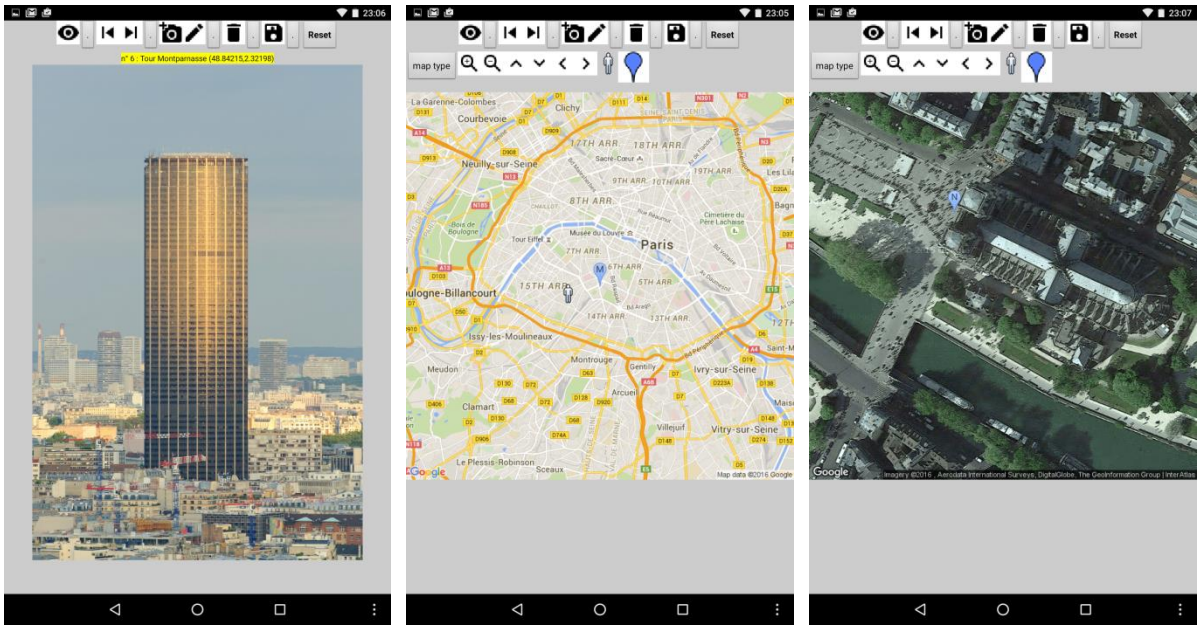
Image and maps display Tutorial with App Inventor



Quick view on resulting app

Image and maps display Tutorial with App Inventor

Overview :

This app displays the pictures of monuments or places of interest and their location on maps or satellite images with the user's position or street-view display.
This document reviews design and development steps and attempts to point at good practices.
It illustrates simultaneous use of multiple App inventor components and external services, and tries to provide enough detail for reuse by teachers' in projects with students.

Trainer's goal :

- Help participants experiment the complete process of an application development from initial design to final tests.
- Increase the participants' practical understanding of the importance of
  - initial conception and of MVP design,
  - key development rules including use of procedures, modular design, local rather than global variables, inclusion of tests, etc.
- Show the practical connection between methods and lower risk for errors, and easier debugging.
- Increase the participants ease of use of app inventor
  - internal resources such as TinyDB, camera, location sensors, (activity starter),
  - or external services such as the Google static Map API which imply to search, read (and discuss) documentation and determine how to use it.

Learning objectives : Participants will be able to :

- paper prototype, make updates to the initial analysis, check emphasis,
- understand the role of conception,
- search for external documentation,
- check the overall benefit of initial conception and "good practices" even if they do not initially fully see why.

Unit outline :

- Paper prototyping : 2 h
- Chapter 1 : 2h - display list of monument pictures
- Chapter 2 : 2h - delete/create/update uses the location sensor & camera
- Chapter 3 : 2h - with display of the monument's and user position on a background map or satellite image with zoom and pan functions or centering on the user's or monument's location.
- Chapter 4 : 2h – streetview and improvements
- Other : zoom on location configurations and functions (GPS, network)
- Discussion and debrief

Content :

Current version of this app is in App Inventor Gallery (search for : "image and mapping" or author "pierre.huguet50") with current .apk and .aia

You may also start or restart from intermediate versions corresponding to the end of each chapter according to teaching purposes.

Note :

This tutorial is a draft to be completed … It will probably be broken down into separate "smaller pieces" to be easier to use.

1. Introduction

Pictures, maps and aerial or satellite images are basic blocks for many apps, in connection with camera and location sensors (GPS, Wifi, GSM). This tutorial will review common methods to acquire and display pictures and handle maps in a mobile background.

This tutorial is provided with the sample app for each step at this link : (TBD)

and you may follow the same tutorial on videos at this link : (TBD)

2. What you'll Build

You will build an application which acquires and displays pictures of monuments and their location on maps or satellite images with interactive change of position (panning), scale (zoom) or map type. You will :

- (chapter 1) view monument pictures from a list provided with the app data,
- (chapter 2) update this list, delete and add new monuments with their location and picture from smartphone sensors (GPS, Wifi, …) and camera,
- (chapter 3) query and display maps and satellite imagery from web services, handle the zoom factor, move horizontally (panning) and change the map type, with added markers showing the location of monuments as well as yours.

You may later add to this base with information on the monuments with audio, video, … measurement of distances, call for new services, … with no limit but your imagination.

3. What you'll Learn

- Draft the app with paper prototyping,
- Upload images and icons at App development stage,
- Load an image from the camera,
- ~~Load an image from smartphone gallery,~~
- Refer to an image location on the web,
- ~~Load an image from the web,~~
- Read and process the smartphone location sensors,
- Use a Web server API with HTTP requests for maps and handling of :
    - location, scale or zoom factor,
    - map type,
    - overlays with markers for your position and other objects,
- Map pixel size,
- Interactive management of the map.

