MapThat StreetView and SQL Triggers

by David Crowther

The DynamicMaps webGIS – **MapThat** – has a year by year development roadmap which outlines new tools, functionality and enhancements that we wish to implement to ensure continual improvement in our software. In addition, this roadmap is supported by our client's requirements and in particular ideas they have which would improve their day to day business processes.

This blog details one such client requirement and how over the course of one day I undertook some RnD, ran some google searches and generated a step by step guide for the client to help them improve their web mapping experience.

A – The Requirement:

Many of our clients utilise MapThat webGIS to visualise their assets and perform analysis without having to leave their office. This can be done using a combination of spatial queries and measuring tools. To compliment those tools, they also utilise resources such as **Google StreetView** to visualise those assets in the field, without having to make costly onsite visits.

Currently our users have MapThat open in one application and then use another screen with Google StreetView open and manually search for each new area of interest, which can be time consuming.

One of our clients asked whether it was possible to have a direct link between their assets in MapThat and Google StreetView?

'Sure,…. That's possible,… let me have a think about the best options and I will come back to you.'

B – The Options:

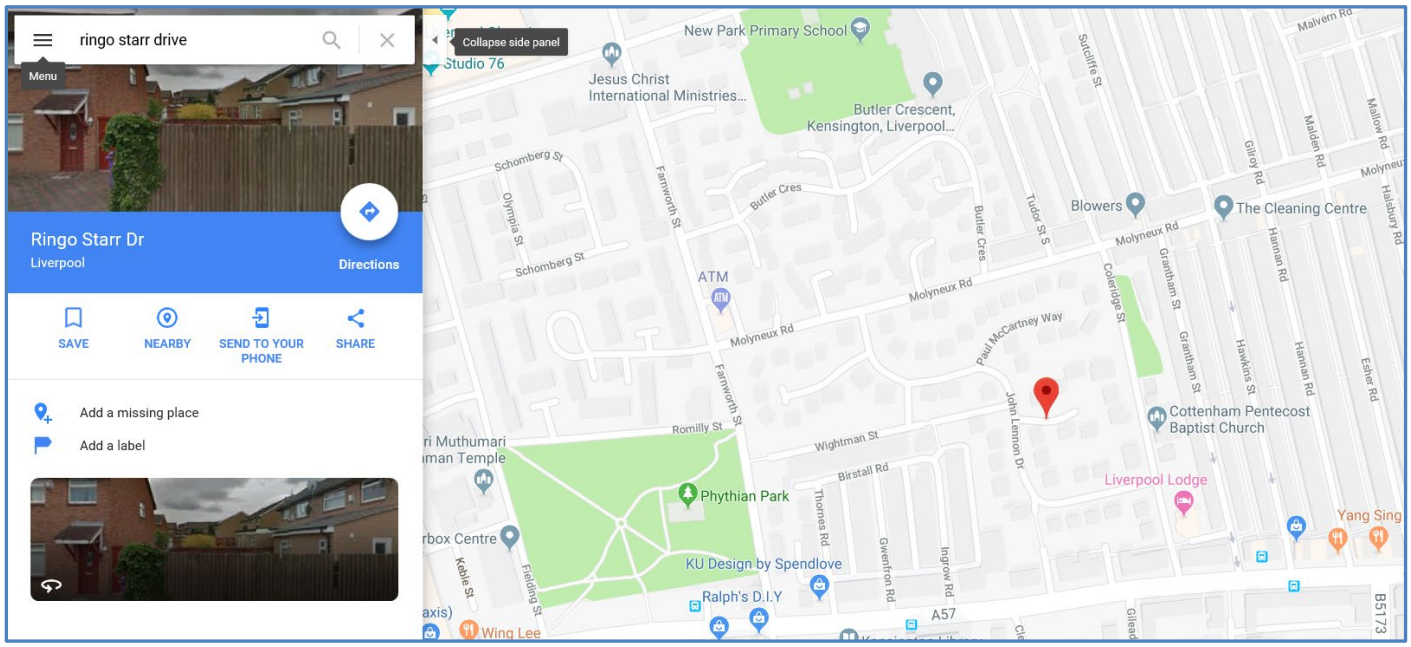
As with most things there are two options; one where we can definitely utilise some existing functionality and create links to Google StreetView using **SQL scripts** and **HTML** and the second which will require some development work and a brand new tool for MapThat.

