WET DOG
WEATHER

wetdogweather.com

PRODUCT

# GEOSERVICE

# How Does GeoService Work?

Our GeoService supports two OGS standards: WMTS and the older WMS. Both are still in active use on the web, and we plan to support them indefinitely.

Both standards are similar, with WMTS being the more modern one. They consist of two kinds of calls:

1. GetCapabilities - Find out what's available
2. GetMap (or tile) - Get a visual representation of the data

Let's start with a bit of bookkeeping on the GeoService itself.

## Your GeoService Endpoint

Our customers have their own endpoints for GeoService, as they do for our other products. These will allow access to all of their data, everything they're using from our standard data sets, and anything they may be buying or providing.

We'll give you an endpoint for our services of the form *<company>*.api.wetdogweather.com. In the rest of this document, we'll refer to that URL as *{{service}}.*

## Authorization

We use the standard authorization header in an HTTP call, but we can also accept an authorization parameter (*&auth=<API key>*) in the URL.

At some point, we will turn on authorization for the GeoService, so it's best to build it into your services now. We can give you auth tokens to use as you like.

## Hiding Our Endpoints & Building Your Own Service

If you're white-labeling our service (we don't mind), you'll have a set of endpoints similar to ours. The simplest approach is to just forward the /geoservice calls on to our service. The endpoint we give you is stable and includes a CDN to cut down on traffic.

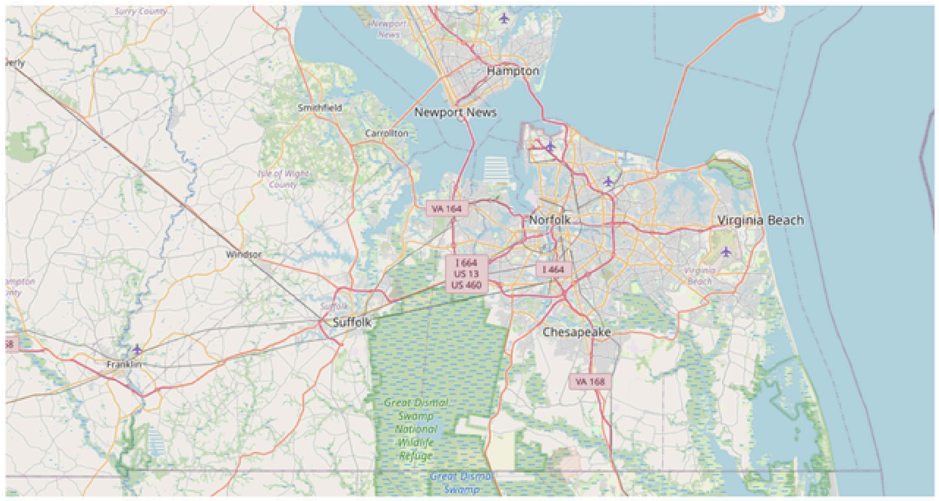
One wrinkle, if you're white-labeling our GeoService, is the URLs. The Capabilities returns have a lot of URLs, and by default, they all point back to us. Luckily, you can override that with the *X-WDW-Base-Address* header. Just set that in the header of your GetCapabilities request, and we'll supply that as the base {{service}} we return.

Now let's get back to WMS and WMTS. We'll start with what they are.

## Web Mapping Service

Born in the late nineties, WMS is a straightforward standard for fetching a chunk of a rendered map, one of the original web mapping services. This was a time when servers were smart, and clients were dumb. Web browsers were improving, but AJAX wouldn't appear for another half-decade, and mobile was still very simple.

WMS works in two parts, though most people skip the first part.
1. Ask the server for Capabilities. That is what it has to offer.
2. Ask the server to construct an image at a given resolution for a given area.