4. General app design : MVP and paper prototyping

First thing to do is to draft a general view of the application.

You should decide what is your "Minimal Viable Product" or MVP which is the minimum that you need to include in your app to start testing with a prototype. Then draw the screens on paper (e.g. "paper prototyping"), show them to your friends (as if they were potential customers) and explain how your app works :

Is it clear for them ? do they like it ?
would they use it they had it ?
what would they like that is not there ?
what is there that they do not understand or would not use ? …

Stick to your main goals but tune according to their comments, until they understand and like it. This paper prototyping will also give you a good view of what you will need such as the list of components, variables, etc.

Time that you spend now will be useful later to remember where you are when coding details. This will also avoid major design errors which could impact the structure of the app.

It is a good practice to have a prototype showing key concepts as early as possible, rather than wait for a full app. Do what's important and easy first, then come back after feedback.

Here is an example of what may come out and that we will use as our guideline.

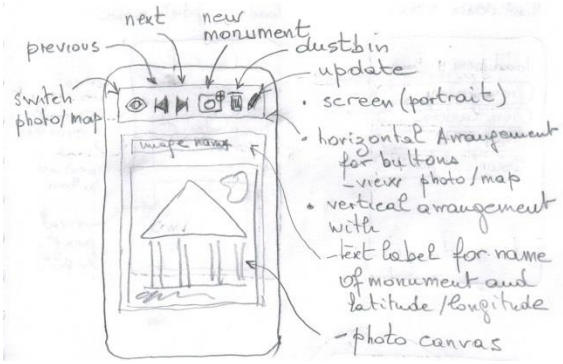
Main monument view : Screen state 1 shows :

A top main horizontal arrangement with

- buttons/icons to switch from the photo to map display, switch to previous or next monument in the list, create, delete or update a new monument item.

A vertical picture arrangement which contains :

- a text label with name and latitude/longitude of the monument,
- a canvas to display the picture of each monument.

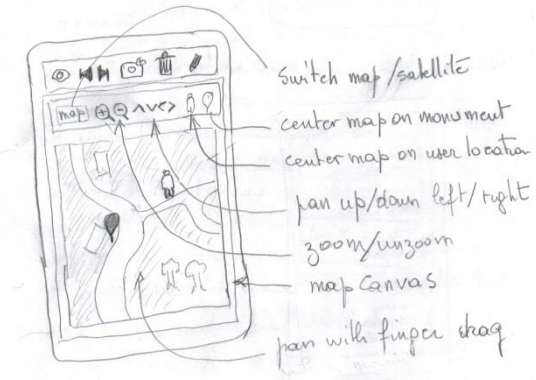


When clicking on the first icon (or on the photo itself) we switch to the

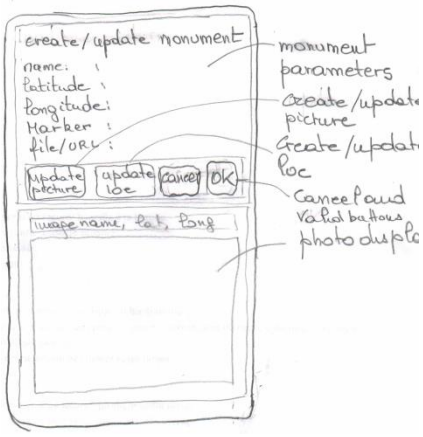
Map view : Screen state 2 contains

Below the main buttons arrangement (which could be hidden), a vertical map arrangement becomes visible and contains :

- an horizontal arrangement with button/icons to handle map/image display
  - 1 toggles from map to image
  - 2-3 zoom in and out
  - 4-5-6-7 pan up/down, left/right
  - 8 center map on user's location
  - 9 center map on monument's location
- an image canvas to hold the map or image



When clicking on the new  or update  icons in the monument view we switch to a “create/update” screen configuration : which shows :



- a vertical arrangement to edit monument parameters with
  - text title label
  - monument name, latitude, longitude, marker and picture url (or file)
  - an horizontal arrangement with 3 buttons to
    - update picture
    - update localization from smartphone sensors
    - cancel update
- the same vertical photo arrangement as the monument view (if photo available).

Picture acquisition screen : when the update picture button is clicked

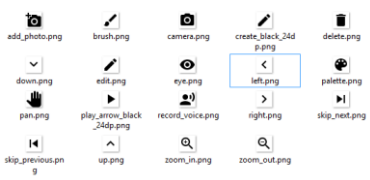
- a vertical arrangement with different picture sources buttons appears
  - picture from camera,
  - ~~load picture from gallery,~~ (to do later)
  - ~~picture as web reference,~~ (to do later)
  - ~~download picture from web.~~ (to do later)

5. Getting started / data preparation

Prepare an initial list of monuments with their name, latitude and longitude and a picture (preferably in portrait mode) with a size which should preferably stay around 512 pixels. You can easily obtain the latitude or longitude of a place from services on the web (ex : http://www.coordonnees-gps.fr/)

name :	Tour Eiffel	Notre Dame de Paris	Prison de la conciergerie
latitude :	48.85826	45.8532788	48.8562756
longitude :	2.294507	2.3490083	2.3457541
Marker (letter) :	T	N	C
filename or URL :	TourEiffel.jpg	NotreDameDeParis.jpg	conciergerie.jpg
			

Prepare icons which you will need from a source in the public domain or Creative Commons. ex : https://design.google.com/icons/. (Use the png format, not svg).



