
        },
        "None": {
            "Name": "None",
            "Code": "None",
            "ParentUnit": null
        }
    }
}
```

D.2.5 Locations

Request

`URL: %api_url%/entity/locations/get`

The locations request has the same parameters as the base entity request (paragraph 0).

The filters are applied as follows:

- **DataSourceCodes**: when applied, only locations in the data source group of the given data sources are returned (connection through DataSourceGroup).
- **DataSourceGroupCodes**: when applied, only locations that are in the a data source group with one of the given codes are returned.
- **LocationIdentifiers**: when applied, only locations that have the given identifiers are returned.

Request example with LocationIdentifiers. Only locations with the given identifier are returned.

```
{
    "LocationIdentifiers": ["HydroNet#MLMO"]
}
```

Response

Response example:

```
{
    "Locations": {
        "HydroNet#MLMO": {
            "Identifier": "HydroNet#MLMO",
            "Name": "DMA LWBG6",
            "Code": "MLMO",
            "X": 5.805045,
            "Y": 53.217092,
            "Z": 0.0,
            "ProjectionId": 3,
            "DataSourceGroupCode": "HydroNet"
        }
    },
    "Projections": {
        "3": {
            "ProjectionId": 3,
            "Name": "WGS84",
            "Epsg": 4326,
            "ProjectionString": "+proj=longlat +ellps=WGS84 +datum=WGS84
+no_defs "
        }
    },
    "DataSourceGroups": {
        "HydroNet": {
            "Code": "HydroNet",
            "Name": "HydroNET System",
            "Description": null,
            "ProjectionId": 3,
            "Settings": {
                "LocationsSource": "HydroNet"
            }
        }
    }
}
```

D.3 Data retrieval

HydroNET 4 Server/the WIWB API can disseminate data for a number of different Structure types. Fig. 1 shows which Structures are available.

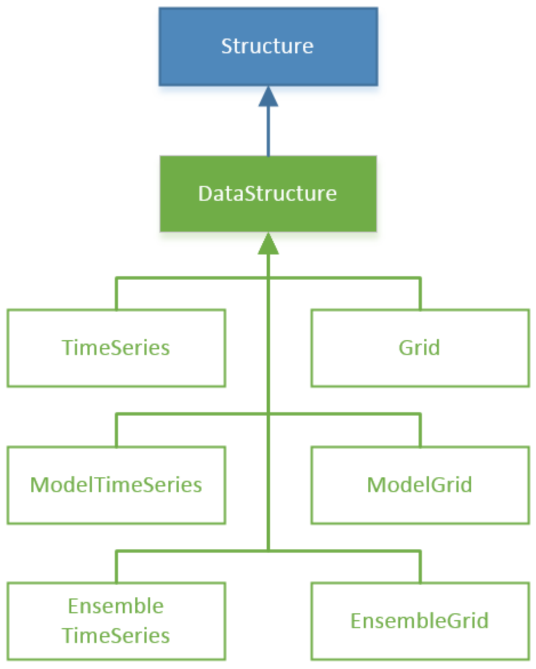


Fig. 1 - Structure model

The following URLs are used for retrieving data (relative to %api_url%):

- timeseries/get
- modeltimeseries/get
- ensembletimeseries/get
- grids/get
- modelgrids/get
- ensemblegrids/get

D.3.1 Data request basics

The general structure of data requests is the same for every type of Structure. A data request (see table below) is always in JSON and consists of:

- One or more Readers
- Zero or one Exporters

```
{
    "Readers": [...],
    "Exporter": {...}
}
```

Readers

A Reader defines from which data source data should be read, and how it should be read. A Reader consists of a DataSourceCode and read settings for that data source. Below is an example of a request. The DataSourceCode is the unique (case-insensitive) code of the data source. The content of the Settings property (given in green) differs for each Structure type. The StructureType property may be omitted. If this property is omitted, the property is derived from the URL (e.g. timeseries/get will set the Structure type of all readers to TimeSeries, but only if they are undefined).

```
{
    "Readers": [
        {
            "DataSourceCode": "Vitens.Proeftuin",
            "Settings": {
                "LocationCodes": ["MLAG"],
                "VariableCodes": ["T"],
                "StartDate": "201401010000",
                "EndDate": "201401020000",
                "StructureType": "TimeSeries"
            }
        }
    ]
}
```

It is possible to give multiple Readers. The Readers are executed in the order in which they are given.

Exporter

The Exporter determines to which data format the data is exported. Most applications use JSON as data format, so this is set as the default Exporter in case it is omitted. If the data should be exported to a different format, the Exporter should be defined as given in the example below. An Exporter has a DataFormatCode and a Settings property. The DataFormatCode should contain the unique (case-insensitive) code of the data format. The settings are specific to the data format to which data is exported.