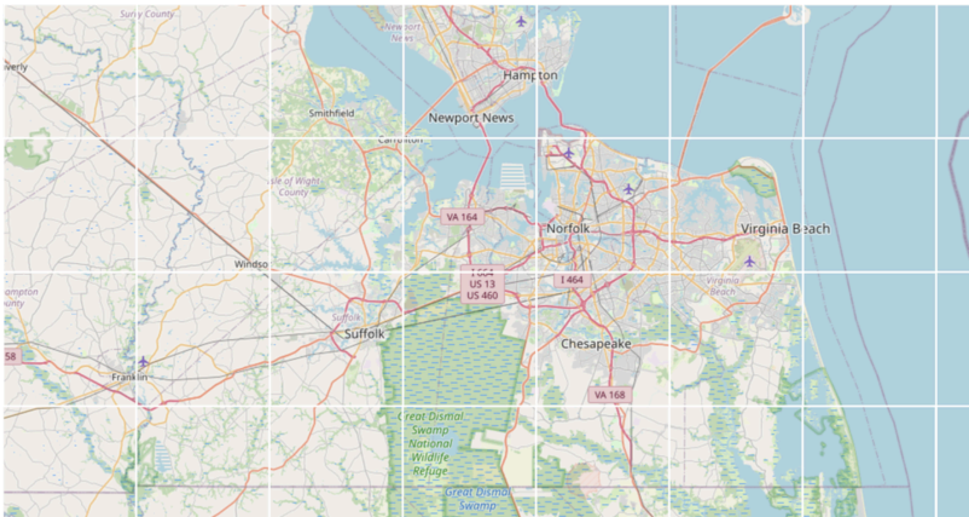
Once you know what the server has, you typically just ask for a new area over and over again. Pretty simple to use, but it's got some significant downsides.

- The user spends a lot of time waiting for an image to appear.
- Server work scales by the number of active users.
- It defies modern caching techniques.

Sometimes you need it anyway, and we'll talk about how to access it at Wet Dog Weather shortly.

## Web Map Tile Service

A newer standard, WMTS, is pretty good for web mapping and not horrible for weather data. With WMTS, the client can request individual tiles in a tile stack.



Each map is broken into a tiling system that usually starts at zero and works its way up to some larger number. WMTS is flexible, but we only use quad trees, with the number of tiles quadrupling as each zoom level increases.

This is what most display packages expect, and you should have no trouble using our tile sources accordingly.

The OGC knew what it was doing and designed WMTS better than WMS. It's pretty good, and it works nicely with modern caching techniques. Image pyramids are broken into predictable sizes and can be fetched as needed by a client and cached in your CDN.

As with WMS, there's a GetCapabilities step, and then most users hardwire the URLs they want to use. This can normally be tricky with weather data, but we've added a few shortcuts to help out.

We'll work through WMTS, which is what you should use if you can't use Terrier. Then we'll circle back to WMS.

## WMTS GetCapabilities

The purpose of the GetCapabilities call is to see what the service has available. The way most people use it is to poke through the contents and figure out how to call it directly. This works well with static map data, but gets a little tricky when constantly updating weather data.

The Capabilities return is XML, and ours will list the various layers we have available. Here is the wind speed from HRRR in the 'web mercator' coordinate system. That is, a normal sort of tile set you could put on top of all the most commented maps.

```
<Layer>
    <ows:Identifier>hrrr-conus-natf-wind_speed-16-epsg_3857</ows:Identifier>
    <ows:WGS84BoundingBox>
        <ows:LowerCorner>-135.0  16.63619187839766</ows:LowerCorner>
        <ows:UpperCorner>-56.25  16.63619187839766</ows:UpperCorner>
    </ows:WGS84BoundingBox>
```

The identifiers are our own and won't change. This one tells us that it's from the HRRR model, covers CONUS, and is wind_speed. We tack the coordinate system on the end and currently provide two, the other being *epsg:4326*.

After the layer information, we provide a list of Styles the data can be rendered with. Each style will have a distinct name and look something like this:

```
<Style>
    <ows:Title>Sequential Red to Yellow</ows:Title>
    <ows:Abstract>Sequential Red-Yellow Colormap</ows:Abstract>
    <ows:Identifier>mp_autumn</ows:Identifier>
```

There is also a *LegendURL* which points to an image you can use as a legend to describe the colormap. It will look something like this:

```xml
<LegendURL format="image/png"
xlink:href="{{service}}/geoservice/colorbar?unit=m/s&amp;min=0&amp;max=100&amp;
title=Wind_speed-80m&amp;cname=mp_gnuplot2&amp;transparent=True&amp;dpi=150"/>
```

Those extra "&amp;"s are a function of XML. The URL is normal when decoded from XML.