At the time of writing this blog ,option 2 is being spec'd out by me ready to send to our Development Team. Meanwhile option 1 sounded simple enough and a fun task for me to do some RnD and learn a few new things!

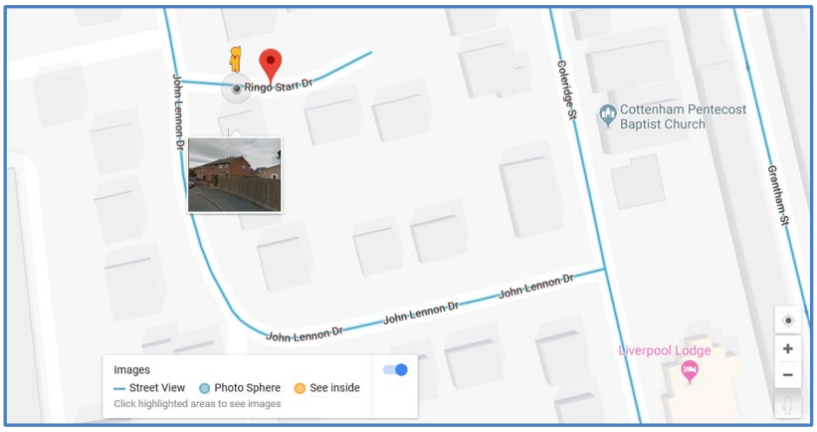
C – Option 1 Research and Development:

'What's a good starting point…..?'

It must be the URL which Google StreetView uses. So I opened Google StreetView and searched for an address in Liverpool - Ringo Starr Drive is my current favourite!



Having found the address in Google Maps I then zoomed into **StreetView** level, dropping the yellow map locator icon onto the map:



And then zooming into Ringo Starr Drive:

I looked at the **URL** being used by Google StreetView and it returned the following:

https://www.google.com/maps/@53.4130709,-2.9515343,3a,75y,127.28h,90t/data=!3m7!1e1!3m5!1sds3FyQazZTbpxyWnbOQ7Ow!2e0!6s%2F%2Fgeo1.ggpht.com%2Fcbk%3Fpanoid%3Dds3FyQazZTbpxyWnbOQ7Ow%26output%3Dthumbnail%26cb_client%3Dmaps_sv.tactile.gps%26thumb%3D2%26w%3D203%26h%3D100%26yaw%3D141.93765%26pitch%3D0%26thumbfov%3D100!7i13312!8i6656

'Wow that's a long string,… there must be something shorter I can use?'

So, my next step was to run a google search and ask: **"build a google street view URL"** and the first search result provided a **Stack Overflow** help page:

https://stackoverflow.com/questions/387942/google-street-view-url

This was a great resource as it provided the following information:

a) **Basic url to display GPS cords location**

http://maps.google.com/maps?q=31.33519,-89.28720

b) **layer= Activates overlays. Current options are "t" traffic, "c" street view. Append (e.g. layer=tc) for simultaneous.**

http://maps.google.com/maps?q=&layer=c

c) **cbll= Latitude,longitude for Street View**

http://maps.google.com/maps?q=&layer=c&cbll=

d) **finally, a full URL**

- Street View/map arrangement, 11=upper half Street View and lower half map, 12=mostly Street View with corner map
- Rotation angle/bearing (in degrees)
- Tilt angle, -90 (straight up) to 90 (straight down)
- Zoom level, 0-2
- Pitch (in degrees) -90 (straight up) to 90 (straight down), default 5

http://maps.google.com/maps?q=&layer=c&cbll=31.335198,-89.287204&cbp=11,0,0,0,0

Further reading suggested that these initial URLS and some parameters have been superseded, although when I have more time I will test these and see if they still work. Instead, it was suggested that the following parameters are currently working:

- q: is ignored, can be skipped For more Google Street View code interpretation
- layer: The parameter must be set to *c* (*t* is no more supported and breaks it)
- cbll: **latitude and longitude (unchanged)**
- cbp: only parameter 2 (rotation angle) and 5 (pitch) are still supported
    1. is ignored, can be 0 or empty string
    2. **Rotation angle/bearing (in degrees)**
    3. is ignored, can be 0 or empty string
    4. is ignored, can be 0 or empty string
    5. **Pitch (in degrees) -90 (straight up) to 90 (straight down)**

And that this would be a good **Base URL** to work from:

http://www.google.com/maps?layer=c&cbll=31.335198,-89.287204

or

http://www.google.com/maps?layer=c&cbll=31.335198,-89.287204&cbp=,30,,,20 which would add a rotation of 30 and pitch of 20.

So, for my trial run I chose the first URL to base my links on and found a sample **Longitude** and **Latitude** value for Liverpool and updated the URL:

http://www.google.com/maps?layer=c&cbll=53.4023,-2.96938&cbp=,30,,,20

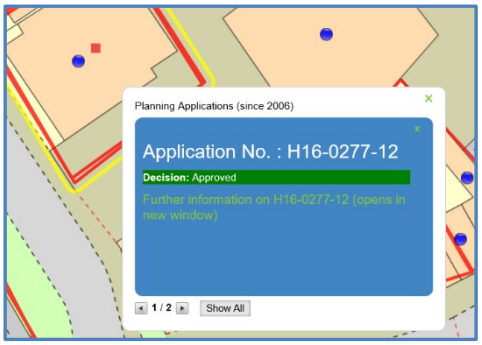
The link worked and opened the correct X and Y location within Google StreetView.

D – How can I link my Assets to Google StreetView?

'Right I have the syntax for a URL how can I link my assets to this URL?'

My next step was to think about the easiest way to link a map feature in MapThat to an external resource. The answer was immediately obvious as we had just updated one of our clients **Planning Application** Layers in MapThat, to link to their Ocella Planning Apps Web Application using the Unique ID of each Planning App.

MapThat uses **Information Bubbles** to expose attributes and those bubbles can be configured using **HTML** to allow clickable links to external web pages, documents, videos, photos etc…



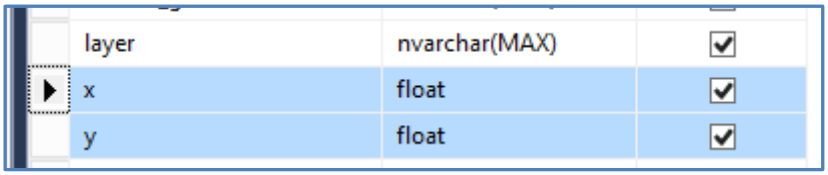
In order to generate this link, we needed a column in the MapThat layer that contained a **URL link** to the correct individual Google StreetView URL. Having now researched the parameters within a StreetView URL, it was obvious that the key values we needed would be the **Longitude** and **Latitude** values for each asset.

I decided to use the Liverpool Streetlights layer in our [TryMapThat] application:

E – Modifying my Spatial Tables

'Ok,. Let's learn some more SQL!'

In order to generate the individual Google StreetView URL, we needed **two new columns** in the Streetlights table called Longitude and Latitude. That's nice and easy, we can simply use the **DESIGN tools** in SQL Server Management Studio and add two new FLOAT fields:



Nice and easy so far, but what is the SQL to extract the relevant Longitude and Latitude values? I like to keep a backup of useful SQL queries and this was one that I already had:

```
Update [dbo].[lighting4326] set x = ogr_geometry.STPointN(1).STX;

Update [dbo].[lighting4326] set y =ogr_geometry.STPointN(1).STY;
```

