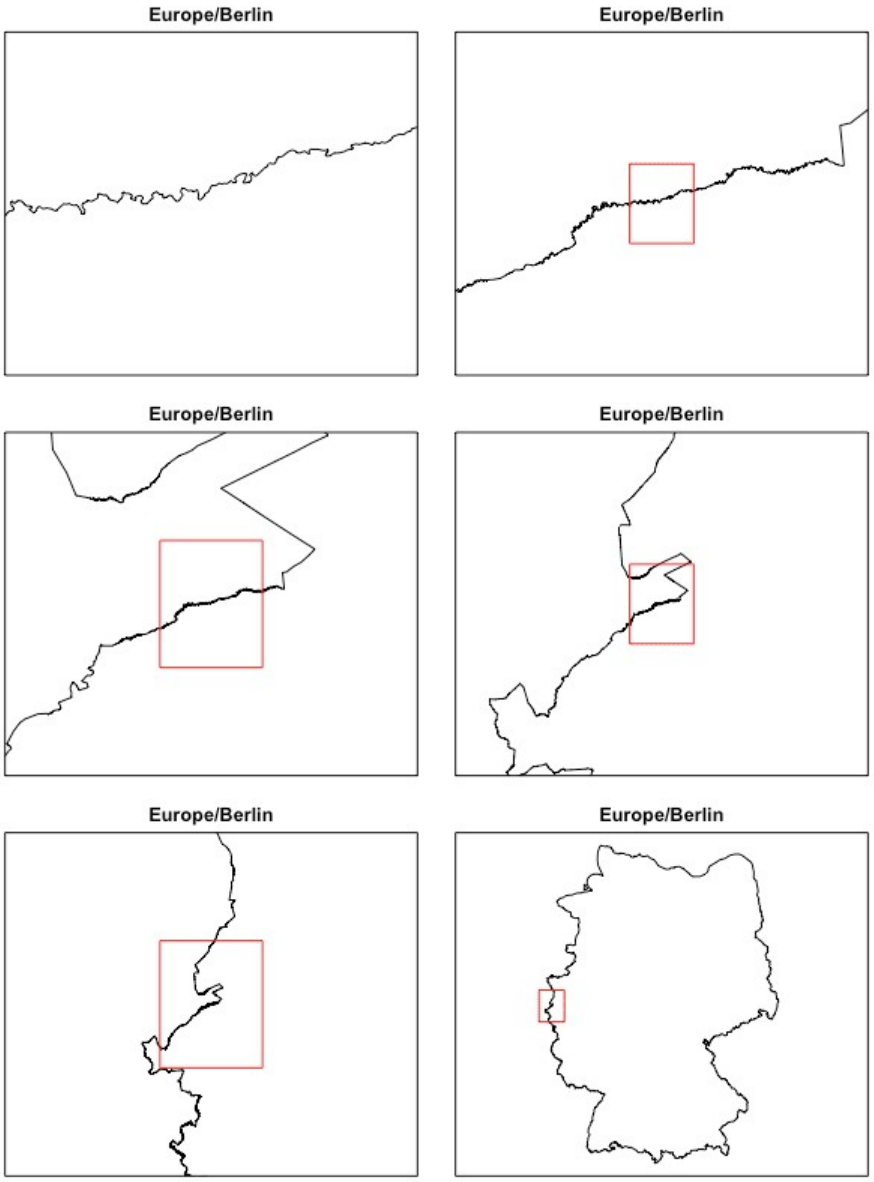
Creating a scalable system

Shapefiles with high resolution features are by nature quite large. Working with the World Timezones dataset we see that the largest single timezone polygon, 'Europe/Berlin', takes up 2.47 Mb of RAM because of the highly detailed, every-curve-in-the-river border delineation between Germany and her neighbors.



Spatial datasets can be large and their conversion from *shapefile* to *SpatialPolygonsDataFrame* can be time consuming. In addition, there is little uniformity to the dataframe data found in these datasets. The **MazamaSpatialUtils** package addresses these issues in multiple ways:

1. It provides a package state variable called `SpatialDataDir` which is used internally as the location for all spatial datasets.
2. It defines a systematic process for converting spatial data into *SpatialPolygonsDataFrames* objects with standardized data columns.
3. A suite of useful spatial data has been pre-processed with topology correction to result in full resolution and multiple, increasingly simplified versions of each dataset.