The secret with our Styles is that we store raw-ish data in the appropriate coordinate system and then convert that data into visuals when asked. We support a dozen or so styles and are always happy to add new ones.

For a model like HRRR, there are two components to the time and one to height. We break the time components into RUN and FORECAST. They look a bit like this.

```
<Dimension>
    <ows:Identifier>RUN</ows:Identifier>
    <Default>2026-02-10T15:00:00Z</Default>
    <Value>2026-02-09T00:00:00Z</Value>
    <Value>2026-02-09T08:00:00Z</Value>
    <Value>2026-02-09T16:00:00Z</Value>
    <Value>2026-02-10T00:00:00Z</Value>
    <Value>2026-02-10T08:00:00Z</Value>
    <Value>2026-02-10T16:00:00Z</Value>
    <Value>2026-02-09T01:00:00Z</Value>
    <Value>2026-02-09T09:00:00Z</Value>
    <Value>2026-02-09T17:00:00Z</Value>
```

```
<Dimension>
    <ows:Identifier>FORECAST</ows:Identifier>
    <Default>PT0H</Default>
    <Value>PT0H</Value>
    <Value>PT1H</Value>
    <Value>PT2H</Value>
    <Value>PT4H</Value>
    <Value>PT5H</Value>
    <Value>PT6H</Value>
    <Value>PT7H</Value>
    <Value>PT8H</Value>
    <Value>PT9H</Value>
```

Together, they specify a forecast slice. You can substitute *"latest"* for RUN and just get the latest complete model run.

Next up is elevation, if it's needed. For something like *wind_speed*, we'd have this.

```
<Dimension>
    <ows:Identifier>ELEVATION</ows:Identifier>
    <Default>10m</Default>
    <Value>10m</Value>
    <Value>100m</Value>
    <Value>122m</Value>
    <Value>305m</Value>
    <Value>30m</Value>
    <Value>457m</Value>
    <Value>50m</Value>
    <Value>61m</Value>
    <Value>80m</Value>
```

That begs the question as to where all these values go. For that, we have a ResourceURL that looks a bit like this.

```xml
<ResourceURL format="image/png" resourceType="simpleProfileFile"
template="{{service}}/geoservice/tile/v2/hrrr/conus/natf/wind_speed/{ELEVATION}
/{RUN}/{FORECAST}/16/{TileMatrix}/{TileCol}/{TileRow}.png?epsg=3857&amp;project
ed=true&amp;mapping=data&amp;cname={Style}"/>
```

This is the meat of WMTS and the URL you're probably looking to tease out. Everything enclosed in {} needs to be filled in, except for {{service}}, which we discussed above.

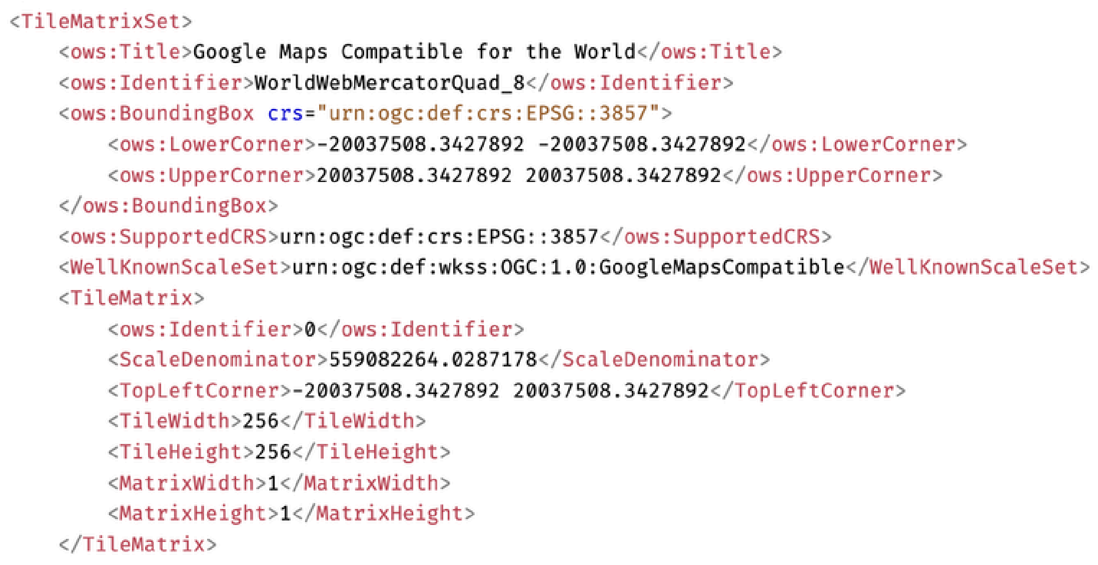
- RUN: One of the model runs given in the Capabilities or "*latest*"
- FORECAST: One of the PT values given in the Capabilities
- ELEVATION: One of the elevation values given in Capabilities
- Style: The name of one of the styles listed in this Layer
- TileMatrix, TileCol, TileRow: Typically, the map toolkit fills these in