```
{
    "Readers": [...],
    "Exporter": {
        "DataFormatCode": "json",
        "Settings": {
            "Formatting": "Indented"
        }
    }
}
```

D.3.2 DataStructure Readers

Each DataStructure (TimeSeries, Grids, ModelTimeSeries, etc.) has a shared base request for reading data. An example is given below.

```
{
    "Readers": [{
        "DataSourceCode": "HydroEstimator",
        "Settings": {
            "StartDate": "20150101000000",
            "EndDate": "20150201000000",
            "VariableCodes": ["P"],
            "Interval": {
                "Type": "Days",
                "Value": 7
            },
            "Extent": {
                "Xll": 0,
                "Yll": 10,
                "Xur": 0,
                "Yur": 10,
                "SpatialReference": {
                    "Epsg": 3857
                }
            }
        }
    }]
}
```

- **StartDate/EndDate**: these properties determine for which period the data should be read. The format `yyyyMMddHHmmss` is used. These properties may not be omitted.
- **VariableCodes**: this property defines for which variables data must be read. The given variables should be enabled and connected to the data source through DataSourceVariables. This property may not be omitted, at least one variable code should be given (case-insensitive).
- **Extent**: this property determines the extent for which data should be retrieved and consists of the corners of an envelope (lower left and upper right) and the projection in which the extent is given (SpatialReference with a property containing the EPSG). This property may be omitted, but care should be taken of the consequences (it could mean a complete grid is downloaded for multiple time steps). When omitted this property defaults to null.

D.3.3 [Model/ensemble] time series

Request

Requesting time series, model time series and ensemble time series is done with the base request given in paragraph D.3.2 including some additional parameters. The base parameters are shown in grey below, the new parameters are shown in orange (all time series types), blue (model time series and ensemble time series) and green (ensemble time series).

```
{
    "Readers": [{
        "DataSourceCode": "Proeftuin.Vitens",
        "Settings": {
            "StructureType": "TimeSeries",
            "LocationCodes": ["MLAG", "MLOM"],
            "ModelDate": "20150101000000",
            "EnsembleNames": [
                "0",
                "1"
            ],
            "StartDate": "20150101000000",
            "EndDate": "20150201000000",
            "VariableCodes": ["T"]
        }
    ]
}
```

- **StructureType**: optional, as discussed earlier. When given it should be TimeSeries, ModelTimeSeries or EnsembleTimeSeries.
- **LocationCodes**: this option can only be used if the primary Structure type of the data source is time series, model time series or ensemble time series. In this case the data is stored per location (e.g. a rain gauge). A spatial filter is required when requesting time series, either in the form of LocationCodes or an Extent. When requesting grids, model grids or ensemble grids as time series, this option cannot be used, since grid data is not stored in locations as such. Instead the Extent property can be used (see paragraph D.3.2).
- **ModelDate**: this option is be used for model time series and ensemble time series to determine which model run should be loaded. In this case start date and end date may be omitted to request a complete model run. This option may not be omitted.
- **EnsembleNames**: this option can be used for requesting ensemble time series, to determine which ensembles should be retrieved. This option may be omitted, in this case all ensembles of the data source are read.

Response

The table below shows an example response of a time series request. It consists of the following:

- Data in blue. This is an array of time series containing the data per location. Note that pixels are also returned as a location. Each time series contains a Data property containing the actual value, date time and optionally availability and quality per time step.
- Locations in brown. This is an object containing information about each location for which time series are returned. Locations are indexed by code.
- Variables in green. This is an object containing information about each variable for which time series are returned. Variables are indexed by code.

The data of model time series and ensemble time series looks slightly different:

- Model time series are the same as time series, only with an additional ModelDate property.
- Each ensemble time series is given as a collection model time series.