Connect to App inventor web site and start a new project call it “monuments” and also name the screen title “monuments”

6. Designing the components

Development will be incremental but since we have a global view with paper prototyping we can try to list and include most of the components we should need :

External or hidden resources

Component type	Palette Group	What you will name it	What do we need it for
Notifier	User Interface	Notifier1	Display warnings, debug
Camera	Media	Camera1	Take monument’s pictures
Location Sensor	Sensors	PositionSensor1	Read position from smartphone (GPS or other sources)
TinyDB	Storage	TinyDB1	Store and restore list of monuments
Activity starter	Connectivity	ActivityStarter1	Download image files from the net

Components for User Interface display (chapters 1 and 2)

Component type	Palette Group	given name	icon	What do we need it for
Screen1				
TOP MENU				
Horizontal arrangement	Layout	HarTopButtons		Insert buttons to display photographs
Button	User Interface	BtnMonumentMap		Toggle between photo and map display
Button	User Interface	BtnPrevious		Display previous monument in list
Button	User Interface	BtnNext		Display next monument in list
Button	User Interface	BtnNew		Add a new monument to the list
Button	User Interface	BtnDelete		Delete current monument from the list
Button	User Interface	BtnEdit		Edit Monument properties
Button	User Interface	BtnSave		Save monuments to TinyDB
MONUMENT DISPLAY				
Vertical arrangement	Layout	VarMonumentDisplay		Display monument pictures and info
Label	User Interface	LblMonument		Show name, latitude and longitude
Canvas	Drawing and animation	CanvasMonument		Display photo of monument

Components for User Interface display for chapter 3

(you may skip at first, but it can be a good idea for a global view with default data)

MAP DISPLAY				
Vertical arrangement	Layout	VarMapDisplay		Display and handle maps
Horizontal arrangement	Layout	HarMapButtons		Show map display buttons
Button	User Interface	BtnMapType	text	Toggle roadmap/satellite
Button	User Interface	BtnZoom		Zoom in
Button	User Interface	BtnUnzoom		Zoom out
Button	User Interface	BtnUp		Pan up/north
Button	User Interface	BtnDown		Pan down (south)
Button	User Interface	BtnLeft		Pan left (west)
Button	User Interface	BtnRight		Pan right (East)
Button	User Interface	BtnHere		Center map on my location
Button	User Interface	BtnLocMonum		Center map on monument's location

It is good practice to keep the UI component's type in the component's name, so that you easily know what you are dealing with when coding blocks. We have used "Btn" for buttons, "Har" for horizontal arrangements, "Var" for vertical arrangements.
You may prefer other conventions, what is important is that you have one which is easy to remember, explain and share.
Think that you might share your code and others will need to understand what you have done. Use coding conventions at group level. This will make development a better experience.

7. Chapter one: Start and display monuments' pictures

7.1. Global data definitions / initialization at application start

Application starts with definitions (or initialization) of global variables which will be shared by all groups of program blocks (each one starting with an event : event handlers).

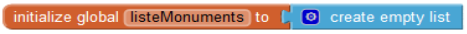
The constants and variables we need to initialize are the following

Variable name	Definition	Comment
listMonuments	List of monuments	Documents are list themselves
indexMonument	Index of current monument	Index of the monument which is currently being displayed or processed
monument 💣 dangerous use … see discussion	List which defines a sample document	This variable is mainly used for temporary storage of the current document, so as to avoid navigation in lists of lists. The problem with this variable is its possible use as a shortcut to existing content which may be changed outside user's intent
monumentDisplayMode	Boolean which indicates the current display state	True if current display state shows monument picture. False otherwise e.g. map display
pictureURL	Text variable for picture URL	This variable is used for temporary storage of the URL when looking for pictures on the web (this may also contain a local file name)
checked	Boolean variable	This variable is used temporarily to check the validity of new input
Constant name	**Definition**	**Comment**
INDEX_NAME	Index of monument LIST holding the name as text	Common name of the monument
INDEX_LAT	Index of monument LIST holding the Latitude as real number	Latitude of the monument
INDEX_LONG	Index of monument LIST holding the Longitude as real number	Longitude of the monument
INDEX_MARKER	Index of monument LIST holding the character to used as a marker	Used when displayed on the map
INDEX_FILE	Index of monument LIST holding the file name or URL	URL or File name. should be jpg, png of gif format

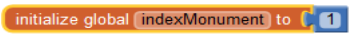
✋ It is good practice to name constants with CAPITAL LETTERS and to start variables with lower-case letters with capital letters used for visual separation of Words (Never use blanks, use under-score if you need).

This leads to the following **initialize** blocks :

- The list of monuments



- The index for the current document item



Each monument is defined itself as a list with name, latitude, longitude, marker type and file or URL. This list may change as we add new information. (It is a good practice to define constants to hold the index of each parameter, so that we will only have to change it at a single place if needed).

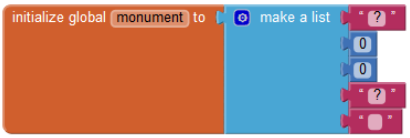
1 for name index

2 for latitude index

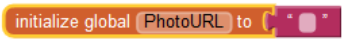
3 for longitude index

4 file (or URL) index

We may also use of single **monument** variable to hold a temporary monument (we will see however that this is dangerous, and it will be safer to handle temporary lists as local variables).

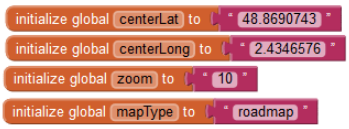


And when searching on the web we will use a variable for URL.



For chapter 3 with the map and satellite image display we will need other variables:

- Center latitude,
- Center longitude,
- Zoom factor (scale),
- Maptype (roadmap or satellite),



then

- HTTP request message (sent to the Map Web server),
- and my current latitude and longitude.

