Accessing EMODnet Seabed Habitats’ OGC services

Jordan Pinder

2023-02-03

Document Control

Version	Date	Author	Comments
0.1	29-03-2021	Jordan Pinder	Initial documentation to access WFS services, basic data manipulation and visualisation
0.2	09-07-2021	Jordan Pinder (Ed. Harriet Allen)	
0.3	13-01-2023	Jordan Pinder (Ed. Ashley Elliott)	Updated the layer names and URL links to direct to the central portal and new map viewer.

Introduction

The European Marine Observation and Data Network (EMODnet) is a network of organisations supported by the EU’s integrated maritime policy. These organisations work together to observe the sea, process the data according to international standards and make that information freely available as interoperable data layers and data products.

Seabed Habitats was one of seven themes of the EMODnet initiative. Since its inception in 2009, EMODnet Seabed Habitats developed, improved and gradually increased the coverage of a broad-scale seabed habitat map for Europe’s seabed, also known as EUSeaMap. It also has a large library of benthic habitat maps from surveys, collections of benthic habitat point observations, composite data products such as Essential Ocean Variables and the OSPAR threatened and/or Declining Habitats database, and environmental data products such as light availability and energy at the seabed.

Seabed Habitats provides all of its spatial data through a series of web services in the Open Geospatial Consortitum (OGC) format, including:

- Web Mapping Services (WMS): access to pre-styled image layers as seen on the interactive mapper;
- Web Feature Services (WFS): access to “real” vectorised data formats including `geoJSON`, `ESRI Shapefile`, `GML`;

This tutorial is to serve as a guide for accessing `WFS` data from EMODnet Seabed Habitats.

Specific blogposts used during this tutorial were:

- `ows4R` - `R` interface for OGC web services. A good tutorial is here, GitHub repo is here.
- Drawing beautiful maps programatically with `R`, `sf`, and `ggplot2` - part one, part two, and part three.
- `xmltools` - useful tools for extracting `xml` based data in a tidy format, a good tutorial is here, GitHub repo is here.

Packages

Firstly we'll need to install and load our packages.

```
# install.packages(c('devtools', 'rgeos', 'janitor',
# 'httr', 'xml2', 'ows4R', 'tidyverse', 'dplyr', 'sf',
# 'ggrepel', 'glue', 'ggthemes', 'rnaturalearth',
# 'rnaturalearthdata'))

# devtools::install_github('dantonnoriega/xmltools')

library(httr)
library(xml2)
library(xmltools)
library(ows4R)
library(tidyverse)
library(dplyr)
library(sf)
library(rvest)
library(tidygeocoder)
library(glue)
library(ggthemes)
library(ggrepel)
library(rnaturalearth)
library(rnaturalearthdata)
library(rgeos)
library(janitor)
library(kableExtra)
```

Accessing WFS data

URL endpoint

Get the URL of the web services. Below is the main one we'll be using, but see here for a full list of web services.

```
wfs_main <- "https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_open/wfs
"
```

Next, we append information to the URL address with the aid of `httr::parse_url` and `httr::build_url`. The former function parses an URL into a list for easier programmatic addition of information to the URL. The latter function does the reverse and builds the URL from the list object.

The `url$query` slot is where you instruct the WFS what information it should return. It is constructed as a list with name-value pairs. For now, we only need to specify the `GetCapabilities` request. Other information such as `version` information (e.g. `version = 2.0.0`) can be added but is not required (by default, the latest version of the WFS service will be chosen).

```
url <- httr::parse_url(wfs_main)

url$query <- list(service = "wfs",
                  #version = "2.0.0", # facultative
                  request = "GetCapabilities"
                  )


request <- httr::build_url(url)
request

## [1] "https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_open/wfs?serv
ice=wfs&request=GetCapabilities"
```

GetCapabilities

With `GetCapabilities`, we obtain a complete overview of all metadata for the web service.

To see all capabilities, you can visit the request in the webbrowser. For instance opening the page in the web browser and searching for `Filter_Capabilities` allows you to see all possible ways to filter the data from a WFS layer (e.g. restrict the downloaded data to a specified bounding box with `SpatialOperator name="BBOX"`).

Instead of searching the page on the web, there are several ways to access specific pieces of information programmatically. We will show here how to do this using functions in the `ows4R` package.

The first thing we need to do is generate a connection to the WFS with the aid of `WFSClient$new()`.

```
emodnet_client <- WFSClient$new(wfs_main, serviceVersion = "2.0.0")  # servic
eVersion must be provided here

emodnet_client
```

Query layers

The features listed can be accessed using `$`. As a first example, the following code will list all available layers for that WFS which will return the layer name and title.

```
emodnet_client$getFeatureTypes(pretty = TRUE)
```

name	title
emodnet_open:art17_hab_1150	2013 Article 17 reporting gridded Annex I habitat distribution - Coastal lagoons (1150)
emodnet_open:art17_hab_1130	2013 Article 17 reporting gridded Annex I habitat distribution - Estuaries (1130)
emodnet_open:art17_hab_1160	2013 Article 17 reporting gridded Annex I habitat distribution - Large shallow inlets and bays (1160)
emodnet_open:art17_hab_1140	2013 Article 17 reporting gridded Annex I habitat distribution - Mudflats and sandflats not covered by seawater at low tide (1140)
emodnet_open:art17_hab_1120	2013 Article 17 reporting gridded Annex I habitat distribution - Posidonia beds (1120)
emodnet_open:art17_hab_1170	2013 Article 17 reporting gridded Annex I habitat distribution - Reefs (1170)
emodnet_open:art17_hab_1110	2013 Article 17 reporting gridded Annex I habitat distribution - Sandbanks (1110)
emodnet_open:art17_hab_1180	2013 Article 17 reporting gridded Annex I habitat distribution - Submarine structures made by leaking gases (1180)

emodnet_open:art17_hab2018_all	2018 Article 17 reporting gridded Annex I marine habitat distribution
emodnet_open:biogenic_substrate_2021	Biogenic substrate in Europe
emodnet_open:habitat_point_eunis	Collated EUNIS Habitat point data - public records
emodnet_open:habitat_point	Collated Habitat point data - All classification systems, public records
emodnet_open:gems_annexi_full	Collection of AnnexI habitats extracted from GeMS (Full resolution)
emodnet_open:coralligenous_platforms_2021	Coralligenous and other calcareous bioconcretions in the Mediterranean
emodnet_open:coralligenous_platforms_points_2021	Coralligenous and other calcareous bioconcretions in the Mediterranean (points)
emodnet_open:eusm2019_bio_full	EUSeaMap (2019) Biozone habitat descriptor - full detail
emodnet_open:eusm2019_ene_full	EUSeaMap (2019) Energy Class habitat descriptor - full detail
emodnet_open:eusm2019_subs_full	EUSeaMap (2019) Substrate habitat descriptor - full detail
emodnet_open:eusm2021_bio_full	EUSeaMap (2021) Broad-Scale Predictive Habitat Map - Biological zones (a habitat descriptor)
emodnet_open:eusm2021_ene_full	EUSeaMap (2021) Energy Class habitat descriptor - full detail
emodnet_open:eusm2021_subs_full	EUSeaMap (2021) Substrate habitat descriptor - full detail
emodnet_open:eusm2019_regions	EUSeaMap 2019 Regions

emodnet_open:eov_2021_points_hard_coral	Essential Ocean Variables in Europe - Live hard coral cover (points, v2021)
emodnet_open:eov_livecoral	Essential Ocean Variables in Europe - Live hard coral cover (polygons, v2019)
emodnet_open:eov_2021_points_Macroalgal_canopy_cover	Essential Ocean Variables in Europe - Macroalgal canopy cover (points, v2021)
emodnet_open:eov_macroalgae	Essential Ocean Variables in Europe - Macroalgal canopy cover (polygons, v2019)
emodnet_open:eov_2021_points_Seagrass_cover	Essential Ocean Variables in Europe - Seagrass cover (points, v2021)
emodnet_open:eov_seagrass	Essential Ocean Variables in Europe - Seagrass cover (polygons, v2019)
emodnet_open:habitat_point_bboxes	Habitat point data - dataset bounding boxes
emodnet_open:ospar2020_points	OSPAR Habitats in the North-East Atlantic Ocean - 2020 Point records
emodnet_open:ospar2020_poly	OSPAR Habitats in the North-East Atlantic Ocean - 2020 Polygons
emodnet_open:ospar_points	OSPAR threatened and/or declining habitats - Points (2018)
emodnet_open:ospar_overview	OSPAR threatened and/or declining habitats - Polygon Overview
emodnet_open:ospar_poly	OSPAR threatened and/or declining habitats - Polygons (2018)
emodnet_open:ICES_VME_Dataset_PublicRecords	Public VME Records from the ICES WFS

This is an `R6` class object and the `$` can be used to chain together several functions, much in the same way as the pipe operator `%>%`.

There are a couple ways you can search the available feature types. The following searches for the `polygon` layer within the OSPAR Threatened & Declining Habitats database:

```
emodnet_client$getCapabilities()$findFeatureTypeByName("emodnet_open:ospar_po
ly")$getDescription() %>%
    map_chr(function(x) {
        x$getName()
    })
```

```
##  [1] "gui"           "recordkey"     "habtype"       "habsubtype"    "habstatu
s"
##  [6] "certainty"     "determiner"    "detdate"       "surveykey"     "startdat
e"
## [11] "enddate"       "datetype"      "placename"     "dataowner"     "accuracy
"
## [16] "althabtype"    "althabclas"    "althabrel"     "shape_length"  "shape_ar
ea"
## [21] "geom"
```

As alternative approach, this time searching for the `point` OSPAR data:

```
emodnet_client$describeFeatureType(typeName = "emodnet_open:ospar_points") %>
%
    map_chr(function(x) {
        x$getName()
    })
```

```
##  [1] "gui"          "recordkey"    "habtype"      "habsubtype"   "habstatus"
##  [6] "certainty"    "determiner"   "detdate"      "surveykey"    "startdate"
## [11] "enddate"      "datetype"     "placename"    "dataowner"    "accuracy"
## [16] "latitude"     "longitude"    "althabtype"   "althabclass"  "althabrel"
## [21] "shape"
```

We can also search by other feature types such as the EUSeaMap regions:

```
emodnet_client$getCapabilities()$findFeatureTypeByName("emodnet_open:eusm2019
_regions")$getDescription() %>%
    map_chr(function(x) {
        x$getName()
    })
```

```
## [1] "euseamap_r" "shape_leng" "shape_area" "geom"
```

Or `point` data from the [EUNIS habitat database](#):

```
emodnet_client$getCapabilities()$findFeatureTypeByName("emodnet_open:habitat_
point_eunis")$getDescription() %>%
    map_chr(function(x) {
```

```
        x$getName()
    })
```

```
##  [1] "objectid"                         "measurementid"
##  [3] "eventid"                          "datasetid"
##  [5] "shorttitle"                       "expectedcitation"
##  [7] "restriction"                      "contactpoints"
##  [9] "eventdate"                        "mindepth"
## [11] "maxdepth"                         "seabedclassificationsystem"
## [13] "seabedclassificationsystem_uri" "seabedtype"
## [15] "seabedtype_uri"                   "seabedstatus"
## [17] "samplingmethod"                   "samplingmethod_uri"
## [19] "seabedtypedetermineddate"         "seabedtypedeterminedmethod"
## [21] "sourcehabitatoccurrenceid"        "relationshiptosourcehabitat"
## [23] "comments"                         "geom"
## [25] "eventenddate"                     "eunis_l3"
```

