Technical documentation

TSBB11 - WeatherMagic

Members:
Hans-Filip Elo - hanel742
Maja Ilestrand - majil334
Magnus Ivarsson - magiv438
Christian Luckey - chrlu350
Alexander Poole - alepo020
Magnus Wedberg - magjo722

Supervisor:
Ingemar Ragnemalm

Product owner:
Ola Leifler

Examiner:
Fahad Khan

December 19, 2016

Contents

1 Introduction

WeatherMagic is a web-application that visualizes climate changes due to different emission levels of carbon dioxide. The user will see two spinning globes with two associated time sliders, the time sliders will make it easy for the user to interact with the application. All data for this visualization are retrieved from SMHI, Swedens Meteorological and Hydrological Institute. The data from SMHI predicts the climate until the year 2100 and have stored data back to the year 1950.

The system consists of two separate code bases written by the authors.

The climate data will be viewed through a web client written in ClojureScript with WebGL. The web client queries the web server for specific data which is delivered over HTTP.

The data handed from SMHI are on the format NetCDF, they contain a lot of different types of data which only a few are of interest to us. This data will be processed and yield output files ready for consumption by the web application. All this will be done offline, beforehand. The files can then effortlessly be delivered by a web server to the client software.

2 Weather Front

Weather front is the name of the front end application. In this section it is described in detail how to setup the application as a developer, in order to add new features and understand the code structure in depth.

2.1 Development tools

Weather front is written in Clojurescript with WebGL and the used build system is leiningen [1].

Clojurescript code will be compiled to javascript so the web browser will be able to read it.

2.2 README

2.2.1 Setup

The front end application is written in Clojurescript and OpenGL Shading Language, with leiningen as build system used. To use this together, leiningen needs to be installed. If on Ubuntu/Debian, run:

```
sudo apt-get install leiningen
```

On macOS with homebrew:

```
brew install leiningen
```

On Windows there is an installer that will do the installation.

2.2.2 Our code

With that installed, clone our repository https://github.com/WeatherMagic/weather-front.git and step into the folder git created for you. With that done, in order to get an interactive development environment run:

```
lein figwheel
```

and open your browser at localhost:3449. This will auto compile and send all changes to the browser without the need to reload. After the compilation process is complete, you will get a Browser Connected REPL. An easy way to try it is:

```
(js/alert "Am I connected?")
```

and you should see an alert in the browser window.

To clean all compiled files:

```
lein clean
```

To create a production build run:

```
lein do clean, cljsbuild once min
```

And open your browser in resources/public/index.html. You will not get live reloading, nor a REPL.

2.3 Code structure

The code is split into a number of files, with specific purposes. Figure 1 below shows how the most important files are arranged in a tree.

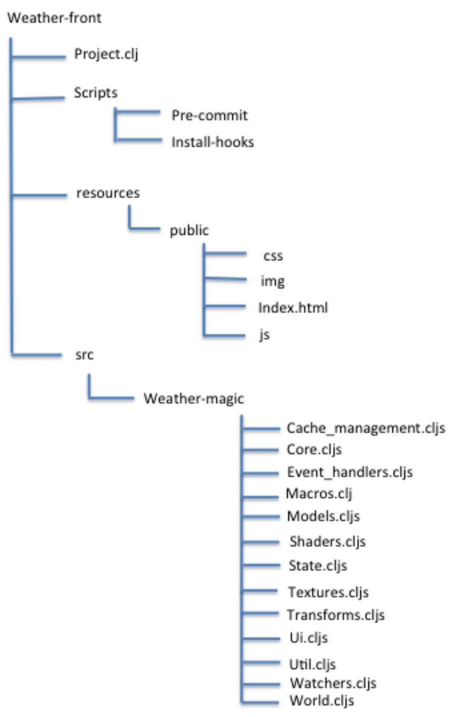

Figure 1: Weather front, code structure

2.3.1 project.clj

This file contains the Leiningen definition of our project, it tells Leiningen where our source code is and how to compile it. Most importantly it contains the list of dependencies we're building on.

2.3.2 Scripts

The scripts folder contains two executables for the pre-commit hook. This checks all our code when we make a commit to git and will make sure that all our code on Github has correct formatting.

2.3.3 css

The css folder only contain one file, style.css. This file hold all the css code to style the website. Mostly, the css code affects the user interface related things.

Most of the buttons and sliders are divided into different classes, e.g. a time slider container that contains all the time slider related objects such as sliders, play/pause buttons and the text displaying the current value. These different objects all have different id's bound to them, making it possible to change different things on different objects.

2.3.4 img

This folder contains all the static images used. For example the texture mapped on the sphere to make it a globe and the space texture mapped onto the background plane.

