App Inventor tutorial (V2)

Mapping with Web services, staticmaps and streetview APIs

using web viewer, canvas, start activity and fusion tables components

Content

1 Lesson plan

1.1 What you will learn :

- What is a Web API (Application Programming Interface)
- What is a URL (Uniform Resource Locator) and how to build it
- Mapping resources on the web or in the cloud :

- Mapping APIs
  - Interactive Web maps (open street map, Google maps, …)
  - Staticmaps and streetview APIs,
- Web Databases with mapping capabilities
  - fusion tables

- Display interactive maps with the Webviewer and a map URL,
- Display map with the Start Activity Component, using a secondary mapping App,
- Display map in a WebViewer, and fusion tables with Web data management,
- Display static maps with the Canvas component, and build interactions from you App,
- Display streetview and visit buildings,
- discuss pros and cons for each solution,
- Optional : Review a few “good practices” (see appendix)

1.2 What you will do :

1. Introduction : Review mapping solutions with App Inventor
2. Review concepts that will be used (URL, URL encoding)
3. Coding exercises
   - Sol. 1 : display a dynamic map with a Webviewer component and URL,
   - Sol. 2 : display map with the StartActivity component and a second mapping App,
   - Sol. 3 : display map with Webviewer and fusion tables with web data sharing,
   - Sol. 4 : display static maps with a Canvas component and interact from your App,
   - Sharable utilities : Key service functions when mapping
   - Sol. 4a : display a list of markers on the map
   - Sol. 4b : extend interaction levels with the canvas component
   - Sol. 4d : extend to streetview display and visit buildings

2 Introduction : Mapping solutions with App Inventor

App Inventor lets us go mobile AND enables to build “location aware” Apps which can display and interact with our spatial environment : build games like Pokemon Go or services like pizza delivery management or manage a fleet of vehicles like Uber.

These apps will generally require 3 kinds of services :

1. Mapping (with Open streetMap, google Maps or other services),

2. Geolocation, indoors or outdoors with GPS, network, SSID, IP, …
3. Web sharing of structured databases (customers, orders, drivers, pokemons, …)

This tutorial, deals with the first subject : mapping (we look at other ones in other tutorials). You will find a draft tutorial on Geolocation at http://www.letustry.org/AppInventor/mapping/GPSTutorial/ (and tutorial on web data sharing with fusion tables should popup sometime).

-0-

App Inventor offers many different ways to do mapping with very different capabilities. We will consider 4 of them :

1. Mapping with a web viewer component and Open street Map or Google maps services,
2. Mapping with the "Start Activity" component and the Google maps App,
3. Mapping with a Web viewer component, fusion tables and web data management,
4. Mapping with a Canvas component with static maps and streetview APIs.

These four techniques enable VERY different App capabilities :

1. Mapping with a Webviewer and WebMap Services is probably the fastest and most simple. It provides good Map display with dynamic web interaction, but has poor capabilities to incorporate user's data and does not allow feedback from the map back to the App.
2. Mapping with the Start Activity has similar capabilities and limitations but offers a nicer set of display options with a well-documented Application Programming Interface. Your App disappears once the map display activity starts (and comes back when you quit display).

These solutions are fine if your requirement is to use map display for interactive output only (directions, locations, …) with limited user data. This is the case for many Apps, and you will find very nice and popular Apps in the gallery, such as "find my car" which use these techniques.

We will have a closer look at solutions 3 and 4 which enable two-way interactions between maps and App data. These interactions are key for games where there is some "action" in the map (like Pokemons) or for business App that interact with fleets of vehicles (like UBER or pizza deliveries).

3. Mapping with fusion tables and a webviewer : The choice here is to use cloud services (fusion tables) to manage data AND maps. Fusion tables enable management and Cloud sharing of data for multiple players. This goes with powerful query capabilities (SQL like), which are necessary to handle interconnected data. Fusion tables also have mapping capability. This solution is fit to run small databases (ex : customers, orders, fleet of vehicles, deliveries, …).

Note here that Mapping AND data management are subcontracted to cloud services. They will still be handled by YOU, but outside App Inventor.

4. Mapping with a Canvas and static APIs : the choice here is to keep full control inside the App, and manage all interactions with the canvas component. In this case, maps are handled as image files. Interactions are unlimited but you have to write code for each interaction : this powerful solution comes naked. We will provide reusable code to help (ex : translate pixel coordinates to geographic coordinates, measure distances in km, zoom, pan, etc.) but there is still more work to do than with solutions 1 and 2. This solution is fit for Pokemon like games, or when you need two-way interactions between App and map.

Solution 4 has very nice extensions such as streetview, for walking in the streets and inside buildings. This can be great for “treasure finding” games (but requires some good maths).

3 Review concepts used : web API, URLs, …

If you know about APIs, URLs and associated practices and rules, you may skip this chapter. Otherwise, run through it a first time to be aware of what is inside, but do not spend too much time. You will come back to it later as you need.

3.1 What is a Web API (Application Programming Interface) ?

"Web API" historically has been virtually synonymous for “web service”. It describes the functionalities that a web service makes available and how: routines, protocols, inputs, outputs, … APIs are one of the most common ways technology companies integrate with each other.

In the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages[1].

Note : HTTP requests frequently return html web pages. They can be displayed with a web viewer component. Staticmaps and streetview APIs only return images. In this case, they can also be displayed on a Canvas component as image files.

The request message which is sent on the Web takes the form of a URL (Uniform Resource Locator) described below. (see more info in appendix)

3.2 URL (Uniform Resource Locator)

3.2.1 What is a URL and how to build it ?

A URL, or commonly a web address, is a reference to a web resource that specifies its location on the net and a mechanism to retrieve it. URLs most commonly reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and other applications.

For example URLs used for static maps use http and contain :

- a scheme (http),
- a host (maps.googleapis.com),

[1] which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format

- a path (/maps/api/staticmap)
- and a query :
  (ex : center=37.7,-122.4&size=320x320&zoom=14&maptype=roadmap),

https://maps.googleapis.com/maps/api/staticmap?center=37.77,-122.45&size=320x320&zoom=14&maptype=roadmap
scheme,//, host , path , query

3.2.2 Query string in a URL

The query string is the part of a URL, which carries the specification of what is requested as the returned result. It comes after the host and path and generally follows the question mark "?" and is encoded as field/value pairs separated by & delimiters :

field1=value1&field2=value2&field3=value3...

The query fields and values are defined by each service provider in its API. For example in the following URL (which you can copy/paste into your web browser address), the query string is what follows the question mark. The delimiter between attribute-value pairs is '&'

https://maps.googleapis.com/maps/api/staticmap?center=37.777,-122.451&size=320x320&zoom=14&maptype=roadmap

In this example the field named "*center*" has the value "*37.777,-122.451*" and the field named "*size*" has the value "*320x320*".

When a field has multiple values, another delimiter is often used. For example, the '|' character separates coordinate values in the static maps API, as in the following list of marker coordinates :

```
&markers=color:blue|label:x|37.78,-122.46|37.77,-122.46|37.78,-122.45
```

3.2.3 Other query formats

We will meet other query formats where parameters do not follow a question mark. This is the case for open street maps or google maps :

http://www.openstreetmap.org/#map=17/48.83577/2.30503&layers=H

https://www.google.fr/maps/@37.7760491,-122.4488144,16.9z

But the URL organization remains similar and you should have no difficulty to handle them.

3.2.4 URL encoding

Some characters have a special meaning in a URL or in html they are "reserved characters". For example, the *space* character is a separator for HTTP messages, so that we cannot use it within query parameters such as someone's address.

Such characters are forbidden or "reserved" characters that must be replaced when they are used in the query by % and their hexadecimal value (see tables below). Replacement is mandatory with "reserved characters" and recommended for other characters which are neither "reserved" or "Non reserved". The space character (which is the most frequent) may also be replaced by '+'.

Values for **reserved characters** after percent-encoding – mandatory

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Values for other (**neither reserved or non reserved**) – recommended but non mandatory

Newline	space	"	%	-	.	<	>	\	^	_	`	{	|	}	~
%0A *or* %0D *or* %0D%0A	%20	%22	%25	%2D	%2E	%3C	%3E	%5C	%5E	%5F	%60	%7B	%7C	%7D	%7E

(see more info in appendix)

For example,

&markers=color:blue|label:x|37.78,-122.46|37.77,-122.46|37.78,-122.45

Should be replaced by

&markers=color:blue%7C label:x%7C 37.78,-122.46%7C 37.77,-122.46%7C 37.78,-122.45

and

“University of San Francisco” should be replaced by “University%20of%20San%20Francisco”
or by “University+of+San+Francisco”

4 App inventor components and Mapping APIs

- the “webViewer” component can display web pages in general, like a browser : http://ai2.appinventor.mit.edu/reference/components/userinterface.html#WebViewer

The webviewer component can be associated with several APIs. We will illustrates it’s use in 2 solutions :

- dynamic maps from Web services (solution 1)
- with fusion tables (solution 3)

- the “canvas” component can only display image files, but if a URL returns a file, then this URL can be used exactly like a local image file name ☺ !

The canvas component can be associated with several APIs, and we will illustrate its use with the google Maps and streetview static API (solution 4)

- The Activity Starter component can start a new App or activity such as Google Maps, and tell it what to do with a DataURI. This API is the Intents API.

We will illustrate its use with the google Maps App (solution 2)

Besides showing what a secondary App can do, we will show examples with the web viewer and the canvas to understand what each one can do and how they differ :

- the webviewer component is very general, but the App loses control on window events. What is nice is that you have the web server interactivity,
- The backdoor solution to recover control with a web viewer is to manage data and mapping on the cloud side : this is what App Inventor can do with fusion tables do,
- the canvas component has no native functions for mapping, but it keeps control on window events (click,drag, …), so that you can decide to do whatever you want on each event (but you have to do the coding work)

We may also mention, the “Web” component which has blocks to query and process web data. We will not use it, but if you wish, look at the tutorial here
http://appinventor.mit.edu/explore/ai2/stockquotes.html

5 Coding examples /exercises

We will see 4 different ways to do mapping. They are independent and you can go directly to the method that seems to fit your needs.

5.1 Solution 1 : Display maps with a web viewer and web map servers

5.1.1 "dynamic" maps APIs : Open street map, Google maps

You will find several map servers on the web including open street map, USGS, IGN or Google maps. They return dynamic web pages (i.e. not image files) and cannot be used with the canvas component. They need a web viewer.

A simple (and fun) thing to do is to test them on your web browser. Go to the Open Street Map or Google maps server and look at the URLs to see how they are built. Start to play with location values (longitude and latitude), map scale, theme. Then copy/paste a sample URL in your App Inventor program, write blocks to change URL content and set it as the Webviewer's URL, for example :

- Open street map URL, where you easily see zoom, latitude, longitude and layers

http://www.openstreetmap.org/#map=17/48.83577/2.30503&layers=H

http://www.openstreetmap.org/#map=16/37.7761/-122.4510&layers=HD

- Or the URL displayed with Google map where you can easily see how to write a request with, lat, long and zoom :

https://www.google.fr/maps/@37.7760491,-122.4488144,16.9z

(note :

make sure that you set the width and height of your webviewer component to a specific value or fill parent - NOT automatic - otherwise it will not display correctly. "Automatic" width and height work with images but not with dynamic pages in general.
Also uncheck the scrollable property on your Scrren1 component).

5.1.2 Coding exercise with dynamic maps and webviewer

Goal : Add a Webviewer component and fill in the display procedure. Test display and interaction with map. Check that you can build the URL with blocks and data from your App.

1. Go to Open Street Maps (https://www.openstreetmap.org) zoom around your area of interest and look at the URL adresses to identify the zoom, latitude and longitude in URL.

https://www.openstreetmap.org/#map=18/48.85302/2.34973&layers=C

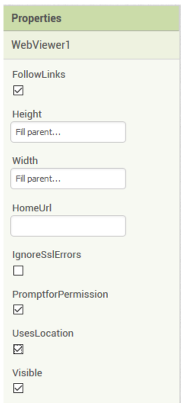
Change zoom, latitude and longitude values and check what works.

2. Try the same with Google maps, look at the URL addresses as you move around or select options

https://www.google.com/maps/
https://www.google.com/maps/@48.8589163,2.3277139,13z
https://www.google.com/maps/@48.8531755,2.3498957,18z/data=!3m1!1e3
https://www.google.com/maps/place/eiffel+tower
https://www.google.com/maps/dir/Eiffel+tower/Notre+Dame+de+Paris/

Google adds more information to your URL when you send it, but the short version works and is easier to code. When editing URLs, Remember about reserved characters, replace spaces by +

3. Open App Inventor and create a new project.

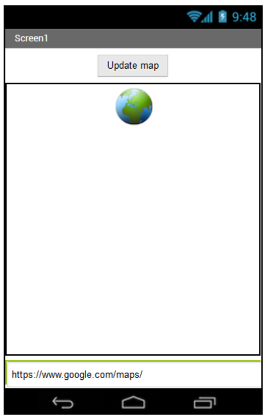

Properties
WebViewer1
FollowLinks
☑
Height
Fill parent...
Width
Fill parent...
HomeUrl

IgnoreSslErrors
☐
PromptforPermission
☑
UsesLocation
☑
Visible
☑

Add a Webviewer component called **WebviewerMap.** Check **Fillparent** for width and height and Check the **uses location** property. Set Home URL to the first address above : https://www.google.com/maps/ or other preferred value.

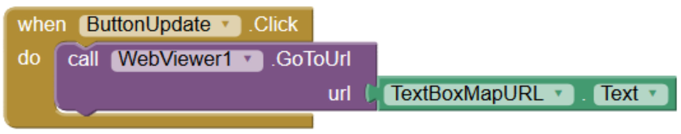
Note : Google's basic URL with no location will use your location as default if your smartphone has location enabled and if your browser allows location sharing (disable it if you worry about privacy).

(Uncheck the Scrollable property of the Screen1 component, and check the responsive sizing property for maps with higher resolution).



Add a **ButtonUpdate** button to update map and a text box **TextBoxMapURL** to display the Map URL. Check the multiline property and set text to another URL such as : https://www.google.com/maps/@48.8589163,2.3277139,13z)

On the blocks side, set the URL to the Textbox text in the button click handler.



```
when ButtonUpdate .Click
do  call WebViewer1 .GoToUrl
                         url  TextBoxMapURL . Text
```

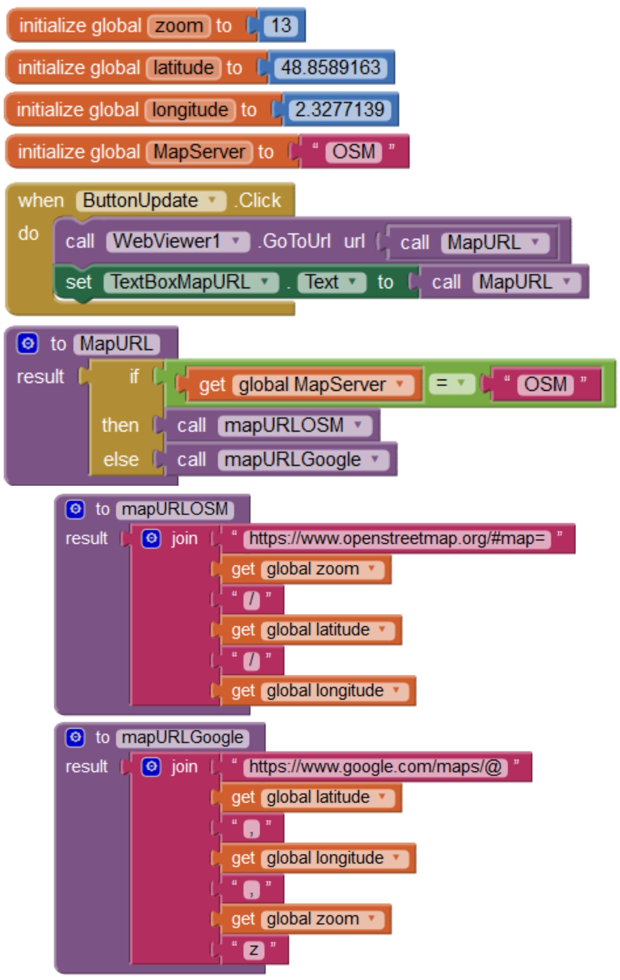
Run your app. Check interaction with the web window (ex : zoom-unzoom with 2 fingers, switch to satellite or traffic maps, …). Click update. Edit zoom factor and/or latitude and longitude in the text box, … You have control of the web window from your App and within the window itself.

4. Now re-Build the URL with program blocks what you did by hand :

You can build the URL from latitude longitude, zoom and other variables from your App (for example a list of places), then update the Webviewer URL, which will update display. Building the URL will be different according to the Web server you choose.

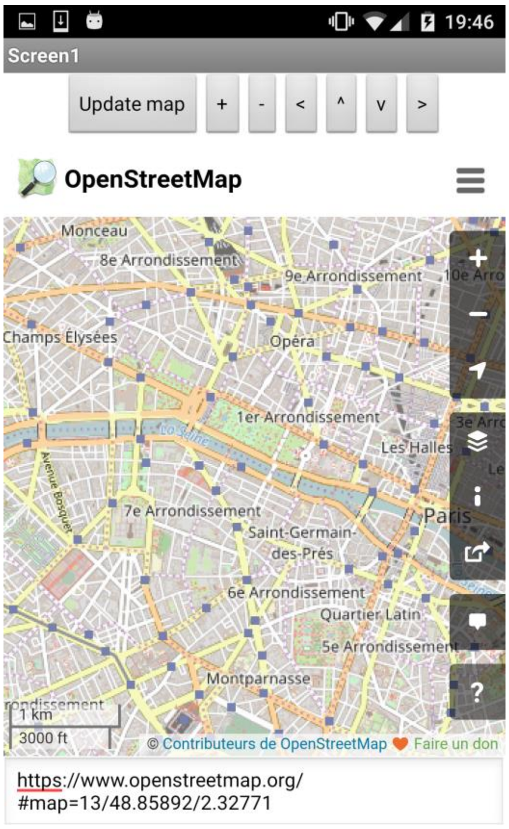
The example below shows how to use either to Open street Map or Google according to the MapServer variable. The Web Viewer URL is returned by the MapURL procedure which calls **mapURLOSM** for open street map and **MapURLGoogle** for Google map server.