Now we've found some layers matching our search, we now want to extract them. First, let's see what operations are available using the WFS:

```
emodnet_client$getCapabilities()$getOperationsMetadata()$getOperations() %>%
    map_chr(function(x) {
        x$getName()
    })
```

```
## [1] "GetCapabilities"       "DescribeFeatureType"   "GetFeature"
## [4] "GetPropertyValue"      "ListStoredQueries"     "DescribeStoredQueries
"
## [7] "CreateStoredQuery"     "DropStoredQuery"
```

The next chunk shows how we can extract the available output formats. We will see later that `GetFeature` is the operation needed to read or download data from the WFS. The metadata for this operation has what we want and we can extract it with a combination of `purrr::map()` and `purrr::pluck()`.

```
emodnet_client$getCapabilities()$getOperationsMetadata()$getOperations() %>%
    map(function(x) {
        x$getParameters()
    }) %>%
    pluck(3, "outputFormat")
##  [1] "application/gml+xml; version=3.2"
##  [2] "GML2"
##  [3] "KML"
##  [4] "SHAPE-ZIP"
##  [5] "application/json"
##  [6] "application/vnd.google-earth.kml xml"
##  [7] "application/vnd.google-earth.kml+xml"
##  [8] "csv"
##  [9] "gml3"
## [10] "gml32"
## [11] "json"
```

```
## [12] "text/csv"
## [13] "text/xml; subtype=gml/2.1.2"
## [14] "text/xml; subtype=gml/3.1.1"
## [15] "text/xml; subtype=gml/3.2"
```

Some extra examples include: extracting the bounding boxes for all layers.

```
emodnet_client$getCapabilities()$getFeatureTypes() %>%
    map(function(x) {
        x$getBoundingBox()
    })
## [[1]]
##          min      max
## x -47.58017 41.95249
## y  24.81188 66.48150
##
## [[2]]
##          min      max
## x -24.26399 41.77421
## y  34.65221 66.48619
##
## [[3]]
##          min      max
## x -47.68894 42.30771
## y  26.94096 66.48399
##
## [[4]]
##          min      max
## x -37.06373 41.77421
## y  29.97137 66.44175
##
## [[5]]
##          min      max
## x -7.186059 37.02481
## y 32.452611 45.67816
##
## [[6]]
##          min      max
## x -51.12328 52.33401
## y  23.41187 66.33389
##
## [[7]]
##          min      max
## x -57.48689 59.53411
## y  23.10754 71.02764
##
## [[8]]
##          min      max
## x -34.14763 12.72759
## y  27.55606 60.78477
```

```
##
## [[9]]
##         min      max
## x -35.56893 34.17924
## y  27.55606 70.16985
##
## [[10]]
##         min      max
## x -17.41497 34.35826
## y  32.58452 64.09822
##
## [[11]]
##         min      max
## x -15.84686 29.59930
## y  34.89477 60.86076
##
## [[12]]
##         min      max
## x -16.35679 41.74913
## y  34.89477 60.86076
##
## [[13]]
##         min         max
## x -11.45586  0.03196578
## y  54.60035 60.81498852
##
## [[14]]
##         min       max
## x -29.08698 -29.08670
## y  12.99385  12.99403
##
## [[15]]
##         min      max
## x  2.939068 28.34579
## y 36.312670 43.80589
##
## [[16]]
##         min      max
## x -36.39353 43.10083
## y  22.06430 80.20107
##
## [[17]]
##         min      max
## x -25.34000 43.34000
## y  24.61456 81.09517
##
## [[18]]
##         min      max
## x -35.77074 41.40184
## y  27.94162 78.51768
```

```
##
## [[19]]
##         min      max
## x -36.39500 43.39500
## y  22.36996 81.09706
##
## [[20]]
##         min      max
## x -36.39500 43.39500
## y  22.36996 81.09706
##
## [[21]]
##         min      max
## x -35.41252 43.02791
## y  27.17631 80.61796
##
## [[22]]
##         min      max
## x -36.39500 43.39500
## y  21.91587 81.09744
##
## [[23]]
##         min      max
## x -17.83332 14.51888
## y  38.79292 65.35979
##
## [[24]]
##             min          max
## x  0.0003220005 0.0006506793
## y -0.0001562665 0.0001543668
##
## [[25]]
##         min      max
## x -17.83332 14.51888
## y  38.79292 65.35979
##
## [[26]]
##             min          max
## x  3.234128e-04 0.0006328213
## y -9.500185e-05 0.0001869170
##
## [[27]]
##         min      max
## x -17.83332 14.51888
## y  38.79292 65.35979
##
## [[28]]
##             min          max
## x  0.0002489462 0.0006332609
## y -0.0001554837 0.0003063432
```

```
##
## [[29]]
##          min      max
## x -16.84621 31.51602
## y  36.21518 71.55264
##
## [[30]]
##          min      max
## x -18.81906 13.03938
## y  36.39022 65.54959
##
## [[31]]
##          min     max
## x -17.44740 12.2748
## y  44.47954 62.0423
##
## [[32]]
##          min      max
## x -26.15375 20.72628
## y  39.97409 70.23447
##
## [[33]]
##   min  max
## x -18 23.0
## y  36 72.5
##
## [[34]]
##          min      max
## x -14.11575 12.25857
## y  47.30682 60.87880
##
## [[35]]
##    min max
## x -180 180
## y  -90  90
```