```
{
    "Data": [
        {
            "LocationCode": "MLAG",
            "LocationIdentifier": "HydroNet#MLAG",
            "Data": [
                {
                    "Value": 0.04884005,
                    "Availability": 1,
                    "DateTime": "20140301085000"
                },
                {
                    "Value": 0,
                    "Availability": 1,
                    "DateTime": "20140301085500"
                },
                {
                    "Value": 0.02442002,
                    "Availability": 1,
                    "DateTime": "20140301090000"
                }
            ],
            "NoDataValue": -9999,
            "StartDate": "20140301000000",
            "EndDate": "20140301001000",
            "DataType": "Single",
            "Interval": {
                "Type": "None",
                "Value": 0
            },
            "VariableCode": "T",
            "DataSourceVariableId": 541
        }
    ],
    "Meta": {
        "Locations": {
            "MLAG": {
                "DataSourceGroupCode": "HydroNet",
                "Identifier": "HydroNet#MLAG",
                "Name": "DMA LWCB1 ",
                "Code": "MLAG",
                "X": 5.834358,
                "Y": 53.216243,
                "Z": 0,
                "ProjectionId": 3
            }
        },
        "Variables": {
            "T": {
                "Name": "Temperatuur",
                "Code": "T",
                "UnitCode": "degrees_celsius"
            }
        }
    },
    "StructureType": "TimeSeries"
}
```

Supported data formats

For time series, model time series and model time series the following data formats are available:

- JSON in HydroNET structure
- CSV in HydroNET structure

D.3.4 [Model/ensemble] grids

Request

Requesting grids, model grids and ensemble grids is done with the base request given in paragraph D.3.2 including some additional parameters. The base parameters are shown in grey below, the new parameters are shown in orange (all grid types), blue (model grids and ensemble grids) and green (ensemble grids).

```
{
    "DataSourceCode": "HydroEstimator",
    "Settings": {
        "StructureType": "Grid",
        "ModelDate": "20150101000000",
        "EnsembleNames": [
            "0",
            "1"
        ],
        "StartDate": "20150101000000",
        "EndDate": "20150201000000",
        "VariableCodes": ["P"],
        "Extent": null
    }
}
```

- **StructureType**: optional, as discussed earlier. When given it should be Grid, ModelGrid or EnsembleGrid.
- **ModelDate**: this option can be used for model time series and ensemble time series to determine which model run should be loaded. In this case start date and end date may be omitted to request a complete model run. This option may not be omitted.
- **EnsembleNames**: this option can be used for requesting ensemble time series, to determine which ensembles should be retrieved. This option may be omitted, in this case all ensembles of the data source are read.

Response

The table below shows a partial example response of a grids request. It consists of the following:

- Data in blue. This is an array of grids containing the data per time step (a single time step is given). Each grid contains a Data property containing the actual values, date time and optionally availability and quality for that time step.

- Variables in green. This is an object containing information about each variable for which time series are returned. Variables are indexed by code.

The data of model grids and ensemble grids looks slightly different:
- Each model grid is given as a collection of grids.
- Each ensemble grid is given as a collection model grids.

Ask a for additional support from our helpdesk if you need to use these data structures.

```
{
  "Data": [
    {
      "Data": [
        30.194828,
        -17.61728,
        15.4995346,
        -4.33504,
        9.981618,
        28.8752041,
        40.7175331,
        26.8732738,
        -46.2580147,
        33.21269,
        -41.0236244,
        -3.491497,
        0.752158344,
        27.7336617,
        32.3882828,
        45.76374,
        2.29221559,
        23.9290085,
        -19.8560352,
        -17.4038811
      ],
      "GridDefinition": {
        "GridDefinitionId": 0,
        "Columns": 5,
        "Rows": 4,
        "CellWidth": 1.0,
        "CellHeight": 1.0,
        "Xll": 0.0,
        "Yll": 0.0,
        "Xur": 5.0,
        "Yur": 4.0,
        "Name": "Random DataSet GridDefinition",
        "ProjectionId": 2,
        "StartColumn": 0,
        "EndColumn": 0,
        "StartRow": 0,
        "EndRow": 0
      },
      "Availability": "Available",
      "NoDataValue": -999.0,
      "StartDate": "20150104020000",
      "EndDate": "20150104030000",
      "DataType": "Single",
      "VariableCode": "Test.Json.Grid.T",
      "DataSourceVariableId": 1021
    }
  ],
  "Meta": {
    "Variables": {
      "Test.Json.Grid.T": {
        "VariableId": 2989,
        "Name": "Temperature",
        "Code": "Test.Json.Grid.T",
        "UnitCode": "K"
      }
    },
    "DataSourceVariables": {
      "1021": {
        "DataSourceVariableId": 1021,
        "VariableId": 2989,
        "DataSourceCode": "Test.Json.Grid",
        "UnitCode": "C",
        "Name": "Temperature",
        "Code": "T",
        "DataType": "Single",
        "NoDataValue": -999.0,
        "MathematicalType": "NotSummable",
        "MeasurementType": "Period",
        "State": 1,
        "IsCumulative": false
      }
    }
  },
  "StructureType": "Grid"
}
```

Calculating coordinates

Grids, model grids and ensemble grids are given in one-dimensional arrays. In order to use this data, a translation must be made to a 2D matrix with coordinates. The pseudo-code below gives the relevant formulas for calculating the current column, row and coordinates for the given (zero-based) index. Note that Xll and Yll in the GridDefinition are the bottom left coordinates of the bottom left cell. Xur and Yur are the top right coordinates of the top right cell.