If your map toolkit has trouble with this kind of replacement URL, you can substitute z/x/z for the Tile logic. That's the order things are in. But the Z brings up a question. How deep does this tile set go?

We have more than enough information on that for you, contained in the Tile Matrix Set. Each Layer will have one of these.

```
<TileMatrixSetLink>
    <TileMatrixSet>WorldWebMercatorQuad_8</TileMatrixSet>
</TileMatrixSetLink>
```

The Tile Matrix Set tells us how the data is spatially structured. It's an XML Link, which probably seemed like a good idea when they were designing XML. Luckily, the meaning is pretty simple here. This is an 8-level web mercator map. You can see all that in glorious detail near the bottom.



```
<TileMatrixSet>
    <ows:Title>Google Maps Compatible for the World</ows:Title>
    <ows:Identifier>WorldWebMercatorQuad_8</ows:Identifier>
    <ows:BoundingBox crs="urn:ogc:def:crs:EPSG::3857">
        <ows:LowerCorner>-20037508.3427892 -20037508.3427892</ows:LowerCorner>
        <ows:UpperCorner>20037508.3427892 20037508.3427892</ows:UpperCorner>
    </ows:BoundingBox>
    <ows:SupportedCRS>urn:ogc:def:crs:EPSG::3857</ows:SupportedCRS>
    <WellKnownScaleSet>urn:ogc:def:wkss:OGC:1.0:GoogleMapsCompatible</WellKnownScaleSet>
    <TileMatrix>
        <ows:Identifier>0</ows:Identifier>
        <ScaleDenominator>559082264.0287178</ScaleDenominator>
        <TopLeftCorner>-20037508.3427892 20037508.3427892</TopLeftCorner>
        <TileWidth>256</TileWidth>
        <TileHeight>256</TileHeight>
        <MatrixWidth>1</MatrixWidth>
        <MatrixHeight>1</MatrixHeight>
    </TileMatrix>
```

It's rare that anyone needs to parse this, and the one piece of information you can take away is that this data goes from zoom level 0 to 8. Most map toolkits can handle that fine.

## WMTS Tiles

Here's the funny thing about WMTS. Once you've got your ResourceURL teased out, picked your style, and figured out the min and max zoom level, that's enough to display. If you stick to the latest RUN, you can just use the list of FORECAST times, and you're off!

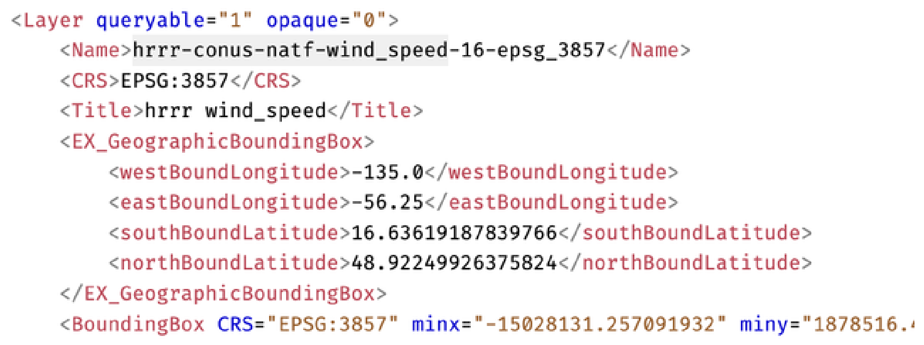
Here’s an example URL from HRRR where we’re pulling the latest wind speed at 10m.

```
None
{{service}}/geoservice/tile/v2/hrrr/conus/natf/wind_speed/10m/latest/default/16
/0/0/0.png?epsg=4326&projected=true&mapping=data&cname=truwx_wind
```

Some of our users have noticed we have our own grammar for these URLs and may start tweaking them on their own. If you do something unusual, let us know so we can keep the old syntax in place for you.