Or, extracting the abstracts for each layer so that you can read the contents of the layer.

```
emodnet_client$getCapabilities()$getFeatureTypes() %>%
    map_chr(function(x) {
        x$getAbstract()
    })
##  [1] "Gridded distribution map for Annex I Coastal lagoons as reported by
EU member states for 2013 Habitats Directive Article 17 reporting. Available
from the European Environment Agency website at: https://www.eea.europa.eu/da
ta-and-maps/data/article-17-database-habitats-directive-92-43-eec-1#tab-metad
ata
"
##  [2] "Gridded distribution map for Annex I Estuaries as reported by EU mem
ber states for 2013 Habitats Directive Article 17 reporting. Available from t
```

```
he European Environment Agency website at: https://www.eea.europa.eu/data-and
-maps/data/article-17-database-habitats-directive-92-43-eec-1#tab-metadata
"
##  [3] "Gridded distribution map for Annex I Large shallow inlets and bays a
s reported by EU member states for 2013 Habitats Directive Article 17 reporti
ng. Available from the European Environment Agency website at: https://www.ee
a.europa.eu/data-and-maps/data/article-17-database-habitats-directive-92-43-e
ec-1#tab-metadata
"
##  [4] "Gridded distribution map for Annex I Mudflats and sandflats not cove
red by seawater at low tide as reported by EU member states for 2013 Habitats
Directive Article 17 reporting. Available from the European Environment Agenc
y website at: https://www.eea.europa.eu/data-and-maps/data/article-17-databas
e-habitats-directive-92-43-eec-1#tab-metadata
"
##  [5] "Gridded distribution map for Annex I Posidonia beds as reported by E
U member states for 2013 Habitats Directive Article 17 reporting. Available f
rom the European Environment Agency website at: https://www.eea.europa.eu/dat
a-and-maps/data/article-17-database-habitats-directive-92-43-eec-1#tab-metada
ta
"
##  [6] "Gridded distribution map for Annex I Reefs as reported by EU member
states for 2013 Habitats Directive Article 17 reporting. Available from the E
uropean Environment Agency website at: https://www.eea.europa.eu/data-and-map
s/data/article-17-database-habitats-directive-92-43-eec-1#tab-metadata
"
##  [7] "Gridded distribution map for Annex I sandbanks as reported by EU mem
ber states for 2013 Habitats Directive Article 17 reporting. \r\n\r\nAvailabl
e from the European Environment Agency website at: https://www.eea.europa.eu/
data-and-maps/data/article-17-database-habitats-directive-92-43-eec-1#tab-met
adata
"
##  [8] "Gridded distribution map for Annex I Submarine structures made by le
aking gases as reported by EU member states for 2013 Habitats Directive Artic
le 17 reporting. Available from the European Environment Agency website at: h
ttps://www.eea.europa.eu/data-and-maps/data/article-17-database-habitats-dire
ctive-92-43-eec-1#tab-metadata
"
##  [9] "Gridded distribution map for Annex I marine features as reported by
EU member states for 2018 Habitats Directive Article 17 reporting. \r\n\r\nAv
ailable from the European Environment Agency website at: https://sdi.eea.euro
pa.eu/catalogue/srv/eng/catalog.search#/metadata/9f71b3e3-f8ec-442b-a2d5-c3c1
90605ac4
"
## [10] "This layer shows the current known extent and distribution of Corall
igenous and other calcareous bioconcretions in the Mediterranean, collated by
EMODnet Seabed Habitats. The purpose was to produce a data product that would
provide the best compilation of evidence for this habitat, as described in th
e \"Action Plan for the protection of the coralligenous and other calcareous
bio-concretions in the Mediterranean\". This data product contains large data
```

```
gaps and should be viewed as incomplete.
"
## [11] "Fully open-access habitat point data from EMODnet Seabed Habitats co
llated point data where presented within the EUNIS habitat classification sys
tem.\r\n\r\nData can be filtered by EUNIS habitat in the \"seabedtype\" field
, or using the web service vendor parameter \"hab_type\".
"
## [12] "All downloadable habitat point data collated by EMODnet Seabed Habit
ats, presented per measurement with relevant sample method joined where possi
ble.
"
## [13] "This dataset gives the extent of all AnnexI habitats within Scottish
waters, extracted from the Nature Scot \"Geodatabase of Marine features adjac
ent to Scotland\" GeMS database.\r\n\r\nNo simplification applied for close v
iews
"
## [14] "This layer shows the current known extent and distribution of Corall
igenous and other calcareous bioconcretions in the Mediterranean, collated by
EMODnet Seabed Habitats. The purpose was to produce a data product that would
provide the best compilation of evidence for this habitat, as described in th
e \"Action Plan for the protection of the coralligenous and other calcareous
bio-concretions in the Mediterranean\". This data product contains large data
gaps and should be viewed as incomplete.
"
## [15] "This layer shows the current known extent and distribution of Corall
igenous and other calcareous bioconcretions in the Mediterranean, collated by
EMODnet Seabed Habitats. The purpose was to produce a data product that would
provide the best compilation of evidence for this habitat, as described in th
e \"Action Plan for the protection of the coralligenous and other calcareous
bio-concretions in the Mediterranean\". This data product contains large data
gaps and should be viewed as incomplete.
"
## [16] "Biological zone at the seabed for various sea basins in Europe. Used
as a habitat descriptor in the creation of the EMODnet broad-scale seabed hab
itat for Europe (EUSeaMap) 2019, as part of EMODnet Seabed Habitats. \r\n\r\n
The extent of the mapped area includes the Mediterranean Sea, Black Sea, Balt
ic Sea, and areas of the North Eastern Atlantic extending from the Canary Isl
ands in the south to the Barents Sea in the north.\r\n\r\nThe map was produce
d using a \"top-down\" modelling approach using classified habitat descriptor
s to determine a final output habitat.\r\n\r\nHabitat descriptors differ per
region but include:\r\nBiological zone\r\nEnergy class\r\nOxygen regime\r\nSa
linity regime\r\nSeabed substrate\r\nRiverine input\r\n\r\nHabitat descriptor
s (excepting Substrate) are calculated using underlying physical data and thr
esholds derived from statistical analyses or expert judgement on known condit
ions.\r\n\r\nThe model is produced using R and Arc Model Builder (10.1). \r\n
\r\nThe model was created using raster input layers with a cell size of 0.001
04dd (roughly 100 metres). The model includes the sublittoral zone only; due
to the high variability of the littoral zone, a lack of detailed substrate da
ta and the resolution of the model, it is difficult to predict littoral habit
ats at this scale.\r\n\r\nThis map follows the EUNIS 2007-11 classification s
```

```
ystem where it is appropriate. It has also been classified according to MSFD
Benthic Broad Habitat types.\r\n\r\nThis report details the methods used in t
he previous version (v2016) - a new report is in progress:\r\nPopulus J. And
Vasquez M. (Eds), 2017. EUSeaMap, a European broad-scale seabed habitat map.
Ifremer\r\nAvailable from: http://archimer.ifremer.fr/doc/00388/49975/"
## [17] "Energy class at the seabed for various sea basins in Europe. Used as
a habitat descriptor in the creation of the EMODnet broad-scale seabed habita
t for Europe (EUSeaMap) 2019, as part of EMODnet Seabed Habitats. \r\n\r\nThe
extent of the mapped area includes the Mediterranean Sea, Black Sea, Baltic S
ea, and areas of the North Eastern Atlantic extending from the Canary Islands
in the south to the Barents Sea in the north.\r\n\r\nThe map was produced usi
ng a \"top-down\" modelling approach using classified habitat descriptors to
determine a final output habitat.\r\n\r\nHabitat descriptors differ per regio
n but include:\r\nBiological zone\r\nEnergy class\r\nOxygen regime\r\nSalinit
y regime\r\nSeabed substrate\r\nRiverine input\r\n\r\nHabitat descriptors (ex
cepting Substrate) are calculated using underlying physical data and threshol
ds derived from statistical analyses or expert judgement on known conditions.
\r\n\r\nThe model is produced using R and Arc Model Builder (10.1). \r\n\r\nT
he model was created using raster input layers with a cell size of 0.00104dd
(roughly 100 metres). The model includes the sublittoral zone only; due to th
e high variability of the littoral zone, a lack of detailed substrate data an
d the resolution of the model, it is difficult to predict littoral habitats a
t this scale.\r\n\r\nThis map follows the EUNIS 2007-11 classification system
where it is appropriate. It has also been classified according to MSFD Benthi
c Broad Habitat types.\r\n\r\nThis report details the methods used in the pre
vious version (v2016) - a new report is in progress:\r\nPopulus J. And Vasque
z M. (Eds), 2017. EUSeaMap, a European broad-scale seabed habitat map. Ifreme
r\r\nAvailable from: http://archimer.ifremer.fr/doc/00388/49975/"
## [18] "Seabed substrate for various sea basins in Europe. Used as a habitat
descriptor in the creation of the EMODnet broad-scale seabed habitat for Euro
pe (EUSeaMap) 2019, as part of EMODnet Seabed Habitats. \r\n\r\nThe extent of
the mapped area includes the Mediterranean Sea, Black Sea, Baltic Sea, and ar
eas of the North Eastern Atlantic extending from the Canary Islands in the so
uth to the Barents Sea in the north.\r\n\r\nThe map was produced using a \"to
p-down\" modelling approach using classified habitat descriptors to determine
a final output habitat.\r\n\r\nHabitat descriptors differ per region but incl
ude:\r\nBiological zone\r\nEnergy class\r\nOxygen regime\r\nSalinity regime\r
\nSeabed substrate\r\nRiverine input\r\n\r\nHabitat descriptors (excepting Su
bstrate) are calculated using underlying physical data and thresholds derived
from statistical analyses or expert judgement on known conditions.\r\n\r\nThe
model is produced using R and Arc Model Builder (10.1). \r\n\r\nThe model was
created using raster input layers with a cell size of 0.00104dd (roughly 100
metres). The model includes the sublittoral zone only; due to the high variab
ility of the littoral zone, a lack of detailed substrate data and the resolut
ion of the model, it is difficult to predict littoral habitats at this scale.
\r\n\r\nThis map follows the EUNIS 2007-11 classification system where it is
appropriate. It has also been classified according to MSFD Benthic Broad Habi
tat types.\r\n\r\nThis report details the methods used in the previous versio
n (v2016) - a new report is in progress:\r\nPopulus J. And Vasquez M. (Eds),
2017. EUSeaMap, a European broad-scale seabed habitat map. Ifremer\r\nAvailab
```

```
le from: http://archimer.ifremer.fr/doc/00388/49975/"
## [19] "Predictive Biological Zone layer produced by EMODnet Seabed Habitats
as an input layer for the 2021 EUSeaMap broad-scale habitat model. The extent
of the mapped area includes the Mediterranean Sea, Black Sea, Baltic Sea, and
areas of the North Eastern Atlantic extending from the Canary Islands in the
south to the Barents Sea in the North. The map of biological zone was produce
d using underlying physical data and thresholds derived from statistical anal
yses or expert judgement on known conditions.\r\n\r\nThe model is produced in
R and Arc Model Builder (10.1). The model was created using raster input laye
rs with a cell size of 0.00104dd (roughly 100 meters). The model includes the
sublittoral zone only (Infralittoral to Abyssal zone).
"
## [20] "Energy class layer produced by EMODnet Seabed Habitats as an input l
ayer for the 2021 EUSeaMap broad-scale habitat model. The extent of the mappe
d area includes the Baltic Sea, and areas of the North Eastern Atlantic and A
rctic extending from the Canary Islands in the south to Norway in the North.
The map of energy classes was produced using underlying wave and current data
and thresholds derived from statistical analyses or expert judgement on known
conditions. This layer is the same as the input used in EUSeaMap 2019.\r\n\r\
nA report on the methods used in the 2021 version of EUSeaMap (Vasquez et al.
, 2021) and reports on previous versions (v2016 and V2019) are linked in Onli
ne Resources.
"
## [21] "Classified seabed substrate types for European seas. Produced by EMO
Dnet Seabed Habitats as an input layer for the 2021 EUSeaMap broad-scale habi
tat model, based on a combination of EMODnet Geology seabed substrate product
s and biological substrates extracted form individual habitat maps from surve
ys around European seas. The extent of the mapped area includes the Mediterra
nean Sea, Black Sea, Baltic Sea, and areas of the North Eastern Atlantic exte
nding from the Canary Islands in the south to the Barents Sea in the north. T
he layer of seabed substrate was produced using data from EMODnet geology at
the following scales:\r\n- 1:25k, 1:30k, 1:45k, 1:60k, 1:70k (a new fine scal
e layer as of 2021)\r\n- 1:50k 1:100k,1:250k (these were updated for 2021)\r\
n- 1:1M (not updated for 2021)\r\nBiological substrates were included in the
2021 version of EUSeaMap to assist in the classification of biogenic habitats
for the 2019 version of EUNIS. The Folk 5 classification of substrate is adop
ted because it is compatible with both the 2007-11 and 2019 versions of EUNIS
, both of which have been applied in EUSeaMap 2021.\r\n\r\nA report on the me
thods used in the 2021 version of EUSeaMap (Vasquez et al., 2021) and reports
on previous versions (v2016 and V2019) are linked in Online Resources.
"
## [22] "The regions boundary layer shows the basins used to subset the model
ling process used in the creation of the 2019 version of EMODnet broad-scale
seabed habitat map for Europe (EUSeaMap 2019).
"
## [23] "This layer shows the current known extent and distribution of live h
ard coral cover in European waters, collated by EMODnet Seabed Habitats. The
points were added in Sept 2021. Lophelia pertusa and Coral gardens are both o
n the OSPAR List of threatened and/or declining species and habitats. The pur
pose was to produce a data product that would provide the best compilation of
```

```
evidence for the essential ocean variable (EOV) known as Hard coral cover and
composition (sub-variable: Live hard coral cover and extent), as defined by t
he Global Ocean Observing System (GOOS). This data product should be consider
ed a work in progress and is not an official product.
"
## [24] "This layer shows the current known extent of Live Hard Coral in Euro
pean waters, collated by EMODnet Seabed Habitats. Lophelia pertusa and Coral
gardens are both on the OSPAR List of threatened and/or declining species and
habitats. The relevant habitats were extracted from the library of maps on th
e EMODnet Seabed Habitats portal and collated into one standardised shapefile
. This data product should be considered a work in progress and is not an off
icial product.
"
## [25] "This layer shows the current known extent and distribution of macroa
lgal canopy in European waters, collated by EMODnet Seabed Habitats. The poin
ts were added in Sept 2021. The purpose was to produce a data product that wo
uld provide the best compilation of evidence for the essential ocean variable
(EOV) known as Macroalgal canopy cover and composition (sub-variable: Areal e
xtent), as defined by the Global Ocean Observing System (GOOS). Kelp and fuco
id brown algae are the dominant species that comprise macroalgal forests. Thi
s data product should be considered a work in progress and is not an official
product.
"
## [26] "This layer shows the current known extent of macroalgal forests in E
uropean waters, collated by EMODnet Seabed Habitats. Kelp and fucoid brown al
gae are the dominant species that comprise macroalgal forests. The relevant h
abitats were extracted from the library of maps on the EMODnet Seabed Habitat
s portal and collated into one standardised shapefile.  This data product sho
uld be considered a work in progress and is not an official product.
"
## [27] "This layer shows the current known extent and distribution of Seagra
ss meadows in European waters, collated by EMODnet Seabed Habitats. The point
s were added in Sept 2021. The purpose was to produce a data product that wou
ld provide the best compilation of evidence for the essential ocean variable
(EOV) known as Seagrass cover and composition (sub-variable: Areal extent of
seagrass meadows), as defined by the Global Ocean Observing System (GOOS). Se
agrasses provide essential habitat and nursery areas for many marine fauna. T
here are approximately 72 seagrass species that belong to four major groups:
Zosteraceae, Hydrocharitaceae, Posidoniaceae and Cymodoceaceae. Zostera beds
and Cymodecea meadows are named on the OSPAR Threatened or Declining Habitats
list. Posidonia beds are protected under Annex I of the EU Habitats Directive
. This data product should be considered a work in progress and is not an off
icial product.
"
## [28] "This layer shows the current known extent of Seagrass meadows in Eur
opean waters, collated by EMODnet Seabed Habitats. Seagrasses provide essenti
al habitat and nursery areas for many marine fauna. There are approximately 7
2 seagrass species that belong to four major groups: Zosteraceae, Hydrocharit
aceae, Posidoniaceae and Cymodoceaceae. Zostera beds and Cymodecea meadows ar
e named on the OSPAR Threatened or Declining Habitats list. Posidonia beds ar
```

```
## e protected under Annex I of the EU Habitats Directive. This data product sho
uld be considered a work in progress and is not an official product.
"
## [29] "Bounding boxes for datasets within EMODnet Seabed Habitats' point da
ta collation.
"
## [30] "This is a compilation of OSPAR habitat point data for the northeast
Atlantic submitted by OSPAR contracting parties. The compilation is coordinat
ed by the UK's Joint Nature Conservation Committee, working with a representa
tive from each of the OSPAR coastal contracting parties. This public dataset
does not contain records relating to sensitive species (e.g. Ostrea edulis) i
n specific areas, or where data are restricted from public release by the own
er's use limitations. This version (v2020) was published in July 2021.
"
## [31] "This is a compilation of OSPAR habitat polygon data for the northeas
t Atlantic submitted by OSPAR contracting parties. The compilation is coordin
ated by the UK's Joint Nature Conservation Committee, working with a represen
tative from each of the OSPAR coastal contracting parties.This public dataset
does not contain records relating to sensitive species (e.g. Ostrea edulis) i
n specific areas, or where data are restricted from public release by the own
er's use limitations.\r\n\r\nThis version (v2020) was published in June 2020.
"
## [32] "Points data from the 2018 OSPAR database of threatened and/or declin
ing habitats, compiled from OSPAR (Oslo Paris Convention) contracting parties
submission of listed habitats. For more information,, see https://www.ospar.o
rg/work-areas/bdc/species-habitats/mapping-habitats-on-the-ospar-list-of-thre
atened-or-declining-species-and-habitats.
"
## [33] "Overview grid (based on ICES sea squares) for the OSPAR polygon laye
r to show location of potentially small polygons at low zoom levels.
"
## [34] "Polygon data from the 2018 OSPAR database of threatened and/or decli
ning habitats, compiled from OSPAR (Oslo Paris Convention) contracting partie
s submission of listed habitats. For more information,, see https://www.ospar
.org/work-areas/bdc/species-habitats/mapping-habitats-on-the-ospar-list-of-th
reatened-or-declining-species-and-habitats.
"
## [35] "VME data portal is a central portal for data on the distribution and
abundance of Vulnerable Marine Ecosystems (VMEs), (and organisms considered t
o be indicators of VMEs) across the North Atlantic that has been set up by th
e Joint ICES/NAFO Working Group on Deep-water Ecology (WGDEC) and is hosted b
y ICES. The database is comprised of 'VME habitats' and 'VME indicators' reco
rds.\r\nThe VME database serves multiple purposes; ICES uses it for providing
scientifically-robust advice on the distribution of VMEs and recommending pos
sible management solutions such as bottom fishing closures within NEAFC (Nort
h East Atlantic Fisheries Commission) waters to protect VMEs."
```

Read vector data

Now that we've queried the `WFS` metadata and seen what layers are available, we now want to extract a specific layer.