For safety reasons (to avoid mistakes) we will use 2 constants SPACE and NULL which look the same in the **text blocks** but are different. (but we will define them as local variables whennecessary.

We will also need a global "checked" used to control the validity of new input.

7.2.Initialization at startup & Save on exit

When starting, the application has to recover previous or default data and check that everything is in good shape. We do this when first screen is initialized screen1.Initialize.

It is good practice to deal at the same time with what happens on application start up and application exit screen1.Backpressed because what you catch when starting is what you saved before exit. These program blocks should be consistent : keep them close to each other on your screen.

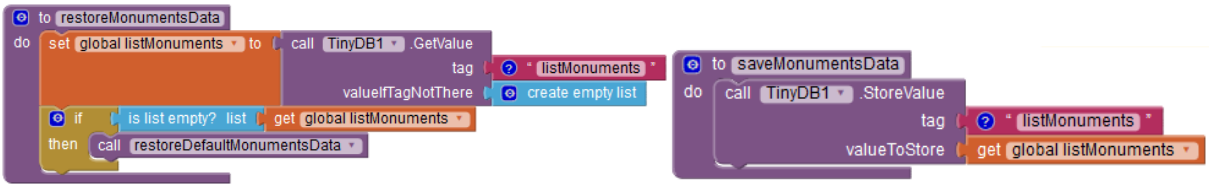
We will also deal with a "save" function to save the list of monuments at anytime.

To keep things manageable for future changes, we will write a procedure **storeMonumentsData** to save data on TinyDB and **restoreMonuments Data** to retrieve it from TinyDB.

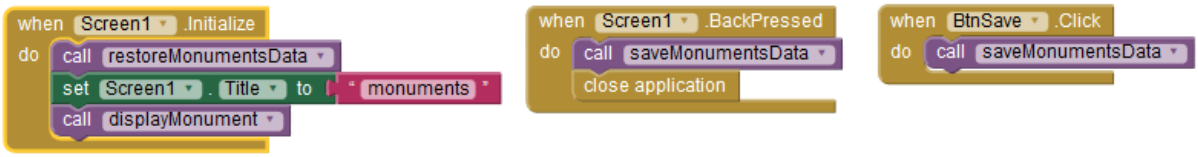
It is good practice to put code in a separate procedure (as if subcontracted) as soon as the blocks do a consistent set of operations which may be used several times, may require dedicated debugging, or may change with time. This will make debugging easier and group programming possible. Keep track of procedures and what they do in your paper files.

To save the information on monuments, we will use TinyDB which handles tag/value pairs with 2 procedures :

- Procedure **storeMonumentsData** uses the TinyDB1. StoreValue to store the list of monuments "listMonuments" under the "listMonuments" tag.
- Procedure **restoreMonuments Data** sets the "**listMonuments**" variable with the result of TinyDB1. GetValue with the "listMonuments" tag or an empty list if not found. I then checks if the list is empty and if yes replaces the list of monuments with a default preset list in the app.



These procedures will simply be called at screen startup, upon exit (backpress) or on demand :



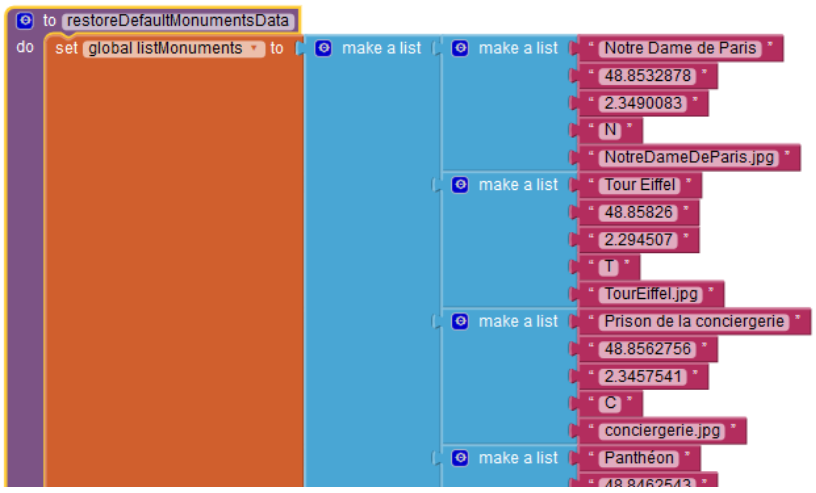
We will however disable the exit function during development time because the **close application** block is not supported in debug mode and we do not want to lose time with this. For now, we will save our list with the "save" button.

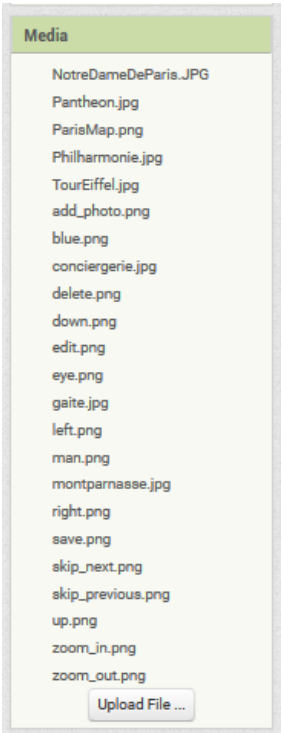
The third procedure **restoreDefaultMonumentsData** is called by **restoreMonuments Data** when nothing is found in the database. The corresponding data should have been prepared (see above § 5) and images preloaded into the app at development time.

Do not to forget ! in case the files are on your phone, but not loaded into the app (and .apk), the app will work on your phone, but not on others …(I did the mistake … :=)

Trying the app on a phone which has not been used for development is a safe and recommended step before app broadcast.

The initialization block which sets the corresponding data is illustrated below for its beginning. And you can check in the components that the corresponding files have been loaded into the app.