Spatial data directory

Users will want to maintain a directory where their .rda versions of spatial data reside. The package provides a `setSpatialDataDir()` function which sets a package state variable storing the location. Internally, `getSpatialDataDir()` is used whenever data need to be accessed. *(Hat tip to Hadley Wickham's description of Environments and package state.)*

Standardized data

The package comes with several `convert~()` functions that download, convert, standardize and clean spatial datasets available on the web. Version 0.7 of the package has 21 such scripts that walk through the same basic steps with minor differences depending on the needs of the source data. With these as examples, users should be able to create their own `convert~()` functions to process other spatial data. Once converted and normalized, each dataset will benefit from other package utility functions that depend on the consistent availability and naming of certain columns in the `@data` slot of each *SpatialPolygonsDataFrame*.

Cleaned and Simplified files

Part of the process of data conversion involves using the cleangeo package to fix any topologies that are found to be invalid. In our experience, this is especially important if you end up working with the data in the **sf** or **raster** packages.

The final step in the data processing is the creation of simplified datasets using the rmapshaper package. If you work with GIS data and are unfamiliar with **mapshaper**, you should go to mapshaper.org and try it out. It's astonishing how well this javascript package performs in real-time.

Simplified datasets are important because the dramatically speed up both spatial searches and creation of plots when fast is more important than hyper-accurate. The conversion of the WorldTimezone dataset used above generates a .rda file at full resolution as well as additional versions with 5%, 2% and 1% as many vertices. In the plot above, the 5% version was used to create the the last three plots where high resolution squiggles would never be seen. File sizes for the WorldTimezone .rda files are 67 M, 3.4 M, 1.4 M and 717 K respectively.

Normalizing identifiers

The great thing about working with spatial data stored as a *shapefile* or *geodatabase* is that these are the *defacto* standard formats for spatial data. We LOVE standards! Many shapefiles, but not all, also use the ISO 3166-1 alpha-2 character encoding for identifying countries. However, there seems to be no agreement at all about what to call this encoding. We have seen 'ISO', 'ISO2', 'country', 'CC' and many more. The ISOcodes package calls this column of identifiers 'Alpha_2' which is not particularly descriptive outside the context of ISO codes. From here on out, we will call this column the `countryCode`.

Of course there are many spatial datasets that do not include a column with the countryCode. Sometimes it is because they use FIPS or ISO 3166-1 alpha-3 or some (non-standardized) version of the plain English name. Other times it is because the data are part of a national dataset and the country is assumed.

Wouldn't it be nice if every spatial dataset you worked with was guaranteed to have a column named `countryCode` with the ISO 3166-1 alpha-2 encoding? We certainly think so!

The heart of spatial data standardization in this package is the conversion of various spatial

datasets into *SpatialPolygonsDataFrames* files with guaranteed and uniformly named identifiers. The package internal standards are very simple:

1) Every spatial dataset **must** contain the following data columns:

- `polygonID` – unique identifier for each polygon
- `countryCode` – country at centroid of polygon (ISO 3166-1 alpha-2)

2) Spatial datasets with timezone data **must** contain the following column:

- `timezone` – Olson timezone

3) Spatial datasets at scales smaller than the nation-state **should** contain the following column:

- `stateCode` – 'state' at centroid of polygon (ISO 3166-2 alpha-2)

If other columns contain these data, those columns must be renamed or duplicated with the internally standardized name. This simple level of consistency makes it possible to generate maps for any data that is ISO encoded. It also makes it possible to create functions that return the country, state or timezone associated with a set of locations.

Searching for 'spatial metadata'

The **MazamaSpatialUtils** package began as an attempt to create an off-line answer the following question: "How can we determine the timezones associated with a set of locations?"

We arrived at that question because we often work with pollution monitoring data collected by sensors around the United States. Data are collected hourly and aggregated into a single multi-day dataset with a shared UTC time axis. So far so good. Not surprisingly, pollution levels show a strong diurnal signal so it is useful do identify measurements as being either during the daytime or nighttime. Luckily, the maptools package has a suite of 'sun-methods' for calculating the local sunrise and sunset if you provide a longitude, latitude and POSIXct object **with the proper timezone**.