For this section, we'll focus on the OSPAR Threatened and/or Declining Habitats data, specifically the `point` data - `emodnet_open:ospar_points`.

Example 1: read an entire layer

Firstly we'll read in the entire layer and transform the coordinate reference system (`CRS`) to the geographic World Geodetic Sytem (`WGS84`, `EPSG:4326`).

```
url <- parse_url(wfs_main)

url$query <- list(service = "wfs",
                  #version = "2.0.0", # optional
                  request = "GetFeature",
                  typename = "emodnet_open:ospar_points",
                  srsName = "EPSG:3857" # Web mercator system
                  )

request <- build_url(url)

ospar_points <- st_read(request) %>%
  st_transform(4326) # Transform to WGS84 for easier plotting

## Reading layer `ospar_points' from data source
##   `https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_open/wfs?servic
e=wfs&request=GetFeature&typename=emodnet_open%3Aospar_points&srsName=EPSG%3A
3857'
##   using driver `GML'
## Simple feature collection with 44511 features and 21 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -4657062 ymin: 4179747 xmax: 2962121 ymax: 20036570
## Projected CRS: WGS 84 / Pseudo-Mercator
```

We can inspect the first 5 rows of the data using `head()`.

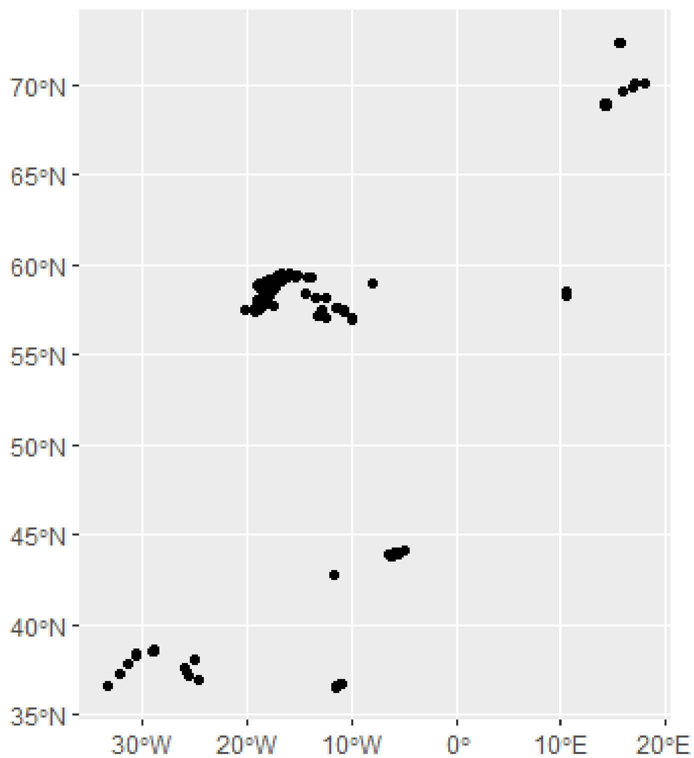
```
head(ospar_points)
```

gml_id	gui	recordkey	habtype	habsubtype	habstatus	certainty	determiner	detdate	survey key	start date
ospar_points.7	OSPARHab2010ES1v0	11000079	Lophelia pertusa reefs	Not applicable	Present	Uncertain	Max Wisshak & Andr, Freiwald / IPAL Erlangen	2006-01-01		
ospar_points.8	OSPARHab2010ES1v0	11000377	Lophelia pertusa reefs	Not applicable	Present	Uncertain	Max Wisshak & Andr, Freiwald / IPAL Erlangen	2006-01-01		1870-01-01
ospar_points.9	OSPARHab2010ES1v0	11000378	Lophelia pertusa reefs	Not applicable	Present	Uncertain	Max Wisshak & Andr, Freiwald / IPAL Erlangen	2006-01-01		1870-01-01
ospar_points.11	OSPARHab2010ES1v0	11000425	Lophelia pertusa reefs	Not applicable	Present	Uncertain	Max Wisshak & Andr, Freiwald / IPAL Erlangen	2006-01-01		1971-01-01
ospar_points.19	OSPARHab2010ES1v0	11001017	Lophelia pertusa reefs	Not applicable	Present	Uncertain	Max Wisshak & Andr, Freiwald / IPAL Erlangen	2006-01-01		1767-01-01
ospar_points.25	OSPARHab2010ES2v0	114	Lophelia pertusa reefs	Not applicable	Present	Uncertain				1967-01-01

end date	datetype	placename	dataowner	accuracy	latitude	longitude	althabtype	althabclass	althabrel	shape
	U				44	-5.57				c(-5.57079999989967, 43.983300000219)
1870-12-31	Y	Iberian Shelf			36.5	-7.27				c(-7.26666699996151, 36.4833330003478)
1870-12-31	Y	Iberian Shelf			36.5	-7.27				c(-7.26666699996151, 36.4833330003478)
1971-12-31	Y	Iberian Shelf			36.3	-7.23				c(-7.23166699976176, 36.2883329998573)
1972-12-31	YY				43.6	-3.6				c(-3.60000000039719, 43.5967000000872)
1967-12-31	Y	T510	IFREMER		44	-6.98				c(-6.97666666965142, 44.0366666700299)

With the points read in, let's filter for any coral garden points and quickly visualise it using `ggplot2`:

```
ospar_points %>%
    filter(habtype %in% c("Coral gardens")) %>%
    ggplot() + geom_sf()
```



Example 2: filter by attributes

Whereas the previous example extracted a full layer from GeoServer, this example will add an attribute filter to return OSPAR habitats, specifically Coral Gardens, Seamounts and *Zostera* beds.

Filtering can be done using either the [standard OGC filter specification] or using a [Common Query Language (CQL) filter]. The OGC filtering approach is very verbose and prone to mistyping the filtering parameters. Hence, since EMODnet Seabed Habitats disseminates their data using [GeoServer] which supports CQL filtering, that's the approach we'll focus on.

In this example we also show how the previously used `R` code can be stitched together in a single command.

First let's remind ourselves what fields which we can filter by.

```
emodnet_client$getCapabilities()$findFeatureTypeByName("emodnet_open:ospar_po
ints")$getDescription() %>%
    map_chr(function(x) {
```

```
        x$getName()
    })
```

```
##  [1] "gui"          "recordkey"    "habtype"      "habsubtype"   "habstatus"
##  [6] "certainty"    "determiner"   "detdate"      "surveykey"    "startdate"
## [11] "enddate"      "datetype"     "placename"    "dataowner"    "accuracy"
## [16] "latitude"     "longitude"    "althabtype"   "althabclass"  "althabrel"
## [21] "shape"
```

We can see that `habtype` field is the one we need to filter for our habitats of interest.

The CQL filter format is much more human readable and easier to code:

```
ospar_points_filtered <- wfs_main %>%
  parse_url() %>%
  list_merge(query = list(service = "wfs",
                          #version = "1.1.0", # optional
                          request = "GetFeature",
                          typeName = "emodnet_open:ospar_points",
                          srsName = "EPSG:3857",
                          cql_filter = glue::glue(
                            "habtype like 'Zos%'", # SQL-like wildcard filter
returning everything beginning with Zos (i.e. Zostera beds)
                          "or", # Add in the SQL `or` statement
                          "habtype like 'Coral%'", # wildcard filter returnin
g everything beginning with Coral
                          "or",
                          "habtype='Seamounts'",
                          .sep = " "
                          ))) %>%
  build_url() %>%
  read_sf() %>%
  st_transform(4326) #  Transform to same CRS

head(ospar_points_filtered)
```

gml_id	gui	recordkey	habtype	habsubtype	habstatus	certainty	determiner	detdate	survey key	start date
ospar_points.1318	OSPARHab2011ES1v1	111	Coral gardens	Not applicable	Present	Certain	Francisco Sanchez	2010-01-01	ES_1	2007-11-01
ospar_points.5008	OSPARHab2014ES1v2	1	Coral gardens	Not applicable	Present	Certain	IEO	2014-12-31	ES_ES1Surv1	2009-06-24
ospar_points.5009	OSPARHab2014ES1v2	10	Coral gardens	Not applicable	Present	Certain	IEO	2014-12-31	ES_ES1Surv1	2009-06-24
ospar_points.5010	OSPARHab2014ES1v2	11	Coral gardens	Not applicable	Present	Certain	IEO	2014-12-31	ES_ES1Surv1	2009-06-24
ospar_points.5011	OSPARHab2014ES1v2	12	Coral gardens	Not applicable	Present	Certain	IEO	2014-12-31	ES_ES1Surv1	2009-06-24
ospar_points.5012	OSPARHab2014ES1v2	13	Coral gardens	Not applicable	Present	Certain	IEO	2014-12-31	ES_ES1Surv1	2009-06-24

enddate	date type	placename	dataowner	accuracy	latitude	longitude	althab type	althab class	althab rel	shape
2009-12-31	Y	Cachucho Bank	Instituto Español de Oceanografía	50	44.1	-4.91				c(-4.9112000001444, 44.0811999998856)
2012-10-19	DD	Cañón de Avilés	IEO/FB	5	43.9	-6.45				c(-6.45019999986729, 43.9370999997054)
2012-10-19	DD	Cañón de Avilés	IEO/FB	5	43.7	-6.1				c(-6.09970000007428, 43.7316000002091)
2012-10-19	DD	Cañón de Avilés	IEO/FB	5	44	-5.91				c(-5.91340000038678, 44.0326000002397)
2012-10-19	DD	Cañón de Avilés	IEO/FB	5	43.9	-5.83				c(-5.82660000039447, 43.9450999998023)
2012-10-19	DD	Cañón de Avilés	IEO/FB	5	44	-5.83				c(-5.83134399981664, 43.9576799998068)

Note, you may find some datasets return the `geometry type` as a `MULTISURFACE` layer. This poses an issue for some `sf` functions, such as `st_buffer`, which do not work with this geometry. It's not necessarily an issue for what we're covering in this tutorial, but it's something to be aware of.

Example 3: bounding box filter

GeoServer is quite flexible in how it can provide data via `GetFeature` requests.

It is possible to query for features based on geometry. While there are limited options available in a GET request for spatial queries, filtering by bounding box (BBOX) is supported.

The BBOX parameter allows you to search for features that are contained (or partially contained) inside a box of user-defined coordinates. The format of the BBOX parameter is `bbox=x1,y1,x2,y2,[crs]` where `x1, y1, x2`, and `y2` represent the coordinate values. The optional `crs` parameter is used to name the CRS for the bbox coordinates (if they are different to the featureTypes native CRS.) The order of coordinates passed to the BBOX parameter depends on the coordinate system used.