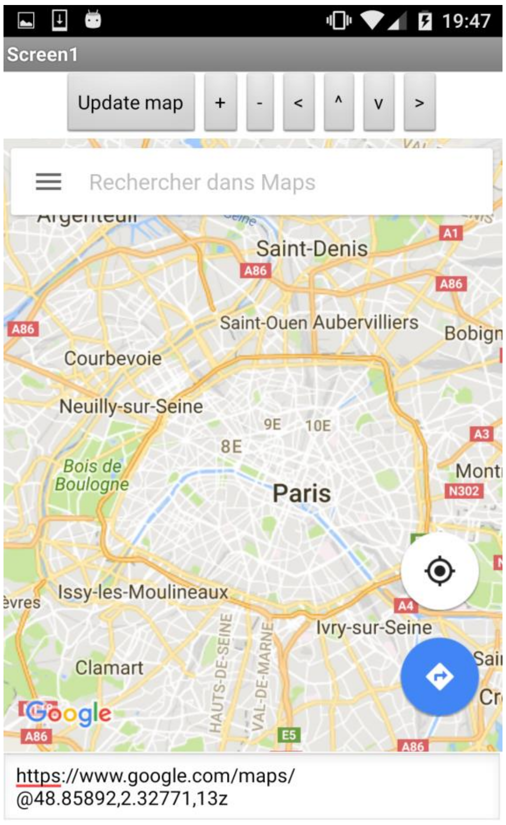
Replace previous blocks by the following :



```
initialize global zoom to 13
initialize global latitude to 48.8589163
initialize global longitude to 2.3277139
initialize global MapServer to " OSM "

when ButtonUpdate .Click
do  call WebViewer1 .GoToUrl url  call MapURL
    set TextBoxMapURL . Text to  call MapURL

to MapURL
result  if    get global MapServer = " OSM "
        then  call mapURLOSM
        else  call mapURLGoogle

to mapURLOSM
result  join  " https://www.openstreetmap.org/#map= "
              get global zoom
              " / "
              get global latitude
              " / "
              get global longitude

to mapURLGoogle
result  join  " https://www.google.com/maps/@ "
              get global latitude
              " , "
              get global longitude
              " , "
              get global zoom
              " z "
```

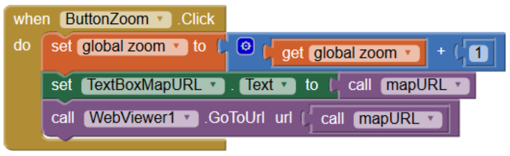
Which will display (on update map) :



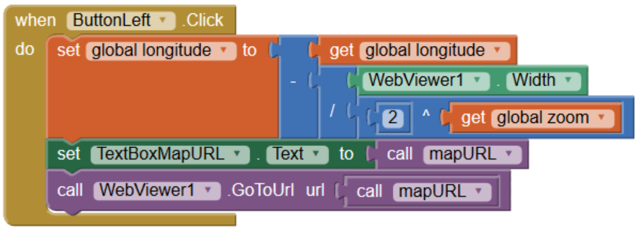


Note that zoom factors are not the same for all servers.

If you wish, you can then add buttons to zoom/unzoom or pan left/right/up/down from your app :



```
when ButtonZoom .Click
do  set global zoom to  get global zoom + 1
    set TextBoxMapURL . Text to  call mapURL
    call WebViewer1 .GoToUrl url  call mapURL
```



```
when ButtonLeft .Click
do  set global longitude to  get global longitude - WebViewer1 . Width / 2 ^ get global zoom
    set TextBoxMapURL . Text to  call mapURL
    call WebViewer1 .GoToUrl url  call mapURL
```

But it is slow and much less practical than interacting with the Web window directly.
(We will come back to this type of interactions when using the canvas component).

5.2 Solution 2 : Display map by starting an other activity (or other App)

5.2.1 Intent APIs and interface to other Apps on your smartphone

With this solution, your App delegates mapping to another App that must be installed on your phone (here the Google maps App). The general idea of an “Intent” request is that an android App throws an intent (i.e. something it wishes to do) to the Android system and waits for some Apps to answer ”I can do it”. If there is only one App, it will start, otherwise it will let you choose between candidates do the job. That’s funny, Android Apps can behave like students in class when one calls for help, and others who know how propose to help !

In our case, we will ask for a VIEWing action (*android.intent.action.VIEW).* This with the data URI may be enough for Android to find the good App (as it is the case on my Nexus) but besides action and data URI it is better to say more precisely what service the App wants, or be more explicit.
With Activity Starter, you can indicate 3 infos : Action (*android.intent.action.VIEW),* Class (*android.maps.MapsActivity*) and Package (*com.google.android.apps.maps*). Activity action, class and package will help Android identify the App that will answer the request. Then your requesting App will pass the "Data URI" to the answering App to tell it what to do.

This Data URI must match the API of the answering App. In our case with Google maps, the API and data URIs are described here. They may be like : `geo:0,0?q=`*`latitude,longitude`*`(`*`label`*`)`

We will see details of this API when we use it below.

If you want to know more you will find :

- a nice video tutorial by Ralph Morelli within the mobile CSP course (see lesson 10 in unit 3) with this Youtube video to show the App and this Youtube video to show how to build it,
- a nice App on the playstore which may help you practice and find solutions with App Inventor is the App Inventor ActivityStarter App. It may help you fill in parameters for the Activity starter component (even if you do not completely understand what you are really doing).

5.2.2 Coding exercise with Intent APIs and the ActivityStarter component

If you have Apps like Google maps installed on you smartphone, you can start it from your app with an **ActivityStarter** component (see in the Connectivity drawer). There are 2 main differences with previous solution :

1. your own App disappears while the new activity or App is running. This means that your App has completely lost control and can no longer interact until you close map display,
2. but you have more display options and there is a good documentation of API available at https://developers.google.com/maps/documentation/android-api/intents#display_a_map.

Create a new project,

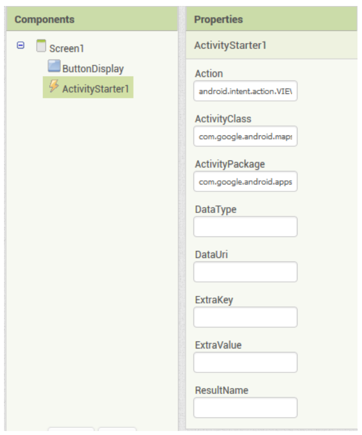
on the design side :

Add a display button and set its text displayMap.

Add a StartActivity component with the following setup :

Action :	android.intent.action.VIEW
Activity Class :	android.maps.MapsActivity
Activity Package :	com.google.android.apps.maps

This will enable Android to find the application that matches these criteria (e.g. Google maps that has to be on your phone).



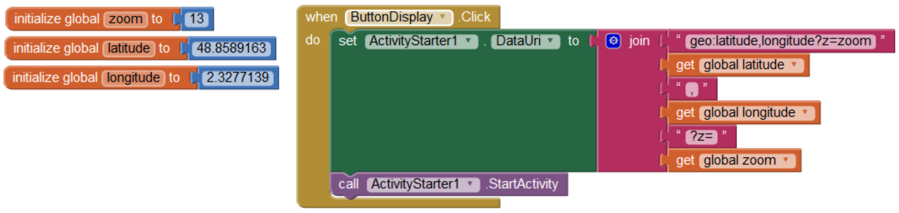
ActivityStarter1

Action: android.intent.action.VIE
ActivityClass: com.google.android.maps
ActivityPackage: com.google.android.apps
DataType:
DataUri:
ExtraKey:
ExtraValue:
ResultName:

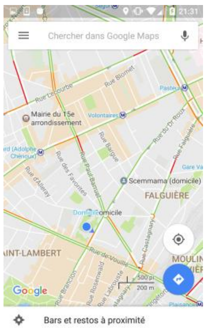
On the blocks side :

Initialize latitude, longitude and zoom global variables with default values, and add a query variable. Then we build the data URI, according to the first case shown in the documentation.

```
geo:latitude,longitude?z=zoom
```



```
initialize global zoom to 13
initialize global latitude to 48.8589163
initialize global longitude to 2.3277139

when ButtonDisplay .Click
do  set ActivityStarter1 . DataUri to  join  " geo:latitude,longitude?z=zoom "
                                             get global latitude
                                             " , "
                                             get global longitude
                                             " ?z= "
                                             get global zoom
    call ActivityStarter1 .StartActivity
```


Chercher dans Google Maps
Mairie du 15e arrondissement
Scemmama (domicile)
FALGUIÈRE
Domicile
SAINT-LAMBERT
Google
Bars et restos à proximité

Clicking on the button will display the map on the right :

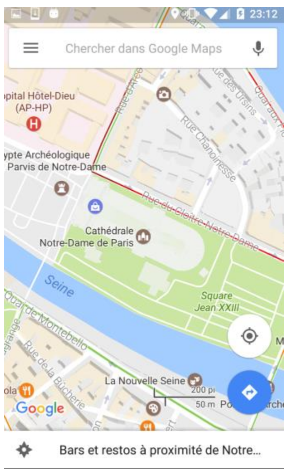
The google maps Android-Intent API offers many other options (see documentation) :

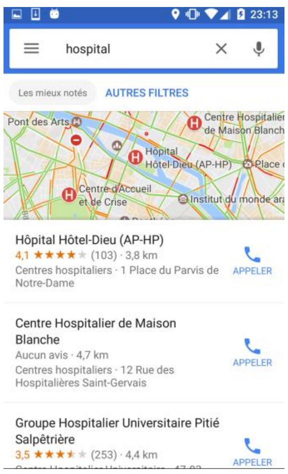
1. `geo:latitude,longitude?z=zoom`	basic display
2. `geo:`*`latitude,longitude`*`?q=`*`query`*	to add search for restaurant, hospital
3. `geo:0,0?q=`*`my+street+address`*	to find by address
4. `geo:0,0?q=`*`latitude,longitude`*`(`*`label`*`)`	to add a label
5. `google.navigation:q=`*`a+street+address`*	to find directions with different travel modes
6. `google.navigation:q=`*`latitude,longitude`* `&mode=d or w or b`	for travel option : drive, walk ,bike

and for streetview :

7. `google.streetview:cbll=`*`latitude,longitude`*`&cbp=`*`0,bearing,0,zoom,tilt`*
8. `google.streetview:panoid=`*`id`*`&cbp=`*`0,bearing,0,zoom,tilt`*

Cases 1 to 7 are illustrated below them with their dataURIs.


23:12
Chercher dans Google Maps
Hôpital Hôtel-Dieu (AP-HP)
Crypte Archéologique Parvis de Notre-Dame
Cathédrale Notre-Dame de Paris
Seine
Square Jean XXIII
La Nouvelle Seine
Google
Bars et restos à proximité de Notre...


23:13

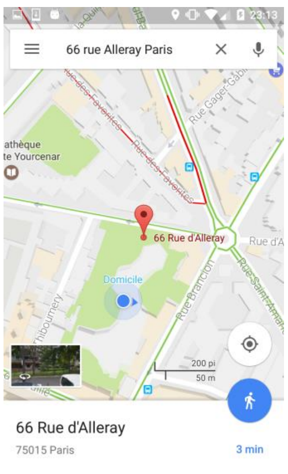
hospital

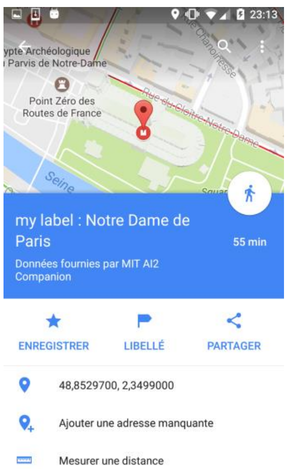
Les mieux notés   AUTRES FILTRES

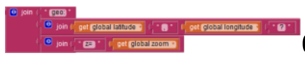
Hôpital Hôtel-Dieu (AP-HP)
4,1 ★★★★★ (103) · 3,8 km
Centres hospitaliers · 1 Place du Parvis de Notre-Dame
APPELER

Centre Hospitalier de Maison Blanche
Aucun avis · 4,7 km
Centres hospitaliers · 12 Rue des Hospitalières Saint-Gervais
APPELER

Groupe Hospitalier Universitaire Pitié Salpêtrière
3,5 ★★★★★ (253) · 4,4 km
APPELER

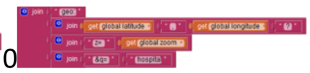



23:13
Crypte Archéologique Parvis de Notre-Dame
Point Zéro des Routes de France
Seine
my label : Notre Dame de Paris
55 min
Données fournies par MIT AI2 Companion
ENREGISTRER
LIBELLÉ
PARTAGER
48,8529700, 2,3499000
Ajouter une adresse manquante
Mesurer une distance

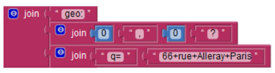


```
join  " geo: "
      join  get global latitude  " , "  get global longitude  " ? "
      join  " z= "  get global zoom
```

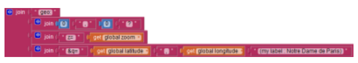
0



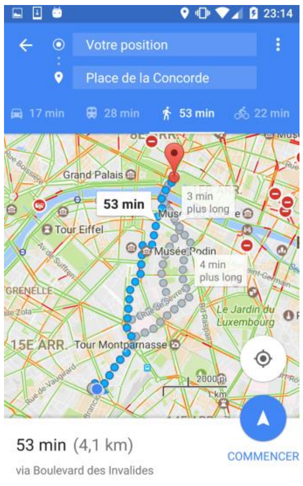
```
join  " geo: "
      join  get global latitude  " , "  get global longitude  " ? "
      join  " z= "  get global zoom
      join  " &q= "  " hospital "
```

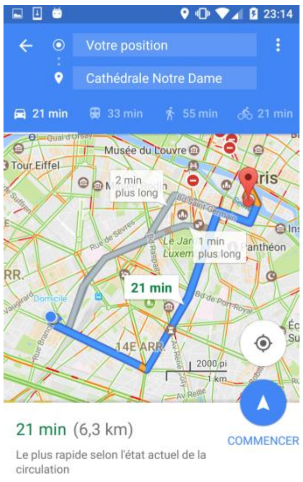


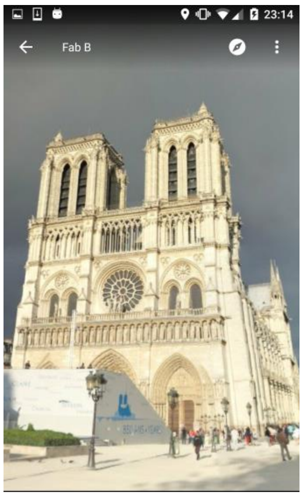
```
join  " geo: "
      join  0  " , "  0  " ? "
      join  " q= "  " 66+rue+Alleray+Paris "
```

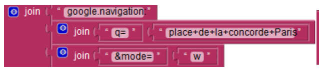


```
join  " geo: "
      join  0  " , "  0  " ? "
      join  " z= "  get global zoom
      join  " &q= "  get global latitude  " , "  get global longitude  " (my label : Notre Dame de Paris) "
```

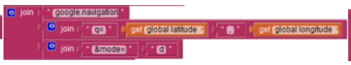




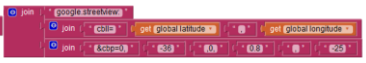




```
join ( "google.navigation:"
       join ( "q=" , "place+de+la+concorde+Paris" )
       join ( "&mode=" , "w" )
```

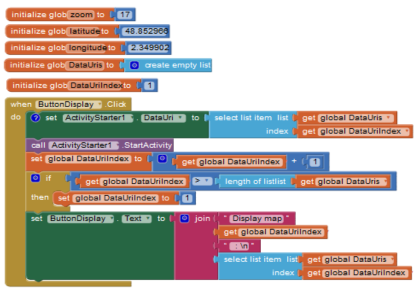


```
join ( "google.navigation:"
       join ( "q=" , get global latitude , "," , get global longitude )
       join ( "&mode=" , "d" )
```

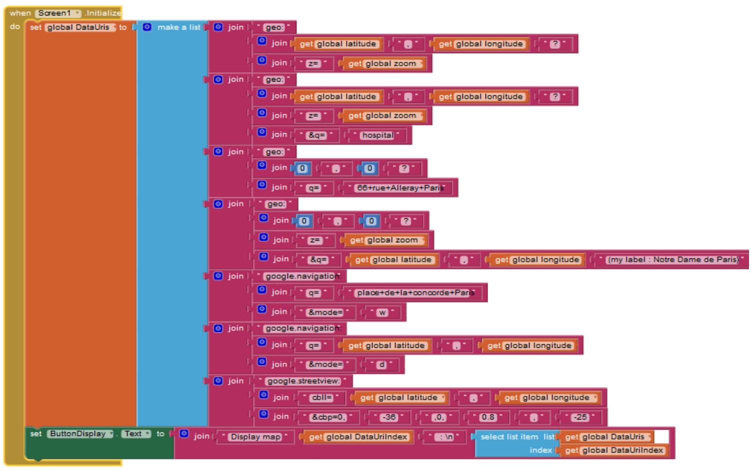


```
join ( "google.streetview:"
       join ( "cbll=" , get global latitude , "," , get global longitude )
       join ( "&cbp=0," , "-36" , ",0," , "0.8" , "," , "-25" )
```

The code - shown below -uses a list of 7 dataURIs. Display switches from one to the next each time the user clicks on the display button.



```
initialize glob zoom to 17
initialize glob latitude to 48.852966
initialize glob longitude to 2.349902
initialize glob DataUris to create empty list
initialize glob DataUriIndex to 1

when ButtonDisplay .Click
do  set ActivityStarter1 . DataUri to  select list item  list  get global DataUris
                                                          index get global DataUriIndex
    call ActivityStarter1 .StartActivity
    set global DataUriIndex to ( get global DataUriIndex + 1 )
    if   get global DataUriIndex > length of listlist get global DataUris
    then set global DataUriIndex to 1
    set ButtonDisplay . Text to join ( "Display map"
                                       get global DataUriIndex
                                       " : \n"
                                       select list item  list  get global DataUris
                                                         index get global DataUriIndex )
```



```
when Screen1 .Initialize
do  set global DataUris to make a list
        join ( "geo:"
               join ( get global latitude , "," , get global longitude , "?" )
               join ( "z=" , get global zoom ) )
        join ( "geo:"
               join ( get global latitude , "," , get global longitude , "?" )
               join ( "z=" , get global zoom )
               join ( "&q=" , "hospital" ) )
        join ( "geo:"
               join ( 0 , "," , 0 , "?" )
               join ( "q=" , "66+rue+Alleray+Paris" ) )
        join ( "geo:"
               join ( 0 , "," , 0 , "?" )
               join ( "z=" , get global zoom )
               join ( "&q=" , get global latitude , "," , get global longitude , "(my label : Notre Dame de Paris)" ) )
        join ( "google.navigation:"
               join ( "q=" , "place+de+la+concorde+Paris" )
               join ( "&mode=" , "w" ) )
        join ( "google.navigation:"
               join ( "q=" , get global latitude , "," , get global longitude )
               join ( "&mode=" , "d" ) )
        join ( "google.streetview:"
               join ( "cbll=" , get global latitude , "," , get global longitude )
               join ( "&cbp=0," , "-36" , ",0," , "0.8" , "," , "-25" ) )
    set ButtonDisplay . Text to join ( "Display map" , get global DataUriIndex , " : \n" ,
                                       select list item  list  get global DataUris
                                                         index get global DataUriIndex )
```

5.3 Solution 3 : Display with a web viewer and fusion tables

We are now switching to a new world where data management and mapping are in the cloud.
We will have two separate developments, one in the cloud to manage-serve data and build maps, and second one inside App Inventor to hold basic App algorithms and display map in Web viewer.