Determining the timezone associated with a location is an inherently spatial question and can be addressed with a point-in-polygon query as enabled by the **sp** package. Once we enabled this functionality with a timezone dataset we realized that we could extract more spatial metadata for our monitoring stations from other spatial datasets: country, state, watershed, legislative district, *etc. etc.*

get~ functions

The package comes with several 'get' functions that rely on the consistency of datasets to provide a uniform interface. Current functionality includes the following functions that are all called in the same way. Any ~ below means there are two versions of this function, one each to return the Code or Name:

- `getCountry~(longitude, latitude, ...)` – returns names, ISO codes and other country-level data
- `getState~(longitude, latitude, ...)` – returns names, ISO codes and other state-level data
- `getUSCounty(longitude, latitude, ...)` – returns names and other county-level data
- `getTimezone(longitude, latitude, ...)` – returns Olson timezones and other

data
- `getHUC~(longitude, latitude, ...)` – returns USGS Hydrologic Unit Codes and other data
- `getSpatialData(longitude, latitude, ...)` – returns all data
- `getVariable(longitude, latitude, ...)` – returns a single variable

Simple search

Here is an example demonstrating a search for Olson timezone identifiers:

```
library(MazamaSpatialUtils)

# Vector of lons and lats
lons <- seq(-120, -80, 2)
lats <- seq(20, 60, 2)

# Get Olson timezone names
timezones <- getTimezone(lons, lats)
print(timezones)
```

```
  [1] NA                  NA                  NA
  [4] NA                  NA                  "America/Hermosillo"
  [7] "America/Denver"    "America/Denver"    "America/Denver"
 [10] "America/Denver"    "America/Chicago"   "America/Chicago"
 [13] "America/Chicago"   "America/Chicago"   "America/Chicago"
 [16] "America/Nipigon"   "America/Nipigon"   "America/Nipigon"
 [19] "America/Iqaluit"   "America/Iqaluit"   "America/Iqaluit"
```

Additional data

Additional information is available by specifying `allData = TRUE`:

```
# Additional fields
names(SimpleTimezones)
[1] "timezone"       "UTC_offset"     "UTC_DST_offset" "countryCode"
[5] "longitude"      "latitude"       "status"         "notes"
[9] "polygonID"

getTimezone(lons, lats, allData = TRUE) %>%
   dplyr::select(timezone, countryCode, UTC_offset)
         timezone countryCode UTC_offset
1                                    NA
2                                    NA
3                                    NA
4                                    NA
5                                    NA
6  America/Hermosillo          MX         -7
7      America/Denver          US         -7
8      America/Denver          US         -7
9      America/Denver          US         -7
10     America/Denver          US         -7
11    America/Chicago          US         -6
12    America/Chicago          US         -6
```

```
13    America/Chicago          US         -6
14    America/Chicago          US         -6
15    America/Chicago          US         -6
16    America/Nipigon          CA         -5
17    America/Nipigon          CA         -5
18    America/Nipigon          CA         -5
19    America/Iqaluit          CA         -5
20    America/Iqaluit          CA         -5
21    America/Iqaluit          CA         -5
```

Subset by country

Becuase every datasets is guaranteed to have a `countryCode` variable, we can use this for subsetting.

```
# Canada only
subset(SimpleTimezones, countryCode == 'CA') %>%
  dplyr::select(timezone, UTC_offset)
                timezone UTC_offset
71       America/Atikokan       -5.0
77   America/Blanc-Sablon       -4.0
81  America/Cambridge_Bay       -7.0
90        America/Creston       -7.0
94         America/Dawson       -7.0
95   America/Dawson_Creek       -7.0
99       America/Edmonton       -7.0
102   America/Fort_Nelson       -7.0
104     America/Glace_Bay       -4.0
105     America/Goose_Bay       -4.0
112       America/Halifax       -4.0
123        America/Inuvik       -7.0
124       America/Iqaluit       -5.0
146       America/Moncton       -4.0
152       America/Nipigon       -5.0
161   America/Pangnirtung       -5.0
169   America/Rainy_River       -6.0
170  America/Rankin_Inlet       -6.0
172        America/Regina       -6.0
173      America/Resolute       -6.0
182      America/St_Johns       -3.5
187 America/Swift_Current       -6.0
190   America/Thunder_Bay       -5.0
192       America/Toronto       -5.0
194     America/Vancouver       -8.0
195    America/Whitehorse       -7.0
196      America/Winnipeg       -6.0
198   America/Yellowknife       -7.0
```