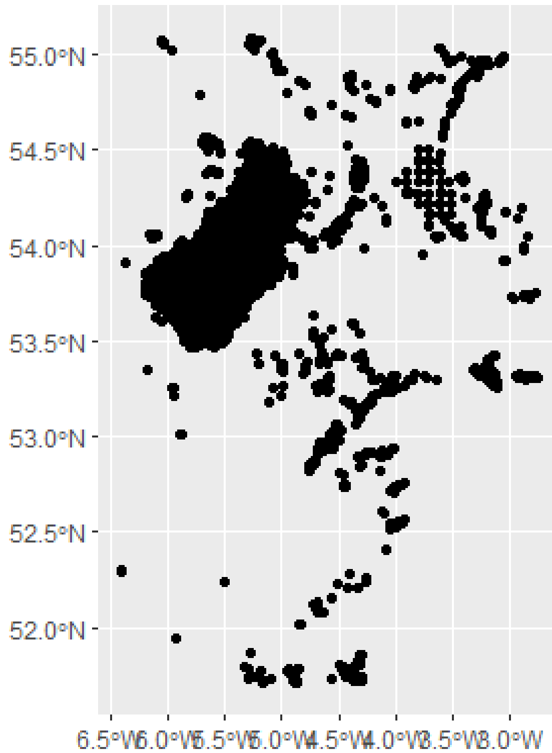
To specify the coordinate system for the returned features, append `srsName=CRS` to the WFS request, where CRS is the Coordinate Reference System you wish to use.

Below is an example request returning any points within the Irish Sea.

```
ospar_bbox_filter <- wfs_main %>%
  parse_url() %>%
  list_merge(query = list(service = "wfs",
                          #version = "1.1.0", # optional
                          request = "GetFeature",
                          typeName = "emodnet_open:ospar_points",
                          srsName = "EPSG:3857", # source CRS is projected, n
ot geographic
                          bbox = "-6.613770,51.713416,-2.746582,55.090944,EPS
G:4326" # bbox box with geographic CRS
                          )) %>%
  build_url() %>%
  read_sf() %>%
  st_transform(4326)
```

Now to quickly plot it.

```
ggplot(ospar_bbox_filter) + geom_sf()
```



This covers the majority of options applicable to EMODnet Seabed Habitats data, but for additional options to read in data from `WFS` services I would strongly recommend reading the previous tutorial listed above (here for reference)

Visualising data

Now that we've had a chance to read in some data let's make some nicer plots. This section will be using the `ospar_points_filtered` object we read in earlier, focusing on Coral Gardens, Seamounts & Zostera beds.

Read in global country data

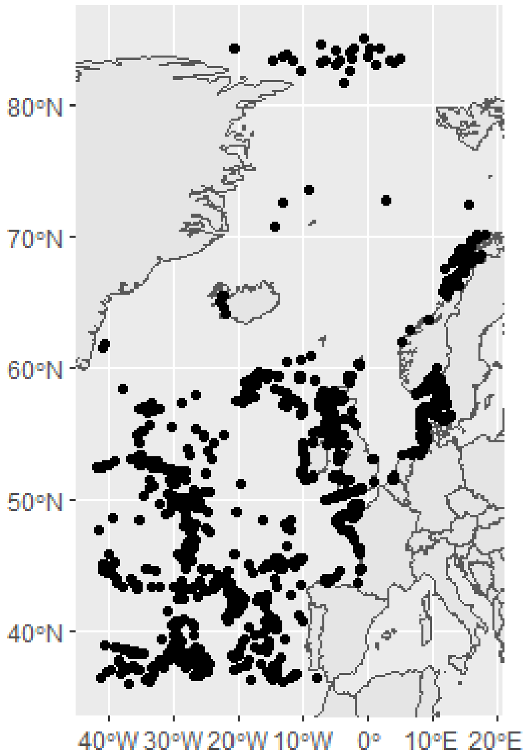
To serve as reference points in our maps we'll need some country boundary data; this dataset is available from the `rnaturalearth` package. We'll extract the data and choose the appropriate scale. By default the function returns data of the `sp` class, but since we're working with `sf` data we'll specify this in one of the arguments.

```
world <- ne_countries(scale = "medium", returnclass = "sf")

class(world)

## [1] "sf"         "data.frame"
```

Basic plot

Now we'll visualise our filtered OSPAR points alongside the world boundaries data. Since the extent of the map will by default show the entire world, we'll need to limit this extent to the range of our OSPAR points. This can be done with the `coord_sf()` function, and we also set `expand = TRUE` to prevent the data and axes from overlapping.

```
ggplot() + geom_sf(data = world) + geom_sf(data = ospar_points_filtered) +
    coord_sf(xlim = st_bbox(ospar_points_filtered)[c(1, 3)],
        ylim = st_bbox(ospar_points_filtered)[c(2, 4)], expand = TRUE)
```



It's good to see that the points are plotting correctly, but since we didn't provided any map symbology (e.g. colours) it's difficult to tell where each habitat is.

Add symbologies

We'll now focus on how we can use the same map layer styles as the interactive mapper, therefore keeping any visualisations consistent with the EMODnet Seabed Habitats project. Here we'll extract the layer styles from GeoServer in an `xml` file format, which will be parsed using the `xmltools` package in a `tidy` format.

First, we'll need to set up a `url` request to the GeoServer `workspace` where all styles & symbologies are stored, which is on the Web Mapping Service `emodnet_view` workspace, or `WMS` for short.

Unfortunately, the `ows4R` package does not currently support `WMS` metadata harvesting, meaning we'll have to explore this manually. However, we can still build the `url` request:

```
wms_main <- "https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_view/wms
"
```

```
url <- parse_url(wms_main)

url$query <- list(service = "wms", version = "1.3.0", request = "GetCapabilit
ies")
request <- build_url(url)

request
```

```
## [1] "https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_view/wms?serv
ice=wms&version=1.3.0&request=GetCapabilities"
```

You can copy and paste the above `url` and enter it into your browser, or go directly to the `GetCapabilities` using this link. This will return a large `xml` data of all layers available on the `emodnet_view` workspace, their metadata, associated styles and so on.

To quickly find our layer style of interest, using `Ctrl + F` and search for `ospar_points`. We can see this brings up `4` results and is provided below.

```
htmltools::img(src = knitr::image_uri(here::here("figs", "ospar_getCapbilitie
s.PNG")))
```

Highlighted is information on the layer style, essentially the information we're need to capture. You can preview this style pasting the `xlink:href` attribute into your browser or by using this link.

So now we know the name of the style is `emodnet_view:ospar_points` (conveniently similar to our layer name!). We can download the style directly from the GeoServer instance using a `GetStyle` request. We'll store it as a temporary file.

```
tf <- tempfile(tmpdir = tdir <- tempdir(), fileext = ".xml")

download.file("https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_view/w
ms?service=wms&version=1.1.1&request=GetStyles&layers=emodnet_view:ospar_poin
ts",
    tf)
```

Let's read the `xml` file and view the `tree` structure.

```
doc <- xml2::read_xml(tf)

doc %>%
    xmltools::xml_view_tree()
```

```
## └── NamedLayer
##     ├── Name
##     └── UserStyle
##         ├── Name
##         ├── IsDefault
##         └── FeatureTypeStyle
##             ├── Name
##             ├── Rule
##                 ├── Name
```

```
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
```

```
##                 └── Mark
##                     ├── WellKnownName
##                     └── Fill
##                         └── CssParameter
##         ├── Rule
##             ├── Name
##             ├── Title
##             ├── Filter
##                 └── PropertyIsEqualTo
##                     ├── PropertyName
##                     └── Literal
##             └── PointSymbolizer
##                 └── Graphic
##                     ├── Size
##                     └── Mark
##                         ├── WellKnownName
##                         └── Fill
##                             └── CssParameter
##         ├── Rule
##             ├── Name
##             ├── Title
##             ├── Filter
##                 └── PropertyIsEqualTo
##                     ├── PropertyName
##                     └── Literal
##             └── PointSymbolizer
##                 └── Graphic
##                     ├── Size
##                     └── Mark
##                         ├── WellKnownName
##                         └── Fill
##                             └── CssParameter
##         ├── Rule
##             ├── Name
##             ├── Title
##             ├── Filter
##                 └── PropertyIsEqualTo
##                     ├── PropertyName
##                     └── Literal
##             └── PointSymbolizer
##                 └── Graphic
##                     ├── Size
##                     └── Mark
##                         ├── WellKnownName
##                         └── Fill
##                             └── CssParameter
##         ├── Rule
##             ├── Name
##             ├── Title
##             ├── Filter
```

```
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
```

```
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PointSymbolizer
##           └── Graphic
##             ├── Size
##             └── Mark
##               ├── WellKnownName
##               └── Fill
##                 └── CssParameter
##       └── Rule
##         ├── Name
##         ├── Title
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
```

```
##              └── Literal
##          └── PointSymbolizer
##            └── Graphic
##              ├── Size
##              └── Mark
##                ├── WellKnownName
##                └── Fill
##                  └── CssParameter
```

This gives a breakdown of the full tree structure, showing the attributes and elements used for styling an OSPAR record. We can see that the styling elements really begin at the `++-- FeatureTypeStyle` section of the tree, which is the 4th level of the tree.

What we need to do is extract the the `terminal nodes`, which are `elements` which do not have any `children`. These nodes contain the information we generally want to extract into a tidy dataframe.

To do this, we'll use the `xml_get_paths` function to find all feasible `xpaths` within the `xml` structure.

```
## we can find all feasible paths then collapse
## what we really want is the parent node of terminal nodes.
## use the `only_terminal_parent = TRUE` to do this

terminal_parent <- doc %>% ## get all xpaths to parents of parent node
  xml_get_paths(only_terminal_parent = TRUE)

terminal_xpaths <- terminal_parent %>% ## collapse xpaths to unique only
  unlist() %>%
  unique()

terminal_nodesets <- lapply(terminal_xpaths, xml2::xml_find_all, x = doc)
```

Now we have the nodesets, we can now extract the `xml` file to a tidy dataframe. As mentioned when we viewed the tree structure, we know that the nodes we are interested begin at the 4th level, so we'll extract from there until the last (8th) level..

```
ospar_styling <- terminal_nodesets[c(4:8)] %>%
    purrr::map(xml_dig_df) %>%
    purrr::map(dplyr::bind_rows) %>%
    dplyr::bind_cols() %>%
    janitor::clean_names() %>%
    dplyr::select(title:css_parameter)

head(as.data.frame(ospar_styling))
```

title	property_name	literal	size	well_known_name	css_parameter
Carbonate mounds	habtype	Carbonate mounds	4	circle	#d3d3d3
Coral gardens	habtype	Coral gardens	7	star	#2ba67d
Deep-sea sponge aggregations	habtype	Deep-sea sponge aggregations	4	circle	#00ff00
Intertidal Mytilus edulis beds on mixed and sandy sediments	habtype	Intertidal Mytilus edulis beds on mixed and sandy sediments	4	circle	#49c2e2
Intertidal mudflats	habtype	Intertidal mudflats	4	circle	#b2393b
Littoral chalk communities	habtype	Littoral chalk communities	4	circle	#f3c4eb

Voila! We now have all the styling information we need for the `ospar_points` layer. A breakdown of the table attributes is below:

- `title` - the value which is displayed in the legend;
- `property_name` - the field name of the GeoSever layer relevant for setting the style (*Note*, we used this field earlier when we extracted the OSPAR data);
- `literal` - the literal `habtype` value that each style will be applied to;
- `size` - denotes the symbol size to be used when plotting;
- `well_known_name` - the specific symbol to be used when plotting;
- `css_parameter` - the hex value specifying the colour to use.