2.3.5 index.html

This file is kept short and only contains the definitions of the div holding the canvases and the span holding the ui. To visualize change of climate data, two canvases with two different gl-contexes is initialized. These are referred to as $left-canvas$ and $right-canvas$.

2.3.6 js

The js folder contains the compiled javascript code and all the libraries used to create Weather Magic.

2.3.7 cache-management.cljs

Contains functions that handles caching and fetching of textures.

- rotate-in-next! - Keeps track of current and next texture
- trigger-data-load! - Get climate data for the area currently in view on the screen

2.3.8 core.cljs

This file contains as the name suggests the core functionality of the application. The animation loop [2] is to be found here, along with the related function **draw-frame!** which refreshes the buffer and draws the scene with the up-to-date transformations.

The following functions are provided:

- set-model-matrix - this function calls the current $earth-animation-fn$ explained in 2.3.13
- trigger-data-load! - get climate data for the area currently in view on the screen
- update-year-month-info - updates the year and month as the play button on the time sliders is clicked
- draw-in-context - Draws the background plane and the sphere in the given context and uploads relevant uniforms
- draw-frame! - this function is calls a number of functions to update textures and year/month info if necessary and finally calls draw-in-context for the left and right canvas
- running - start the animation
- ui-mounted? - start the function mount-ui that displays the user interface buttons on screen
- hooked-up? - start the event-handler function hook-up-events!

2.3.9 event-handlers.cljs

This file contains all the event handlers [3] added. These event handlers listen to event caused by the user, it can be double click, scroll-event or a resize of the window.

The functionality it provides to Weather Magic are as follows:

Aspect ratio
To make the globe have a spherical shape on all screens, the ratio between the height and width of the screen need to be taken under consideration. This is done after either the web page is loaded or the web page is resized. This is done by comparing the actual size of the canvas and the Web-GL rendered canvas. If they do not match the gl-context of the canvas is set to the actual size of the canvas. After that the cameras view rectangle is set to fit this actual value. Last the viewport is set to the aspect-ratio of the camera.

Panning
The event that triggers the function that handles the panning is a mousedown event on the canvas. When that happens, the x- and y-position of the mouse is read and three more eventlisteners are added - mousemove, mouseup and mouseleave. As long as there is no mouseup event i.e. the mouse is still pressed while moving or the mouse is till on the canvas, the initial x- and y-positions are compared to the current mouse position. This delta movement is used to rotate the globe around an axis orthogonal to the movement in the xy-plane as the mouse moves.

This is done by modifying the matrix that holds the world rotation by a few rotations according to the mouse movements.

Alignment
On the web page there is a compass that tells the orientation of the earth. By clicking on this compass it will, by an animation, rotate the globe so the north is up according to where the user is looking.

This is done by getting the z-angle that corresponds to the rotation the world by algorithm 1. The arguments xdiff and ydiff are always set to zero when the function is used for this puspose. It may be some other values when used in zoom-to-mouse that is described below.

Algorithm 1 get-uprighting-z-angle

```
1: function GET-UPRIGHTING-Z-ANGLE($xdiff, ydiff$)
2:     $futureEarthOrientation = identityMatrix \cdot rotateZ(-1 \cdot arctan\frac{ydiff}{xdiff}) \cdot rotateY(\sqrt{ydiff^2 + xdiff^2} \cdot zoomlevel \cdot 0.1) \cdot rotateZ(arctan\frac{ydiff}{xdiff}) \cdot earthOrientationMatrix$
3:     $northpoleX = firstcolumnoffutureEarthOrientation$
4:     $northpoleY = secondcolumnoffutureEarthOrientation$
5:     $northpoleZ = thirdcolumnoffutureEarthOrientation$
6:     $northpoleY_{norm} = \frac{northpoleY}{\sqrt{northpoleY^2 + northpoleX^2}}$
7:     $uprightingzAngle = cos(northpoleY_{norm}) \cdot sign(northpoleX)$
8: end function
```

Zooming to mouse when double-click
If a double-click occur somewhere on the globe and an animation will start. This animation rotates the globe so that the place where the double click occurred will be rotated to $(0,0,1)$ in world coordinates facing the user. During the rotation, zooming along the z-axis and alignment of the world such that the northpole will be placed in the yz-plane pointing towards the camera will occur.

This is done by calculating the difference between the place where the double click occured and the middle of the globe, xdiff and ydiff. These values are then sent in to algortihm 1.

Zooming to mouse when scrolling
When the user scrolls, zooming along the z-axis will occur at the same time as the point where the mouse are at will be rotated towards $(0,0,1)$ in world coordinates facing the user.

