MDD – WMS Image Caching

|  |  |
| --- | --- |
| Project | Mike Operations Web |
| Author | DK83719 |
| Date | 2018-04-18 |

# Introduction

The WMS[[1]](#footnote-1) functionality used by MO Web have been update so the MIKE Provider now uses MzChart to render the images. And despite this having given a significant performance improvement (10x faster), the servers still have a vast resource load because of the calculations needed to serve each image. To resolve this load and make the rendering even faster the caching mechanism herein is introduced.

## Background

When images are served back to the client the request goes through the following call stack to render an image that fits the bounds of the client.

|  |  |  |  |
| --- | --- | --- | --- |
|  | UI (Polymer) | ■ | Request image |
| DHI.Services.Web.Controllers.MC | ■ | Validate request, respond to client caching and forward request |
| DHI.Services.GIS | ■ | Forward request |
| DHI.Services.Provider.MIKE | ■ | Read time step data from dfs2, dfsu or particle file |
| MzChart | ■ | Create spatial elements, convert geometry and create image within specific bounds |

CPU Load: ■ Low ■ Medium ■ High

Each request for an image at a certain time step results in a lot of processing because data is read from the source file and compiled into an image on the fly. Only images that a specific client has requested previously is cached through client caching. For data files with a limited number of elements this has no apparent performance impact. But with a large number of elements in each time step the calculations needed to create an image is becoming resource intensive.

# High-level design

The caching works by creating an image based on the bounds of the data file and this image is assigned quadrants. For each request the image is cropped to fit the request bounds and served back to the client. Areas in the request bounds that are outside the image bounds are transparent and areas of the image bounds that are outside the request bounds are cut off.

|  |  |
| --- | --- |
|  | Screen bounds |
|  |  |

If the request bounds are within a quadrant a new higher resolution image is created that contains the quadrant and this image is cropped to fit the request bounds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

Subsequent requests in other quadrants will gradually complete the higher resolution image.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |
|  | |  |
|  | | |  |

To accommodate for good resolutions at multiple zoom levels each quadrant is again divided into quadrants. This division continuous recursively while the size of the quadrants are above 20x20 pixel. The quadrants at each level are added to the image of that level; making the lowest level a very high resolution image.

For scenarios where the screen bounds are narrower than a quadrant, in either the horizontal or vertical direction; effectively spanning multiple quadrants in either direction.

|  |  |
| --- | --- |
|  |  |
|  |  |

This requires a higher resolution image and it is therefore necessary to load multiple quadrants before the image can be cropped. This is needed to provide a consistent resolution at the same zoom level and thereby providing a steady detail level when panning.

|  |  |
| --- | --- |
| **!** | Please note, using this caching will not always provide the best resolution available for a given area. But the difference should be unnoticeable and at a quality suited for websites. |

## Building the cache

The cache is built on the fly allowing for subsequent requests in the same area to be served with a cached image. There is therefore no lead time to preload the cache and areas of no interest does not take up space. Drawback is of cause that first request will be the slowest, but since single request rendering is not a performance issue this should not be a problem. For requests spanning multiple quadrants the lead time will at least double for the first request.



Each request is evaluated to see what detail level is needed and if no request have previously been made, to the entire data area or the specific region, a new image is created and stored. The newly created or cached image is cropped and sent back to the client.

The cache is cleared if the data source is updated.

## Storage considerations

An initial image width of 800 pixel will give a maximum of 6 images in different resolutions; i.e. level 0 to level 5.

**2^5 < 800 / 20 < 2^6**

An image is typically 20KB per quadrant and a filled cached will take up around 27MB of storage per time step.

**20KB x (1 + 4 + 4^2 + 4^3 + 4^4 + 4^5) = 27MB**

# High-level architecture

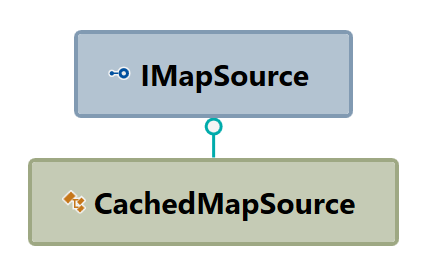
The caching is implemented in the DHI.Services.GIS[[2]](#footnote-2) module and utilized in the MapService class.