We'll now join this style to our filtered OSPAR points dataset that we read in earlier, giving us a single tidy `sf` object. We'll join the two objects by the `habtype` and `literal` column, which contain the exact free-text match of the OSPAR habitat.

```
ospar_points_complete <- ospar_points_filtered %>%
    left_join(ospar_styling, by = c(habtype = "literal"))
```

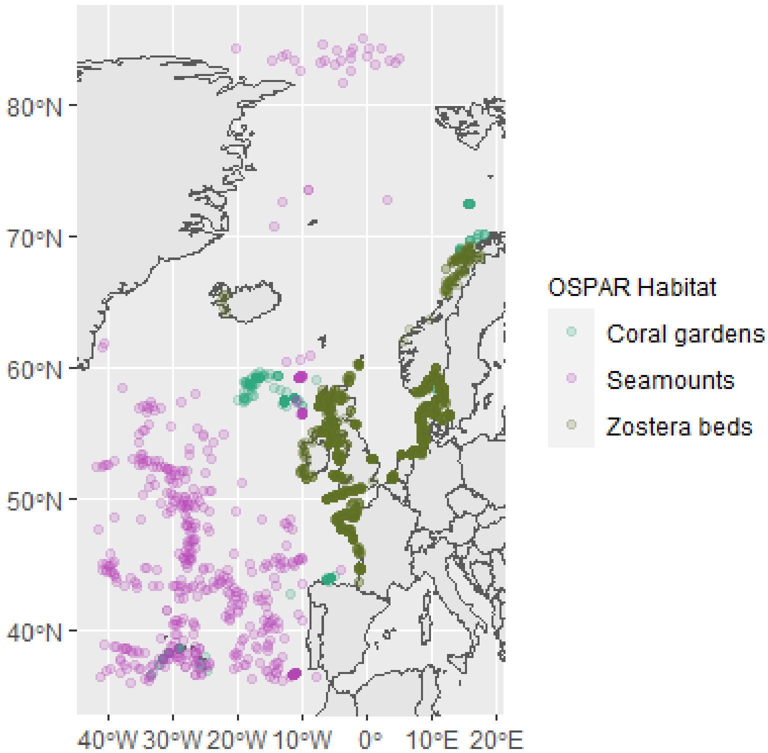
```
names(ospar_points_complete)
```

```
##  [1] "gml_id"          "gui"             "recordkey"       "habtype"
##  [5] "habsubtype"      "habstatus"       "certainty"       "determiner"
##  [9] "detdate"         "surveykey"       "startdate"       "enddate"
## [13] "datetype"        "placename"       "dataowner"       "accuracy"
## [17] "latitude"        "longitude"       "althabtype"      "althabclass"
## [21] "althabrel"       "shape"           "title"           "property_name"
## [25] "size"            "well_known_name" "css_parameter"
```

We can see that the styling columns have been appended to our `sf` object.

Now, let's visualise the points again with the correct stlying applied.

```
col_group <- as.character(ospar_points_complete$css_parameter) # Assign colou
r values to vector
names(col_group) <- as.character(ospar_points_complete$habtype) # Assign grou
ping names to vector

ggplot(data = world) +
  geom_sf() +
  geom_sf(data = ospar_points_complete,             # OSPAR points
          aes(color = habtype),
          alpha = 0.2) +                           # Legend item, point trans
parency
  scale_colour_manual(name = "OSPAR Habitat",       # Legend name
    values = col_group) +
  coord_sf(xlim = st_bbox(ospar_points_complete)[c(1, 3)],
           ylim = st_bbox(ospar_points_complete)[c(2, 4)],
           expand = TRUE) +
  theme(legend.title = element_text(size = 9))
```

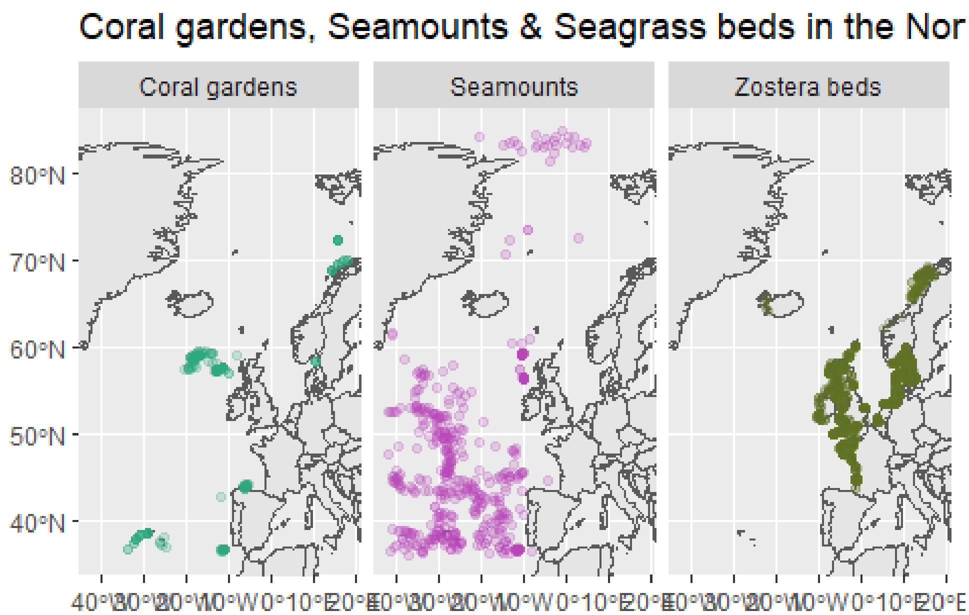


That’s the colour sorted, and by adding a `alpha` argument to change the transparency this helps us to visualise clusters of points. However the map is still pretty busy and doesn’t tell us much, so how can we improve this?

Facet maps

One way is to simply add in the `facet_wrap` argument, which will create a plot for each habitat using the same extents.

```
colour_scheme <- as.character(ospar_points_complete$css_parameter) # Assign c
olour values to vector
names(colour_scheme) <- as.character(ospar_points_complete$habtype) # Assign
grouping names to vector

ggplot() +
  geom_sf(data = world) +
  geom_sf(data = ospar_points_complete,
          aes(color = habtype),
          alpha = 0.2) +
  scale_colour_manual(values = colour_scheme) + # Specify OSPAR colour scale
  facet_wrap(~habtype) + # Create individual panel by habtype
  coord_sf(xlim = st_bbox(ospar_points_complete)[c(1, 3)],
           ylim = st_bbox(ospar_points_complete)[c(2, 4)],
           expand = TRUE) +
  ggtitle("Coral gardens, Seamounts & Seagrass beds in the Northeast Atlantic
") + # Add title
  guides(colour=FALSE) # Remove legend as not needed for this plot
```



We can see where there are clusters of each habitat. To provide more context we will create a series of inset maps for each habitat.

Inset maps

An inset map is a smaller map featured on the same page as the main map. Traditionally, inset maps are shown at a larger scale (smaller area) than the main map. Often, an inset map is used as a locator map that shows the area of the main map in a broader, more familiar geographical frame of reference.

Coastline data

The world coastlines from the `rnaturalearth::ne_countries()` dataset are very coarse and are only suitable for viewing at lower geographic scales. For the inset plots we will use a higher resolution coastline provided by the European Environment Agency.

```
temp_download <- tempfile()
temp_files <- tempfile()

# URL for coastline download saved into 'temp_download'
download.file("https://www.eea.europa.eu/data-and-maps/data/eea-coastline-for
-analysis-1/gis-data/europe-coastline-shapefile/at_download/file",
    temp_download, mode = "wb")

# unzip the contents in 'temp' and save unzipped content in
# 'temp_files'
unzip(zipfile = temp_download, exdir = temp_files)
```

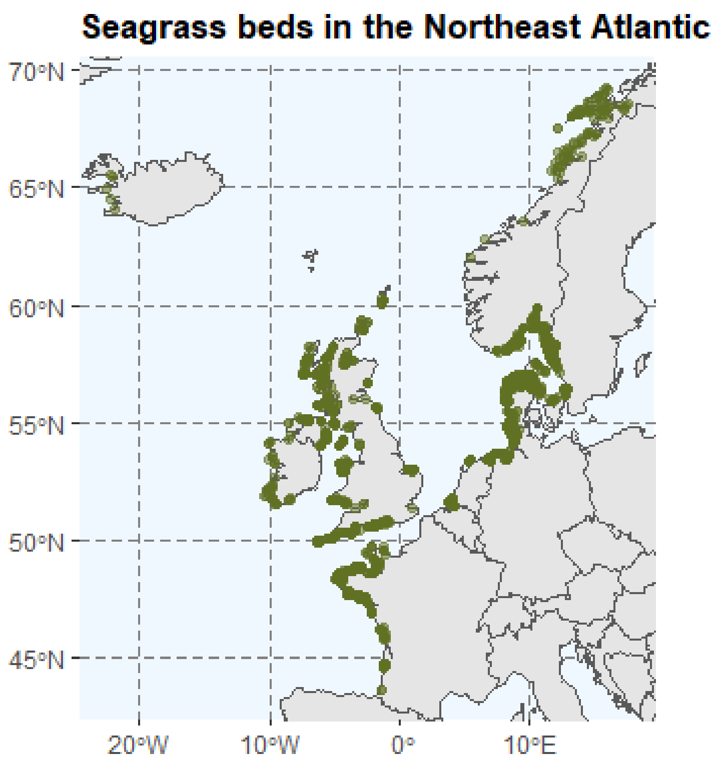
```
# Find the filepath for the polygon dataset (i.e.
# 'poly.shp$' will string match that file)
coastline_file <- list.files(temp_files, pattern = "poly.shp$",
    full.names = TRUE)

# read to sf object and transform to WGS84
eu_coast_detailed <- sf::st_read(coastline_file) %>%
    st_transform(4326)

## Reading layer `Europe_coastline_poly' from data source
##   `C:\Users\Ashley.Elliott\AppData\Local\Temp\RtmpuUDyXV\file2e9c55963b01\
Europe_coastline_poly.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 71520 features and 1 field
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 943609.8 ymin: -375446 xmax: 7601958 ymax: 6825119
## Projected CRS: ETRS89-extended / LAEA Europe
```

Zostera basemap

Now with the extra data gathered, we'll now create the basemap map of *Zostera* beds observations, this is essentially the previous plot filtered for *Zostera* beds. In this example, we'll also make some changes the map background in the `theme()` function; we'll change graticule lines to `dashed` and the background to `aliceblue` to show the marine areas.

```
# Create a zostera-only variable to use in mapping to the correct extent
ospar_zostera <- ospar_points_complete %>%
      filter(str_detect(habtype, "Zostera")) # Wildcard search for Zostera

zostera_colour <- as.character(ospar_zostera$css_parameter) # Assign colour v
alues to vector
names(zostera_colour) <- as.character(ospar_zostera$habtype) # Assign groupin
g names to vector

(gg_zostera <- ggplot() +
  geom_sf(data = world) + # Note: we use the world coastline data for broad v
isualisation
  geom_sf(data = ospar_zostera,
          aes(color = habtype),
          alpha = 0.5) +
  scale_colour_manual(values = zostera_colour) +
  coord_sf(xlim = st_bbox(ospar_zostera)[c(1, 3)],
          ylim = st_bbox(ospar_zostera)[c(2, 4)],
          expand = TRUE) +
    ggtitle("Seagrass beds in the Northeast Atlantic") + # Add title
  theme(plot.title = element_text(size = 13, face = "bold"),
        legend.position = "none", # Remove legend as it's not needed for thes
```

```
e plots
        panel.grid.major = element_line(color = gray(0.5), # Setting graticul
e lines to dashed
                                        linetype = "dashed",
                                        size = 0.5),
        panel.background = element_rect(fill = "aliceblue"))) # Background fi
ll is blue
```



From the above map, we can see that some interesting study areas could be:

- Isle of Arran, UK
- The Gulf of Morbihan, France
- Kattegat Straits

Location data

For study areas, we'll collate the coordinates of some nearby towns/cities to provide additional reference in the plots. This is done using the `geocode()` function from the `tidygeocoder` package. In this example we already know the locations so we can pass these directly to the function.