Having ran my Spatial Update query I had successfully extracted the X and Y coordinate values from my Streetlights table into the X and Y fields:

site_code	feature_gr	layer	x	y
23600210	CLAL	Lighting	-2.96938328898225	53.4022869707307
23600304	CLAL	Lighting	-2.96703926646906	53.4004003367244
23600304	CLAL	Lighting	-2.9668624085082	53.4005635236888
23600304	CLAL	Lighting	-2.96683770269538	53.4007974588261
23600310	CLAL	Lighting	-2.97570448918668	53.4031164042109
23600404	CLAL	Lighting	-2.94364912310759	53.3751578561976
23600404	CLAL	Lighting	-2.94311387169234	53.3754227820401
23600420	CLAL	Lighting	-2.9765263065259	53.3910554636395
23600420	CLAL	Lighting	-2.97639600490511	53.391272292454
23600420	CLAL	Lighting	-2.9761300981612	53.3901687838325

***Note** – In order to ensure that the coordinates extracted were Longitude and Latitude values I had to ensure the projection of my Streetlights table was 4326.*

Also NOTE - *If the layer that you wish to use is not a point layer, but maybe a polygon e.g. planning apps, the SQL script to extract the coordinates of the centre of each polygon is:*

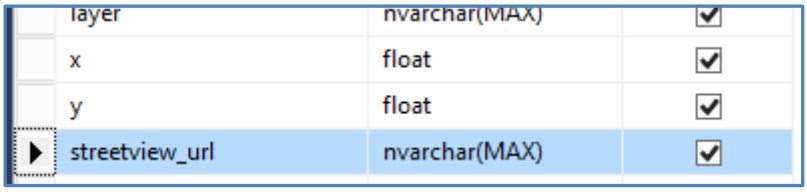
-- Update XCentre

update [dbo].[your_polygon_table] set XCentre = ogr_geometry.STCentroid().STX;

- Update YCentre

update [dbo].[your_polygon_table] set YCentre = ogr_geometry.STCentroid().STY;

Now I had the Longitude and Latitude values in columns X and Y, I could then use those to update a third column to contain the full **Google StreetView URL**. Adding a third field called **streetview_url** using the DESGIN tools:

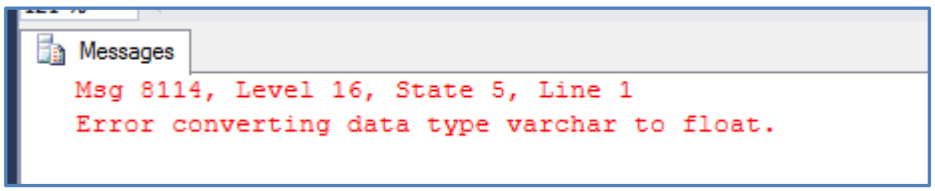


Finally, we needed to run an **Update Statement** to update the new column with the first part of the Google StreetView URL, concatenated with the X and Y fields and then with the final part of the URL.

```
update [dbo].[lighting4326]

set streetview_url = 'http://www.google.com/maps?layer=c&cbll='+[y]+','+[x]+'&cbp=,30,,,20'
```

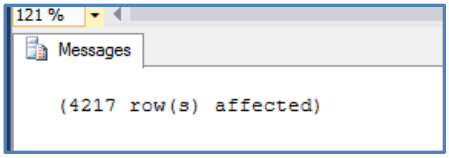
However, the update script failed as it didn't like creating a URL string where the X and Y fields were set as type = FLOAT



To resolve this, I used a **CAST statement** to set the X and Y field as **VARCHAR** as they were passed into the Update Statement:

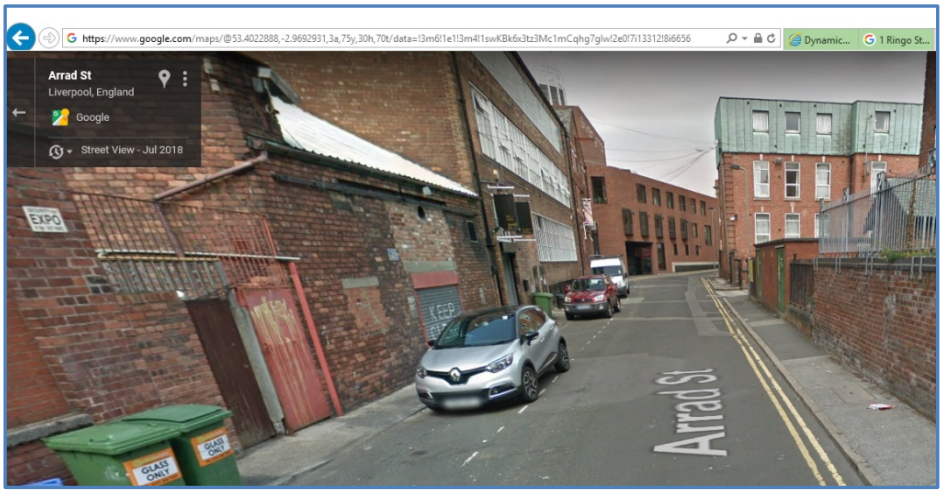
```
update [dbo].[lighting4326]
set streetview_url = 'http://www.google.com/maps?layer=c&cbll='+CAST(y AS VARCHAR(20))+','+CAST(x AS
VARCHAR(20))+'&cbp=,30,,,20'
```

This time the Update Statement was successful and the new URL column was updated:



feature_gr	layer	x	y	streetview_url
CLAL	Lighting	-2.96938328898225	53.4022869707307	http://www.google.com/maps?layer=c&cbll=53.4023,...
CLAL	Lighting	-2.96703926646906	53.4004003367244	http://www.google.com/maps?layer=c&cbll=53.4004,...
CLAL	Lighting	-2.9668624085082	53.4005635236888	http://www.google.com/maps?layer=c&cbll=53.4006,...
CLAL	Lighting	-2.96683770269538	53.4007974588261	http://www.google.com/maps?layer=c&cbll=53.4008,...
CLAL	Lighting	-2.97570448918668	53.4031164042109	http://www.google.com/maps?layer=c&cbll=53.4031,...
CLAL	Lighting	-2.94364912310759	53.3751578561976	http://www.google.com/maps?layer=c&cbll=53.3752,...
CLAL	Lighting	-2.94311387169234	53.3754227820401	http://www.google.com/maps?layer=c&cbll=53.3754,...
CLAL	Lighting	-2.9765263065259	53.3910554636395	http://www.google.com/maps?layer=c&cbll=53.3911,...
CLAL	Lighting	-2.97639600490511	53.391272292454	http://www.google.com/maps?layer=c&cbll=53.3913,...
CLAL	Lighting	-2.9761300981612	53.3901687838325	http://www.google.com/maps?layer=c&cbll=53.3902

I tested the new URL and the Google StreetView location for that record successfully opened.