## WMS GetCapabilities

In many ways, WMS is even simpler to use, even if it is a lot slower. It starts out the same with a GetCapabilities call. From our HRRR example earlier, the structure is similar.



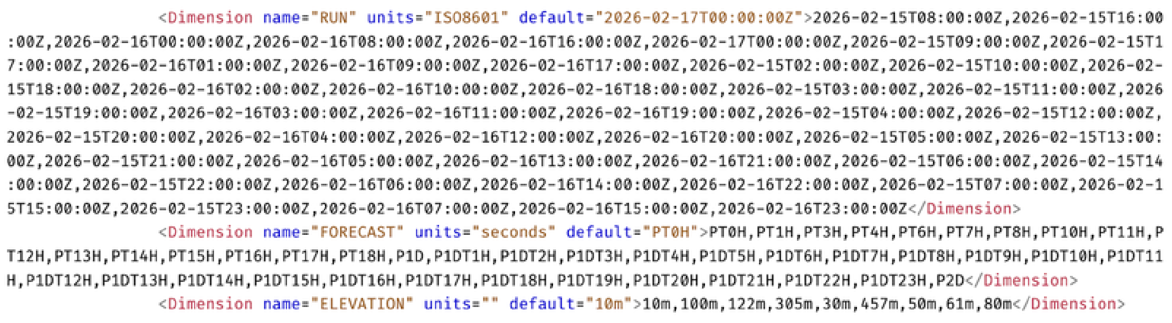
```
<Layer queryable="1" opaque="0">
    <Name>hrrr-conus-natf-wind_speed-16-epsg_3857</Name>
    <CRS>EPSG:3857</CRS>
    <Title>hrrr wind_speed</Title>
    <EX_GeographicBoundingBox>
        <westBoundLongitude>-135.0</westBoundLongitude>
        <eastBoundLongitude>-56.25</eastBoundLongitude>
        <southBoundLatitude>16.63619187839766</southBoundLatitude>
        <northBoundLatitude>48.92249926375824</northBoundLatitude>
    </EX_GeographicBoundingBox>
    <BoundingBox CRS="EPSG:3857" minx="-15028131.257091932" miny="1878516.
```

Those will also have LegendURLs that might look like this:

```
XML
<LegendURL>
 <Format>image/png</Format>
 <OnlineResource xlink:type="simple"
xlink:href="{{service}}/geoservice/colorbar?unit=m/s&amp;min=0&amp;max=100&amp;
title=Wind_speed-80m&amp;cname=mp_viridis&amp;transparent=True&amp;dpi=150"/>
</LegendURL>
```

The {{service}} portion will point back to your endpoint, of course. This is otherwise identical in function to the WMTS legends.