2.3.10 macros.clj

Contains different macros that may be useful.

2.3.11 models.cljs

This file contains the definition of the sphere that represents the earth and the plane that represents the universe. The sphere has a radius of 1 and consists of 8192 vertices? and the plane has a width of 10 and and a height of 8 and consists of 4096 vertices?.

2.3.12 Shaders.cljs

In this file the shaders are defined. A shader specification is given as an instance of the data type map with the keys vertex shader, fragment shader, uniform, varying, attribs and state. The these values are then given as strings with the source code for each entry.

The shaders included with weather-front are.

- Standard
- Temperature
- Precipitation

Standard shader
This is a very basic shader that is used to visualize the earth without any climate data. A texture of the earth

is uploaded as an uniform and wrapped around the sphere. To improve the visualization phong-shading is applied on the sphere as well.

Temperature shader
In this shader, all features described in the Standard shader is included and additionally a texture with temperature data is uploaded. The temperature is visualized in two different ways. The first way is to map the temperature into an interpolated value between two colors. To accurately visualize the temperature spectra with a high resolution, a total of six colors is used described below:

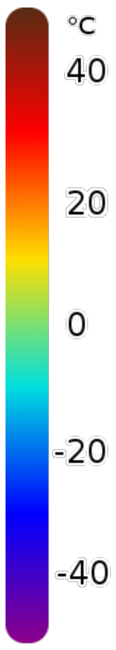

Figure 2: Temperature gradient

To give the user a better grasp of larger areas with a certain temperature, a contour map is used. The contours are white lines. The temperature values are mapped between 0 and 1 in a texture, and the contours are drawn at temperatures in a small interval around multiples of 10 degrees Celsius by using a threshold value. The contours are only drawn when the z component of the eye vector in the camera is smaller than 1.2, which means that they are only visible when the user has zoomed in quite near the surface of the earth.

Precipitation shader
In this shader, all features described in the Standard shader is included and additionally a texture with precipitation data is uploaded.

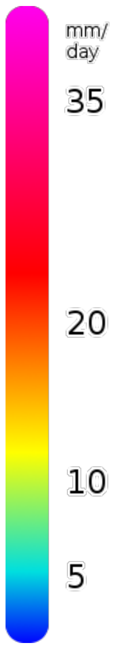

Figure 3: Precipitation gradient

2.3.13 state.cljs

This file holds all the global mutable and immutable state that are only defined once and can then be used and changed in all the other files. The following atom is used to hold different states of the program:

- camera-left/right - holds the camera matrices for the left and right canvases that is used when drawing the spheres
- background-camera-left/right - holds the camera matrices that is used when drawing the background.
- data-layer-atom - holds the information of which data to show, temperature or precipitation
- date-atom - holds the current year/month value for the left and right canvas
- earth-animation-fn - holds the function which is used to update the earth-rotation matrix
- static-scene-coordinates - holds the coordinates of the current static view scene
- earth-orientation - holds the earths rotation matrix which is uploaded to the shader every frame
- visibility-atoms - decides which page is shown for the user
- climate-model-info - holds which climate-model and exhaust levels that is currently visualized
- models - hold keys for the buffers of the sphere and plane of the left and right canvas
- texture-atoms - holds the information of which texture that is loaded to which shader
- dynamic-texture-keys - holds the information of what data that is to be requested from backend
- shaders-left/right - holds the shader specification of the shaders used to display different data or the background
- current-shader-key - holds the key to the which shader to use the following frames
- time-of-last-frame

- pointer-zoom-info - holds the information of how much the earth should be rotated around what axis when the user double-clicks on the globe.
- year-update - holds the information when year/month was last updated if the sliders are in play mode
- texture-info - holds the information where to display the data on the sphere
- space-offset - holds the information which part of the space texture that is mapped onto the background plane
- pan-speed - holds the information of how fast and around what axis the earth should rotate after panning

2.3.14 textures.cljs

This file contains necessary functions for loading textures and climate data.

- load-texture - Loads a given URL into a WebGL texture for a given WebGL context.
- load-texture-if-needed - Adds a caching mechanism on top of load-texture, will only request the image if it isn't already in the given map of textures.
- load-data - Constructs a data request to our back end Thor from the given arguments like e.g. year, month, from-latitude, to-latitude, and runs load-texture-if-needed for that.
- load-data-into-atom-and-return-key! - Convenience function which mutates the given atom storing the currently loaded data from Thor and returns the key.
- load-data-for-current-viewport-and-return-key! - Tightly coupled function which stitches together a whole bunch of functions and does what its name says it does.