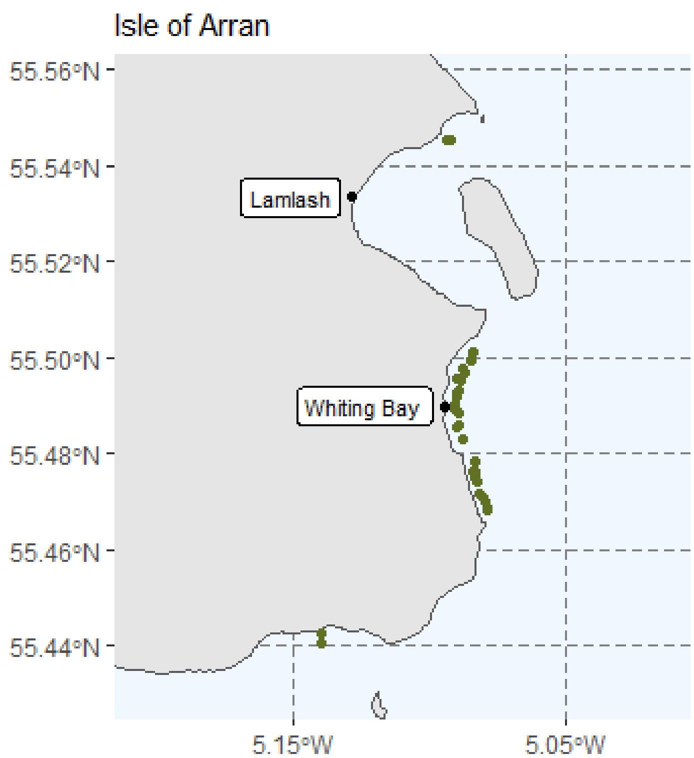
```
location_data <- tibble(location = c("Lamlash", "Whiting Bay", # Isle of Arra
n
                                     "Vannes", # Gulf of Morbihan locations
                                     "Gothenburg")) # Skagerrak locations


locations_sf <- location_data %>%
```

```
  tidygeocoder::geocode(city = location,
                        method = 'osm') %>%
  st_as_sf(coords = c("long", "lat"), crs = 4326) %>%
  mutate(lon = unlist(purrr::map(.$geometry, 1)), # Keep coords as explicit c
olumns
         lat = unlist(purrr::map(.$geometry, 2)))
```

Zostera inset maps

We'll focus on the Isle of Arran inset map first. To do this, we'll need to manually pass the lat/lon extents to the `coord_sf` function. We'll also add a title the plot denoting the region, provide the town/cities location as reference points via `ggrepel::geom_label_repel()` and use the `st_crop()` function on the EU coastline to speed up plotting.

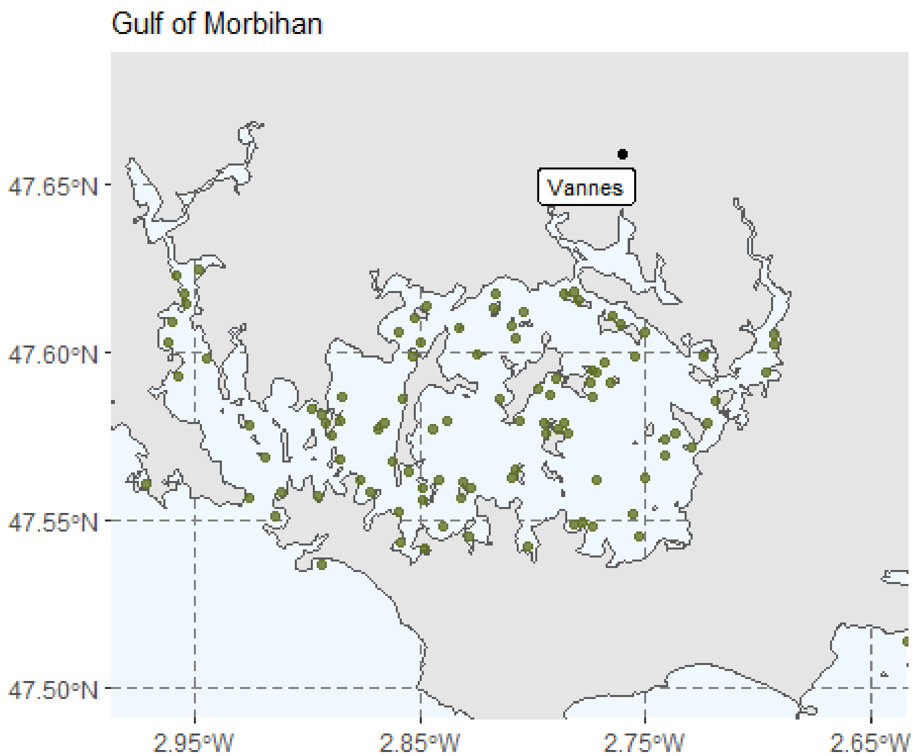
```
(gg_arran <- ggplot() +
    geom_sf(data = st_crop(eu_coast_detailed, # Speed up plotting
                           xmin = -6, ymin = 54,
                           xmax = -4, ymax = 57)) +
    geom_sf(data = ospar_zostera, aes(color = habtype), alpha = 1) + # Add Zo
stera points
    scale_colour_manual(values = zostera_colour) + # Add colour
    geom_sf(data = locations_sf %>% # Town point data
              filter(location %in% c("Lamlash", "Whiting Bay"))) +
    geom_label_repel(data = locations_sf %>% # Town labels
                       filter(location %in% c("Lamlash", "Whiting Bay")),
                     aes(x = lon, y = lat,
                         label = location),
                     size = 3,
                     fontface = "plain",
                     nudge_x = c(-0.02, -0.02), # Move along the x axis
                     nudge_y = c(0, 0)) + # Move along the y axis
    coord_sf(xlim = c(-5.206146, -5.013885),
             ylim = c(55.430962, 55.556602), expand = TRUE) +
    scale_x_continuous(breaks = c(-5.15, -5.05)) + # Set longitude breaks
    scale_y_continuous(breaks = c(55.44, 55.46, 55.48,
                                  55.50, 55.52, 55.54, 55.56)) + # Set latitu
de breaks
    ggtitle("Isle of Arran") +
    theme(legend.position = "none",
          plot.title = element_text(size = 11),
          axis.title.x  = element_blank(),
          axis.title.y = element_blank(),
          panel.grid.major = element_line(color = gray(0.5),
                                          linetype = "dashed",
                                          size = 0.5),
          panel.background = element_rect(fill = "aliceblue")))
```



Now repeat for the Gulf of Morbihan plot.

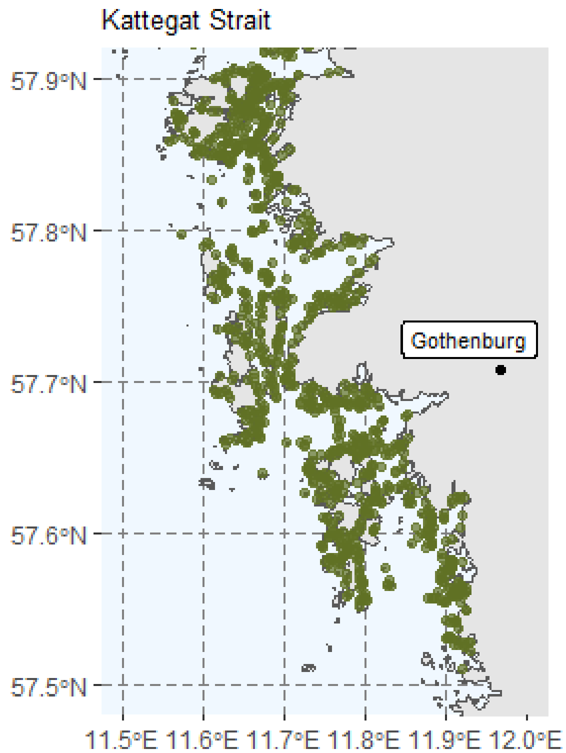
```
(gg_morbihan <- ggplot() +
    geom_sf(data = st_crop(eu_coast_detailed,
                           xmin = -5, ymin = 45,
                           xmax = -0, ymax = 49)) +
    geom_sf(data = ospar_zostera, aes(color = habtype), alpha = 0.8) +
    scale_colour_manual(values = zostera_colour) +
    geom_sf(data = locations_sf %>%
              filter(location %in% c("Vannes"))) +
    geom_label_repel(data = locations_sf %>%
                       filter(location %in% c("Vannes")),
                     aes(x = lon, y = lat,
                         label = location),
                     size = 3,
                     fontface = "plain",
                     nudge_x = c(0),
                     nudge_y = c(0)) +
    coord_sf(xlim = c(-2.97, -2.65), ylim = c(47.5, 47.68), expand = TRUE) +
    scale_x_continuous(breaks = c(-2.95, -2.85, -2.75, -2.65)) + # Set longit
ude breaks
    ggtitle("Gulf of Morbihan") +
    theme(legend.position = "none",
          plot.title = element_text(size = 11),
          axis.title.x  = element_blank(),
          axis.title.y = element_blank(),
          panel.grid.major = element_line(color = gray(0.5),
```

```
                                                linetype = "dashed",
                                                size = 0.5),
          panel.background = element_rect(fill = "aliceblue")))
```



And then the Kattegat plot.

```
(gg_kattegat <- ggplot() + geom_sf(data = st_crop(eu_coast_detailed,
    xmin = 9, ymin = 55, xmax = 14, ymax = 60)) + geom_sf(data = ospar_zoster
a,
    aes(color = habtype), alpha = 0.7) + scale_colour_manual(values = zostera
_colour) +
    geom_sf(data = locations_sf %>%
        filter(location %in% c("Gothenburg"))) + geom_label_repel(data = loca
tions_sf %>%
    filter(location %in% c("Gothenburg")), aes(x = lon, y = lat,
    label = location), size = 3, fontface = "plain", nudge_x = c(0),
    nudge_y = c(0.02)) + coord_sf(xlim = c(11.5, 12), ylim = c(57.5,
    57.9), expand = TRUE) + ggtitle("Kattegat Strait") + theme(legend.positio
n = "none",
    plot.title = element_text(size = 10), axis.title.x = element_blank(),
    axis.title.y = element_blank(), panel.grid.major = element_line(color = g
ray(0.5),
        linetype = "dashed", size = 0.5), panel.background = element_rect(fil
l = "aliceblue")))
```



Before creating the final map, we need to create the arrows to show the position of each inset map relative to the position in the main map.

