Serving map tiles with MediaFlux

by implementing a WMS-GetMap-like service

## Background

James Cook University currently uses MapServer to deliver map tiles to mapping software running inside client web browsers. An example of this is the CliMAS project:

<http://climas.hpc.jcu.edu.au/maps/>

(select a species name, e.g. “cane toad”).

Tiles are derived from GeoTIFF files. For example, the current distribution of cane toads across Australia is represented as a single GeoTIFF, and delivered to the browser as a dozen or so map tiles. QCIF (TSV), located at JCU, has requested that capability be added to MediaFlux to be able to derive and deliver map tiles from GeoTIFF assets.

Duplicating all MapServer functionality, or implementing a mapping standard like WMS, **is not required**. This document describes a **minimal acceptable product** for serving map tiles.

## Contents

Background 1

Contents 1

Scope 2

Overview of the map tile service 3

Identifying the asset 4

Projection 4

Colour mapping 5

Colour mapping examples 6

The map tile request 8

Acceptance Criteria 9

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Modification** |
| 3 June 2015 | daniel.baird@jcu.edu.au | Incomplete version, adapted from document  “2015-03-ReplacingMapServerWithMediaFlux.docx” |
| 19 June | daniel.baird@jcu.edu.au | Complete except for performance & correctness tests. Circulated for comment |
| 24 June | daniel.baird@jcu.edu.au | Clarified that static strings in URL are okay; added perf and tile examples (github repo) |

# Scope

This spec includes:

* Requesting map tiles using the GET method of the HTTP / HTTPS protocols
* RESTful / stateless tile requests
* GeoTIFF data sources
* Mapping WGS84 lat/long input data to EPSG:3857 “spherical Mercator” output data
* Output mapping using linear gradients of colour and opacity
* A single optional cutoff value

This spec does NOT include:

* Using POST or any HTTP method other than GET
* Implementing any part of the WMS standard
* SOAP or any protocols other than HTTP / HTTPS
* Input formats such as ASCIIgrid, ESRI grid, NetCDF, or any raster format other than GeoTIFF
* Handle input projections other than WGS84 latitude and longitude
* Handle output projections other than EPSG:3857
* Mapping of data values to discrete steps

# Overview of the map tile service

In practice, performance optimisations are likely to affect the order of these processing steps.

### Initiate service

The service will run continuously for all assets in selected namespaces, and so should be convenient to start and keep running. Note that the service should support tile serving for multiple groups of assets. The implementation may support this by allowing multiple instances of the service, each serving assets from a single root namespace, or by a single instance of the service supporting multiple root namespaces (and multiple colour configurations).

### Receive request: Identify the asset

URL elements must be able to identify the asset being addressed. We expect this information is primarily contained in the URL path. The URL must be predictable based on the asset name.

This is expanded on in the “Identifying the asset” section.

### Re-project data

Data in JCU’s GeoTIFFs use latitude and longitude on a WGS84 ellipsoid to specify location. Browser mapping libraries all employ a spherical Mercator (EPSG:3857) map projection. To generate a map tile, the server must convert data from lat/long to spherical Mercator.

This is expanded on in the “Projection” section.

### Mapping data values to colour values

The GeoTIFFs JCU intends to serve from MediaFlux are storing geographic data, not images. Each pixel value is not a colour; it is a data value. For example, in a GeoTIFF of cane toad climate suitability, pixel values vary from 0 to 1. This value represents the likelihood of a cane toad thriving at that geographic position. In the GeoTIFF, this survival likelihood datum is encoded as a greyscale pixel value.

To display data in a visually useful map tile, pixel data values from the input GeoTIFF need to be mapped to pixel colours for the output image.

This is expanded on in the “Colour mapping” section.

# Identifying the asset

The URL used will include information identifying the source data asset. Clients must be able to construct this URL from knowledge of the (user-constructed) namespace hierarchy and asset names, and not require MediaFlux-generated ids or other non-user information.