2.3.15 transforms.cljs

This file contains a number of functions that either get certain coordinates or convert coordinates from one coordinate system into another. The following functions are provided:

- lat-lon-to-uv - converts latitude and longitude to uv-coordinates of the texture
- lat-lon-to-model-cords - converts latitude and longitude to model coordinates
- model-coords-to-lat-lon
- get-center-model-coords - this function takes the point in the middle of the earth facing the viewer i.e. $(0,0,1)$ in world coordinates and converts it to model coordinates. This is done by applying the inverse of the earth-rotation matrix onto the vector $(0,0,1)$
- get-lat-lon-map - this function returns a map of which longitude and latitude data is asked for. First it gets the model coords of the center position and converts it to latitude and longitude. Depending on the zoom-level, three different sizes of data can be requested
- get-texture-position-map - converts the latitude and longitude request map into which uv-coordinates the data should be displayed
- northpole-pole-rotation-aound-z - returns the angle of how much the northpole is rotated around the z axis

2.3.16 ui.cljs

This file holds all the user interface related code such as code for defining buttons and sliders. This is done in HTML mixed with clojurescript code. This file also contains several of functions that is described below. It is recommended to read the userguide first in order to fully understand this section.

- set-static-view - resets the worlds rotation to the predefined continent position corresponding to the button that was clicked
- hide-unhide -Returns the key hidden if the key visible is send as argument and vice versa
- toggle-about-page - uses hide-unhide on two atoms.
- go-from-landing-page - removes landing page from program
- update-climate-model-info - updates the info given by the climate model
- toggle-play-stop - starts and stops the time sliders
- update-shader-and-data-layer - update so right shader is used when switching between normal, temperature and precipitation data.
- button - creates a button with a given HTML id which when clicked does function on an atom with arguments

Data
When the user clicks on the button standard, temperature or precipitation, an atom is set to choose the correct shader when drawing the following frames. If temperature or precipitation is chosen a request for new data is also sent to Thor.

This menu also provides two buttons which controls which climate-model and exhaust-levels that is requested during a data request.

Navigation
The navigation menu provides a number of buttons which sets the earth-animation to different states. If the buttons which shows the different continents is clicked, the atom which holds the earth-rotation matrix is set to a static value.

If the spin-earth button is clicked, the atom $earth - animation - fn$ is set to hold the function $spin - earth!$ which updates the earth-orientation explained in 2.3.19.

If the about button is clicked a div which blurs the window is set to visible and a text about the webpage is displayed.

Sliders
When the play-button is pressed, an atom which holds which year and/or month to show data from is updated with specific intervals. This information is also passed as an uniform to the shader in order to visualize for the given time. The state of the atom is also updated every time the user move the slider.

2.3.17 util.cljs

This file holds all the general help functions that can be helpful in other files. These are:

- transparent-println - print something and return that something
- toggle - Add item to set if it's not in set, remove item from set if it's already in it. In other words, toggle the item in the set

- get-filename - get the name of the file in the given path
- map->query-string - Turns a map into a query string
- north-pole-rotation-around-z -

2.3.18 watchers.cljs

This section contains functions that adds watchers to different data events. For example when the button "precipitation" is clicked a watcher catches this and will trigger a new data load from the back end.

The watchers that are defined in this file are:

- Rotation - when rotating the earth
- Date-change - when manually chaning the date with the time sliders
- Zoom - When zooming in and out
- Temperature/precipitation - when changing data to look at
- Climate model/exhaust-level - when changing climate model or exhaustlevel

2.3.19 world.cljs

The functions that updates the rotation-matrix of the globe are defined here. Below is a list of the different rotation-functions:

- set-earth-orientation - takes in three angles and returns a matrix that is the multiplication of the x, y and z rotation matrices
- show-static-view - sets the earth-orientation matrix to the matrix returned by set-earth-orientation
- spin-earth! - updates the earth-orientation matrix to the multiplication of a rotation matrix around the y-axis and the current earth-orientation matrix
- stop-spin! - does nothing to the earth-orientation matrix
- after-pan-spin! - this function updates the earth-orientation matrix with information provided by the atom pan-speed explained in 2.3.13
- align-animation - this function updates the earth-orientation matrix with information provided by the atom pointer-zoom-info explained in 2.3.13
- zoom-camera - updates the z-value of the camera position. The value is clamped between 1.1 and 3.0
- reset-zoom - resets the camera position to $(0, 0, 3.0$

3 Thor

Thor is the back end of Weather Magic. It reads climate data from netcdf files downloaded from Eearth System Grid Foundation servers [4] and then Weather Front or other applications can get climate data through direct HTTP requests using the Thor HTTP API 3.4.

3.1 System requirements

In order to run thor, you need:

- A POSIX [5] compliant system with python3 and pip.

