Tile Server Client

What is a Tile Service?

We are starting to see implementations of servers that are "something" like a web map server. What is that something?

- view of the world split up into tiles
- tiles avaiable in a single projection
- tiles available at known zoom levels

There are a number of these servers available:

- Nasa World Wind
- Virtual Earth
- OSGeo Tile Map Service
- WMS Tile Profile

Links

- Add-on:Virtual_Earth
- Virtual Earth (VE) in NASA WorldWind (WW)
- Tile_Map_Service_Specification
- WMS_Tiling_Client_Recommendation

Servers

- http://labs.metacarta.com/wms-c/

TileServer

So what does this client interface going to look like:

- TileService - a good name, describes what we have
  - Configure with a single URL
  - deleage to TileServer strategy objects (one implementation per service type)
- GeoResource
  - configured with a single URI
- GeoResourceInfo describing the service
  - TileInfo extension descrbing zoom levels (and anything else needed)

TileServer API

Represents a single tile server, actual protocol negotiated by TileStrategy object.

```
class TileServer {
    TileServer( URL server, ProgressMonitor
monitor );
    TileServer( GridInfo info ); // incase
info was created beforehand

    GridInfo getInfo();
    List<TileMap> getTileMapList();
}
```

Content is described and accessed through TileMap instances, these have enough information for rendering:

```
class TileMap {
    TileMap( TileMapInfo info );

    // data access
    CoordinateReferenceSystem getCRS();
    TileMapInfo getInfo(); // describes tile
sets

    TileSet getTileSet( ZoomLevel zoom );
}
class TileSet {
    TileSet( TileMap, ZoomLevel zoom );

    TileMap getTileMap();
    ZoomLevel getZoomLevel();

    //GridCoverage getTile( Coordinate
position );
    GridCoverage getTile( int row, int col
);

    TileRange getTileRange( Envelope bbox );
// must match CRS
}
```

Tile Service/Resource Metadata

Metadata for the TileServer needs to capture enough information for discovery:

```
class TileService extends Service {
    TileService ( URI, Memento ); // Memento
holds associated state

    TileInfo info( ProgressMonitor monitor
);
    List<TileResource> children(
ProgressMonitor monitor );
}
interface TileServiceInfo extends
ServiceInfo {
    int getTileWidth(); // often 256
    int getTileHeight(); // often 256
}
```

Metadata for each TileMap captures enough information to set up a display pipeline.

```
class TileResource extends GeoResource {
    TileResource( URI, Memento );

    TileInfo getInfo( ProgressMonitor );
    TileMap getTileMap( ProgressMonitor );
}
class TileInfo extends GeoResourceInfo {
    CoordinateReferenceSystem getCRS();
    SortedSet<ZoomLevel> getZoomLevels();
}
class ZoomLevel(){
    int getScaleDenominator();
    int getSize(); // number of tiles
    int getNumberOfRows();
    int getNumberOfColumns();
}
```

Extension Interface for Protocol Support

Individual protocol support is handled by TileStrategy objects. Once again this is similar to the WMS implementation.

TileStrategy Extention

We need a strategy object capturing the protocol specific behavior:

```
interface TileStrategy {
    CRS getCoordinateReferenceSystem();
    ReferencedEnvelope getBounds(); //
Bounds include CRS
    GridCoverage getTile( int x, int y, int
zoom );
}
```

The factory has the difficult decision of deciding what URI it can support; factory negotiation determines the protocol used. Addition information may may be required if protocol version negotiation is also needed (as in the WMS case).

```
interface TileStrategyFactory {
    boolean canTile( URL uri );
    TileInfo createInfo( URI uri ); // used
to parse descriptors
    TileService createTileService( URI uri
);
}
```

TileCache Extension

Part of the fun of having TileRange is the ability to interact smoothly with a Cache.

```
interface TileCache {
     TileRange createRange( TileDraw draw,
int row, int col, int rows, int cols );
}
```

TileDraw is used to isolate the TileCache; for reuse by those working with WMS or other renderers.

```
interface TileDraw {
     GridCoverage drawPlaceholder( int row,
int col );
     GridCoverage drawTile( int row, int col
);
}
```

The following implementations are desired:

- PassthroughTileCache - The initial implementaion can be virtual, and just create a standalone TileRange
- MemoryTileCache - uses memory up to a specified capacity
- DiskTileCache - uses diskspace up to a specified capacity

The most common compound request is handled with a special data structure, a progress monitor is used to provide feedback on during the loadTile process, the getTiles method will return a set of the correct size; although some GridCoverages may be transparent (or pixelated) during the loading process.

```
class TileRange {
    ReferencedEnvelope getBounds(); //
bounds of tiles represented, w/ CRS

    load( ProgressMonitor ); // monitor
advances as each tile is available
    boolean isLoaded();
    refresh( ProgressMonitor ); // leaves
tiles as is, but redraws

    Set<GridCoverage> getTiles(); // tiles
in range
}
```