For example, a MediaFlux data collection may include these assets:

/myMaps/mammals/Felis\_catus/current.tiff

/myMaps/mammals/Felis\_catus/2050.tiff

/myMaps/amphibians/Rhinella\_marina/current.tiff

/myMaps/amphibians/Rhinella\_marina/2050.tiff

The client application already knows the project identifier (/myMaps), and can collect species name (e.g. the mammal species Felis catus), and time point (e.g. the year 2050) from the user. With these data points, plus any other fixed strings identifying the project or tile service, the client must be able to construct the correct URL. For example:

http://mf.jcu.edu.au/tiles/myMaps/mammals/Felis\_catus/2050.tiff?BBOX=*[…]*

http://mf.jcu.edu.au/myMapTiles/Felis\_catus.mammals.2050.tiff?BBOX=*[…]*

http://mf.jcu.edu.au/tilesGalore/?asset=Felis\_catus.mammals.2050&BBOX=*[…]*

can be generated from information known to the client (assuming that tiles, myMapTiles, and tilesGalore are known strings), so is an acceptable URL for this implementation. However, URLs such as these:

http://mf.jcu.edu.au/tiles/myMaps/asset1902?BBOX=*[…]*

http://mf.jcu.edu.au/myMapTiles/?uuid=*[…]*&BBOX=*[…]*

cannot be generated from information known to the client, and so would not be acceptable.

# Projection

Features of the more-or-less spherical world can be mapped to rectilinear images in a variety of ways. These are called projections. This spec requires a re-mapping of data from the source projection (latitude and longitude values on a WGS84 ellipsoid), to output projection (EPSG:3857, a terrible projection often referred to as spherical Mercator or web Mercator).

### Notes about EPSG:3857

<https://en.wikipedia.org/wiki/Web_Mercator>

The north and south poles are not representable in Mercator projections. Okay fine there are transverse Mercators, but let’s not try to out-nerd each other. For EPSG:3857 the representable world is defined as ±180° latitude and ±85.051129° longitude.

Coordinates in EPSG:3857 are measured in metres, with the x value measuring east–west and y measuring north–south. The origin is at 0°, 0° just like the lat/long origin, with positive x values to the east and positive y values to the north. The world bounds are roughly ±20,037,508.34278 metres on both x and y axes, giving a square world map (which was probably the motivation for picking 85.051129° as the longitude limit).

### Zoom

The GeoTIFF asset may not cover this entire area. The ratio of the asset’s data pixel density to the required output pixel density provides a “zoom” level.

Most zoom levels will use a lower output resolution than the data’s resolution, which will require reducing the number of pixels in the selected bounds. There are a number of strategies for reducing data values; the simplest is to copy the value of the source data point that is nearest to the output point. Alternative techniques include finding the median or average of the source data points that contribute to the output point. The nearest-neighbour technique is acceptable.

Some zoom levels may require more dense output pixels than the input data include. The nearest-neighbour technique can be applied in this situation too.

It may be that a library is employed to perform re-projection that supports “better” techniques for remapping data pixels; in that case please advise.

# Colour mapping

The mapping of data value to colour value must be configurable. It is acceptable to have a single configuration applicable to all assets contained within a given namespace. However note that it must be possible to simultaneously serve tiles from multiple namespaces, each using a different configuration.

Colour mapping should support four methods of configuration. All methods describe a linear colour-and-transparency scale (by giving a colour for each end of the scale) that data points are mapped to. The methods vary in how the scale is matched to the input data range, and whether there is a cutoff value specified.

For the following discussion, a colour value includes transparency information – that is, each colour value contains red, green, blue, and alpha channels.

### Tabular summary of colour mapping configuration methods

|  |  |  |
| --- | --- | --- |
|  | **Colour scale**  **matches data range** | **Colour scale**  **fixed** |
| **No cutoff value** | Method 1 | Method 3 |
| **Cutoff value specified** | Method 2 | Method 4 |

### Method 1: Maximum and minimum relative values

The configuration specifies two colour-plus-transparency values.

* maxC: a colour+alpha level for the highest data value
* minC: a colour+alpha level for the lowest data value

The tile service will need determine the numerically highest and lowest data values across the entire GeoTIFF. These data values will map to the configured colour and transparency levels. All other data points will fall between those values; the colour and transparency values of a data point is calculated by linear interpolation of the red, green, blue and alpha colour channels.

### Method 2: Maximum and minimum relative values with cutoff

Similar to Method 1, but the configuration specifies an additional numeric value.