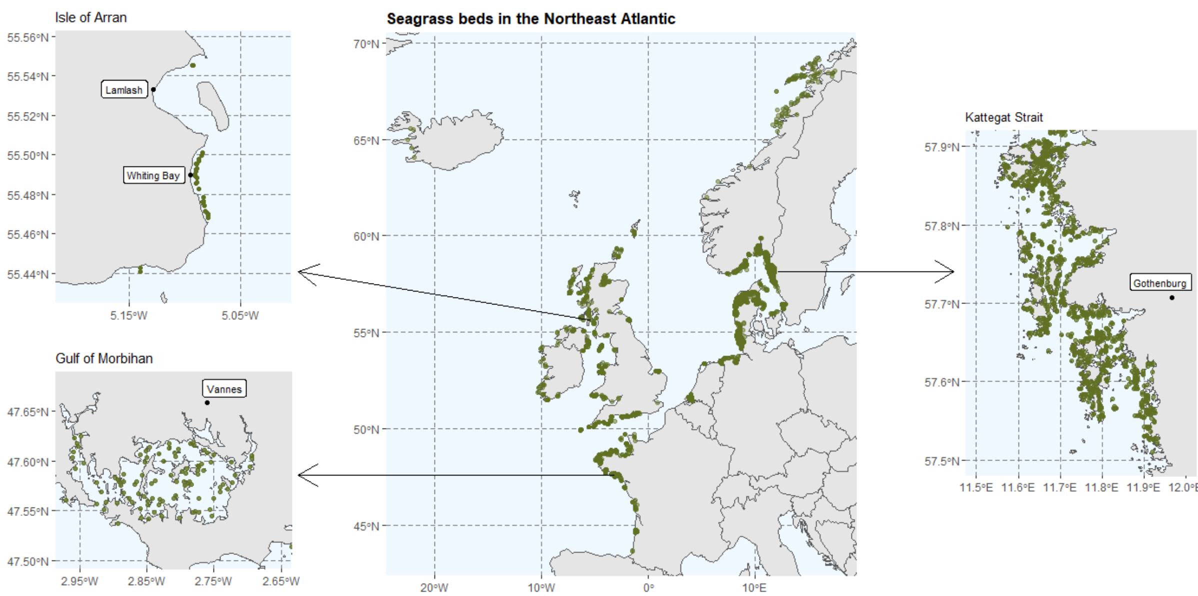
We can connect the three subplots to the main map using arrows which are passed through to the `geom_segment()` function. We'll create a `data.frame` storing the start and end point on the panel (`x1`, `x2` and `x3` on the x-axis, `y1`, `y2` and `y3` on the y-axis).

```
arran_arrow <- data.frame(x1 = 98, x2 = 50, y1 = 47, y2 = 55)

morbihan_arrow <- data.frame(x1 = 102, x2 = 50, y1 = 21.5, y2 = 21.5)

kattegat_arrow <- data.frame(x1 = 129, x2 = 158, y1 = 55, y2 = 55)
```

We can now create the final map. **Note** that we have set the background elements in the `theme()` argument to blank to ensure we only plot our `ggplot` objects.

```
ggplot() + coord_equal(xlim = c(0, 200), ylim = c(0, 100), expand = FALSE) +
    annotation_custom(ggplotGrob(gg_zostera), xmin = 31, xmax = 170,
        ymin = 1, ymax = 99) + annotation_custom(ggplotGrob(gg_arran),
    xmin = 1, xmax = 50, ymin = 45, ymax = 100) + annotation_custom(ggplotGro
b(gg_morbihan),
    xmin = 1, xmax = 50, ymin = 0, ymax = 45) + annotation_custom(ggplotGrob(
gg_kattegat),
    xmin = 152, xmax = 199, ymin = 0, ymax = 100) + geom_segment(aes(x = x1,
    y = y1, xend = x2, yend = y2), data = arran_arrow, arrow = arrow(),
    lineend = "round") + geom_segment(aes(x = x1, y = y1, xend = x2,
    yend = y2), data = morbihan_arrow, arrow = arrow(), lineend = "round") +
```

```
    geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2), data = kattegat_a
rrow,
        arrow = arrow(), lineend = "round") + theme(axis.line = element_blank
(),
    axis.text.x = element_blank(), axis.text.y = element_blank(),
    axis.ticks = element_blank(), axis.title.x = element_blank(),
    axis.title.y = element_blank(), legend.position = "none",
    panel.background = element_blank(), panel.border = element_blank(),
    panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
    plot.background = element_blank())
```



Further development

This was a first attempt at extracting EMODnet Seabed Habitats data. During the testing, some minor bugs were found.

Webscraping multiple city locations

The example provided in this tutorial focused on manually passing locations to the `tidygeocoder::geocode()` function. However, if you are wanting to gather coordinates for multiple locations, you can do so by scraping webpages and passing these to the `geocode()` function. Below is an example using the `rvest` and `purrr` packages, web scraping towns & cities for England, Sweden & Norway.

```
html.english_locations <- xml2::read_html("https://en.wikipedia.org/wiki/List
_of_cities_in_the_United_Kingdom")

df.english_locations <- html.english_locations %>% # Pass html file
  rvest::html_nodes("table") %>% # Extract table nodes
  .[c(1)] %>% # Tables of interest
```

```
  purrr::map_dfr(html_table) %>%
  filter(str_detect(`Nation/region`, "England")) %>% # Filter for cities in E
ngland only
  dplyr::select(1) %>%
  dplyr::mutate(`City[3][1]` =
                stringr::str_replace_all(`City[3][1]`, # Some extra tidying
using regex to clean city name
                                         glue::glue("[:digit:]|", # Remove
all numbers
                                         "\\[|\\]|", # Remove square bracke
ts
                                         "\\s*\\([^\\)]+\\)"), # Remove all
text within parenthesis
                                         "")) %>%
  arrange(1) %>%
  drop_na()



html.swedish_locations <- xml2::read_html("https://en.wikipedia.org/wiki/List
_of_cities_in_Sweden")
df.swedish_locations <- html.swedish_locations %>%
  rvest::html_nodes("table") %>%
  .[c(2)] %>%
  purrr::map_dfr(html_table) %>%
  dplyr::select(1) %>%
    dplyr::mutate(`City` =
                stringr::str_replace_all(`City`, # Some extra tidying using
regex to clean city name
                                         glue::glue("[:digit:]|", # Remove
all numbers
                                                    "\\[|\\]|", # Remove sq
uare brackets
                                                    "\\s*\\([^\\)]+\\)"), #
Remove all text within parenthesis
                                         "")) %>%
  arrange(1) %>%
  drop_na()

html.norwegian_locations <- xml2::read_html("https://en.wikipedia.org/wiki/Li
st_of_towns_and_cities_in_Norway")
df.norwegian_locations <- html.norwegian_locations %>%
  rvest::html_nodes("table") %>%
  .[c(1:3)] %>%
  purrr::map_dfr(html_table) %>%
  dplyr::select(1) %>%
  dplyr::mutate(`City/town` =
                stringr::str_replace_all(`City/town`, # Some extra tidying
using regex to clean city name
                                         glue::glue("[:digit:]|", # Remove
```

```
all numbers
                                                                 "\\[|\\]|", # Remove sq
uare brackets
                                                                 "\\s*\\([^\\)]+\\)"), #
Remove all text within parenthesis
                                                      "")) %>%
  arrange(1) %>%
  drop_na()

# Merge into single dataframe
webscraping_location_data <- list(df.english_locations,
                      df.swedish_locations,
                      df.norwegian_locations) %>%
  purrr::map(., setNames, "location") %>% # using purrr::map to set names of
dataframes to the same
  bind_rows() %>% # merge into a single object
  as_tibble()
```

With the locations in a single object, pass this to the `geocode()` function.

```
webscraping_locations_sf <- webscraping_location_data %>%
  tidygeocoder::geocode(city = location,
                        method = 'osm') %>%
  drop_na() %>%
  st_as_sf(coords = c("long", "lat"), crs = 4326) %>%
  mutate(lon = unlist(purrr::map(.$geometry, 1)), # Keep coords as explicit c
olumns
         lat = unlist(purrr::map(.$geometry, 2)))
```

We can now inspect the first 5 rows using `head()`.

```
head(as.data.frame(webscraping_locations_sf))
```

[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]

And the last 5 rows using `tail()`.

```
tail(webscraping_locations_sf)
```

[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]
[[character]]	[[character]]	[[character]]	[[character]]

You can now pass these coordinates to any plots similar to the above or perform additional analysis.

Extracting complex styles

Some styles, such as the Marine Strategy Framework Directive and Biozone habitat descriptor (This is currently not available and will be updated in the future), layers have specific polygon styles to make the features more distinguishable. These include polygons with strokes, graphic fills and hatching fills.

Complex polygon styling means the GeoServer styles have more nested layers and cannot be parsed easily.

```
# Download style files
## Make a temporary file (tf) and a temporary folder (tdir)
tf <- tempfile(tmpdir = tdir <- tempdir(), fileext = ".xml")

download.file("https://ows.emodnet-seabedhabitats.eu/geoserver/emodnet_view/wms?service=wms&version=1.1.1&request=GetStyles&layers=emodnet_view:eusm2019_bio_full",
              tf)

styles_doc <- xml2::read_xml(tf)

# Let's view the XML tree to understand the structure
styles_doc %>%
  xmltools::xml_view_tree()
```

```
## └── NamedLayer
##     ├── Name
##     ├── UserStyle
##         ├── Name
##         ├── IsDefault
##         └── FeatureTypeStyle
##             ├── Name
##             ├── Rule
```

```
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
```

```
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
```

```
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
```

```
##          └── PolygonSymbolizer
##            └── Fill
##              └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── GraphicFill
##                └── Graphic
##                  ├── Size
##                  └── Mark
```

```
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── GraphicFill
##                └── Graphic
##                  ├── Size
##                  └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── GraphicFill
##                └── Graphic
##                  ├── Size
##                  └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##        ├── Rule
##          ├── Name
##          ├── Title
##          ├── MaxScaleDenominator
##          ├── Filter
##            └── PropertyIsEqualTo
##              ├── PropertyName
##              └── Literal
##          └── PolygonSymbolizer
##            └── Fill
##              └── GraphicFill
##                └── Graphic
##                  ├── Size
```

```
##                   └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── GraphicFill
##               └── Graphic
##                 ├── Size
##                 └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── GraphicFill
##               └── Graphic
##                 ├── Size
##                 └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── GraphicFill
##               └── Graphic
```

```
##                   ├── Size
##                   └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##       ├── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── GraphicFill
##               └── Graphic
##                 ├── Size
##                 └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
## 11 └── CssParameter
##       └── Rule
##         ├── Name
##         ├── Title
##         ├── MaxScaleDenominator
##         ├── Filter
##           └── PropertyIsEqualTo
##             ├── PropertyName
##             └── Literal
##         └── PolygonSymbolizer
##           └── Fill
##             └── GraphicFill
##               └── Graphic
##                 ├── Size
##                 └── Mark
## 10 ├── WellKnownName
## 10 └── Stroke
##    └── UserStyle
##      ├── Name
##      ├── Title
##      ├── Abstract
##      └── FeatureTypeStyle
##        ├── Name
##        └── Rule
##          ├── Name
##          ├── Title
##          ├── Abstract
##          └── PolygonSymbolizer
##            ├── Stroke
```

```
##           └── Fill
##             └── CssParameter

styles_terminal_parent <- styles_doc %>% ## get all xpaths to parents of pare
nt node
  xml_get_paths(only_terminal_parent = TRUE)

styles_terminal_xpaths <- styles_terminal_parent %>% ## collapse xpaths to un
ique only
  unlist() %>%
  unique()
styles_terminal_nodesets <- lapply(styles_terminal_xpaths,
                            xml2::xml_find_all,
                            x = styles_doc)

# This code has been included for reference, but bugs have been found during
testing. Due to the more complex polygon styling of the GeoServer styles, sty
les_terminal_nodesets has nested layers of differing list lengths and cannot
be parsed easily. Some amendments to xml_dig_df and xml_to_df could address t
his issue.
biozone_parse <- styles_terminal_nodesets %>%
  purrr::map(xml_dig_df) %>% ## does not dig by default
  purrr::map(dplyr::bind_rows) %>%
  dplyr::bind_cols()

biozone_parse <- styles_terminal_nodesets[6][[1:17]] %>%   #select specifics
parts of the style
  purrr::map(xml_dig_df) %>% ## does not dig by default
  purrr::map(dplyr::bind_rows) %>%
  dplyr::bind_cols()
```