Our goal here is not to make a tutorial on fusion tables, so we will mainly use an existing table :
PizzaCustomers to hold the list of registered pizza customers with their name, address and location
https://www.google.com/fusiontables/DataSource?docid=1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC

Within an optional exercise, we will also use :
PizzaOrders to hold the list of orders with customer name, pizza and drink
https://www.google.com/fusiontables/DataSource?docid=1-WStaXfak4yGpXMibqX9TfSNSMamSNycsfozTpHd

PizzaMergeOrdersA... PizzaMergeOrdersAndCustomers, which merges or joins orders and customers on name https://www.google.com/fusiontables/DataSource?docid=1Om-ntvLmRaZ63iR4YdqKABbZWjMBcsTy7Fq2VKmh

These tables have been shared with anyone having the link in the fusion table sharing parameters. You can click or copy/paste each one of the fusion tables URL to open it and see what is inside.

Now, let us start with preparing the map on the cloud side.

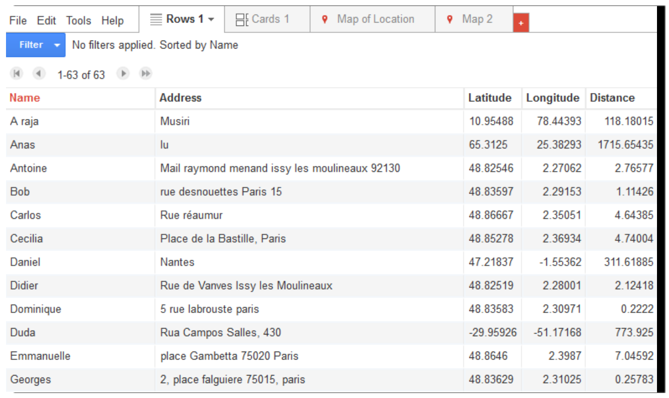
5.3.1 Fusion tables : My pizza company, take orders and deliver worldwide

You are now running a pizza company, that has worldwide customers with name and address. Your customers order pizzas to be delivered at their location. You and your customers want to monitor on a map what's going on : where's my pizza ? is it on its way ? when will I eat ?

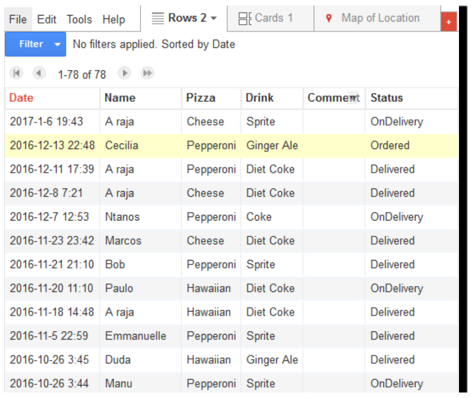
What data that we have to handle :

- **customers** with :
  name, address and/or latitude-longitude and other fields which you may need, such as distance to the pizza bakery.
- **orders** (taken by customers) with :
  date of order, name of customer, pizza ordered, drink, comment and delivery status (ordered, on- delivery, or delivered)

Tables below illustrate the beginning of corresponding fusion tables for **customers** and **orders**.



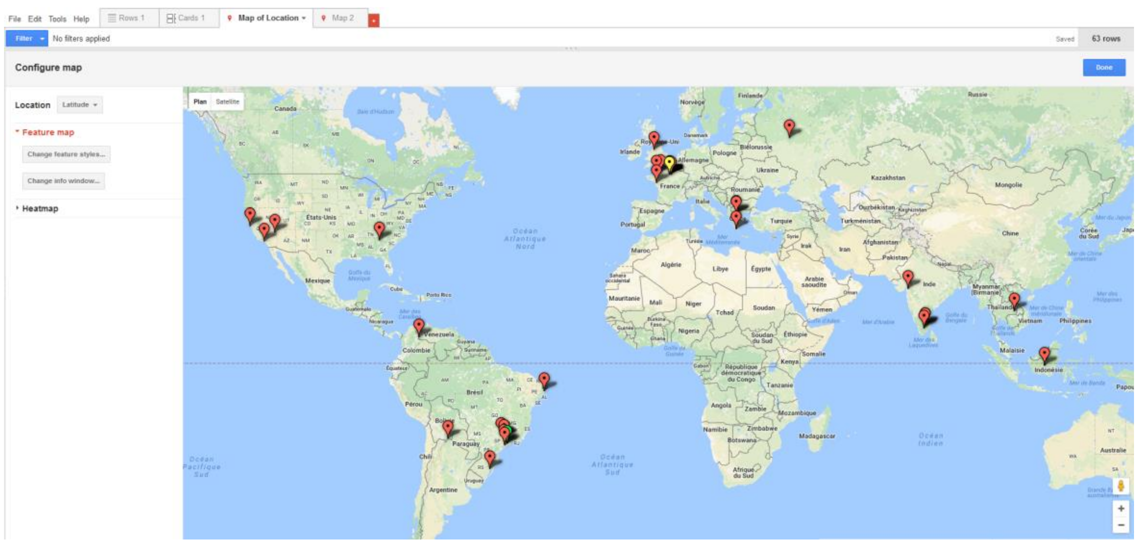
Name	Address	Latitude	Longitude	Distance
A raja	Musiri	10.95488	78.44393	118.18015
Anas	lu	65.3125	25.38293	1715.65435
Antoine	Mail raymond menand issy les moulineaux 92130	48.82546	2.27062	2.76577
Bob	rue desnouettes Paris 15	48.83597	2.29153	1.11426
Carlos	Rue réaumur	48.86667	2.35051	4.64385
Cecilia	Place de la Bastille, Paris	48.85278	2.36934	4.74004
Daniel	Nantes	47.21837	-1.55362	311.61885
Didier	Rue de Vanves Issy les Moulineaux	48.82519	2.28001	2.12418
Dominique	5 rue labrouste paris	48.83583	2.30971	0.2222
Duda	Rua Campos Salles, 430	-29.95926	-51.17168	773.925
Emmanuelle	place Gambetta 75020 Paris	48.8646	2.3987	7.04592
Georges	2, place falguiere 75015, paris	48.83629	2.31025	0.25783



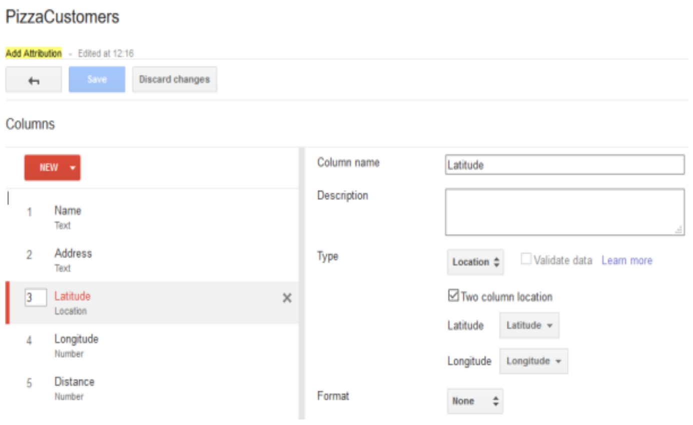
Date	Name	Pizza	Drink	Comment	Status
2017-1-6 19:43	A raja	Cheese	Sprite		OnDelivery
2016-12-13 22:48	Cecilia	Pepperoni	Ginger Ale		Ordered
2016-12-11 17:39	A raja	Pepperoni	Diet Coke		Delivered
2016-12-8 7:21	A raja	Cheese	Diet Coke		Delivered
2016-12-7 12:53	Ntanos	Pepperoni	Coke		OnDelivery
2016-11-23 23:42	Marcos	Cheese	Diet Coke		Delivered
2016-11-21 21:10	Bob	Pepperoni	Sprite		Delivered
2016-11-20 11:10	Paulo	Hawaiian	Diet Coke		OnDelivery
2016-11-18 14:48	A raja	Hawaiian	Diet Coke		Delivered
2016-11-5 22:59	Emmanuelle	Pepperoni	Sprite		Delivered
2016-10-26 3:45	Duda	Hawaiian	Ginger Ale		Delivered
2016-10-26 3:44	Manu	Pepperoni	Sprite		OnDelivery

We will do mapping of customers, then mapping of orders as an option.

<u>One click mapping of **customers**</u> : click on the **Map of Location** tab and here is your map !



This one click mapping works because Customers have a *location* criteria which can be either (*latitude* + *longitude*) or address :


PizzaCustomers

Add Attribution - Edited at 12:16

Save | Discard changes

Columns

NEW

1	Name Text
2	Address Text
3	Latitude Location
4	Longitude Number
5	Distance Number

Column name	Latitude
Description	
Type	Location ☐ Validate data Learn more
	☑ Two column location
	Latitude: Latitude
	Longitude: Longitude
Format	None

Here we have chosen *latitude and longitude* as location criteria for performance and privacy reasons. (We did not want addresses to travel continuously over the web, nor to translate addresses to geo-locations at each request).

Here, you can see that address is typed as "text" and Latitude as "location" with Longitude as secondary location info. It may be easier for you to use address as the location criteria.

NOTE : This table is used by an App which is in the Gallery ("Pizza delivery Mapping with Fusion Tables" by pierre.huguet50@gmail.com) what this App does is to :

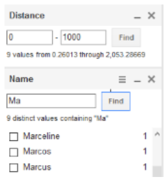
1. convert address to geo locations when creating or updating customer info,
2. compute distance to pizza shop as last attribute, so that it can be a query parameter.

Then we can add query criteria such as distance or name content :

- In the *Map of location* tab, Click on the Filter button Filter

- Select an attribute on where you want a filter, for example : Address
- Insert criteria, for example : Paris
- Then click : Find
  ➔ only customers with "Paris" in their address will be displayed

- Try again with distance between 0 and 1000 km and name containing "Ma" *
  ➔ selection changes, we have customers outside of Paris, but within 1000 km and their name contains "ma"

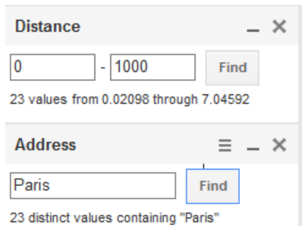


Distance
0 - 1000 Find
9 values from 0.26013 through 2,053.28669
Name
Ma Find
9 distinct values containing "Ma"
☐ Marceline 1
☐ Marcos 1
☐ Marcus 1

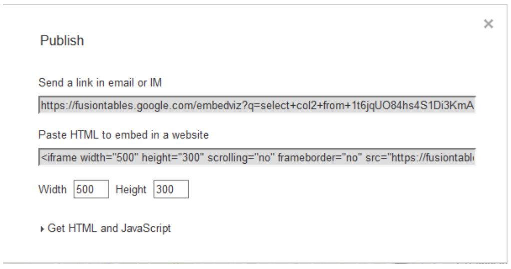
Last thing to do, before we go back to App Inventor, is to catch the URL which holds this map.

- Go back to your fusion table main screen,
- Click on the **Map of location** tab to display map,
- Adjust to your area of interest and zoom factor,
- Adjust your criteria (or insert all you may need)

  (we have kept distance and address here)
- Then click on the ***Tools*** tab and **Publish** option
- and copy and save the link for the Webviewer
- You can also check that it works in a browser



Distance
0 - 1000 Find
23 values from 0.02098 through 7.04592
Address
Paris Find
23 distinct values containing "Paris"



Publish

Send a link in email or IM
https://fusiontables.google.com/embedviz?q=select+col2+from+1t6jqUO84hs4S1Di3KmA

Paste HTML to embed in a website
<iframe width="500" height="300" scrolling="no" frameborder="no" src="https://fusiontabl

Width 500 Height 300

▸ Get HTML and JavaScript

For our example, we get this URL :

https://fusiontables.google.com/embedviz?q=select+col2+from+1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC+where+col6+%3E%3D+0+and+col6+%3C%3D+1000+and+col1+contains+ignoring+case+'Paris'&viz=MAP&h=false&lat=48.832371363477534&lng=2.33880412470695&t=1&z=12&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG

Let's look at it closer (we have replaced <,>, = characters by their readable value)

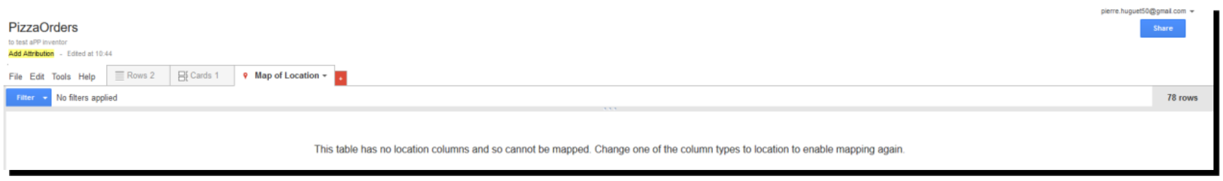
URL scheme, host and path	https://fusiontables.google.com/embedviz?	
Start of query	q=	
Select columns or attribute	select+	col2
From TABLE_ID	+from+	1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC
Where clause (optional)	+where+ +and+	col6+>=+0+and+col6+<=+1000 col1+contains+ignoring+case+'Paris'
Map display type	&viz=	MAP
	&h=	false
Center latitude	&lat=	48.832371363477534
Center longitude	&lng=	2.33880412470695
	&t=	1
Zoom factor	&z=	11
column for location	&l=	col2
	&y=	2
Template	&tmplt=	2
Location type (Latitude, longitude)	&hml=	TWO_COL_LAT_LNG

Where we see the data query criteria (SQL like) followed by the display parameters.

OPTIONAL Exercise : How can we get a map of orders ?

Try the same with the **orders** fusion table : open it and click on the **Map of Location** tab.

This does not work : display tells you that this table has no location …



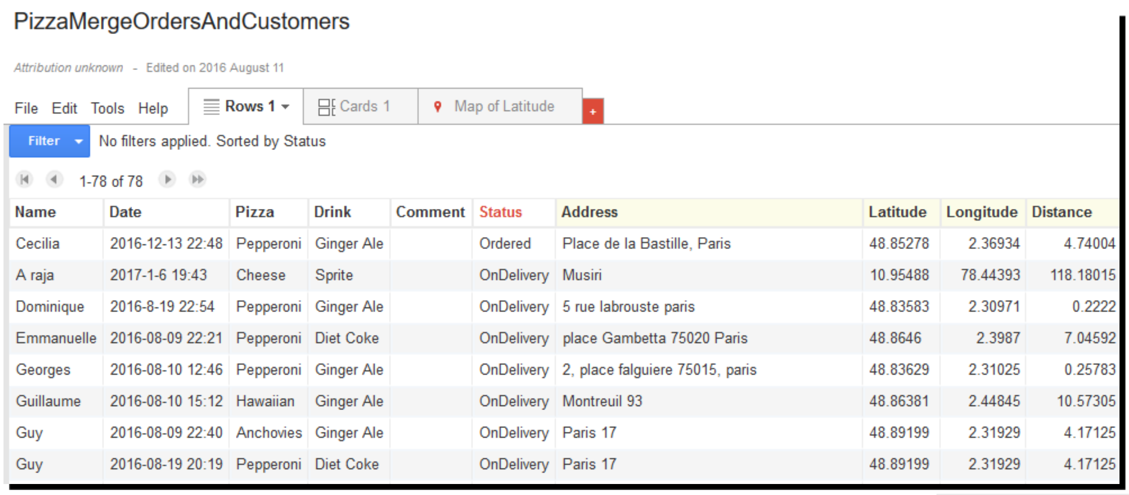
This table has no location columns and so cannot be mapped. Change one of the column types to location to enable mapping again.