Note : Image files which are not stored with TinyDB should remain available (as local file or URL)

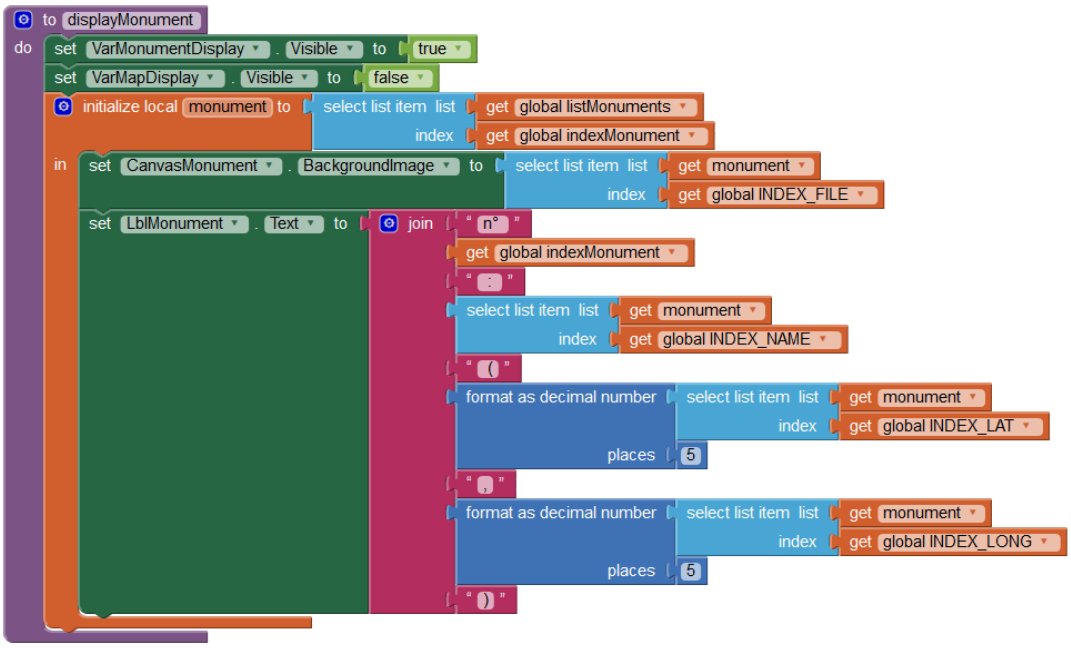
7.3. Write a procedure to display the monument's picture

Display of the current monument's picture occurs at startup and will occur many times later. We may also want to customize it, add information, audio, video ... and who knows what you will imagine.

For this we need a unique "displayMonument" procedure to display the current monument, e.g. monument at index "indexMonument" in the list of monuments : "listMonuments" .

What we have done in the components design is to group all this information in a vertical arrangement called "VarMonumentDisplay" and we have grouped the Map display in the horizontal arrangement called "VarMapDisplay". We will often switch from to the other and the routine will :

- Set the map display off (in case it was visible)
- Set the document display on (in case it was off)
- Copy the current document from the list into the temporary "monument" variable
- Set the canvas image "canvasMonument.backgroundImage" to the monument image file (which is at index defined at index "INDEX_FILE")
- Update the information displayed in the label "LblMonument" with the name of the monument (INDEX_NAME), the latitude (INDEX_LAT) and the longitude (INDEX_LONG) with trimming to 5 decimals for a resolution around 1 m, which is far enough.



Note : the above block shows the monument variables as a local instead of global variable (used in early versions of the code). Both worked, but this is a security improvement which will be discussed later.

7.4. Navigate form one monument to the next or previous

OK, now we have this first monument displayed, but nothing to do …

Let us begin with writing the code to switch display to previous or next monuments with the previous button btnPrevious ⏮ and next button and BtnNext ⏭.

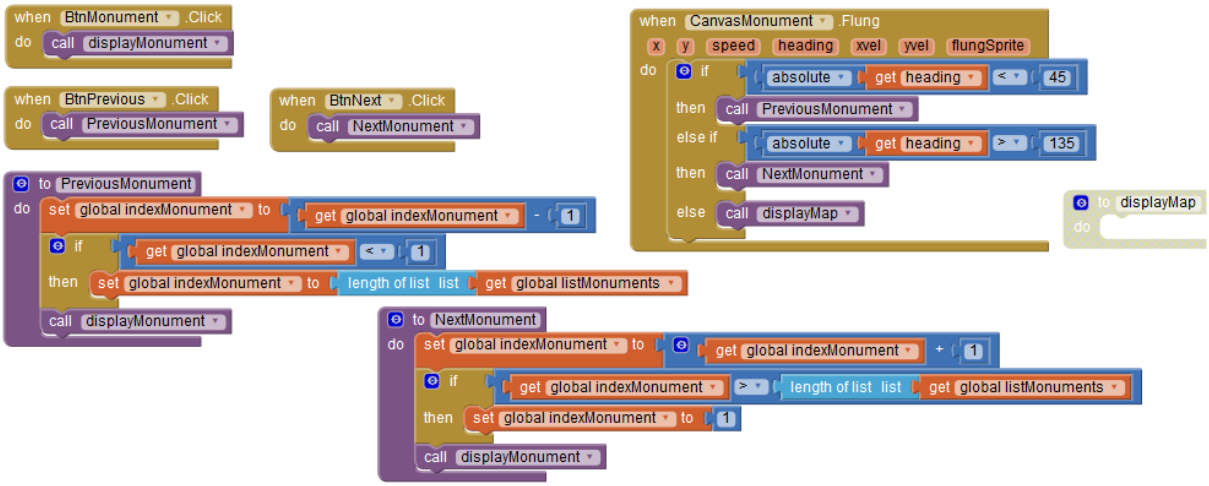
This may be called many times so we will write 2 procedures : previousMonument & Next Monument

Try to code these routines, then check and continue reading … :

What we have to do for previous is substract 1 to the monument index, and if smaller than 1, set it to the length of the list and call displayMonument.