It is recommended that you either run a Linux disribution based on Debian 8 or later, or macOS 10.11 or later. Thor will run on pretty any modern machine, but for a production level system running the included maximum + historic datasets - hardware recommendations are as follows:

- At least two x86-64 capable processor cores.
- At least 8 GiB of RAM (if in-memory caching).
- At least 15 GiB of free disk space (for NetCDF files).

3.2 Installation

3.2.1 Dependencies

Thor is built using Python 3. You need to install Python and its package manager pip in order to run thor. You'll also need some NetCDF libraries.

On Debian or Ubuntu:

```
sudo apt-get install python3 python3-pip netcdf-bin libhdf5-serial-dev libnetcdf-dev
```

On macOS:

```
brew install python3 findutils
pip3 install virtualenv
```

3.2.2 Installation for development

Development requires a couple additional dependencies (see also additional pip dependencies after virtualenv is set up):

On Debian/Ubuntu:

```
sudo apt-get install python3-dev python-virtualenv liblapack-dev libatlas-dev gfortran
```

It's recommended to use virtualenv for development which allows for setup and other possibly system damaging procedures without actually running the risk of doing so. To set up the virtual environment for the first time, stand in the source code folder and run:

```
virtualenv -p python3 env
```

Then activate the environment, this is the only thing you need to do on consecutive shells you want to develop in:

```
source env/bin/activate
```

You're now ready to install the additional python dependencies into your virtual environment using pip:

```
USE_SETUPCFG=0 HDF5_INCDIR=/usr/include/hdf5/serial pip3 install -U -r dev-requirements.txt
```

To keep everything nice and clean we should also lint our code before committing it, still standing in the root of the source code folder:

```
ln -s src/commit-hook.bash .git/hooks/pre-commit
```

3.2.3 Exiting virtualenv

In order to escape the virtualenv one can either close the terminal or run:

```
deactivate
```

3.2.4 Getting files

In order to get files an account on earthsystemcog.org needs to be created.

After an account is created permission to download files from the CORDEX project needs to be granted. This can be gained by trying to download the following file

CORDEX netcdf file

To download all files required for Thor the following script should be run:

```
./PATH/TO/THOR/scripts/get_ESCOG_files.sh
```

3.3 Thor program structure

In figure 4 a flow chart of how Thor works logically can be seen.