That's right, <u>orders have no location by themselves</u>, they must be delivered at customer's location, and this information is in the customers table.

The trick, to solve this problem is to **Merge** the orders and customers table to get a third one that will contain orders data AND location of customers. This will be a dynamic table which will be automatically updated when you add, remove or modify a customer or an order :

In the orders table, click on the merge command in the file tab, and merge with the customers table, with the name as the source for matching on both sides.

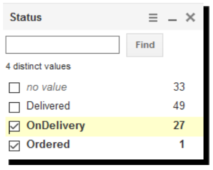
This creates a new table Merge of Customers and orders where we find :



PizzaMergeOrdersAndCustomers
Attribution unknown - Edited on 2016 August 11
File Edit Tools Help
Rows 1
Cards 1
Map of Latitude
Filter
No filters applied. Sorted by Status
1-78 of 78

Name	Date	Pizza	Drink	Comment	Status	Address	Latitude	Longitude	Distance
Cecilia	2016-12-13 22:48	Pepperoni	Ginger Ale		Ordered	Place de la Bastille, Paris	48.85278	2.36934	4.74004
A raja	2017-1-6 19:43	Cheese	Sprite		OnDelivery	Musiri	10.95488	78.44393	118.18015
Dominique	2016-8-19 22:54	Pepperoni	Ginger Ale		OnDelivery	5 rue labrouste paris	48.83583	2.30971	0.2222
Emmanuelle	2016-08-09 22:21	Pepperoni	Diet Coke		OnDelivery	place Gambetta 75020 Paris	48.8646	2.3987	7.04592
Georges	2016-08-10 12:46	Pepperoni	Ginger Ale		OnDelivery	2, place falguiere 75015, paris	48.83629	2.31025	0.25783
Guillaume	2016-08-10 15:12	Hawaiian	Ginger Ale		OnDelivery	Montreuil 93	48.86381	2.44845	10.57305
Guy	2016-08-09 22:40	Anchovies	Ginger Ale		OnDelivery	Paris 17	48.89199	2.31929	4.17125
Guy	2016-08-19 20:19	Pepperoni	Diet Coke		OnDelivery	Paris 17	48.89199	2.31929	4.17125

Now click on the Map of Latitude tab and the Map will show orders.

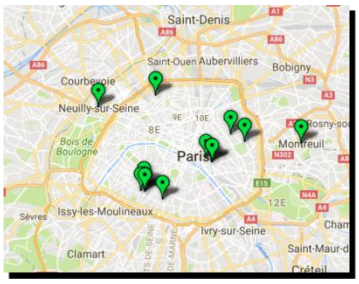


4 distinct values

☐ *no value*	33
☐ Delivered	49
☑ **OnDelivery**	27
☑ **Ordered**	1

If you add a filter on the pizza delivery **status** (ex : check Ordered or onDelivery and uncheck Delivered pizzas), only those will show.

If you further click, on an order, you will see all its details.



You may also colour icons according to a numeric variable, but tis exceeds the scope of the current tutorial (see the fusion tables tutorial).

5.3.2 Display the fusion table maps with a Web Viewer component

Would you take this bet ? build an interactive mapping App with no blocks !!!

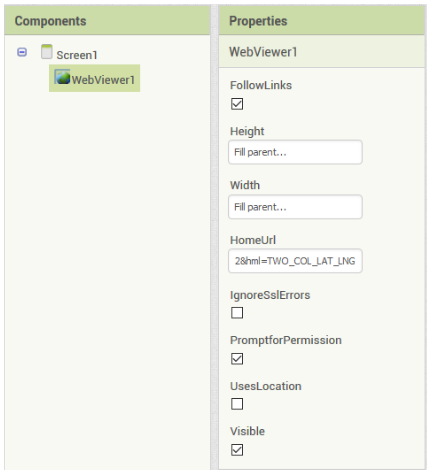
Open App Inventor and create a new project, then add a web viewer component, set its width and height to fill parent, set the Home URL to the URL you saved previously :

> https://fusiontables.google.com/embedviz?q=select+col2+from+1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC+where+col6+%3E%3D+0+and+col6+%3C%3D+1000+and+col1+contains+ignoring+case+'Paris'&viz=MAP&h=false&lat=48.832371363477534&lng=2.33880412470695&t=1&z=12&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG

Run the App !

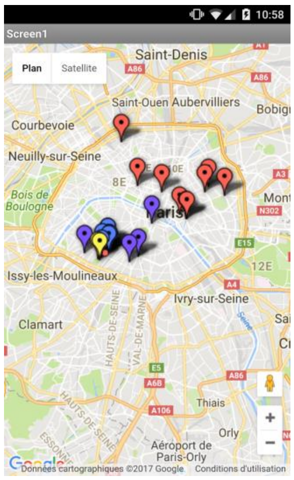
(No blocks, we won the bet)

You can interact with the map : click on customer icons to get their name and address, zoom, pan, switch to satellite display or go to streetview with the pegman … !



Components
Screen1
WebViewer1

Properties
WebViewer1
FollowLinks ☑
Height: Fill parent...
Width: Fill parent...
HomeUrl: 2&hml=TWO_COL_LAT_LNG
IgnoreSslErrors ☐
PromptforPermission ☑
UsesLocation ☐
Visible ☑

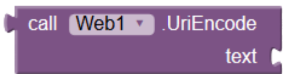


Now let us go back to the URL and see how we can modify it with program blocks.
Here we have added a second URL (from the fusion table “Rows” tab) to display data in list format.
Within the SQL query (select,where and order by) we have replaced col0, col1, col2, col4 and col6 by their attribute name : Name, Address, Latitude, Longitude, Distance.

		MAP DISPLAY URL	LIST DISPLAY URL
URL scheme, host, path		https://fusiontables.google.com/embedviz?	https://fusiontables.google.com/embedviz?
Start of query	q=		
Select cols or attribute	+select+	Latitude	Name,+Address,+Latitude,+Longitude,+Distance
From TABLE_ID	+from+	1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC	1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC
Where clause (optional) And …(other condition)	+where+ +and+	Address+contains+ignoring+case+'Paris' Distance>=0+and+Distance<=1000	Address+contains+ignoring+case+'Paris' Distance>=0+and+Distance<=1000
Order by clause (option)	+order+by+		Name+asc
Center latitude	&lat=	48.832371363477534	48.832371363477534
Center longitude	&lng=	2.33880412470695	2.33880412470695
Zoom factor	&z=	11	
Map display type	&viz=	MAP	GVIZ
	&t=	1	TABLE
	&h=	false	
column for location	&l=	col2	
	&y=	2	
Template	&tmplt=	2	
Location type lLat, long)	&hml=	TWO_COL_LAT_LNG	
Container Id	&containerId=		googft-gviz-canvas

Notes :
. you can change orders between URL parameters separated by &.
. we have kept the + encoding of space in this table to show where blank spaces are (they must be replaced before calling Uri Encode)

Note : you may want to code the URL with readable characters (instead of “reserved characters” translated into %3D …), in which case you can ask App Inventor to do the URL encoding with the **Uriencode** procedure.
(I usually don’t do it, because addresses copied from fusion tables are already encoded



So that we have these 2 URLs to test in a program :

MAP display URL

> https://fusiontables.google.com/embedviz?q=select+Latitude+from+1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC+where+Distance>=+0+and+Distance<=+1000+and+Address+contains+ignoring+case+'Paris'&lat=48.832371363477534&lng=2.33880412470695&z=12&viz=MAP&h=false &t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG

List Display URL

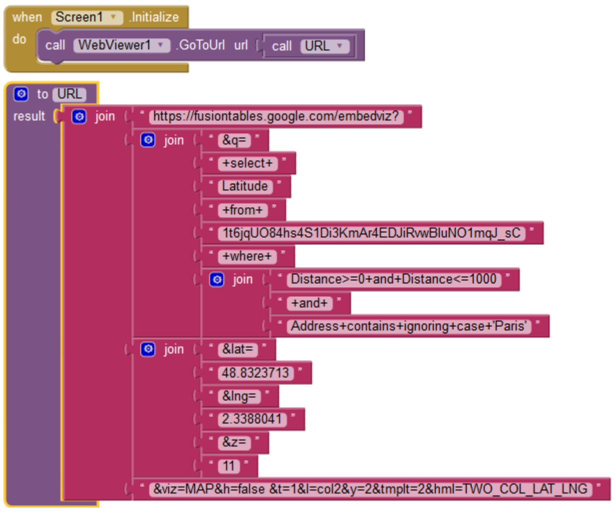
> https://fusiontables.google.com/embedviz?q=select+Name,+Address,+Latitude,+Longitude,+Distance+from+1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC+where+Address+contains+ignoring+case+'Paris'+and+Distance+>=+0+and+Distance+<=+1000+order+by+col0+asc&viz=GVIZ&t=TABLE&containerId=googft-gviz-canvas

Note that we are using reserved characters but these links should still work as long as you keep + instead of 'space'. (In fact you should replace , = < > by their URL encoding)

OK, let us code these cases and enable program :

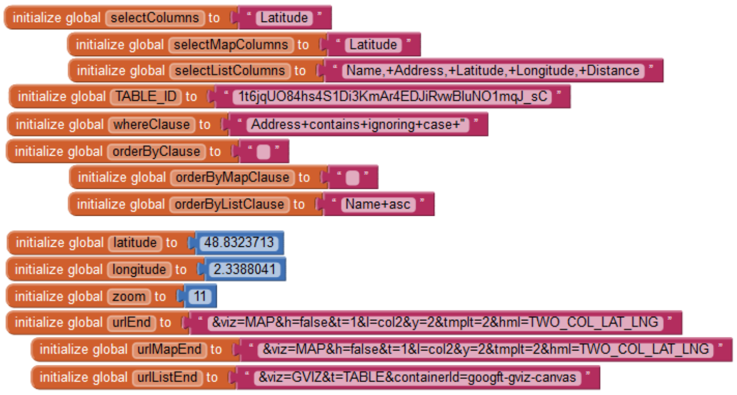
- Open previous App inventor display project which had no block
- create a procedure that computes the URL and start with the previous URL value
- Check that it (still) works by setting the Webviewer URL on screen initialize
- Then break down the URL into manageable pieces, and build it with joins in the procedure

Note : do this with step by step (checking at each step) to avoid headaches with errors

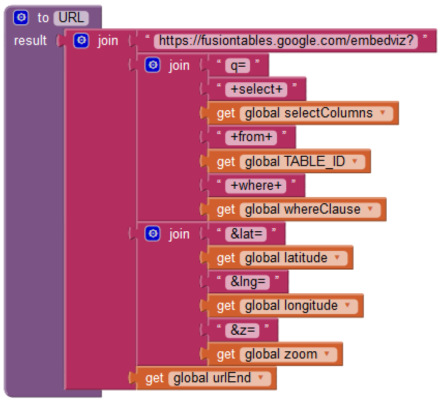


```
when Screen1 .Initialize
do  call WebViewer1 .GoToUrl url  call URL

to URL
result  join  "https://fusiontables.google.com/embedviz?"
              join  "&q="
                    "+select+"
                    "Latitude"
                    "+from+"
                    "1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC"
                    "+where+"
                    join  "Distance>=0+and+Distance<=1000"
                          "+and+"
                          "Address+contains+ignoring+case+'Paris'"
              join  "&lat="
                    "48.8323713"
                    "&lng="
                    "2.3388041"
                    "&z="
                    "11"
              "&viz=MAP&h=false &t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG"
```

- Replace initial values by variables (for later change by program blocks), **check that it works**



```
initialize global selectColumns to "Latitude"
    initialize global selectMapColumns to "Latitude"
    initialize global selectListColumns to "Name,+Address,+Latitude,+Longitude,+Distance"
initialize global TABLE_ID to "1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC"
initialize global whereClause to "Address+contains+ignoring+case+""
initialize global orderByClause to ""
    initialize global orderByMapClause to ""
    initialize global orderByListClause to "Name+asc"

initialize global latitude to 48.8323713
initialize global longitude to 2.3388041
initialize global zoom to 11
initialize global urlEnd to "&viz=MAP&h=false&t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG"
  initialize global urlMapEnd to "&viz=MAP&h=false&t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG"
  initialize global urlListEnd to "&viz=GVIZ&t=TABLE&containerId=googft-gviz-canvas"
```



```
to URL
result  join  "https://fusiontables.google.com/embedviz?"
              join  "q="
                    "+select+"
                    get global selectColumns
                    "+from+"
                    get global TABLE_ID
                    "+where+"
                    get global whereClause
              join  "&lat="
                    get global latitude
                    "&lng="
                    get global longitude
                    "&z="
                    get global zoom
              get global urlEnd
```

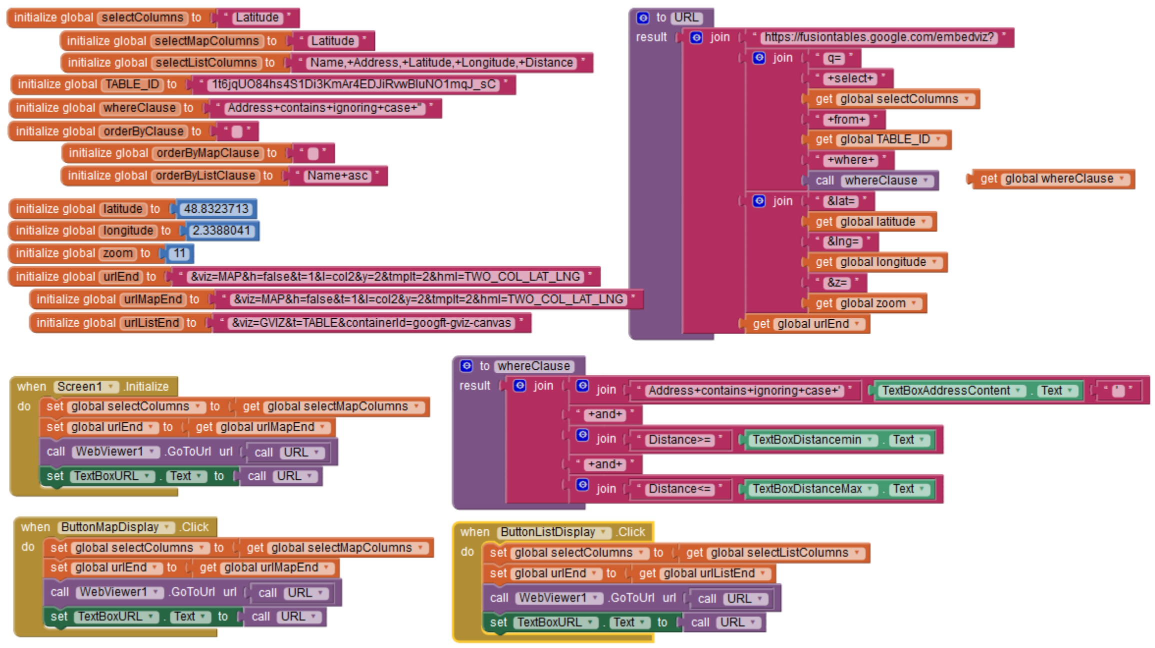
On the design side :

- Add horizontal arrangement with 2 buttons to switch to Map or List display,
- Add horizontal arrangement, labels and textboxes for min and Max distance criteria (check numbers only property and set default values to valid values 0 and 21000),
- Add horizontal arrangement with label and textbox for text criteria on address (check that default text is the empty string),
- Add textbox to show (and check URL) with multiline enabled

Go back to the blocks side :

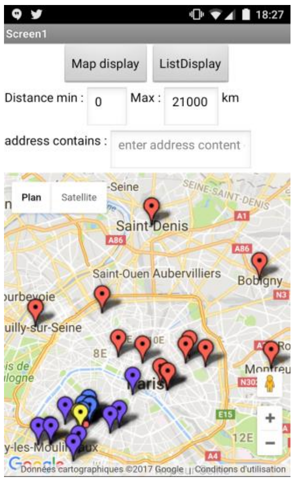
- Create a whereClause procedure that builds it from textbox values and returns it,
- Replace the whereClause variable in the URL procedure by a call to the whereClause procedure,
- Then add the Map/list button click handlers to set the selectcolumns and endURL to the right value, and update the textbox with URL.
- Run and check (step by step) then set URL textbox visibility off.