Caching

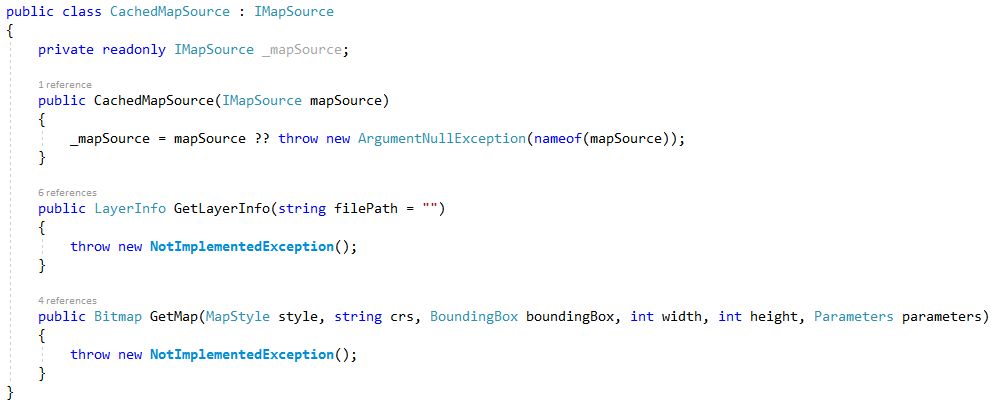
This allows for the functionality to be readily available to other components using domain services and separates client-side and server-side caching functionality in two logical components while adhering to the principle of the MIKE provider only exposing functionality from MIKE Zero.

## Decorator Pattern

Technically, the implementation is done using the [Decorator Pattern](https://en.wikipedia.org/wiki/Decorator_pattern). A new CachedMapSource is introduced. CachedMapSource is a decorator for *any* IMapSource implementation - for example the Dfs2MapSource, DfsuMapSource and ParticleMapSource.



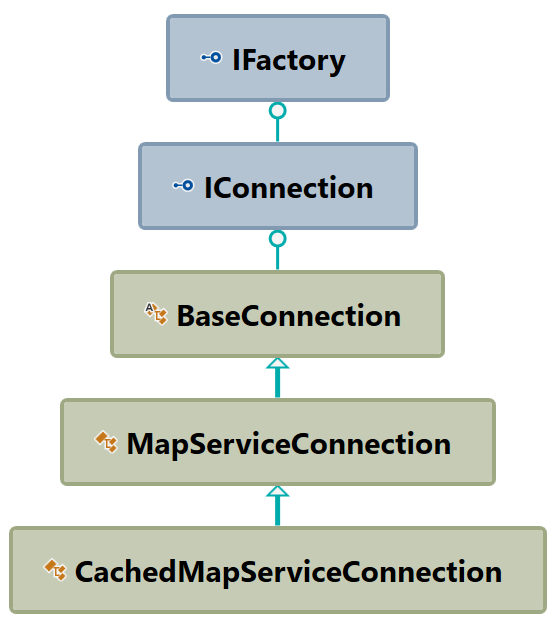
Conceptual code:



The IMapSource-implementation to be decorated with caching capabilities is injected through the constructor at runtime. This is an elegant solution because it treats caching as a truly cross-cutting concern that can be introduced as a loosely coupled and optional *extension* of the original map sources instead of as a *modification* of the original map sources code.

Furthermore, a new **Cached**MapSourceConnection is introduced for configuration and creation of a Map Service that uses the new caching decorator. The configuration of a CachedMapSourceConnection is identical to that of the original MapSourceConnection so the switch between the two options will be seamless.

The CachedMapSourceConnection inherits the original MapSourceConnection because they share all the configuration properties (only the creation is slightly different).



## Caching Technology

The [MemoryCache](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.caching.memorycache?view=netframework-4.5.1) class is selected as the caching technology. MemoryCache is an in-memory object cache implementation of the [ObjectCache](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.caching.objectcache) class.

For maximum flexibility, we could have decided to make an abstraction of a caching mechanism (e.g. an ICache interface) which was injected into the CachedMapSource, or maybe using the [LazyCache](https://github.com/alastairtree/LazyCache/wiki/2.0-Extending-LazyCache-with-Redis,-Cassandra-etc) which is about to introduce [extensibility for 3rd party implementations of ObjectCache](https://github.com/alastairtree/LazyCache/wiki/2.0-Extending-LazyCache-with-Redis,-Cassandra-etc), inclusive Redis, Cassandra etc.