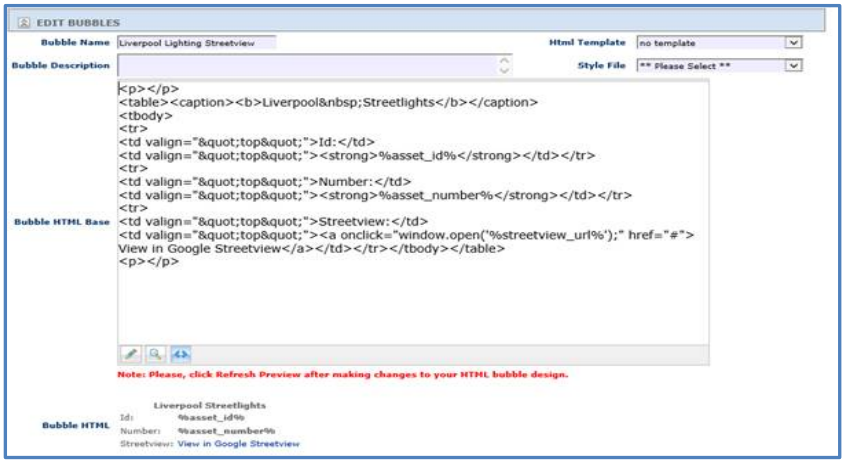


E – Use HTML to create Clickable Links

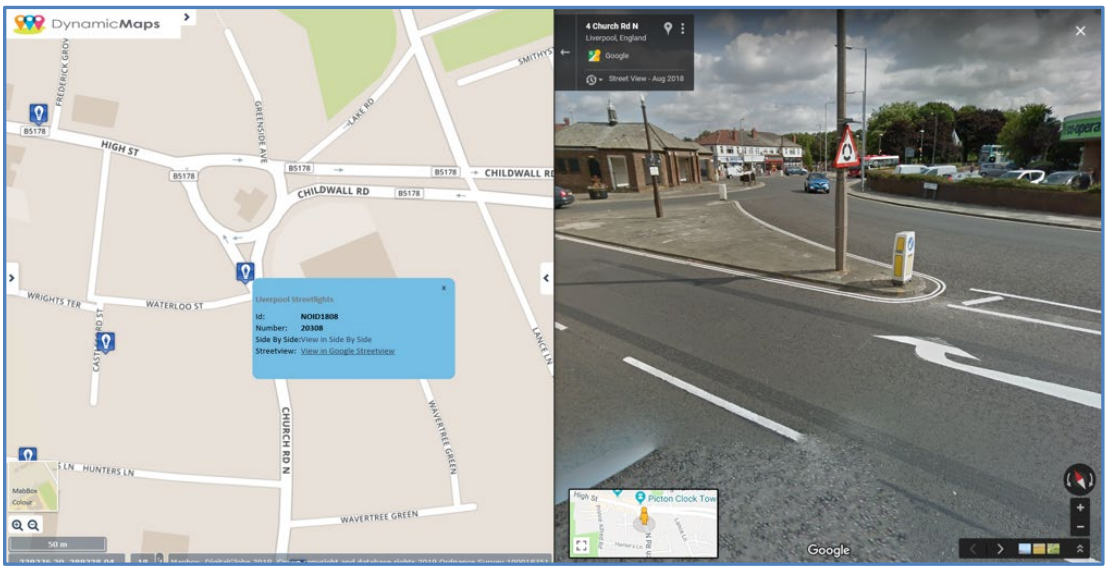
'That's pretty good, now we need a way for the user to activate the link.'

Now that the database table had an individual URL for each Asset, we needed to update the **Information Bubble** in MapThat to allow the user to click on the link and open the StreetView URL. MapThat's Information Bubbles can utilise HTML so that they can be both cosmetically pleasing and highly powerful!

An example of the **HTML text** used to generate the Information Bubble is shown below, where the **Click Open** Window statement is linked to the new Google StreetView field.



Now within MapThat the Streetlights layer had a clickable URL that allowed users to click on the asset and open Google Streetview for that exact location.



G – Create Database Triggers

'I've done it.... oh wait, what happens if a new record is created or an existing one moved?'

So,.. we needed to think about how to handle future data updates, and running the SQL Update Statement manually each day wasn't really an option. After a little more RnD I learnt the basics of generating **Triggers** within a SQL Database Table.

For the Streetlights table I created a **New Trigger** as per the below:

```
USE [DB_NAME]
GO
/****** Object:  Trigger [dbo].[UpdateLATLONG_TRIG]    Script Date: 24-Jan-19 2:58:42 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER TRIGGER [dbo].[UpdateLATLONG_TRIG]
   ON  [dbo].[lighting4326]
   AFTER INSERT,DELETE,UPDATE
AS
BEGIN

       SET NOCOUNT ON;

    -- Insert statements for trigger here
Update [dbo].[lighting4326] set x = ogr_geometry.STPointN(1).STX;
Update [dbo].[lighting4326] set y =ogr_geometry.STPointN(1).STY;


update [dbo].[lighting4326]

set streetview_url = 'http://www.google.com/maps?layer=c&cbll='+CAST(y AS VARCHAR(20))+','+CAST(x AS
VARCHAR(20))+'&cbp=,30,,,20'

END
```

The trigger did the following:

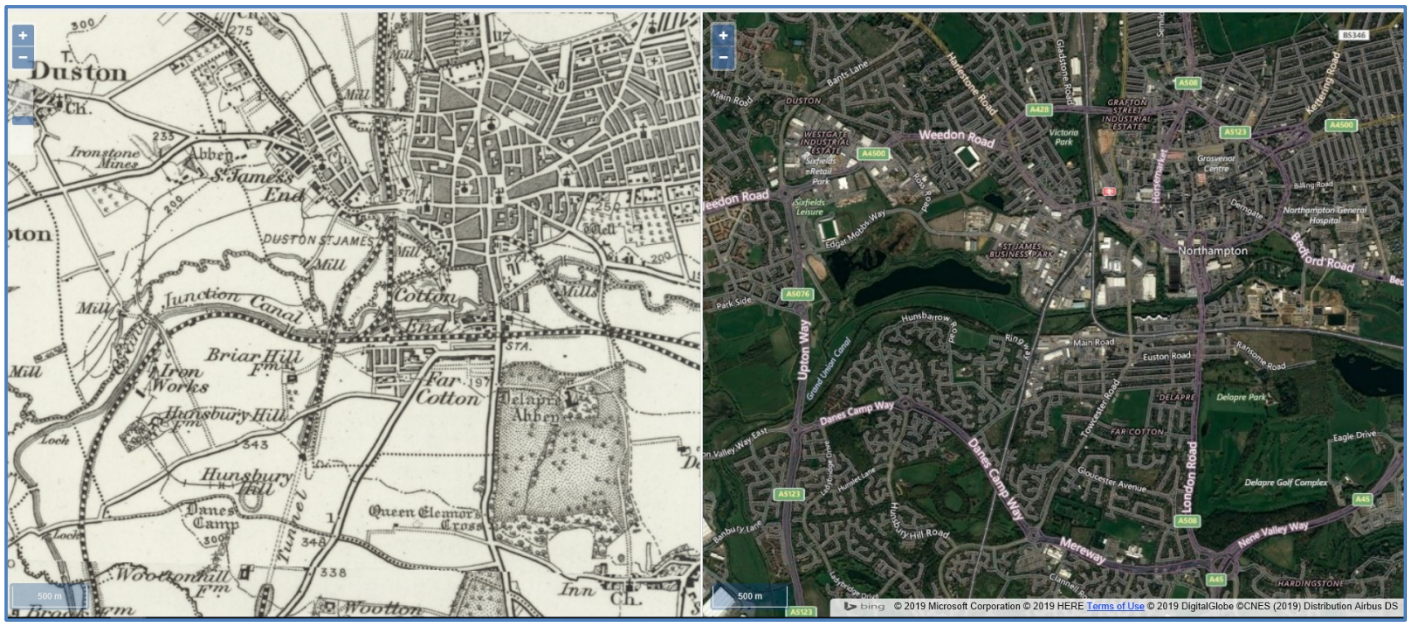
- On an INSERT, DELETE or UPDATE of a record in the Streetlight table
- Run the Update X and Y coordinates
- Then run the Update of the streetview url with the new/updated coordinates

This now meant that if I added a new feature into the table (or moved one) via MapThat or a desktop GIS such as QGIS, that record would automatically have the X and Y and streetview URL generated/updated, so I would retain the integrity of the records.

H – Other External Links

I also remembered another external application that we could link to – **Side By Side** – which provided historic mapping for locations within the UK.

https://maps.nls.uk/



Again by reviewing the URL below it was possible to determine the parameters that could be passed into the URL to open a map for a specific location:

https://maps.nls.uk/geo/explore/side-by-side/#zoom=18&lat=53.4023&lon=-2.96938&layers=1&right=BingHyb

In this case;