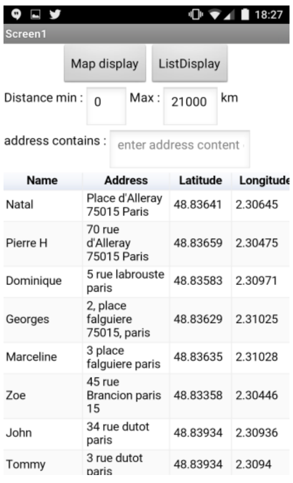
Make sure to things step by step with checks for each change.



```
initialize global selectColumns to "Latitude"
    initialize global selectMapColumns to "Latitude"
    initialize global selectListColumns to "Name,+Address,+Latitude,+Longitude,+Distance"
initialize global TABLE_ID to "1t6jqUO84hs4S1Di3KmAr4EDJiRvwBluNO1mqJ_sC"
initialize global whereClause to "Address+contains+ignoring+case+'"
initialize global orderByClause to " "
    initialize global orderByMapClause to " "
    initialize global orderByListClause to "Name+asc"

initialize global latitude to 48.8323713
initialize global longitude to 2.3388041
initialize global zoom to 11
initialize global urlEnd to "&viz=MAP&h=false&t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG"
  initialize global urlMapEnd to "&viz=MAP&h=false&t=1&l=col2&y=2&tmplt=2&hml=TWO_COL_LAT_LNG"
  initialize global urlListEnd to "&viz=GVIZ&t=TABLE&containerId=googft-gviz-canvas"

to URL
result join "https://fusiontables.google.com/embedviz?"
            join "q="
                 "+select+"
                 get global selectColumns
                 "+from+"
                 get global TABLE_ID
                 "+where+"
                 call whereClause                get global whereClause
            join "&lat="
                 get global latitude
                 "&lng="
                 get global longitude
                 "&z="
                 get global zoom
            get global urlEnd

when Screen1.Initialize
do  set global selectColumns to get global selectMapColumns
    set global urlEnd to get global urlMapEnd
    call WebViewer1.GoToUrl url call URL
    set TextBoxURL.Text to call URL

to whereClause
result join join "Address+contains+ignoring+case+'"  TextBoxAddressContent.Text  "'"
            "+and+"
            join "Distance>="  TextBoxDistancemin.Text
            "+and+"
            join "Distance<="  TextBoxDistanceMax.Text

when ButtonMapDisplay.Click
do  set global selectColumns to get global selectMapColumns
    set global urlEnd to get global urlMapEnd
    call WebViewer1.GoToUrl url call URL
    set TextBoxURL.Text to call URL

when ButtonListDisplay.Click
do  set global selectColumns to get global selectListColumns
    set global urlEnd to get global urlListEnd
    call WebViewer1.GoToUrl url call URL
    set TextBoxURL.Text to call URL
```

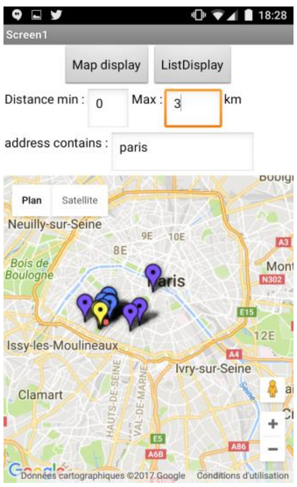
Display results :





18:27
Screen1
Map display ListDisplay
Distance min : 0 Max : 21000 km
address contains : enter address content

Name	Address	Latitude	Longitude
Natal	Place d'Alleray 75015 Paris	48.83641	2.30645
Pierre H	70 rue d'Alleray 75015 Paris	48.83659	2.30475
Dominique	5 rue labrouste paris	48.83583	2.30971
Georges	2, place falguiere 75015, paris	48.83629	2.31025
Marceline	3 place falguiere paris	48.83635	2.31028
Zoe	45 rue Brancion paris 15	48.83358	2.30446
John	34 rue dutot paris	48.83934	2.30936
Tommy	3 rue dutot paris	48.83934	2.3094



5.4 Solution 4 : display map with a Canvas component and the statics map API

5.4.1 General principles

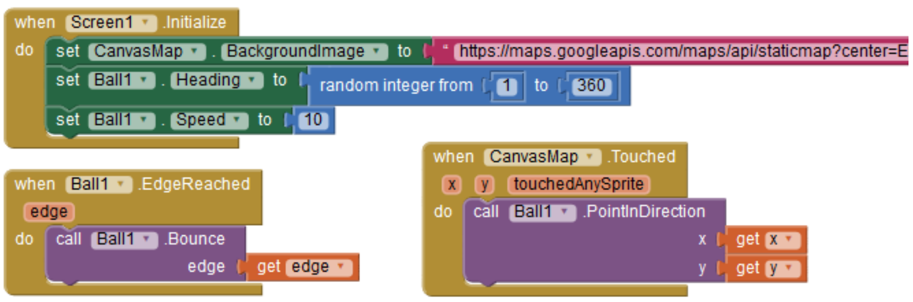
Solutions 1 and 2 handed control to a web service, with immediate benefit of interactive functions provided by the Web site. Here, the map will be the background picture of a canvas. The app keeps control and can add balls and sprites, but we start with a “static” map, it will be the App’s programmer job to make it move, there is no website to do the job.

Let’s start with an example : we want a map around the Eiffel tower and a ball bouncing on the map :

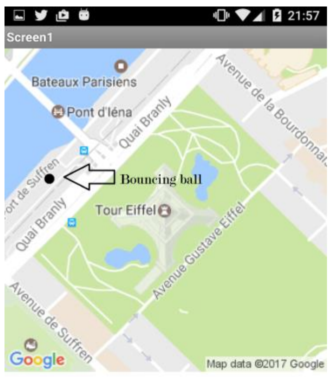
Create a new App Inventor project, with Screen orientation in “portrait mode” and “Fixed” sizing. From the “Drawing and animation” drawer, add a canvas and name it canvasMap and set its width and height to “fill parent”. Also add a ball.

On the blocks side :

- When screen.initialize,
  - Set the canvas background image to https://maps.googleapis.com/maps/api/staticmap?center=Eiffel+tower&zoom=16&size=320x320
- Set the ball heading to a random value between 1 and 360, and set ist speed to 10
- When the ball reaches the end, make it rebound,
- and when user touches the canvas, head the ball towards the point that has been touched.



```
when Screen1 .Initialize
do  set CanvasMap . BackgroundImage to " https://maps.googleapis.com/maps/api/staticmap?center=E
    set Ball1 . Heading to random integer from 1 to 360
    set Ball1 . Speed to 10

when Ball1 .EdgeReached
 edge
do  call Ball1 .Bounce
               edge  get edge

when CanvasMap .Touched
 x  y  touchedAnySprite
do  call Ball1 .PointInDirection
                      x  get x
                      y  get y
```



Run the program, touch the canvas, …

What are the differences between this and mapping with a Web Viewer ? The map is just sitting there, and has no reaction (for now) but App keeps control of outputs to the window (add balls or sprites), and also keeps control of real time inputs from the window : That means that you can build Apps or games which behave according to what happens on the window (like clicks, collisions, edges reached, …). This is a key point for games, including geospatial games like Pokemons.

You may have enough with this map tile, where your hero will play but you may also want to play all around the city, looking for treasures with multiple map scales, multiple players and no geographic boundary to your game.

This is what we will see now, but we have to deal with 2 different subjects :

- Matching the screen pixel space and the geographic space in degrees or meters
- Managing maps with the Static maps API

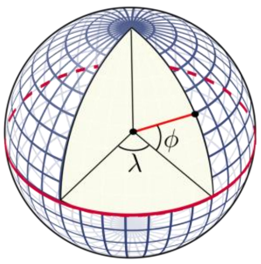
5.4.2 Matching spaces : canvas pixels, geographic lat & long and metric coordinates

Let us consider a game where the map displays my location as well as the location of other players, and all sorts of characters with whom we may interact.

- When writing code for sprites, we will use canvas or pixel coordinates,
- Our position as well as the position of other players and characters will be known in degrees of longitude and latitude,
- Maps will be spatially defined by their center in longitude and latitude, with a pixel size which depends on the zoom factor,
- Distances are measured in meters.

We have what we call different coordinate systems, and we will need conversions to display other players (with longitude and latitude coordinates) on our canvas map (with pixel coordinates), or when we need to measure a distance in kilometers.

May be we can remind what are the Longitude, Latitude and altitude of a point :



- Longitude λ is an angle that specifies the east-west position of a point on the Earth's surface. 0° at the Prime Meridian in Greenwich. It ranges from −180° westward to +180° eastward.
- Latitude ϕ is an angle that specifies the north–south position of a point on the Earth's surface. It ranges from -90° at the south pole, 0° at the equator to +90° at the north pole.
- Altitude is the height in meters above mean sea level (MSL), not to be confused with height above ground (AGL). Terrain elevation is the altitude (MSL) of the ground itself

The functions we need an that are provided are :

- length in meters of a degree of latitude : **degreeOfLatitudeMeters**
- length in meters of a degree of longitude : **degreeOfLongitudeMeters(latitude)**
- distance in meters between 2 points from their longitude and latitude (on the spherical earth) : **distanceMeters(lat1, long1, lat2,long2)**
- width in degrees of a pixel : **pixelWidthInDegrees(zoom)**
- height in degrees of a pixel : **pixelHeightInDegrees(zoom, latitude)**
- size of a pixel in meters : **pixelsizeInMeters(zoom, latitude)**
- longitude of pixel at x : **longitudeFromX(x, canvasWidth,centerlat, centerLong,zoom)**
- latitude of pixel at y : **latitudeFromY (y, canvasHeight, centerLatitude, zoom)**
- X pixel coordinate from latitude longitude :

  xFromLatLong (longitude, canvasWidth, centerLatitude, centerLongitude, zoom)
- Y pixel coordinate from latitude :

  yFromLat (latitude, canvasHeight, centerLatitude, zoom)

Don't be afraid : You will not have to code these functions, and you do not need to understand the details of what is inside. But You need to have a clear view of what they do and when to use them (we will see examples). You should also be able to check that they work well on practical cases. There has been errors and there still may be. When you reuse : trust but check.

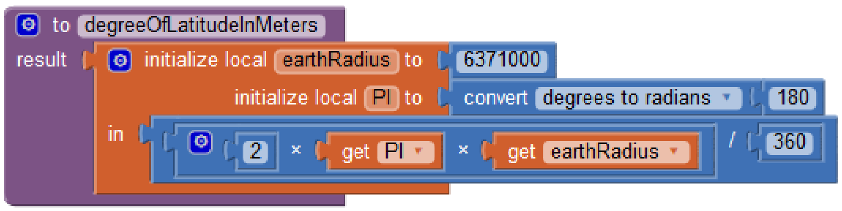
5.4.2.1 Length in meters of one degree of latitude and of longitude

As an effort to share universal units, the meter was first defined in 1791 as 1/10 000 000 of half of a meridian of the earth.

A degree of **latitude** is then equal to 10 000 000 m / 90° ~ 111 111 m.
However, we will here assume a spherical earth with an average radius rounded to 6 371 km
leading to **1° = 111 195** m. The reason is that we want to use a single constant for all formulas : the earth radius.

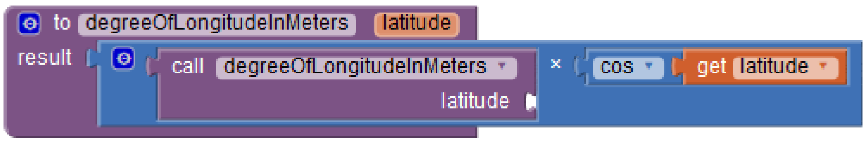
A degree of **longitude** changes with latitude degree of longitude = degree of latitude * cosine (latitude)with the corresponding App inventor code here

The **length of a degree of latitude** in meters is given by :



```
to degreeOfLatitudeInMeters
result  initialize local earthRadius to 6371000
        initialize local PI to convert degrees to radians 180
        in  ( 2 × get PI × get earthRadius ) / 360
```

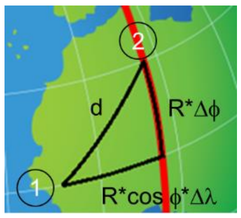
The **length of a degree of longitude** in meters is given by :



```
to degreeOfLongitudeInMeters latitude
result  call degreeOfLongitudeInMeters
             latitude
        × cos get latitude
```

5.4.2.2 *Distance between 2 points known by their longitude λ and latitude ϕ*

The shortest distance between 2 points over the earth is also called the “great circle” distance (different from the straight line through the earth).



When distances are limited you may use the Pythagorean theorem :

$$d = \sqrt{(R*\cos\phi*\Delta\lambda)^2 + (R*\Delta\phi)^2}$$

leading to $d = R\sqrt{\cos\phi_1*\cos\phi_2*\Delta\lambda^2 + \Delta\phi^2}$

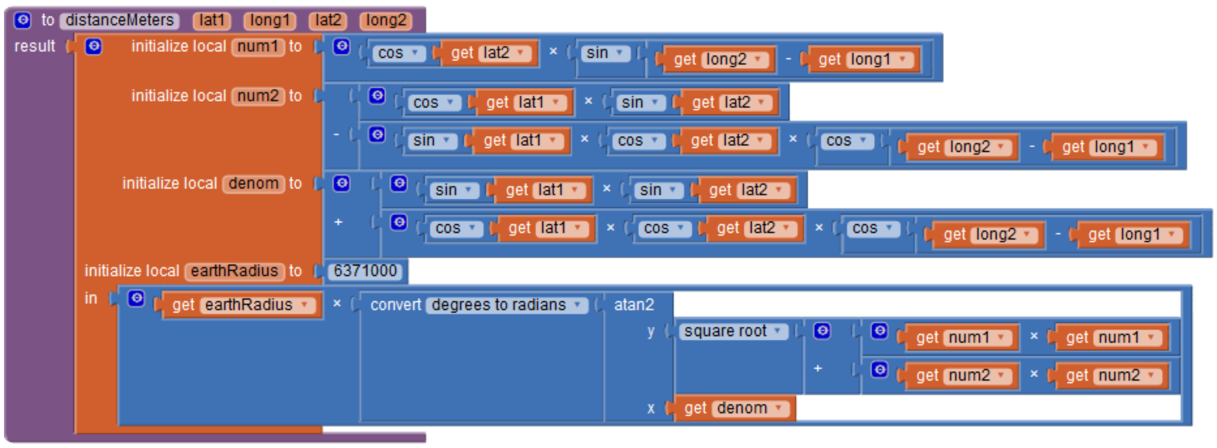
Which has a precision of 1% within 4000 km.

But we will use the more precise Vincenty formula (see Wikipedia) of the angular distance :

$$\Delta\sigma = \arctan\frac{\sqrt{(\cos\phi_2\cdot\sin(\Delta\lambda))^2 + (\cos\phi_1\cdot\sin\phi_2 - \sin\phi_1\cdot\cos\phi_2\cdot\cos(\Delta\lambda))^2}}{\sin\phi_1\cdot\sin\phi_2 + \cos\phi_1\cdot\cos\phi_2\cdot\cos(\Delta\lambda)}$$

Which we multiply by the radius of the earth, to get the distance in meters.

The distance in meters between 2 points on the earth known by their longitudes and latitudes lat1, long1, lat2,long2 is :



```
to distanceMeters lat1 long1 lat2 long2
result  initialize local num1 to  cos get lat2 × sin ( get long2 - get long1 )
        initialize local num2 to  ( cos get lat1 × sin get lat2 )
                                - ( sin get lat1 × cos get lat2 × cos ( get long2 - get long1 ) )
        initialize local denom to ( sin get lat1 × sin get lat2 )
                                + ( cos get lat1 × cos get lat2 × cos ( get long2 - get long1 ) )
        initialize local earthRadius to 6371000
        in  get earthRadius × convert degrees to radians atan2
                                                          y  square root ( get num1 × get num1 ) + ( get num2 × get num2 )
                                                          x  get denom
```

5.4.2.3 Pixel Width and height in degrees of longitude and latitude

The operation that converts latitude and longitude (λ,ϕ) to pixel coordinates (x,y) is the "web Mercator" projection (see Wikipedia) with the following formulas :

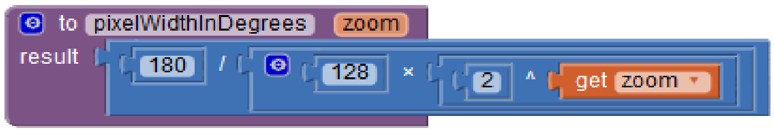
$$x = \frac{128}{\pi} 2^{\text{zoom level}} (\lambda + \pi) \text{ pixels}$$
$$y = \frac{128}{\pi} 2^{\text{zoom level}} \left(\pi - \ln\left[\tan\left(\frac{\pi}{4} + \frac{\varphi}{2}\right)\right]\right) \text{ pixels}$$

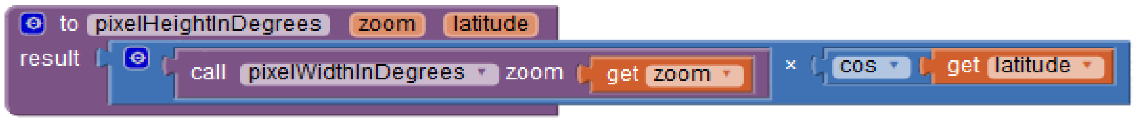
From which follows in radians :

$$\textit{pixel width in radians} = d\lambda/_{dx} = \frac{\pi}{128} 2^{-zoom}$$
$$\textit{pixel height in radians} = d\phi/_{dy} = \cos(\phi) \frac{\pi}{128} 2^{-zoom}$$

or for us in degrees :

$$\textit{pixel width in degrees} = d\lambda/_{dx} = \frac{180}{128} 2^{-zoom}$$
$$\textit{pixel height in degrees} = d\phi/_{dy} = \cos(\phi) \frac{180}{128} 2^{-zoom}$$


```
to pixelWidthInDegrees zoom
result  180 / ( 128 × ( 2 ^ get zoom ) )
```

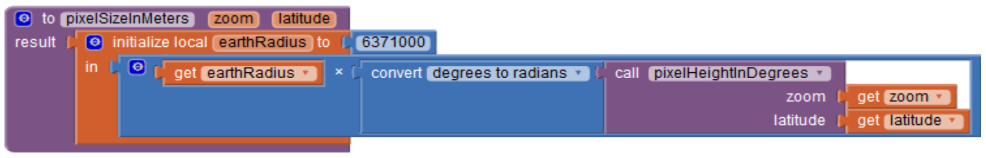

```
to pixelHeightInDegrees zoom latitude
result  call pixelWidthInDegrees zoom get zoom  ×  cos get latitude
```

5.4.2.4 Pixel size in meters

- Pixel height in meters = pixel height in radians multiplied by radius of a meridian [i.e. R_{earth}]
- Pixel width in meters = pixel width in radians multiplied by radius of the parallel at given latitude [i.e. R_{earth} * cos(ϕ)]

And we find the same result for width and height (i.e. square pixels):

$$\textit{pixel width in meters} = R_{earth} \cos(\phi) \frac{180}{128} 2^{-zoom}$$
$$\textit{pixel height in meters} = R_{earth} \cos(\phi) \frac{180}{128} 2^{-zoom}$$

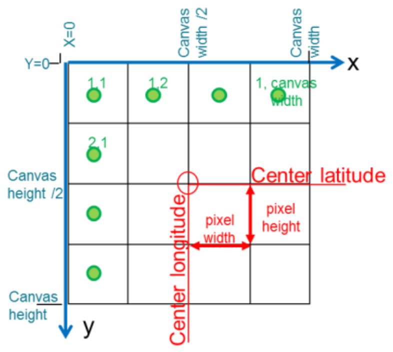

```
to pixelSizeInMeters zoom latitude
result  initialize local earthRadius to 6371000
        in  get earthRadius × convert degrees to radians  call pixelHeightInDegrees
                                                                   zoom      get zoom
                                                                   latitude  get latitude
```

5.4.2.5 longitude and latitude from pixel-canvas coordinates

Canvas pixel coordinates are illustred by the figure on the right with origin is top left.

in green : canvas pixels with center position and x,y number,
in red : center latitude and longitude, pixel height and width,
in blue : physical screen coordinates (canvas units).