Similarly for next, add 1 to the monument index, and if greater than length of the list, set it to 1, and call displayMonument.

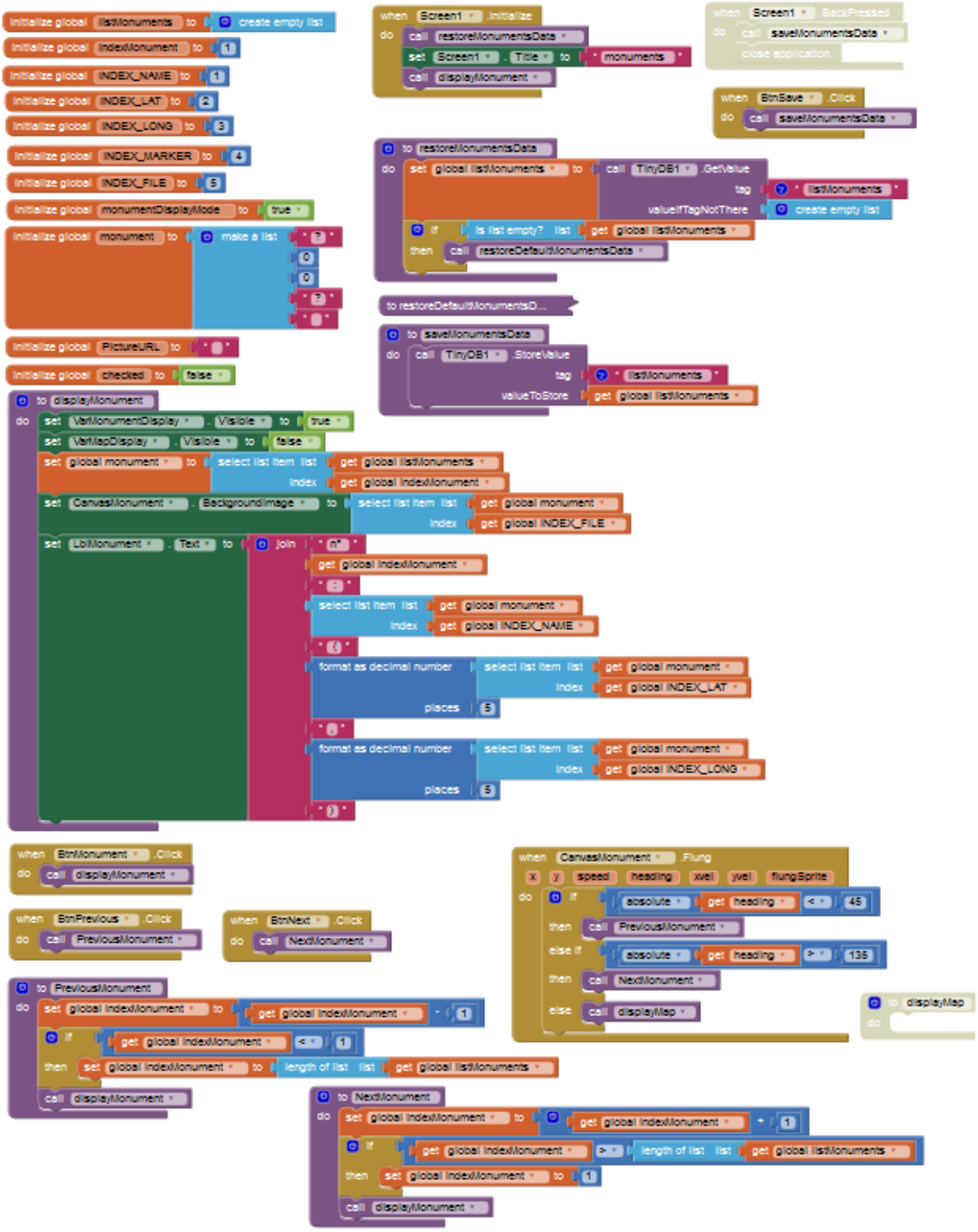
To make it easier for big fingers on small icons, we will also swap images when the user flings left and right on the image, and - to prepare for chapter 3 - we will also anticipate for map display with the following rule : previous picture when swing to the right (-45° to +45°), next picture when swing to the left (opposite e.g. <-135° and >135°), otherwise (up and down) switch to map display. This last will not work for now but we will be ready. This sums up to the following blocks :



7.5.End of chapter one

This how your blocks could look like

(with backpressed and displayMap disabled)



8. Chapter two : delete-add-update monuments' list

Now that we can view the list of monuments with their picture and location we have to deal with :

- Deletion of an existing document,
- Creation of a new one with picture from the camera and location from the smartphone sensor,
- Update / change parameters of existing monuments.

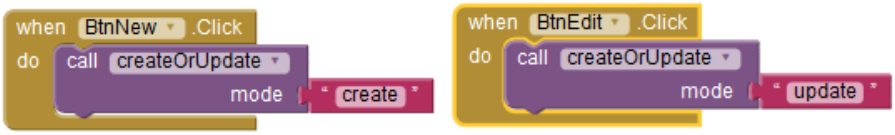
8.1. Delete monuments from list

The delete function is straightforward. The only thing to do is to suppress the current index "indexMonument" in the list of monuments "listMonuments" (you may wish to add a confirmation checkbox).

A reset button has also been added to restore default monuments (helpful for debug)

8.2. Create and update monuments

Create and update, have many things in common, from UI display utility functions which check inputs. So what we will do is write a single set of functions for both with "**mode**" as a calling parameter wich will be "**create**" or "**update**" according to what we want to do.