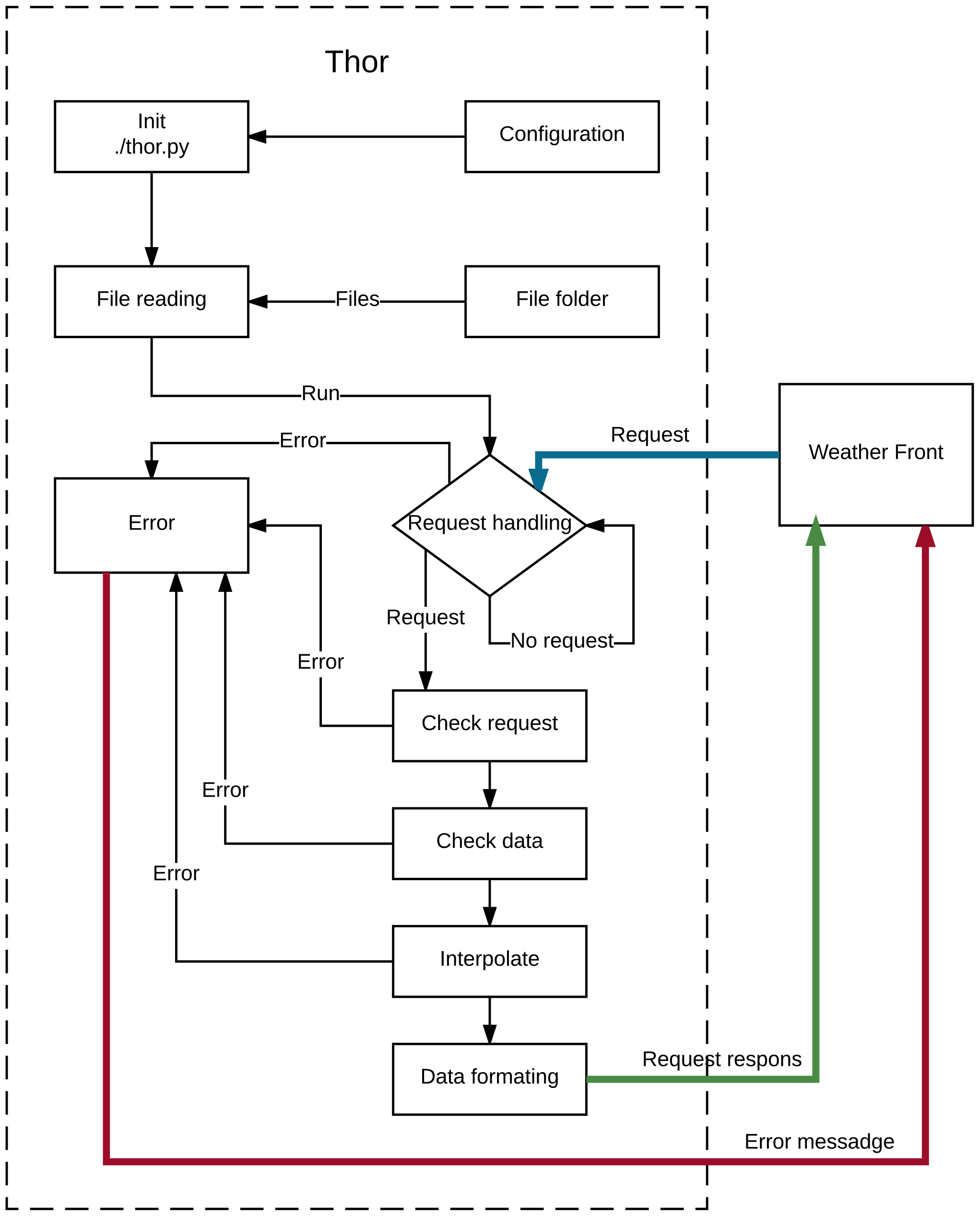

Figure 4: A system overview of Thor

Upon launch of thor, the default configuration is read from the file default.py. In case there are any command line arguments given to thor that change the defaul values, these are changed. All configuration is then put into non-mutable state so that a malicious request can't rewrite the configuration.

After this the application reads the NetCDF-files are loaded into RAM using the class "reader" located in thor/reader.py.

After this the flask app "thorApp" is launched, ready to serve requests. Requests are translated from URLs and POST-bodies into functions and Python dictionaries by the micro framework Flask [6].

For the latest information on Thors request and response format, please see the Thor API, section 3.4 also contains API documentation.

3.3.1 Reading netCDF files

When reading the NetCDF files, Thor organizes them into a tree structure with dictionarys categorizing the data into different geographical domains, climate models and exhaust levels see section4. The leaf nodes in the tree consists of lists with data-files The tree structure have the following levels:

```
domain
    model
        exhaust-level
            files-with-data-over-given-period
```

This tree makes it easier to find the correct data files when someone makes a request to thor.

Interpolation of data When fetching data from a NetCDF file, the data fetched from the NetCDF file are first interpolated to match the resolution requested by the client. This is done through linear interpolation according to equation 1 in each dimension (2D, lat/lon) [7].

$$L(x) = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \cdot (x_2 - x_1) + f(x_1) \tag{1}$$

Where x_1 and x_2 are two different points in the requested area and $f(x_1)$ and $f(x_2)$ are their respective values, wether that is a temperature or some other value.

This interpolation gives Weather-Front the possibility to always request PNGs of the size needed on the screen. It gives Weather-Front the ability to focus its efforts on the interface and not on transforming data to fit different ammounts of data to the screen. This is flexible since the client can ask both for higher and lower resolution data than is available on the server - and does not need to match the resolution of the data stored on the server itself.

Stitching data from different files If a client requests data spanning over different geographical domains[1], then Thor fetches data from different files in order to fill the requested area the best it can given the available data-files. In order to stitch these, Thor first initializes an array with the entire requested area. Thor then calculates how much of the requested area that overlaps with different geographical domains. Thor then fills the initialized array with data from the different geographical domains.

[1] 4

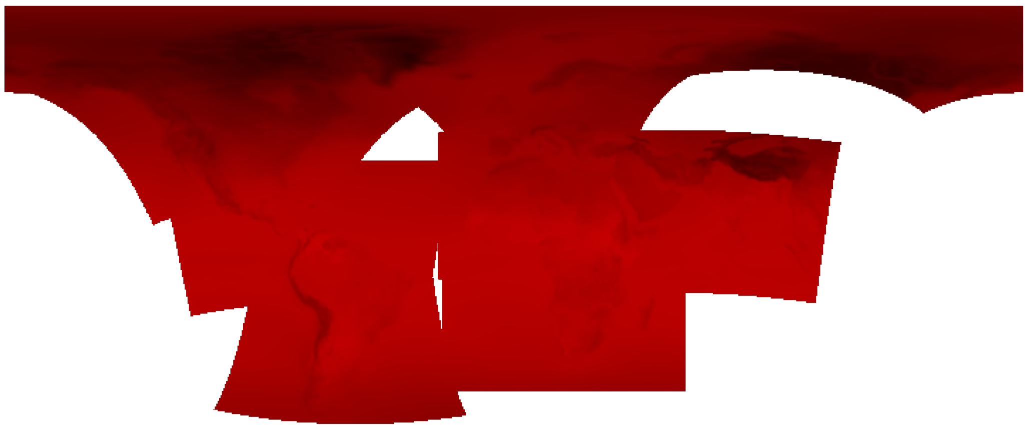

Figure 5: A PNG returned from Thor with data stitched from different NetCDF-files