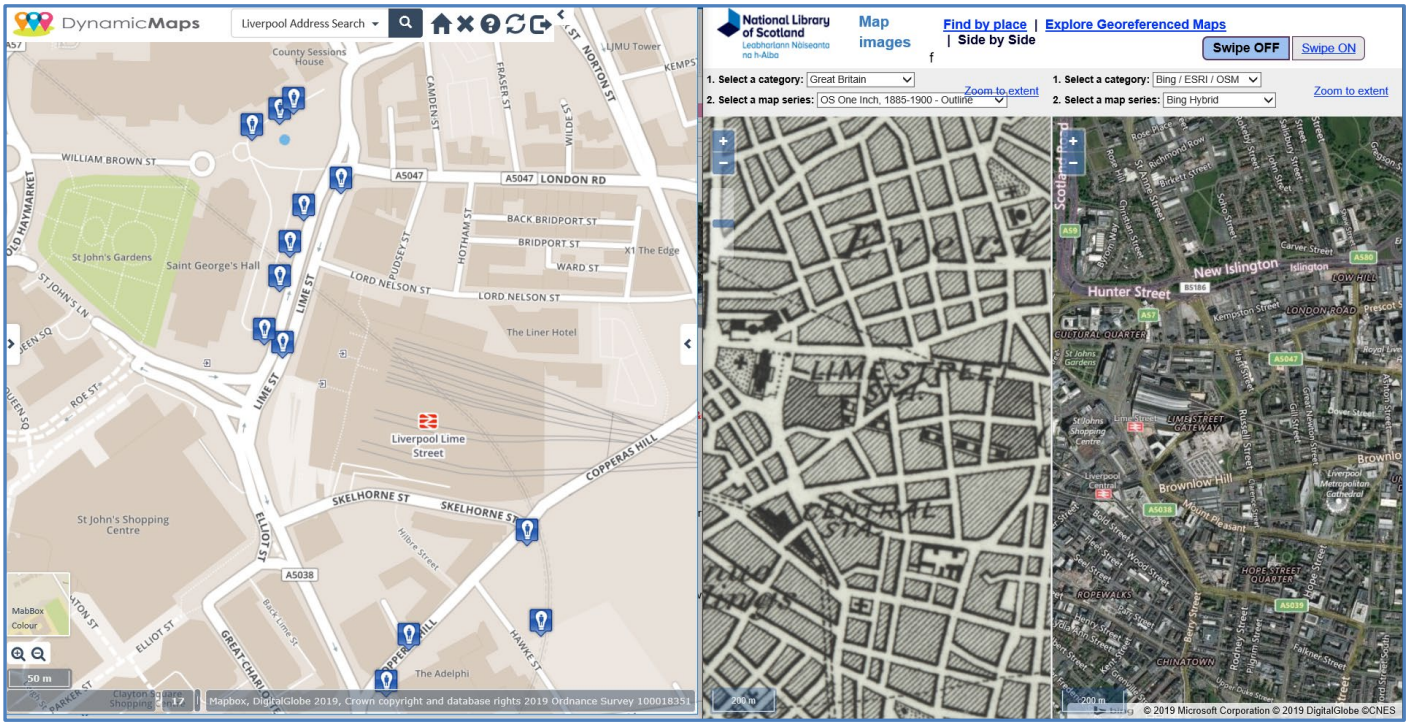
- Zoom Level = 18
- Longitude and Latitude
- The Historic Map Layer to show on the left
  - Where 1 = OS One Inch 1885-1900
  - 2 = OS Six Inch 1888-1913
- The Basemap to show on the right
  - Where BingHyb = Bing Hybrid
  - BingRoad = Bing Road

Using the Historic Map Layer (1) and Bing Hybrid as the mapping I built up a new UPDATE Statement to update my **SidebySide** field:

```
update [dbo].[lighting4326]

set sidebyside = 'https://maps.nls.uk/geo/explore/side-by-
side/#zoom=18&lat='+[y_var]+'&lon='+[x_var]+'&layers=1&right=BingHyb'
```

Then after updating my HTML Information Bubble and SQL Trigger I now had a new clickable Link from my assets in MapThat to historic mapping layers for the same location.



So that was a day well spent!..

I learnt a few new tips and tricks in SQL and understood the parameters involved in calling both Google StreetView and Side By Side mapping. Next onto designing how we will implement this as a new tool within our webGIS MapThat!