```
foreach (grid in response.Data) {
  grid = this;
  gridDefinition = grid.GridDefinition;
  columns = gridDefinition.Columns;
  rows = gridDefinition.Rows;
  cellWidth = gridDefinition.CellWidth;
  cellHeight = gridDefinition.CellHeight;
  xll = gridDefinition.Xll;
  yll = gridDefinition.Yll;

  for(i = 0; i < grid.Data.length; i++) {
    column = i % columns;
    row = floor(i / columns);
    cellLowerLeftX = xll + (column * cellWidth);
    cellLowerLeftY = yll + ((rows – row - 1) * cellHeight;
    cellUpperLeftX = xll + ((column + 1) * cellWidth);
    cellUpperLeftY = yll + ((rows - row) * cellHeight);
    cellCenterX = xll + ((column + 0.5) * cellWidth);
    cellCenterY = yll + ((rows - row - 0.5) * cellHeight);
  }
}
```

The following URLs are used for retrieving the GridDefinitions and Projections (relative to %api_url%):

- /entity/griddefinitions/get
- /entity/projections/get

Stereographic KNMI Radar projection:
The KNMI Radar DataSources contain the Stereographic KNMI Radar projection. This is a projection used by the KNMI and is not linked to an EPSG projection code. In WIWB an EPSG projection code has to be defined, as such the Stereographic KNMI Radar projection is linked to a custom EPSG projection in WIWB (code = 100000). The projection has the following projection string: +proj=stere +lat_0=90 +lat_ts=60 +lon_0=0 +x_0=0 +y_0=0 +a=6378140 +b=6356750 +units=km

D.3.5 Supported data formats

It is not recommended to request gridded data as JSON. More suitable data formats are given below.

For grids the following data formats are available in addition to the default JSON format:

- GeoTIFF: DataFormatCode = ‘geotiff’ (one file per grid).
- ESRI ASCII Grid: DataFormatCode = ‘aaigrid’ (one file per grid).
- HDF5: DataFormatCode = ‘hdf5’ (fixed file layout, one file per grid).
- NetCDF: DataFormatCode = ‘netcdf4.cf1p6’ (fixed file layout, not all DataSources exports follow CF1.6 conventions, number of timestamps per NetCDF file is limited to 1200).
- NetCDF zipped: DataFormatCode = ‘netcdf4.cf1p6.zip’ (fixed file layout, one NetCDF file per day, given as a zip, not all DataSources exports follow CF1.6 conventions).

D.4 Sending download requests

In addition to retrieving data directly from the API, it is also possible to send a download request that is processed by the platform at a convenient time. This is useful for requesting large data sets. Instead of using the %api_url%`/%structuretype%/get` action, use %api_url%`/%structuretype%/createdownload`.

The request body is the same as with the get action and the HTTP method is POST. The result is a data flow id that can be used to retrieve the file with the following call (HTTP GET method):

%api_url%`/data/downloadfile?dataflowid=%dataflowid%`

The file is only returned if the data flow has finished successfully.

D.4.1 Download request example

Generate a download request:
Download 1day precipitation (uncorrected radar) with a 5 minute interval - GeoTIFF

URL	%api_url%/grids/createdownload

request body	`{`
	` "Readers": [{`
	`  "DataSourceCode": "Knmi.Radar.Uncorrected",`
	`  "Settings": {`
	`   "StartDate": "20171205090000",`
	`   "EndDate": "20171206090000",`
	`   "VariableCodes": ["P"]`
	`  }`
	` }],`
	` "Exporter": {`
	`  "DataFormatCode": "geotiff"`
	` },`
	` "DataFlowTypeCode": "Download",`
	` "DataSourceCode": "Knmi.Radar.Uncorrected"`
	`}`

The DataFlowId is returned in the response.

Request the download status:

URL	%api_url%/entity/dataflows/get

request body	`{`
	` "DataFlowIds": [%dataflowid%]`
	`}`

The download status is returned in the response (Idle / Executing / Finished).

To download the data:
- If the download status is finished, change from POST to GET.
- %api_url%/grids/downloadfile?dataflowid=%dataflowid%
- Change %dataflowid% with the DataFlowId. Execute and download the file.