When map is oriented north, x increases with longitude and y decreases with latitude (opposite direction).

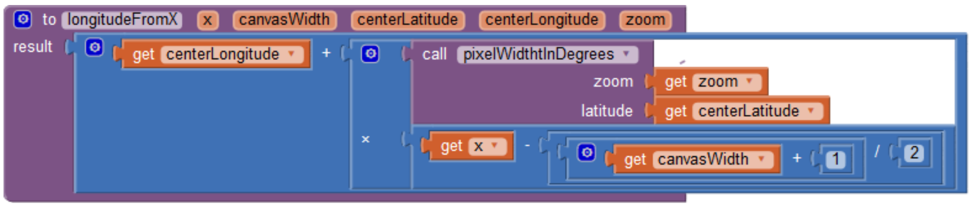


For pixel x,y returned by App Inventor blocks it follows :

$$longitude(x) = center\ longitude + pixel\ width * \left(x - \frac{canvas\ width + 1}{2}\right)$$

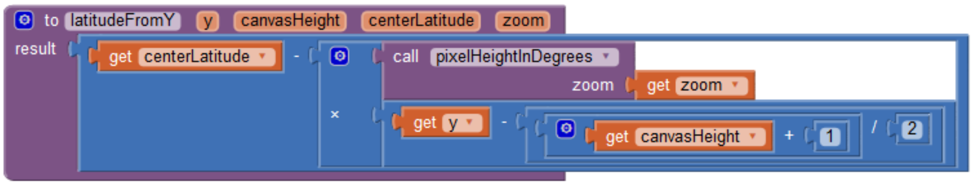
$$latitude(y) = \ center\ latitude\ - pixel\ height * \left(y - \frac{canvas\ height + 1}{2}\right)$$

longitude from pixel x coordinate and canvas center lat



```
to longitudeFromX x canvasWidth centerLatitude centerLongitude zoom
result  get centerLongitude + call pixelWidthtInDegrees
                                     zoom     get zoom
                                     latitude get centerLatitude
                              × get x - get canvasWidth + 1 / 2
```

Latitude from pixel y coordinate



```
to latitudeFromY y canvasHeight centerLatitude zoom
result  get centerLatitude - call pixelHeightInDegrees
                                  zoom get zoom
                             × get y - get canvasHeight + 1 / 2
```

5.4.2.6 pixel-canvas coordinates from longitude and latitude

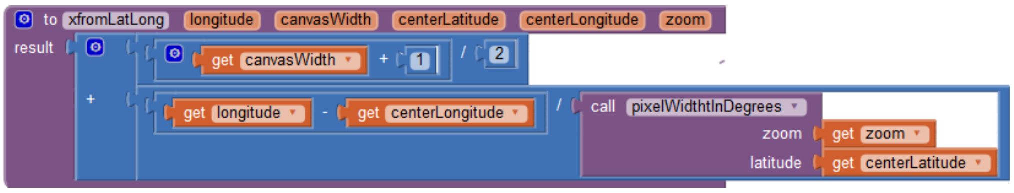
Inversion of the previous functions is straightforward,
a point with "latitude, longitude" coordinates should be plotted at x, y given by :

$$x = \frac{canvas\ width + 1}{2} + \frac{longitude - center\ longitude}{pixel\ width}$$

$$y = \frac{canvas\ height + 1}{2} - \frac{latitude - center\ latitude}{pixel\ height}$$

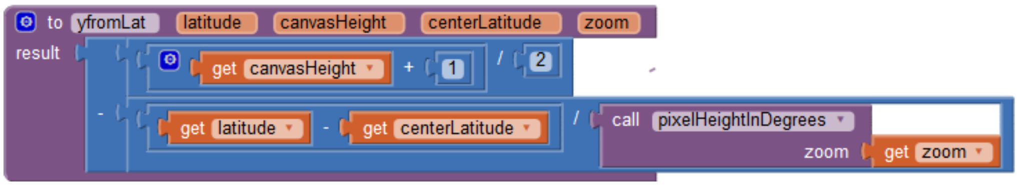
If x or y are less than 0 or greater than canvas width or height, point is out of current map view.

X pixel coordinate from latitude longitude and center



```
to xfromLatLong longitude canvasWidth centerLatitude centerLongitude zoom
result    get canvasWidth + 1 / 2
        + get longitude - get centerLongitude / call pixelWidthtInDegrees
                                                     zoom     get zoom
                                                     latitude get centerLatitude
```

Y pixel coordinate from latitude and center



```
to yfromLat latitude canvasHeight centerLatitude zoom
result    get canvasHeight + 1 / 2
        - get latitude - get centerLatitude / call pixelHeightInDegrees
                                                   zoom get zoom
```

5.4.3 Static maps and streetview APIs

*"The Google Static Maps API lets you embed a Google Maps **image** on your web page without requiring JavaScript or any dynamic page loading. The Google Static Maps API service creates your map based on URL parameters sent through a standard HTTP request and **returns the map as an image** you can display on your web page."*

The documentation for the staticmaps and streetview APIs can be found here :

https://developers.google.com/maps/documentation/static-maps/intro
https://developers.google.com/maps/documentation/streetview/intro
(you can add ?/hl=en or ?/hl=fr or ?/hl=es to change the language)

what is useful for us is :

The base address (scheme and host) :

for static maps	https://maps.googleapis.com/maps/api/staticmap?
or	http://maps.google.com/maps/api/staticmap?
for streetview	https://maps.googleapis.com/maps/api/streetview?
or	http://maps.googleapis.com/maps/api/streetview?

The static maps query fields (separated by '&'):

`center` : defines latitude and longitude of map center (e.g. "40.714728,-73.998672"). It may also be an address (replace space by +)

`size` : rectangular dimensions of the map image (e.g. canvas size) with a string of the form *{horizontal_value}*x*{vertical_value}*.
Note : be careful that the size is not too big to avoid delay times with low bandwidth.

`maptype` : defines the type of map to construct. We will use and switch between `roadmap` and `satellite` (also available in API: `hybrid` and `terrain`).

`zoom` : defines the display scale from : 0 for world, 10 for cities and 21 for street level. Adding 1 to zoom level will divide pixel size by 2. Estimated pixel height and width I degrees of latitude and longitude is :
$dp_{lat} = 360 * 0.707 * 2^{(zoom+8)}$ in degrees of latitude
$dp_{long} = dp_{lat} * \cos(latitude)$ in degrees of longitude

`markers` : defines markers overlaid on the map (the '`|`' or %7C separates multiple values)

and for the streetview query fields :

`location` :	same as center above
`size` :	same as above.
`fov` :	field of view (default 90°)
`heading` :	0 to 360 : 0 north, 90 east, 180 south, 270 west
`pitch` :	vertical orientation, 0 by default, values from -90° (down) to 90° (up)

Note : at the time of writing this tutorial, no API key was necessary.

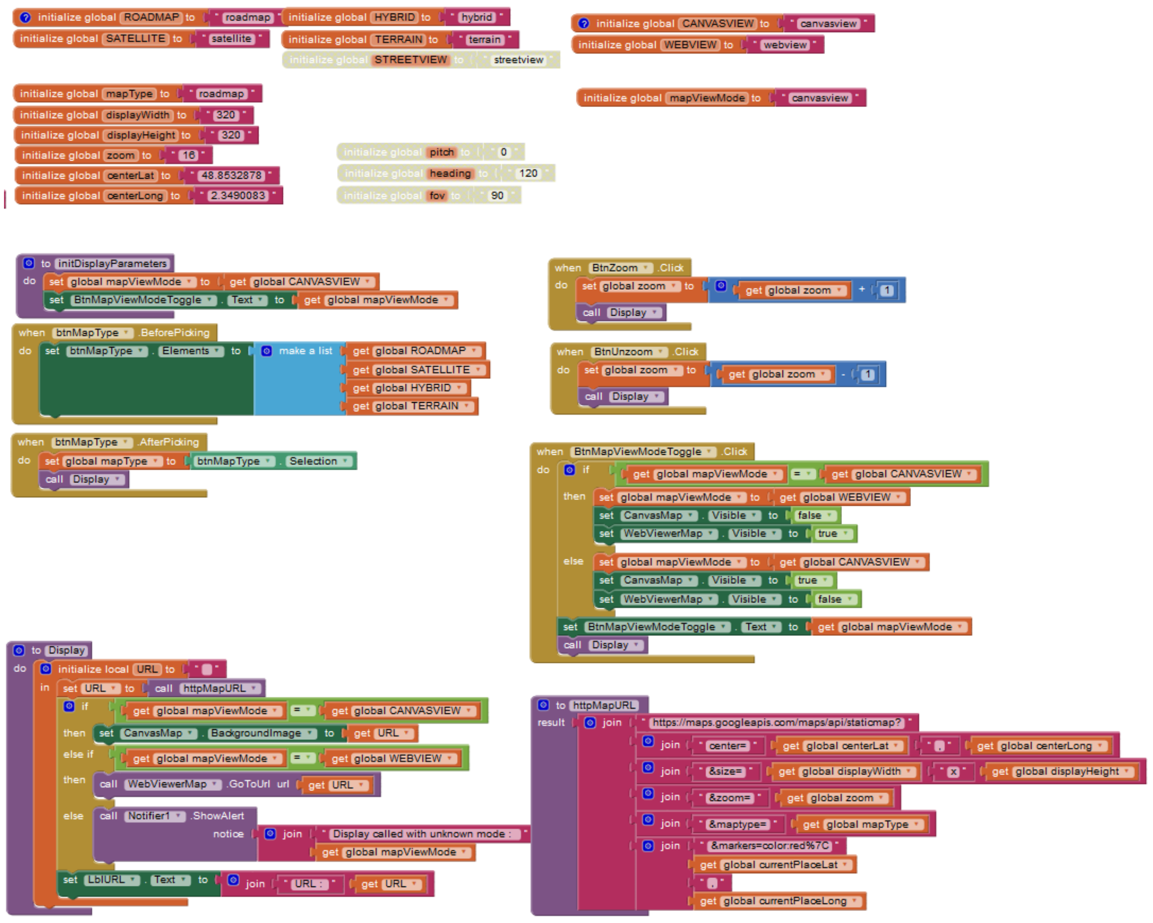
5.4.4 Coding exercise with Static maps and streetview APIs

5.4.5 TODO

1. enable variables and constants which will be used : mapViewMode, CANVASVIEW, WEBVIEW
2. toggle the mapViewMode variable with the **BtnMapViewModeToggle** button, switch visibility of the canvasMap and WebviewerMap accordingly, then call **Display**.
3. You can set preferred startup value in the **initDisplayParameters** procedure
4. in the **Display** procedure, according to the value of mapViewMode branch to :
   - existing webviewer display code,
   - to a new canvas display code where you will set the canvas background image (**canvasMap.BackgroundImage**) to the same URL.
5. Discuss with peers the difference between canvas and webviewer :
   - Is there a display difference between **canvas** view mode and **webviewer** view mode ?
   - what can you do with webview, you cannot with canvas ?
   - what can you do with canvas that you cannot with webview ?

   Discuss with your peers : what do you prefer

5.4.6 suggested solution



```
initialize global ROADMAP to "roadmap"
initialize global SATELLITE to "satellite"
initialize global HYBRID to "hybrid"
initialize global TERRAIN to "terrain"
initialize global STREETVIEW to "streetview"
initialize global CANVASVIEW to "canvasview"
initialize global WEBVIEW to "webview"

initialize global mapType to "roadmap"
initialize global displayWidth to "320"
initialize global displayHeight to "320"
initialize global zoom to "16"
initialize global centerLat to "48.8532878"
initialize global centerLong to "2.3490083"
initialize global mapViewMode to "canvasview"

initialize global pitch to "0"
initialize global heading to "120"
initialize global fov to "90"

to initDisplayParameters
do  set global mapViewMode to get global CANVASVIEW
    set BtnMapViewModeToggle . Text to get global mapViewMode

when btnMapType .BeforePicking
do  set btnMapType . Elements to make a list  get global ROADMAP
                                              get global SATELLITE
                                              get global HYBRID
                                              get global TERRAIN

when btnMapType .AfterPicking
do  set global mapType to btnMapType . Selection
    call Display

when BtnZoom .Click
do  set global zoom to get global zoom + 1
    call Display

when BtnUnzoom .Click
do  set global zoom to get global zoom - 1
    call Display

when BtnMapViewModeToggle .Click
do  if get global mapViewMode = get global CANVASVIEW
    then set global mapViewMode to get global WEBVIEW
         set CanvasMap . Visible to false
         set WebViewerMap . Visible to true
    else set global mapViewMode to get global CANVASVIEW
         set CanvasMap . Visible to true
         set WebViewerMap . Visible to false
    set BtnMapViewModeToggle . Text to get global mapViewMode
    call Display

to Display
do  initialize local URL to " "
    in  set URL to call httpMapURL
        if get global mapViewMode = get global CANVASVIEW
        then set CanvasMap . BackgroundImage to get URL
        else if get global mapViewMode = get global WEBVIEW
        then call WebViewerMap .GoToUrl url get URL
        else call Notifier1 .ShowAlert
                  notice join "Display called with unknown mode : "
                              get global mapViewMode
        set LblURL . Text to join "URL : " get URL

to httpMapURL
result join "https://maps.googleapis.com/maps/api/staticmap?"
            join "center=" get global centerLat "," get global centerLong
            join "&size=" get global displayWidth "x" get global displayHeight
            join "&zoom=" get global zoom
            join "&maptype=" get global mapType
            join "&markers=color:red%7C"
                 get global currentPlaceLat
                 ","
                 get global currentPlaceLong
```

5.5 Part 4: pan map horizontally by dragging on the map

Goal : pan map horizontally with finger dragging – in canvas mode

We can interact from our App with the canvas map window, with events (dragged, flung, …), add animations (balls, sprites). This is not the case with webviewer ☹ . Web interactions are nice ☺ but the app has lost control on window events …

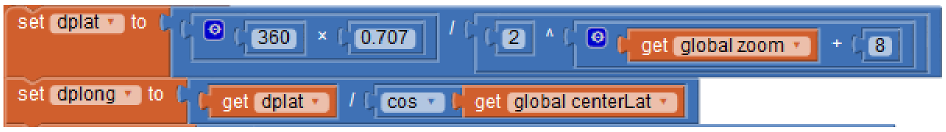
Matching pixel size with latitude and longitude differences

We need to know the pixel width and height (in degrees of latitude and longitude) to compute the change in latitude and longitude.
The **pixel height** dplat and **pixel width** dplong can be approximated by the following formulas :

$dp_{lat} = 360 * 0.7 * 2^{(zoom+8)}$ in degrees of latitude

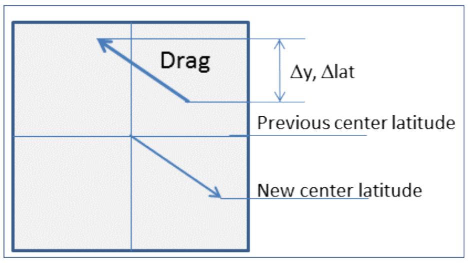
$dp_{long} = dp_{lat} * \cos(latitude)$ in degrees of longitude

or with the App Inventor language :



```
set dplat to ( 360 × 0.707 ) / ( 2 ^ ( get global zoom + 8 ) )
set dplong to get dplat / cos get global centerLat
```

(This is an estimate, let us know if you find better).



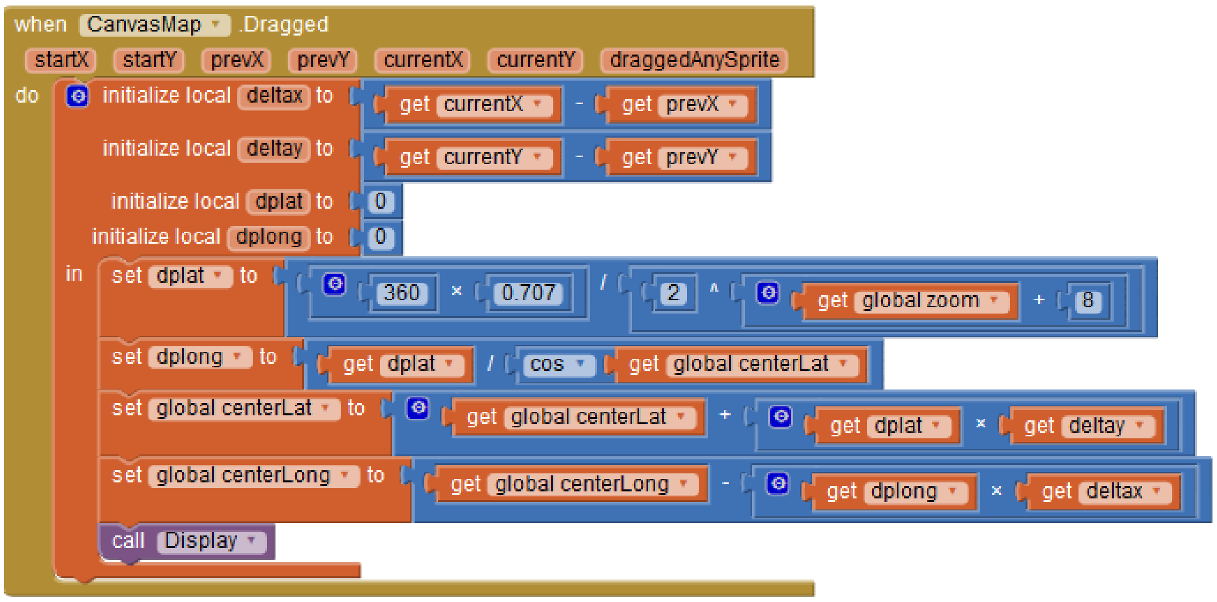
When we drag canvas by Δx, Δy pixels, the longitude and latitude changes (Δlong, Δlat) will be :
($\Delta long = \Delta x * dp_{lat}$, $\Delta lat = \Delta y * dp_{long}$)
in degrees which we will substract to the previous values, before calling **Display**.