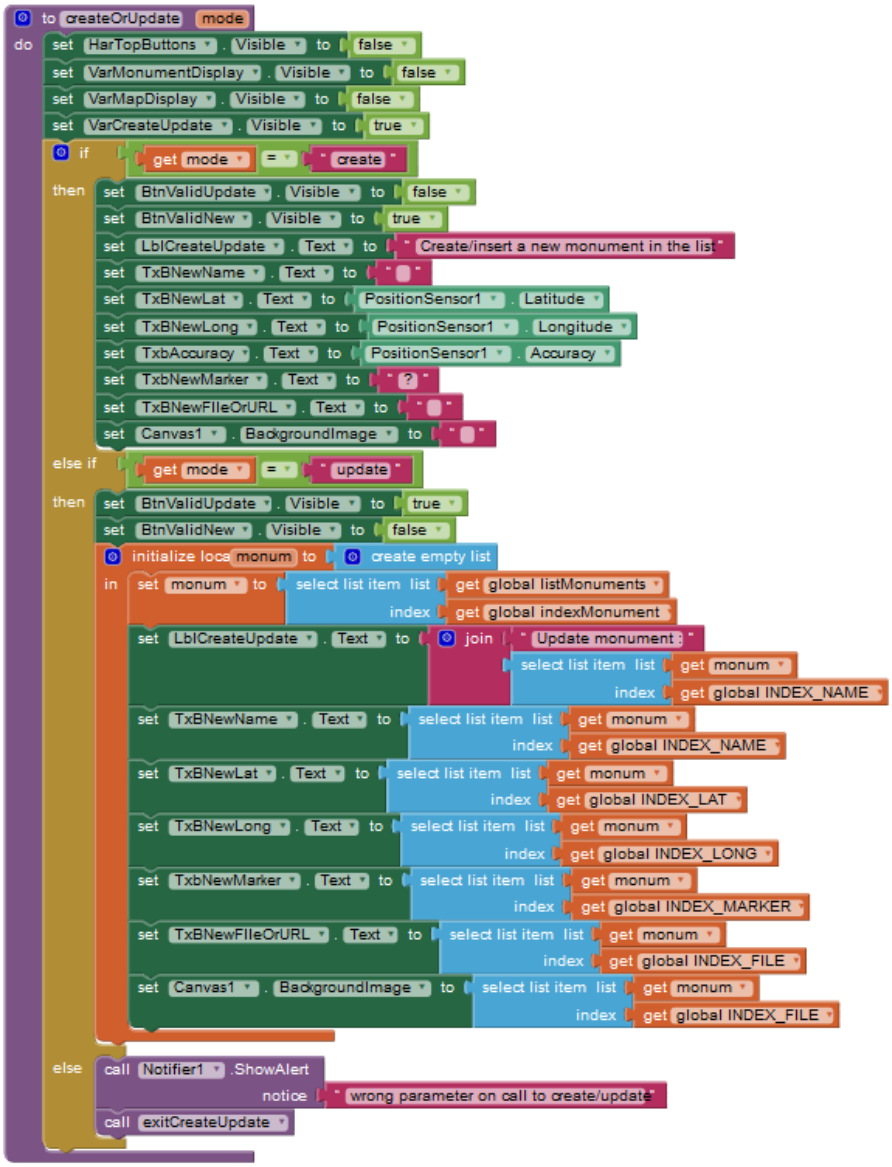
The createOrUpdate procedure.

The CreateOrUpdate procedure will display and init the adequate UI then exit .
App will then wait for the user to click on one of the buttons.

- Setup the GUI and init values by calling the CreateOrUpdate procedure which will :
  - Hide the top buttons horizontal arrangement **HarTopButtons**
  - Hide Vertical arrangement for monument display **VarMonumentDisplay**
  - Hide Vertical arrangement for map display (anticipate) **VarMapDisplay**
  - And make the Create or update vertical arrangement visible **VarCreateUpdate**

  (we could have put this in a procedure)
- Then initialize the display content according to the **mode** argument ("create" or "update") :
  - if in create mode, default new values will be set in the UI as initial proposal
  - if in update mode, the values of the monument at current index are used.

The following procedure has many blocks, because there are 2 cases and many parameters, but is not very difficult.



Note : you may wonder why a local **monum** variable is used instead of the **monument** variable which could well do the job. Try to say why before reading on … … … …

…

…

The reason is that monument variable can be changed at anytime by any routine. Probably not now, but keep in mind that with this "event driven" codding, anything can happen at anytime. You may well start doing something else before completing your update, and who knows … this may well change your **monument** which is a global variable.

So what we do here, since we do not know what can happen, is to make **monum** a local copy of the **monument** global variable. This local copy will not be changed from outside the simple reason that this variable is NOT known outside this routine.

Yeah ! we are safe … but remember this "good practice" with local variables will decrease the number of headaches on bugs which are very difficult to handle because you do not know where to find the error.

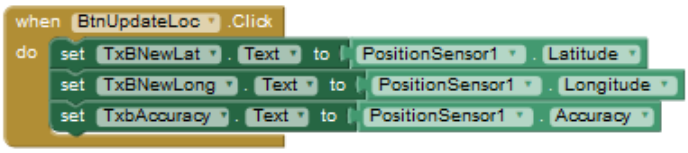
Once create/update UI is set with default values, user has the choice to :

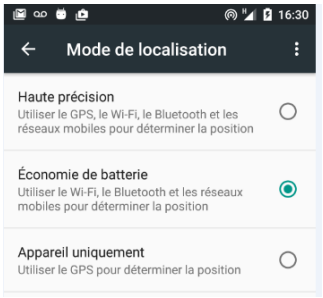
- Update textboxes manually (name, latitude, longitude, marker, file name or URL) . this is the only way to update the name which should not be blank,
- Click on the following buttons :
  - New picture : to take a picture from the smartphone camera,
  - Lat, long to update position from the smartphone loc sensors,
  - Cancel to exit without creation or update,
  - Either Valid create or update.