* maxC: a colour+alpha level for the highest data value
* minC: a colour+alpha level for the lowest data value
* cutoff: a numeric value

The tile service maps highest and lowest data values to the maxC and minC colour values just as in Method 1, but additionally checks to see if the data value is less than or equal to the configured cutoff value. Any data value below or equal to cutoff must have its alpha channel set to full transparency.

Note that if cutoff is higher than or equal to the minimum data value in the asset, no pixel will be mapped to the minC colour; if cutoff is set lower than the minimum data value, then no pixel will be cutoff.

### Method 3: Low and high absolute values

The configuration specifies two numeric values and two colour-plus-transparency values. The numeric values can be negative; they are “absolute” in the sense that they are fixed numbers not relative to the data range of the asset.

Note that in this method, maxV and minV are NOT the maximum and minimum values across the data set; they are just the top and bottom ends of the colour scale.

* minV: a numeric value that should map to minC
* minC: a colour+alpha level for the value minV
* maxV: a numeric value that should map to maxC
* maxC: a colour+alpha level for the value maxV

minV should be less or equal to maxV. The tile service maps data values equal to or less than minV to the colour and transparency level minC, data values equal to or greater than maxV to the colour and transparency level maxC, and data values in between have red, green, blue and alpha levels linearly interpolated between minC and maxC.

Note that data values below minV are fixed at minC, not interpolated beyond. Similarly, data values above maxV are capped at maxC.

### Method 4: Low and high absolute values with cutoff

Similar to Method 3, but the configuration specifies an additional numeric value.

* minV: a numeric value that should map to minC
* minC: a colour+alpha level for the value minV
* maxV: a numeric value that should map to maxC
* maxC: a colour+alpha level for the value maxV
* cutoff: a numeric value

The tile service maps data values to colour+alpha just as in Method 3, but additionally compares the data value to the configured cutoff. If the data value is less than or equal to cutoff, the resulting colour must have its alpha channel set to full transparency.

Note that if cutoff is set above minV, no data values will end up mapping to the minC colour.

## Colour mapping examples

The following example colour configurations are presented in JSON, and express colour plus transparency values using a CSS-like rgba() function, defined as:

rgba(<red>, <green>, <blue>, <alpha>)

<red>, <green> and <blue> represent colour components as a percentage, where 0% means no presence of that colour, and 100% means that colour is present at maximum intensity. <alpha> is the degree of opacity of the colour, with 1.0 as completely opaque, and 0.0 is completely transparent.

### Example 1

Map the source file’s minimum value to light blue at 10% opacity; map the source file’s maximum value to darker blue at 90% opacity.

rainfallMapConfig = {

“minC”: “rgba(50%, 75%, 100%, 0.1)”,

“maxC”: “rgba( 0%, 10%, 100%, 0.9)”

}

In Example 1:

* just minC and maxC are specified, so this is a Method 1 colour mapping.
* the lowest value pixel would be mapped to minC, a light denim blue that is mostly transparent
* the highest value pixel would be mapped to maxC, a rich blue
* a data value midway between the source file’s maximum and minimum values would be mapped to a colour close to rgba(25%, 42.5%, 100%, 0.5)

### Example 2

Map values from a red colour at 50% opacity up to green at 90% opacity. Blank out pixels below 0.1.

climateSuitabilityMapConfig = {

“minC”: “rgba(100%, 20%, 20%, 0.5)”,

“maxC”: “rgba( 0%, 100%, 0%, 0.9)”,

“cutoff”: 0.1

}

In Example 2:

* minC, maxC and cutoff are specified, so this is a Method 2 colour mapping.
* any value <= 0.1 would be 100% transparent (below cutoff)
* a value midway between source min and source max, assuming it was > 0.1, would be mapped to rgba(50%, 60%, 10%, 0.7).