5.5.1 TODO

1. Code the response to the **when canvasMap.dragged** event:
   - compute new coordinates of the center : centerLong and centerLat variables
     - compute drag coordinate differences deltax, deltay in pixel coordinates,
     - compute latitude and longitude difference for one pixel dplong, dplat in degrees of latitude and longitude (from above formula)
     - then multiply deltax by dplong and deltay by dplat
       substract deltax * dplong from centerLong
       add from deltay * dplat from centerLat,
       (remember y axis on tablet goes down from top)
   - call **Display**.

5.5.2 suggested solution

add following event handler (no other change)



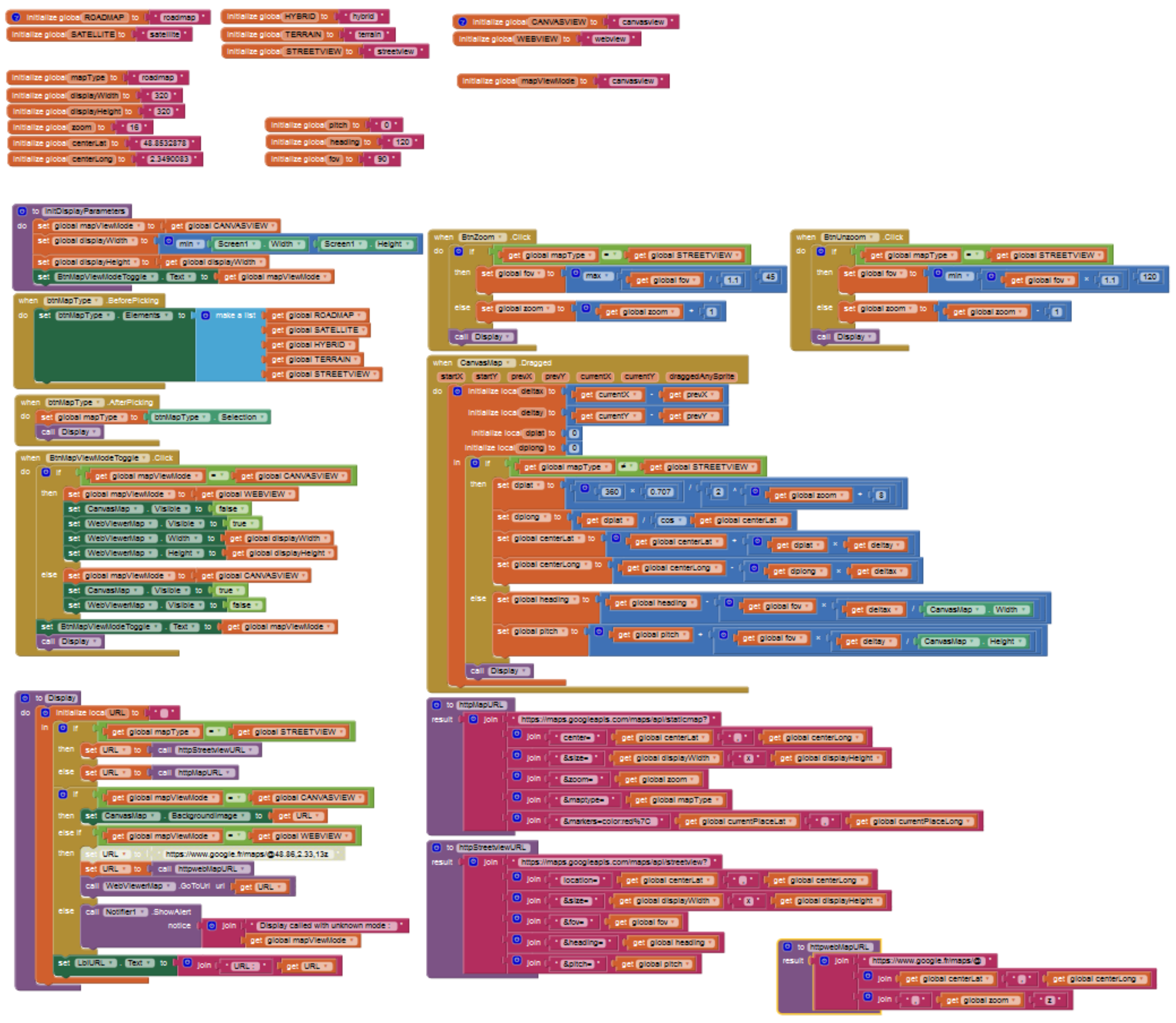
```
when CanvasMap .Dragged
  startX  startY  prevX  prevY  currentX  currentY  draggedAnySprite
do  initialize local deltax to  get currentX - get prevX
    initialize local deltay to  get currentY - get prevY
    initialize local dplat to  0
    initialize local dplong to  0
    in  set dplat to  360 × 0.707 / 2 ^ get global zoom + 8
        set dplong to  get dplat / cos get global centerLat
        set global centerLat to  get global centerLat + get dplat × get deltay
        set global centerLong to  get global centerLong - get dplong × get deltax
        call Display
```

5.6 Part 5 : add streetview display as a new map type

5.6.1 TODO

- Enable the STREETVIEW constant set to "streetview", and of the pitch, heading and fov variables with initial values (pitch =0 for horizontal , heading = 0 for north)
- add STREETVIEW to the list of maptypes (set btnMapType.elements)
- when btnZoom.Click : if in streetview mode, decrease fov by 10% (with 45° lower limit)
- when btnunZoom.Click : if in streetview mode, increase fov by 10% (with 120° upper limit)
- when canvasMapDragged : if in streetview mode, change heading and pitch
  substract to heading fieldOfView *deltax /canvaswidth
  add to pitch fieldOfView *deltay /canvaswidth
- Write the httpStreetViewURL procedure to build the URL with
  scheme, host and path : https://maps.googleapis.com/maps/api/streetview?
  then query parameters : location, size, fov (field of view), heading and pitch
- In **Display** Procedure, set URL to httpMapURL or httpStreetviewURL if maptype si STREETVIEW

5.6.2 suggested solution



```
initialize global ROADMAP to "roadmap"
initialize global SATELLITE to "satellite"
initialize global HYBRID to "hybrid"
initialize global TERRAIN to "terrain"
initialize global STREETVIEW to "streetview"
initialize global CANVASVIEW to "canvasview"
initialize global WEBVIEW to "webview"

initialize global mapType to "roadmap"
initialize global displayWidth to "320"
initialize global displayHeight to "320"
initialize global zoom to "16"
initialize global centerLat to "48.8532878"
initialize global centerLong to "2.3490083"
initialize global pitch to "0"
initialize global heading to "120"
initialize global fov to "90"
initialize global mapViewMode to "canvasview"

to initDisplayParameters
do  set global mapViewMode to get global CANVASVIEW
    set global displayWidth to min(Screen1.Width, Screen1.Height)
    set global displayHeight to get global displayWidth
    set BtnMapViewModeToggle.Text to get global mapViewMode

when btnMapType.BeforePicking
do  set btnMapType.Elements to make a list(get global ROADMAP, get global SATELLITE,
        get global HYBRID, get global TERRAIN, get global STREETVIEW)

when btnMapType.AfterPicking
do  set global mapType to btnMapType.Selection
    call Display

when BtnMapViewModeToggle.Click
do  if get global mapViewMode = get global CANVASVIEW
    then set global mapViewMode to get global WEBVIEW
         set CanvasMap.Visible to false
         set WebViewerMap.Visible to true
         set WebViewerMap.Width to get global displayWidth
         set WebViewerMap.Height to get global displayHeight
    else set global mapViewMode to get global CANVASVIEW
         set CanvasMap.Visible to true
         set WebViewerMap.Visible to false
    set BtnMapViewModeToggle.Text to get global mapViewMode
    call Display

to Display
do  initialize local URL to ""
    in  if get global mapType = get global STREETVIEW
        then set URL to call httpStreetViewURL
        else set URL to call httpMapURL
        if get global mapViewMode = get global CANVASVIEW
        then set CanvasMap.BackgroundImage to get URL
        else if get global mapViewMode = get global WEBVIEW
        then set URL to "https://www.google.fr/maps/@48.86,2.33,13z"
             set URL to call httpwebMapURL
             call WebViewerMap.GoToUrl url get URL
        else call Notifier1.ShowAlert
                 notice join("Display called with unknown mode : ", get global mapViewMode)
        set LblURL.Text to join("URL : ", get URL)

when BtnZoom.Click
do  if get global mapType = get global STREETVIEW
    then set global fov to max(get global fov / 1.1, 45)
    else set global zoom to get global zoom + 1
    call Display

when BtnUnzoom.Click
do  if get global mapType = get global STREETVIEW
    then set global fov to min(get global fov × 1.1, 120)
    else set global zoom to get global zoom - 1
    call Display

when CanvasMap.Dragged
  startX startY prevX prevY currentX currentY draggedAnySprite
do  initialize local deltax to get currentX - get prevX
    initialize local deltay to get currentY - get prevY
    initialize local dplat to 0
    initialize local dplong to 0
    in  if get global mapType ≠ get global STREETVIEW
        then set dplat to (360 × 0.707) / (2 ^ (get global zoom + 8))
             set dplong to get dplat / cos get global centerLat
             set global centerLat to get global centerLat + (get dplat × get deltay)
             set global centerLong to get global centerLong - (get dplong × get deltax)
        else set global heading to get global heading - (get global fov × (get deltax / CanvasMap.Width))
             set global pitch to get global pitch + (get global fov × (get deltay / CanvasMap.Height))
        call Display

to httpMapURL
result join("https://maps.googleapis.com/maps/api/staticmap?",
            join("center=", get global centerLat, ",", get global centerLong),
            join("&size=", get global displayWidth, "x", get global displayHeight),
            join("&zoom=", get global zoom),
            join("&maptype=", get global mapType),
            join("&markers=color:red%7C", get global currentPlaceLat, ",", get global currentPlaceLong))

to httpStreetViewURL
result join("https://maps.googleapis.com/maps/api/streetview?",
            join("location=", get global centerLat, ",", get global centerLong),
            join("&size=", get global displayWidth, "x", get global displayHeight),
            join("&fov=", get global fov),
            join("&heading=", get global heading),
            join("&pitch=", get global pitch))

to httpwebMapURL
result join("https://www.google.fr/maps/@",
            join(get global centerLat, ",", get global centerLong),
            join(",", get global zoom, "z"))
```

5.7 part 6 (optional) : display a list of markers on the map

Goal : add display of a marker for each place (with different color) in the **httpMessageMap** procedure.

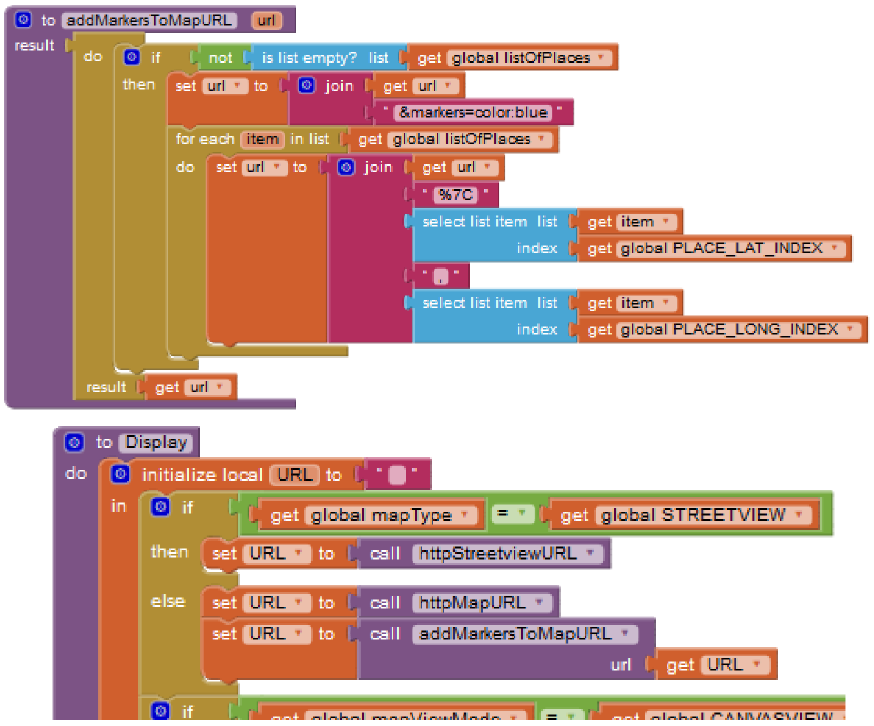
Note : We can do this with the URL or with image sprites (in which case you need to go from latitude, longitude to pixel coordinates, which is the reverse of what has been done previously when dragging). We will do it here with URL

5.7.1 TODO

- Create a procedure **addMarkersToMapURL** which returns the url with added markers (one for each place)
- Call this procedure after creating the basic httpMapURL

Note : this is not quite straightforward because we have used function which return a value (url) and we also avoided global variables

5.7.2 suggested solution



```
to addMarkersToMapURL url
result  do  if  not  is list empty? list  get global listOfPlaces
            then  set url to  join  get url
                                    "&markers=color:blue"
            for each item in list  get global listOfPlaces
            do  set url to  join  get url
                                  "%7C"
                                  select list item  list   get item
                                                    index  get global PLACE_LAT_INDEX
                                  ","
                                  select list item  list   get item
                                                    index  get global PLACE_LONG_INDEX
        result  get url

to Display
do  initialize local URL to " "
in  if  get global mapType = get global STREETVIEW
    then  set URL to  call httpStreetviewURL
    else  set URL to  call httpMapURL
          set URL to  call addMarkersToMapURL
                                          url  get URL
    if  get global mapViewMode = get global CANVASVIEW
```

5.8 part 7 (optional) : make your display routine generic and reusable

- Make the display procedure independent of the calling background, so that it can easily be used with some “interface documentation”

6 Appendix

6.1 Reminder on a few good practices

6.1.1 CONSTANTS and variables

Whenever a value (text or number) has an important significance or may be used as input besides display, define it as a variable or as constant if it does not change troughout the App. This will avoid spelling errors and will make your program safer and easier to read and debug.

Naming conventions :

- write constants in CAPITAL letters with ‘_’ to separate words,ex : MY_CONSTANT
- start variables with lower case letter, use upper case letter to separate words, ex : myVariable
- do not use spaces in variable names.

Practical application :

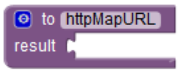
- Each place (in the list of places) is a list with name, latitude and longitude as indexes 1,2,3. These indexes should be initialized as global constants :
  PLACE_NAME_INDEX = 1, PLACE_LAT_INDEX =2 and PLACE_LONG_INDEX =3.

- To avoid errors when building URLs or when branching according to maptype or viewmode , we should alos define these values as constants :
  ROADMAP = roadmap, SATELLITE = satellite, HYBRID =hybrid , TERRAIN =terrain
  CANVASVIEW =canvasview, WEBVIEW = webview

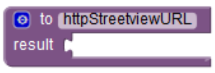
Names of these constants should be self explanatory, and there is no spelling mistakes possible.

6.1.2 Procedures an local variables

You should isolate functions in a procedure when the corresponding code :

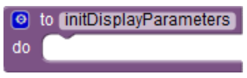
- is doing something important or special, such as building a URL in a function such as 

```
to httpMapURL
result
```

  or 

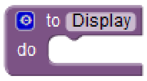
```
to httpStreetviewURL
result
```

  which will return the URL to the caller

- or has shared interest such as 

```
to initDisplayParameters
do
```

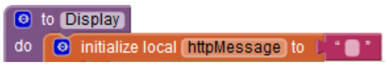
  which returns nothing but does what is necessary on startup for display. If something needs to be done once for display (such as taking into account tablet configuration), do it there. You will know where to look for afterwards,

- or is doing something with common interest or used often as will be the case for 

```
to Display
do
```

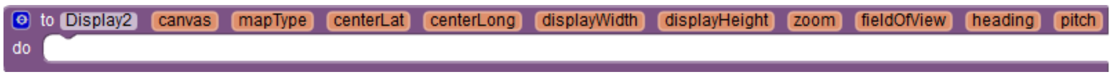
Defining procedures often goes with defining variables as local (inside a procedure) or global (i.e. known by all). Current practice for new programmers is to use global, but as you become a pro and want to share your code with others, think of the following :

- A global variable is known by all App components which may be necessary. But it may also be changed at anytime by any component. If this happens, you may be in trouble …because you may not know where to look to debug,

A local variable is protected from external access (which may not be adequate) but it is then protected. You should make this choice when possible. For example the URL or **httpMessage** variable is only necessary in the **Display** procedure and can be initialized at the beginning of this procedure : 

```
to Display
do  initialize local httpMessage to " "
```

Local variables will contribute to cleaner code, avoiding variables all over the place, and you do not remember which procedures uses them.
One way to switch from global to local variables is to use calling parameters. An example could be a display function on canvas which uses no global variables :



```
to Display2 canvas mapType centerLat centerLong displayWidth displayHeight zoom fieldOfView heading pitch
do
```

Think that if your procedure relies on NO external or global information, it will then be very easy to reuse by for friends.