Optimized searches

One important feature of the package is the ability to optimize spatial searches by balancing speed and accuracy. By default, the `getTimezone()` function uses the `WorldTimezones_02`

dataset to return results quickly. But, if you are very concerned about getting the right timezone on either side of the Roode Beek/Rothenbach border between the The Netherlands and Germany, then you will want to use the full resolution dataset. Luckily, the function signature for `getTimezone()` and the other 'get' functions includes a dataset parameter:

```
getTimezone(
   longitude,
   latitude,
   dataset = "SimpleTimezones",
   countryCodes = NULL,
   allData = FALSE,
   useBuffering = FALSE
)
```

By specifying `datasaset = WorldTimezone` you can perform hyper-accurate (and hyper-slow) searches.

Buffering

For timezone and country searches, we have chosen default datasets that merge detailed borders on land and smoothed, all-encompassing borders off-shore. This avoids issues with peninsulas and islands disappearing with low-resolution datasets. But many datasets attempt to follow coastlines quite closely and lose some of the finer details. When working with these datasets it is useful to specify `useBuffering = TRUE`. This will make an initial pass at finding the polygon underneath each point location. Any locations that remain unassociated after the first pass will be expanded into a small circle and another pass will be performed looking for the overlap of these circles with the spatial polygons. This process is repeated with increasing radii up to 200 km.

Available data

Pre-processed datasets can be viewed and installed locally with the `installSpatialData()` function. Currently available data include:

- `CA_AirBasins` – California regional air basin boundaries
- `EEZCountries` – Country boundaries including Exclusive Economic Zones
- `EPARegions` – US EPA region boundaries
- `GACC` – Geographic Area Coordination Center (GACC) boundaries
- `GADM` – GADM administrative area boundaries
- `HIFLDFederalLands` – US federal lands
- `HMSSmoke` – NOAA Hazard Mapping System Smoke (HMSS) areas
- `HouseLegislativeDistricts` – US state legislative districts, by chamber
- `MTBSBurnAreas` – MTBS burn areas from 1984 – 2017
- `NaturalEarthAdm1` – State/province/oblast level boundaries
- `NWSFireZones` – NWS fire weather forecast zones
- `OSMTimezones` – OpenStreetMap time zones
- `PHDs` – Public Health Districts for Washington, Oregon, Idaho, and California
- `SimpleCountries` – Simplified version of the TMWorldBorders
- `SimpleCountriesEEZ` – Simplified version of EEZCountries
- `SimpleTimezones` – Simplified version of WorldTimezones
- `TerrestrialEcoregions` – Terrestrial eco-regions

- `TMWorldBorders` – Country level boundaries
- `USCensus116thCongress` – US congressional districts
- `USCensusCBSA` – US Core Based Statistical Areas
- `USCensusCounties` – US county level boundaries
- `USCensusStates` – US state level boundaries
- `USCensusUrbanAreas` – US urban areas
- `USFSRangerDistricts` – US Forest Service ranger districts
- `USIndianLands` – US tribal boundaries
- `WBDHU2` – Watershed boundary level-2 hydrologic units
- `WBDHU4` – Watershed boundary level-4 hydrologic units
- `WBDHU6` – Watershed boundary level-6 hydrologic units
- `WBDHU8` – Watershed boundary level-8 hydrologic units
- `weatherZones` – NWS public weather forecast zones
- `WorldEEZ` – Country boundaries including Exclusive Economic Zones over water
- `WorldTimezones` – Timezone

We encourage interested parties to contribute `convert~()` functions for their own favorite spatial datasets. If they produce *SpatialPolygonDataFrames* that adhere to the package standards, we'll include them in the next release.

Happy Mapping!