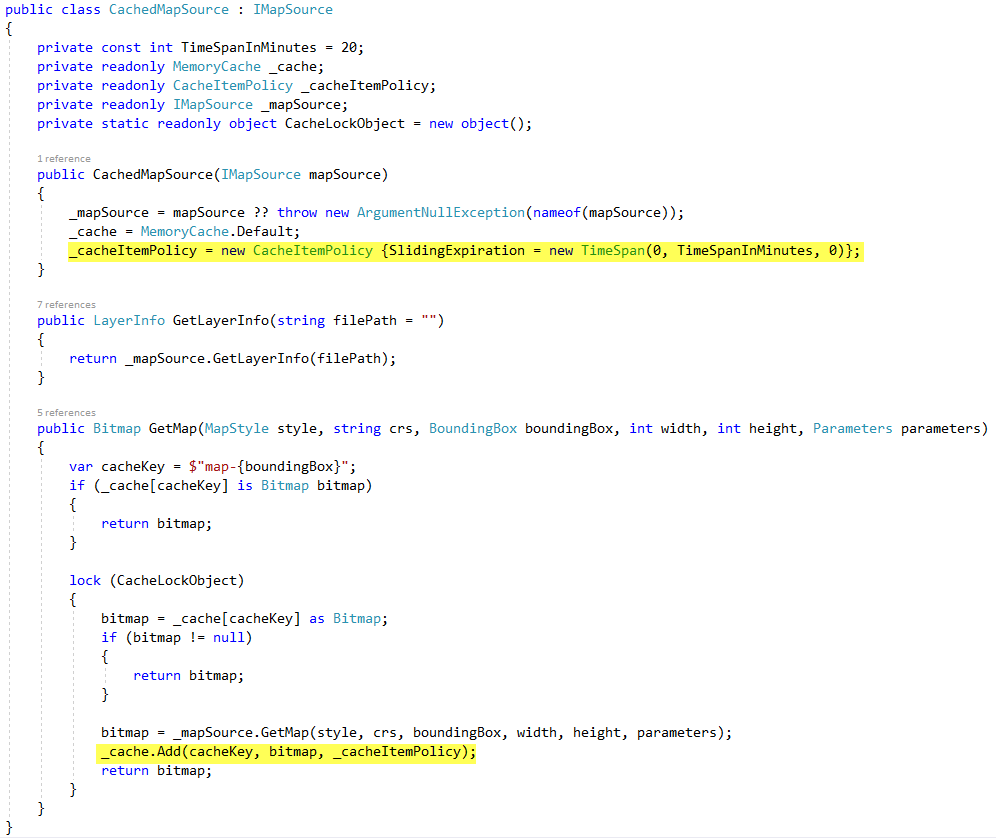
However, for simplicity and adherence to the [YAGNI](https://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it)-principle, let´s stick with MemoryCache for now.

## Caching Policy

MemoryCache supports a [CacheItemPolicy](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.caching.cacheitempolicy?view=netframework-4.5.1) that defines expiration details for cache entries.

[SlidingExpiration](https://docs.microsoft.com/en-us/dotnet/api/system.runtime.caching.cacheitempolicy.slidingexpiration?view=netframework-4.5.1#System_Runtime_Caching_CacheItemPolicy_SlidingExpiration) is used to specify the duration for which an object will reside in the cache when there is no activity—i.e., the item will be removed from the cache when the specified duration of inactivity elapses. This value shall be configurable and default to a period of 20 minutes.

Conceptual code:



# Design and Architecture Review Checklist

|  |  |
| --- | --- |
| **Reviewer** |  |
| **Date** |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Item** | **Yes** | **No** | **Comments/Suggested changes** | **Team actions** |
| Is the high-level functional design sufficiently resolved, and in accordance with the design principles in the context it will be part of? |  |  |  |  |
| Are the dependencies sufficiently resolved? |  |  |  |  |
| Are functionality put in the right components or libraries and given the right names? |  |  |  |  |
| Do we have similar functionality elsewhere that we could exploit/extend/supersede? |  |  |  |  |
| Is the class diagram sufficiently detailed? |  |  |  |  |

1. Web Map Service [↑](#footnote-ref-1)
2. sscm://solscm.dhi.dk:4900//SOLSoftware//SOLSoftware/Domain%20Services/GIS/DHI.Services.GIS [↑](#footnote-ref-2)