Converting data to PNG The stitched data is then converted into the red channel of a PNG [8] image with RGBA-channels [9], with the alpha channel set to mask out the areas were no data is returned. Due to this, the range and resolution of the data is fit into an 8-bit unsigned integer. This does mean some information is lost, however the sent data is very compressed and transfers are fast. For information on how to translate the PNG integer range into real units, please see the section HTTP API3.4 or preferably the Thor HTTP API

The reason for using the RGBA-color space and not just a grayscale image with alpha channel, even though only two channels are used, is that WebGL [10] version 1 can't handle that type of image data. Version 2 of WebGL can do this but is currently not enabled by default in all browsers, and are in all cases seen as experimental. [11]

3.4 Thor HTTP API

This section outlines the Thor Web API.

3.4.1 URL-scheme

The URL-scheme are as follows:

```
/api/<variable>
```

Returned data variable and dimensionality is decided by in-arguments given by the client.

variable is one of the following:

- temperature
- precipitation

Each request needs to provide parameters either in URL or as a json (OBS: Set the HTTP header "Content-type" to "application/json") data object:

Table 1: Thor API arguments

Field	**Type**
year	int
month	int
from-longitude	float
to-longitude	float
from-latitude	float
to-latitude	float
exhaust-level	int
climate-model	int
height-resolution	int

All methods must be called using HTTP(S). Arguments can be passed as GET or POST params, either in URL or as "Content-Type: application/json"when doing a POST.

Example

```
{
        "year": "2083",
        "month": 1,
        "from-longitude": "-180",
        "to-longitude": "180",
          "from-latitude": "-80",
        "to-latitude": "80",
        "exhaust-level": "rcp45",
        "climate-model": "CNRM-CERFACS-CNRM-CM5",
    "height-resolution": 1024
}
```

Response If the result is successful, the response is a PNG image containing the data. in the event of a non-successfull request the response contains a json object, which at the top level contains a boolean value indicating success status as well as an error message.

Example of a non successful response

```
{
    "ok": false
    "error": "ERROR MESSAGE"
}
```

Temperature Temperature is returned as a PNG image clamped to an integer range of 0-255 since the PNG itself is limited to that range. The temperature is centered with 0 degrees Celcius around 128, and the total temperature range is -64 to 63 degrees celcius. This means that each step in the PNG integer range represents half a degree Celius/Kelvin. Formula for getting temperature in Celcius from a pixel value in returned image is as follows.

$$Temperature[C^{\circ}] = \frac{pixelvalue - 128}{2.0} \tag{2}$$

Precipitation Percipitation is returned as a PNG image is clamped to 0-255 integer range since limited by PNG integer range. The total precipitation range is 0 to 62 mm/day. This means that each step in the PNG integer range represents 0,25 mm/day of rain. Formula for getting precipitation in Celcius from a pixel value in returned image is as follows.

$$Percipitation\left[\frac{mm}{day}\right] = \frac{pixelvalue}{4.0} \tag{3}$$

Finding available climate models It is possible to ask the server for available data. To do this, send a GET or POST request to:

`/api/question`

This will return a javascript with something like this:

```
{
        "exhaust-levels": [
                "rcp45",
                "rcp85"
        ],
        "models": [
                "ICHEC-EC-EARTH",
                "CNRM-CERFACS-CNRM-CM5",
                "IPSL-IPSL-CM5A-MR"
        ],
        "ok": true,
        "domains": [
                "NAM-44i",
                "CAM-44i",
                "EUR-11i",
                "WAS-44i",
                "EUR-44i",
                "MNA-22i",
                "AFR-44i",
                "SAM-44i",
                "ARC-44i",
                "MNA-44i"
        ]
}
```

It is also possible for the server to return the entire internal tree structure of the server by setting the argument "with-tree". This is however very taxing for the server and will give a much longer response time (seconds!).

4 Climate data

Climate data is generated by weather institutes with the help of super computers all over the world, like here in Linköping at NSC. The data can be accessed on the website www.earthsystemcog.org.