### Example 3

Map value <= 200 to a yellow at 50% opacity; map values >= 1000 to green at 100% opacity. Blank out non-positive values.

speciesDensityMapConfig = {

“minV”: 200,

“minC”: “rgba(100%, 100%, 0%, 0.5)”,

“maxV”: 1000,

“maxC”: “rgba( 0%, 100%, 0%, 1.0)”,

“cutoff”: 0

}

In Example 3:

* minV, maxV and cutoff are all specified, so this is a Method 4 colour mapping
* value -100 gets transparent (below cutoff)
* value 0 gets transparent (at cutoff)
* values 100 and 200 get rgba(100%, 100%, 0%, 0.5) (below minV)
* value 800 would be about rgba(75%, 100%, 0%, 0.875)
* value 3000 would get rgba(0%, 100%, 0%, 1.0) (above maxV)

# The map tile request

A tile request arrives in the form of a HTTP/HTTPS GET request. Parameters to the tile request should be supplied in the query string following normal URL convention.

### Required parameters

These parameters are required for this implementation. Other parameters should be ignored by the implementation, but must be tolerated – the presence of parameters not listed here should not prevent a tile from being served (except where mentioned in the “WMS parameters” section following).

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| bbox | Bounding box for tile extent. Value is minx,miny,maxx,maxy in EPSG:3857 metres. |
| width | Width of map output, in pixels. |
| height | Height of map output, in pixels. |

### WMS parameters (not required)

The request parameters listed above appear in the WMS GetMap service. This list of additional parameters can be used to make the service more compatible with clients expecting a WMS service, primarily by erroring out if the client requests anything unsupported.

Note that **the implementation is not required to handle these parameters at all**. They are listed here as optional exemptions to the “don’t crash on unknown parameters” rule given in the previous section.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **WMS Description** | **Acceptable behaviour** |
| service | Service name | Error if not WMS |
| version | Service version [one of 1.0.0, 1.1.0, 1.1.1, 1.3] | Ignore |
| request | Operation name | Error if not GetMap |
| layers | Layers to display on map | Require DEFAULT |
| styles | Styles in which layers are to be rendered | Ignore |
| srs *or* crs | Spatial Reference System for map output | Error if not EPSG:3857 |
| format | Format for the map output | Error if not image/png |
| transparent | Allow transparency | Error if not true |

### Example URL

Newlines have been added for readability.

http://mediaflux.jcu.edu.au/path/to/geotiff/whatever/themap.tif

?HEIGHT=256

&WIDTH=256

&BBOX=15028131.257091932,-4383204.9499851465,15654303.392804096,-3757032.814272983

### Example URL with additional WMS parameters

Newlines have been added for readability.

http://mediaflux.jcu.edu.au/path/to/geotiff/whatever/themap.tif

?SERVICE=WMS

&REQUEST=GetMap

&VERSION=1.1.1

&LAYERS=DEFAULT

&STYLES=

&FORMAT=image%2Fpng

&TRANSPARENT=true

&HEIGHT=256

&WIDTH=256

&SRS=EPSG%3A3857

&BBOX=15028131.257091932,-4383204.9499851465,15654303.392804096,-3757032.814272983

# Acceptance properties

These examples refer to files stored in a GitHub repository at

https://github.com/jcu-eresearch/mediaflux-maptileserver-data

Paths are relative to the root directory of the repository.

### Sample output

Using file SampleData/input/Corvus\_tasmanicus.tif as the source asset, the BBOX values given in the table below should produce outputs closely matching the listed files.

These tiles use a colour scheme approximately matching this Method 3 config:

minV: 0,

minC: rgba(100%, 100%, 100%, 0.7),

maxV: 1,

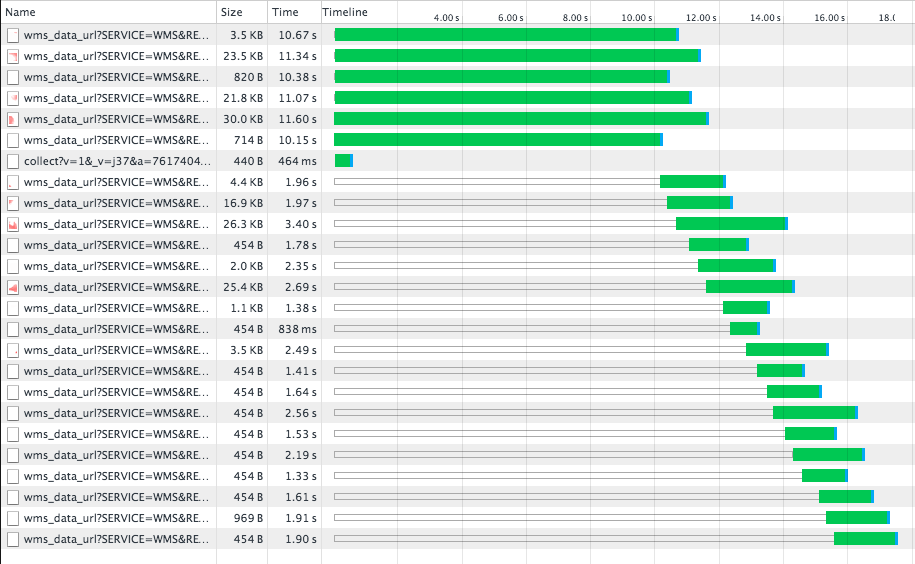
maxC: rgba(0%, 50%, 100%, 0.7)

|  |  |  |
| --- | --- | --- |
| **BBOX parameter value**  (linebreaks added)  values copied from a Leaflet/Mapserver system | **Output tile** | |
| SampleData/output/ **filename** | **Tile image** |
| -20037508.342789244,-20037508.342780735,  20037508.342789244,20037508.34278071 | C\_t\_wholeworld.png |  |
| 0,-7.081154551613622e-10,  20037508.342789244,20037508.34278071 | C\_t\_z1x1y1.png  (a blank tile) |  |
| 0,-20037508.342780735,  20037508.342789244,-7.081154551613622e-10 | C\_t\_z1x1y-1.png |  |
| 16280475.52851626,-6261721.357121639,  17532819.799940586,-5009377.085697311 | C\_t\_easterntas.png |  |

### Caching and performance

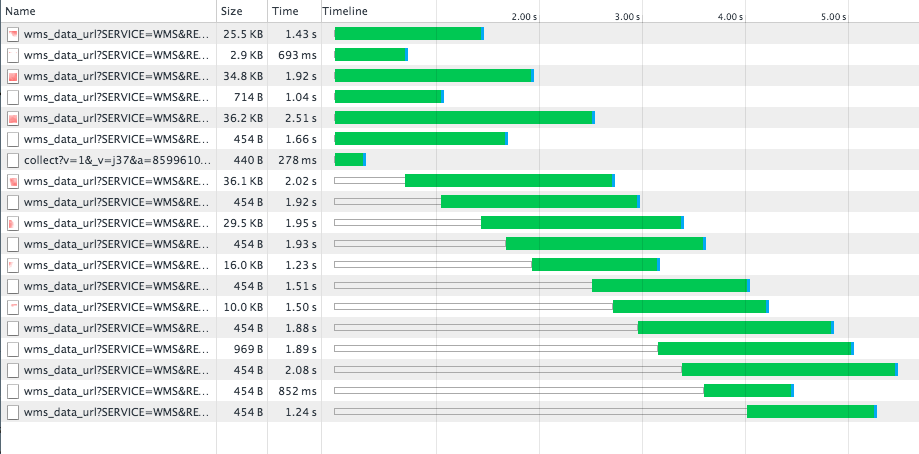
Tile generation should be fast. We expect that a caching strategy will be required to achieve reasonable performance. One option is to do re-projection and colour-mapping for the entire dataset for the requested zoom level, and cache the result. This would allow subsequent tile requests at that zoom level to be fulfilled by subsetting the cached output.

The screenshot at /Screenshots/uncached.png is an example of acceptable performance for 256×256 pixel tiles from an uncached map:



The image above shows around a 10 second cache delay while the tile server fetches the asset, then tiles being served in around 2 seconds per tile. The browser is making six requests in parallel, so tiles arrive at around 3 tiles per second.

The screen shot at /Screenshots/cached.png shows an example of acceptable performance fetching a recently viewed species (after a browser cache clean and restart). The tile service has cached the species’ map and is able to deliver tiles in around 2 seconds per tile, with the first arriving after 0.7 seconds.



On JCU’s MediaFlux hardware setup, we expect time to first tile should be ten seconds or less, and after this warmup time, tiles should be deliverable in less than three seconds per tile.