6.1.3 Ordering blocks on screen

As you add on events, procedures and variables, your code becomes harder and harder to read., So that you have to find some kind of organization … You can agree on one or the other with people you work with. What's most important is to share the same principles when working together. The following principles have been used for the current lesson :

Organize blocks by domain from left to right :

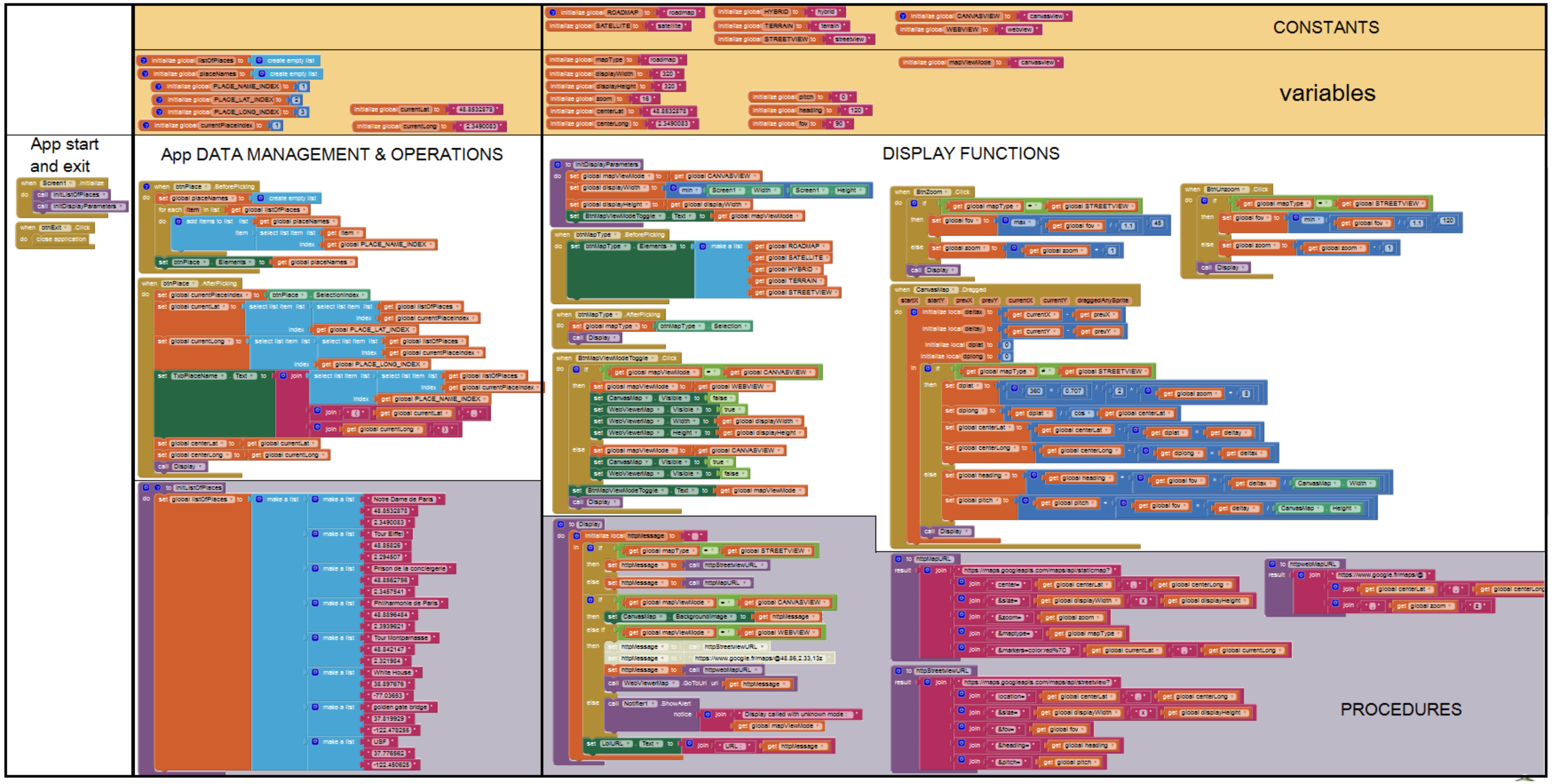
- Application data management (i.e. places) : initialize, backup, access, selection
- application management : startup and initialize and exit App
- Display functions
- GPS functions (not in this lesson) …

Then from top to bottom

- Initialize CONSTANTs
- Initialize global variables
- Events
- Procedures[2]

The next figure shows a possible block organization for the result of this lesson : It reads from left to right and from top to bottom, with constants and variables placed on top of the domain where they are used and procedures below events which uses them.

[2] The init procedure for each domain may be practical above events because it comes first and only once.



CONSTANTS

```
initialize global ROADMAP to "roadmap"
initialize global SATELLITE to "satellite"
initialize global HYBRID to "hybrid"
initialize global TERRAIN to "terrain"
initialize global STREETVIEW to "streetview"
initialize global CANVASVIEW to "canvasview"
initialize global WEBVIEW to "webview"
```

variables

```
initialize global listOfPlaces to create empty list
initialize global placeNames to create empty list
initialize global PLACE_NAME_INDEX to 1
initialize global PLACE_LAT_INDEX to 2
initialize global PLACE_LONG_INDEX to 3
initialize global currentPlaceIndex to 1
initialize global currentLat to "48.8532878"
initialize global currentLong to "2.3490083"
initialize global mapType to "roadmap"
initialize global displayWidth to "320"
initialize global displayHeight to "320"
initialize global zoom to "16"
initialize global centerLat to "48.8532878"
initialize global centerLong to "2.3490083"
initialize global pitch to "0"
initialize global heading to "120"
initialize global fov to "90"
initialize global mapViewMode to "canvasview"
```

App start and exit

```
when Screen1 .Initialize
do  call InitListOfPlaces
    call InitDisplayParameters

when btnExit .Click
do  close application
```

App DATA MANAGEMENT & OPERATIONS

```
when btnPlace .BeforePicking
do  set global placeNames to create empty list
    for each item in list get global listOfPlaces
    do  add items to list  list get global placeNames
                           item select list item  list get item
                                                  index get global PLACE_NAME_INDEX
    set btnPlace . Elements to get global placeNames

when btnPlace .AfterPicking
do  set global currentPlaceIndex to btnPlace . SelectionIndex
    set global currentLat to select list item  list select list item  list get global listOfPlaces
                                                                      index get global currentPlaceIndex
                                               index get global PLACE_LAT_INDEX
    set global currentLong to select list item  list select list item  list get global listOfPlaces
                                                                       index get global currentPlaceIndex
                                                index get global PLACE_LONG_INDEX
    set TxbPlaceName . Text to join  select list item  list select list item  list get global listOfPlaces
                                                                              index get global currentPlaceIndex
                                                       index get global PLACE_NAME_INDEX
                                     join  " ( "  get global currentLat  " , "
                                     join  get global currentLong  " ) "
    set global centerLat to get global currentLat
    set global centerLong to get global currentLong
    call Display

to InitListOfPlaces
do  set global listOfPlaces to make a list
        make a list  "Notre Dame de Paris"  "48.8532878"  "2.3490083"
        make a list  "Tour Eiffel"  "48.85826"  "2.294507"
        make a list  "Prison de la conciergerie"  "48.8562756"  "2.3457541"
        make a list  "Philharmonie de Paris"  "48.8896484"  "2.3939621"
        make a list  "Tour Montparnasse"  "48.842147"  "2.321984"
        make a list  "White House"  "38.897676"  "-77.03653"
        make a list  "golden gate bridge"  "37.819929"  "-122.478255"
        make a list  "USF"  "37.776562"  "-122.450625"
```

DISPLAY FUNCTIONS

```
to InitDisplayParameters
do  set global mapViewMode to get global CANVASVIEW
    set global displayWidth to min  Screen1 . Width  Screen1 . Height
    set global displayHeight to get global displayWidth
    set BtnMapViewModeToggle . Text to get global mapViewMode

when btnMapType .BeforePicking
do  set btnMapType . Elements to make a list  get global ROADMAP
                                              get global SATELLITE
                                              get global HYBRID
                                              get global TERRAIN
                                              get global STREETVIEW

when btnMapType .AfterPicking
do  set global mapType to btnMapType . Selection
    call Display

when BtnMapViewModeToggle .Click
do  if  get global mapViewMode = get global CANVASVIEW
    then  set global mapViewMode to get global WEBVIEW
          set CanvasMap . Visible to false
          set WebViewerMap . Visible to true
          set WebViewerMap . Width to get global displayWidth
          set WebViewerMap . Height to get global displayHeight
    else  set global mapViewMode to get global CANVASVIEW
          set CanvasMap . Visible to true
          set WebViewerMap . Visible to false
    set BtnMapViewModeToggle . Text to get global mapViewMode
    call Display

when BtnZoom .Click
do  if  get global mapType = get global STREETVIEW
    then  set global fov to max  get global fov / 1.1   45
    else  set global zoom to get global zoom + 1
    call Display

when BtnUnzoom .Click
do  if  get global mapType = get global STREETVIEW
    then  set global fov to min  get global fov / 1.1   120
    else  set global zoom to get global zoom - 1
    call Display

when CanvasMap .Dragged
  startX  startY  prevX  prevY  currentX  currentY  draggedAnySprite
do  initialize local deltax to get currentX - get prevX
    initialize local deltay to get currentY - get prevY
    initialize local dplat to 0
    initialize local dplong to 0
    in  if  get global mapType ≠ get global STREETVIEW
        then  set dplat to ( 360 × 0.707 ) / 2 ^ ( get global zoom + 8 )
              set dplong to get dplat / cos get global centerLat
              set global centerLat to get global centerLat - ( get dplat × get deltay )
              set global centerLong to get global centerLong - ( get dplong × get deltax )
        else  set global heading to get global heading + ( get global fov × ( get deltax / CanvasMap . Width ) )
              set global pitch to get global pitch + ( get global fov × ( get deltay / CanvasMap . Height ) )
        call Display
```

```
to Display
do  initialize local httpMessage to " "
    in  if  get global mapType = get global STREETVIEW
        then  set httpMessage to call httpStreetviewURL
        else  set httpMessage to call httpMapURL
        if  get global mapViewMode = get global CANVASVIEW
        then  set CanvasMap . BackgroundImage to get httpMessage
        else if  get global mapViewMode = get global WEBVIEW
        then  set httpMessage to call httpStreetviewURL
              set httpMessage to "https://www.google.fr/maps/@48.86,2.33,13z"
              set httpMessage to call httpwebMapURL
              call WebViewerMap .GoToUrl  url get httpMessage
        else  call Notifier1 .ShowAlert
                notice join  "Display called with unknown mode : "
                             get global mapViewMode
        set LblURL . Text to join  "URL : "  get httpMessage
```

PROCEDURES

```
to httpMapURL
result join  "https://maps.googleapis.com/maps/api/staticmap?"
             join  "center="  get global centerLat  " , "  get global centerLong
             join  "&size="  get global displayWidth  "x"  get global displayHeight
             join  "&zoom="  get global zoom
             join  "&maptype="  get global mapType
             join  "&markers=color:red%7C"  get global currentLat  " , "  get global currentLong

to httpStreetviewURL
result join  "https://maps.googleapis.com/maps/api/streetview?"
             join  "location="  get global centerLat  " , "  get global centerLong
             join  "&size="  get global displayWidth  "x"  get global displayHeight
             join  "&fov="  get global fov
             join  "&heading="  get global heading
             join  "&pitch="  get global pitch

to httpwebMapURL
result join  "https://www.google.fr/maps/@"
             join  get global centerLat  " , "  get global centerLong
             join  " , "  get global zoom  "z"
```

6.2 General Definitions

6.2.1 Web API : Application Programming Interface (written from Wikipedia)

An API is a set of routine definitions, protocols, and tools for building software and applications. It defines the functionalities that an external software component (ex : web services) makes available with its operations, inputs, outputs, …
APIs are one of the most common ways technology companies integrate with each other.

When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format. (note : for the static maps API, the response is an image).

In our case, it will be more simple, staticmaps and streetview APIs will return images.

"Web API" historically has been virtually synonymous for web service.

6.2.2 What is a URL (Uniform Resource Locator) and how to build it

A URL, or commonly a web address, is a reference to a web resource that specifies its location on the network and a mechanism for retrieving it (a URL is a form of URI). URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and many other applications.

URLs were defined in 1994 by Tim Berners-Lee, the inventor of the World Wide Web.

HTTP URLs conforms to the following generic (URI) form :
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

It comprises :

- a **scheme** : here ‘http’, but may be : ‘ftp’, ‘mailto’, ‘file’ and ‘data’
- **//** : required by some schemes. When authority is absent, path cannot begin with //
- an authority with :
  - optional authentication: user name and password, separated by a colon, then @
  - "**host**", consisting of a registered name (or an IP address)
  - optional port number, separated from the hostname by a colon
- a **path**, to the data, usually in hierarchical form, as segments separated by slashes. The path must begin with a single slash (/) if an authority part was present, and may also if not,
- an optional **query**, separated from the preceding part by a question mark (?). see below.
- an optional **fragment** (not used here).

for example :

https://maps.googleapis.com/maps/api/staticmap?center=37.7766,-122.4506&size=320x320&zoom=14&maptype=roadmap

scheme,//,   host   ,   path   ,   query

6.2.3 the Query string within a URL : (written from Wikipedia)

The query string is the part of a URL, which contains data that does not fit conveniently into a hierarchical path structure, and specifies user's criteria. The query string commonly includes fields added to a base URI or URL. Its syntax is not well defined, but is most often a sequence of attribute–value pairs separated by a delimiter.

When a form containing the fields field1, field2, field3 is submitted, the content of the fields is encoded as a query string as follows: *field1=value1&field2=value2&field3=value3...*

For example in the following URL (which you can copy/paste into your web browser address), the query string is what follows the question mark and the delimiter between attribute-value pairs is '&'

https://maps.googleapis.com/maps/api/staticmap?center=37.777,-122.451&size=320x320&zoom=14&maptype=roadmap

- The query string is composed of a series of field-value pairs.
- Within each pair, the field name and value are separated by an equals sign, '='.
- The series of pairs is separated by the ampersand, '&'

When fields have multiple values a delimiter is used. For example the '|' character separates coordinate values in the static maps API, in a list of "markers" coordinates :

&markers=color:blue|label:x|37.78,-122.46|37.77,-122.46|37.78,-122.45

6.2.4 the Query string within a URL : (written from Wikipedia)

The query string is the part of a URL, which contains data that does not fit conveniently into a hierarchical path structure, and specifies user's criteria. The query string commonly includes fields added to a base URI or URL. Its syntax is not well defined, but is most often a sequence of attribute–value pairs separated by a delimiter.

When a form containing the fields field1, field2, field3 is submitted, the content of the fields is encoded as a query string as follows: *field1=value1&field2=value2&field3=value3...*

For example in the following URL (which you can copy/paste into your web browser address), the query string is what follows the question mark and the delimiter between attribute-value pairs is '&'

https://maps.googleapis.com/maps/api/staticmap?center=37.777,-122.451&size=320x320&zoom=14&maptype=roadmap

- The query string is composed of a series of field-value pairs.
- Within each pair, the field name and value are separated by an equals sign, '='.
- The series of pairs is separated by the ampersand, '&'

When fields have multiple values a delimiter is used. For example the '|' character separates coordinate values in the static maps API, in a list of "markers" coordinates :

&markers=color:blue|label:x|37.78,-122.46|37.77,-122.46|37.78,-122.45

6.2.5 URL encoding

Some characters cannot be part of a URL (for example, the space) and some other characters have a special meaning in a URL: for example, the character # can be used to further specify a subsection (or fragment) of a document. In HTML forms, the character = is used to separate a name from a value. The URI generic syntax uses URL encoding to deal with this problem, while HTML forms make some additional substitutions rather than applying percent encoding for all such characters. SPACE is encoded as '+' or "%20".

- Letters (`A–Z` and `a–z`), numbers (`0–9`) and the characters '`*`','`-`','`.`' and '`_`' are left as-is
- SPACE is encoded as '`+`' or '`%20`'
- All other characters are encoded as `%HH` hex representation

Values for reserved characters after percent-encoding

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Values for other characters after percent-encoding (neither reserved or non reserved)

newline	space	"	%	-	.	<	>	\	^	_	`	{	|	}	~
%0A *or* %0D *or* %0D%0A	%20	%22	%25	%2D	%2E	%3C	%3E	%5C	%5E	%5F	%60	%7B	%7C	%7D	%7E

Characters outside the list of NON-reserved character (`A–Z` and `a–z`, numbers `0–9` , '`*`','`-`','`.`','`_`' see above) should be encoded. Reserved characters MUST be.

For the following example '|' should be encoded to %7C (but it seems to work in App inventor if you do not). On the contrary, if there is a blank space in the URL, it will not (try it).

Note : web browsers are more tolerant … they may remove inadequate "space". A wrong message may work with your browser and not in your app inventor program. Use your text editor to show hidden characters and build your URI step by step.

Sample URL with multiple markers (added linefeeds to be removed):

```
https://maps.googleapis.com/maps/api/staticmap?
center=37.7766,-122.4506
&size=320x320&zoom=14
&maptype=roadmap
&markers=color:blue|label:x|37.78,-122.46|37.77,-122.46|37.78,-122.45|37.77,-122.45|37.78,-122.44|37.77,-122.44
```