For the RUN, FORECAST, and Elevation, it’s there but specified a little differently.



```
                <Dimension name="RUN" units="ISO8601" default="2026-02-17T00:00:00Z">2026-02-15T08:00:00Z,2026-02-15T16:00
:00Z,2026-02-16T00:00:00Z,2026-02-16T08:00:00Z,2026-02-16T16:00:00Z,2026-02-17T00:00:00Z,2026-02-15T09:00:00Z,2026-02-15T1
7:00:00Z,2026-02-16T01:00:00Z,2026-02-16T09:00:00Z,2026-02-16T17:00:00Z,2026-02-15T02:00:00Z,2026-02-15T10:00:00Z,2026-02-
15T18:00:00Z,2026-02-16T02:00:00Z,2026-02-16T10:00:00Z,2026-02-16T18:00:00Z,2026-02-15T03:00:00Z,2026-02-15T11:00:00Z,2026
-02-15T19:00:00Z,2026-02-16T03:00:00Z,2026-02-16T11:00:00Z,2026-02-16T19:00:00Z,2026-02-15T04:00:00Z,2026-02-15T12:00:00Z,
2026-02-15T20:00:00Z,2026-02-16T04:00:00Z,2026-02-16T12:00:00Z,2026-02-16T20:00:00Z,2026-02-15T05:00:00Z,2026-02-15T13:00:
00Z,2026-02-15T21:00:00Z,2026-02-16T05:00:00Z,2026-02-16T13:00:00Z,2026-02-16T21:00:00Z,2026-02-15T06:00:00Z,2026-02-15T14
:00:00Z,2026-02-15T22:00:00Z,2026-02-16T06:00:00Z,2026-02-16T14:00:00Z,2026-02-16T22:00:00Z,2026-02-15T07:00:00Z,2026-02-1
5T15:00:00Z,2026-02-15T23:00:00Z,2026-02-16T07:00:00Z,2026-02-16T15:00:00Z,2026-02-16T23:00:00Z</Dimension>
                <Dimension name="FORECAST" units="seconds" default="PT0H">PT0H,PT1H,PT3H,PT4H,PT6H,PT7H,PT8H,PT10H,PT11H,P
T12H,PT13H,PT14H,PT15H,PT16H,PT17H,PT18H,P1D,P1DT1H,P1DT2H,P1DT3H,P1DT4H,P1DT5H,P1DT6H,P1DT7H,P1DT8H,P1DT9H,P1DT10H,P1DT11
H,P1DT12H,P1DT13H,P1DT14H,P1DT15H,P1DT16H,P1DT17H,P1DT18H,P1DT19H,P1DT20H,P1DT21H,P1DT22H,P1DT23H,P2D</Dimension>
                <Dimension name="ELEVATION" units="" default="10m">10m,100m,122m,305m,30m,457m,50m,61m,80m</Dimension>
```

It's a bit dense, but the idea is the same. We list the model RUNs, the FORECAST offsets, and available ELEVATIONS.

The way you use this in WMS is very, very different.

## WMS GetMap

The heart of WMS is the GetMap call. You pass in a bunch of parameters, and the service returns an image to you. This is slower than using tiles, and it's from a pre-smart browser era, but it works. And it works a little like this.

```None
{{service}}/geoservice?VERSION=1.3.1&REQUEST=GetMap&SERVICE=WMS&LAYERS=hrrr-con
us-natf-wind_speed-16-epsg_3857&WIDTH=1024&HEIGHT=1024&BBOX=-10797990.606947536
,2999080.943470641,-9684795.6990148,4439106.787250585&STYLES=mp_viridis&RUN=lat
est&FORECAST=PT18H&ELEVATION=10m
```

That will return to you a very purple map at the moment.

How do you know to do that with the VERSION number, URL, and such? Well, it's all in the Capabilities, but most people just hardwire it. Let's look at each of the parameters.

| **Parameter** | **Purpose** |
|---|---|
| VERSION | In theory, support for multiple versions of 1.3.1. Just set it to 1.3.1 |
| REQUEST | GetMap. It's a GetMap Request. |
| WMS | Using the WMS service. This is pretty old school web 1.0 boilerplate. |
| LAYERS | The layer identifier we want to use. We got this from the Capabilities file. |
| WIDTH, HEIGHT | Size of the image you'd like returned. |
| BBOX | The bounding box of the image to render in the local coordinate system. This is Web Mercator in this case, but your map toolkit should understand this part if it speaks WMS. |
| STYLES | Usually, just one style to render the data. Consult the Capabilities for a list of styles. |

| | |
|---|---|
| RUN | Either one of the RUNs listed in Capabilities, or *latest* to just pick the most recent completed model run. |
| FORECAST | Forecast offset from the run. In this case, it's PT18H, so the 18th hour of the HRRR forecast. |
| ELEVATION | If an elevation is needed, provide it here. The list is in the Capabilities return. The default will be the lowest, if not provided. |

You can leave RUN, FORECAST, and ELEVATION empty, and it will provide defaults. We'd recommend filling them in anyway, just to avoid ambiguity.