4.1 How to get data

This is a simple guide on how to get climate data netCDF files from www.earthsystemcog.org

4.1.1 Getting started

1. Go to www.earthsystemcog.org
2. Click create account or follow this link
3. Login and go to the Browse Projects list on the right side of the screen. Each projects contains different reasources, they are not all of intrest to us.
    - For global data go to CMIP5.
    - For regional data go to CORDEX.

4.1.2 Accesing CMIP5 - global data

1. Press Search for CMIP5 project data or follow this link.
2. To the left are different categorial filters that can be played with. Good filters to begin with are:
    - Project-CMIP5
    - Realm-land
    - Variable-tas
3. Press search after choosing filters.
4. Datasets matching your filters will now come up.
5. To download single netCDF files from the datasets press Show Files and the HTTPServer on the choosen file.
    - If this is your first time downloading from CMIP5 you have to register.

4.1.3 Accesing CORDEX - regional data

1. Press Search for CMIP5 project data or follow this link.
2. To the left are different categorial filters that can be played with. Good filters to begin with are:
    - Domain: EUR-11i
    - Variable: tas
3. Press search after choosing filters.
4. Datasets matching your filters will now come up.
5. To download single netCDF files from the datasets press Show Files and the HTTPServer on the choosen file.
    - If this is your first time downloading from CMIP5 you have to register.

4.1.4 Viewing netCDF files

1. A simple way to view the netCDF files is with ncview it can be downloaded from the following link.
2. Once a netCDF file is downloaded and opened with ncview the data can be visualized. This is done by pressing the variable of interest within the GUI.
3. OBS! With most climate data models there is only one variable of interest often by the same name as the first letters of the filename. Press this variable and nothing else.

4.1.5 Variable categori

When choosing a project in earthsystemCoG and searching for data in it there are a lot of different categories to choose from. This text is ment to explain what those different categories are and what the variables in them stand for.

Project What project do you want to search for data in? Often the same as the project you are in but certain projects can have "sub" projects. Two types of projects are CMIP54.1.2 and CORDEX4.1.3 which are described above.

Domain Which geographical region is the data from? The regions available in climate data simulations are:

- AFR - Africa
- ANT - Antarctica
- ARC - Arctic
- AUS - Oceania
- CAM - Central America
- EAS - East America
- EUR - Europe
- MNA - Middle East and North Africa

- NAM - North America
- SAM - South America
- WAS - West Asia

After the region name there is always a number if the number is high say 44 it means low resolution. If the number is low it means high resolution.

After the number there can be and "i" if it is there it means that it is a 2D map if it is not there it means that the map has been projected on a sphere.

Institute The weather institute responsible for the simulation.

Driving model Driving model is the climate model used for calculating the simulating values. Examples of climate models are:

- CNRM-CERFACS-CNRM-CM5 a french model from 1995
- ECMWF-ERAINT is a global atmospheric reanalysis from 1979, continuously updated in real time.
- ICHEC-EC-EARTH Irish model
- IPSL-IPSL-CM5A-MR The Institute Pierre Simon Laplace active since 1995
- MOHC-HadGEM2-ES Met Office Hadley Centre active from 1990 from UK, uses data from the past 100 years to predict the next 100 years
- MPI-M-MPI-ESM-LR Max-Planck-institute for meteorology German

More technical information about the driving models can be found at the following link.

Experiment What type of data are we looking at?

- Evaluation - Used to evaluate model with historical data
- Historical - Historical data
- rcp[2] - prescribed greenhouse-gasconcentration pathways throughout the 21st century corresponding to different radiative forcing stabilization levels by the year 2100.
  - rcp26 - lvl 2.6 W/m^2
  - rcp45 - lvl 4.5 W/m^2 high probability.
  - rcp85 - lvl 8.5 W/m^2 high probability. [**?**]

Other abbreviations Other abbreviations at earthsystemCoG that might need explaining are:

- MR - medium resolution
- LR - low resolution
- A - Atmosphere
- ES - Earth system
- CC - Carbon cycle

[2]http://tntcat.iiasa.ac.at:8787/RcpDb/dsd?Action=htmlpage&page=welcome

Appendices

References

[1] Phil Hagelberg. Leiningen, 2012.

[2] Robert Nyström. Game programming patterns, 2009.

[3] Mozilla developer network and individual contributors, javascript events, 2016.

[4] Sylvia Murphy. Earth system cog, 2016.

[5] Posix, 2016.

[6] Armin Ronacher. Flask, 2016.

[7] Encyclopedia of math, linear interpolation, 2012.

[8] Portable network graphics format, 2016.

[9] Rgba-color space, 2016.

[10] Rgba-color space, 2016.

[11] teoli. Mozilla developer network, 2016.