We will see that these functions use 3 utility procedures (checkinput, exitCreateUpdate and resetMonument) which we will describe after the code behind each button.

The BtnUpdateLoc click event:

This function will simply read from the phone loc sensor and update the corresponding text boxes for latitude, longitude and accuracy. These values should be checked before validation of update or create.



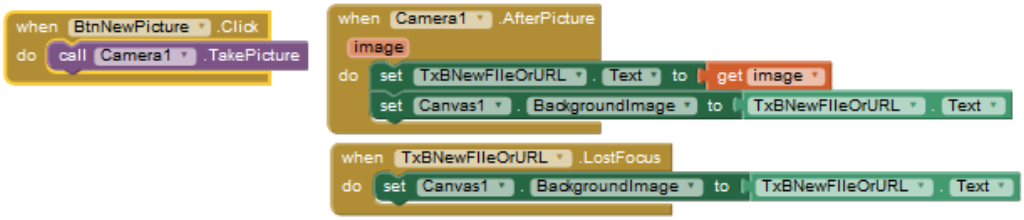


When you are trying this app when working inside, the GPS will probably not work. Most smartphones should still output a less precise value from the WIFI or GSM information, but this does not seem to be always the case, so you may prefer (at least when inside) to setup the localization mode to this.

The BtnNewPicture click event:

This function will simply call for the camera function of your phone and wait for the Camera1.AfterPicture event to return the filename for the picture, which will then be displayed.

An other way to update the picture is to copy/paste the URL of the picture in the adequate textbox. It Will then be displayed when this textbox loses cursor focus.



Note : web URL for images are fine (you can copy/paste their address) but they will read from he internet each time they are called, so do not choose pictures which are too big.

Note : selection of the back/front camera can be done on the smartphone camera app, as well as selection of image resolution.

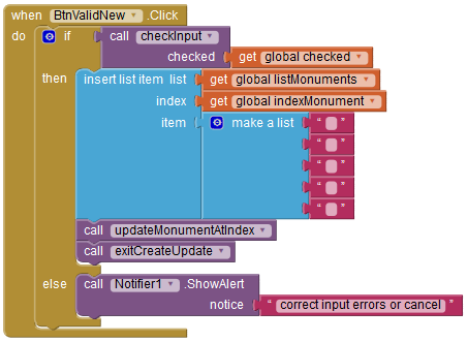
The BtnCancel click event :

This will simply reset the monument global variable (not mandatory) and call an exit function which will reset visibility of adequate UI with monument display.

The BtnValidNew click event :

This will do sanity checking on the input by calling the checkinput procedure. If this procedure returns true, it will insert a new item in the "listMonuments" list of monuments then call for updateMonumentAtIndexMonument which will update of the contents of this new item with what is in the textboxes. then exit through the common exitCreateUpdate procedure.

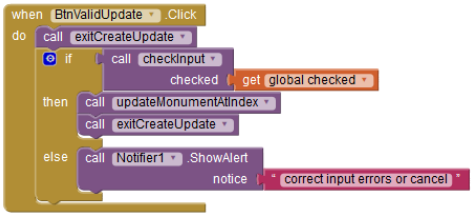
If checkinput returns false, it will warn the user.



The BtnValidUpdate click event :

This will do the same except for insertion of a new item in the list of monuments. The current inde will therefore be updated.

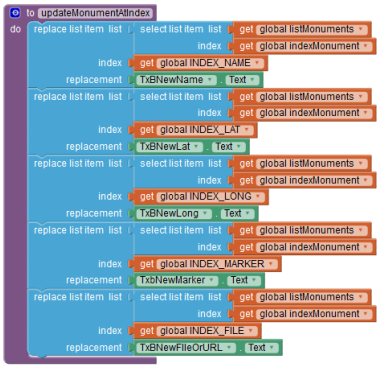
If false a message is returned to help the user understand why creation is refused.



The updateMonumentAtIndex procedure :

This procedure updates the content of the monument which is at index "indexMonument" in the list of documents "listMonuments".

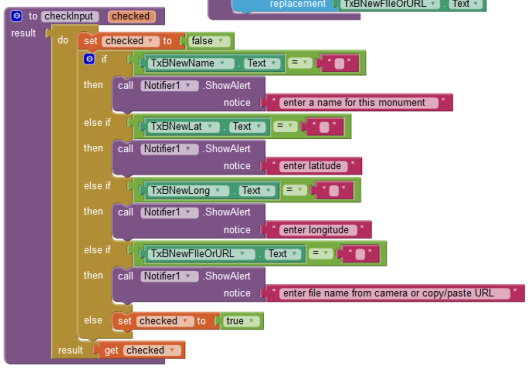
It reads textboxes and replaces the name, latitude, longitude, marker and file name or URL for the corresponding indexes defiens with the constants.



The Checkinput procedure :

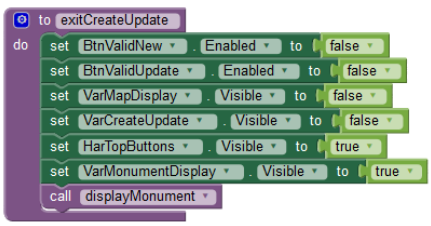
This procedure simply checks that the name, latitude, longitude or filename are not empty. It returns true if OK, otherwise false with a warning displayed.

Control algorithms should be improved but we wanted to give a flavor.

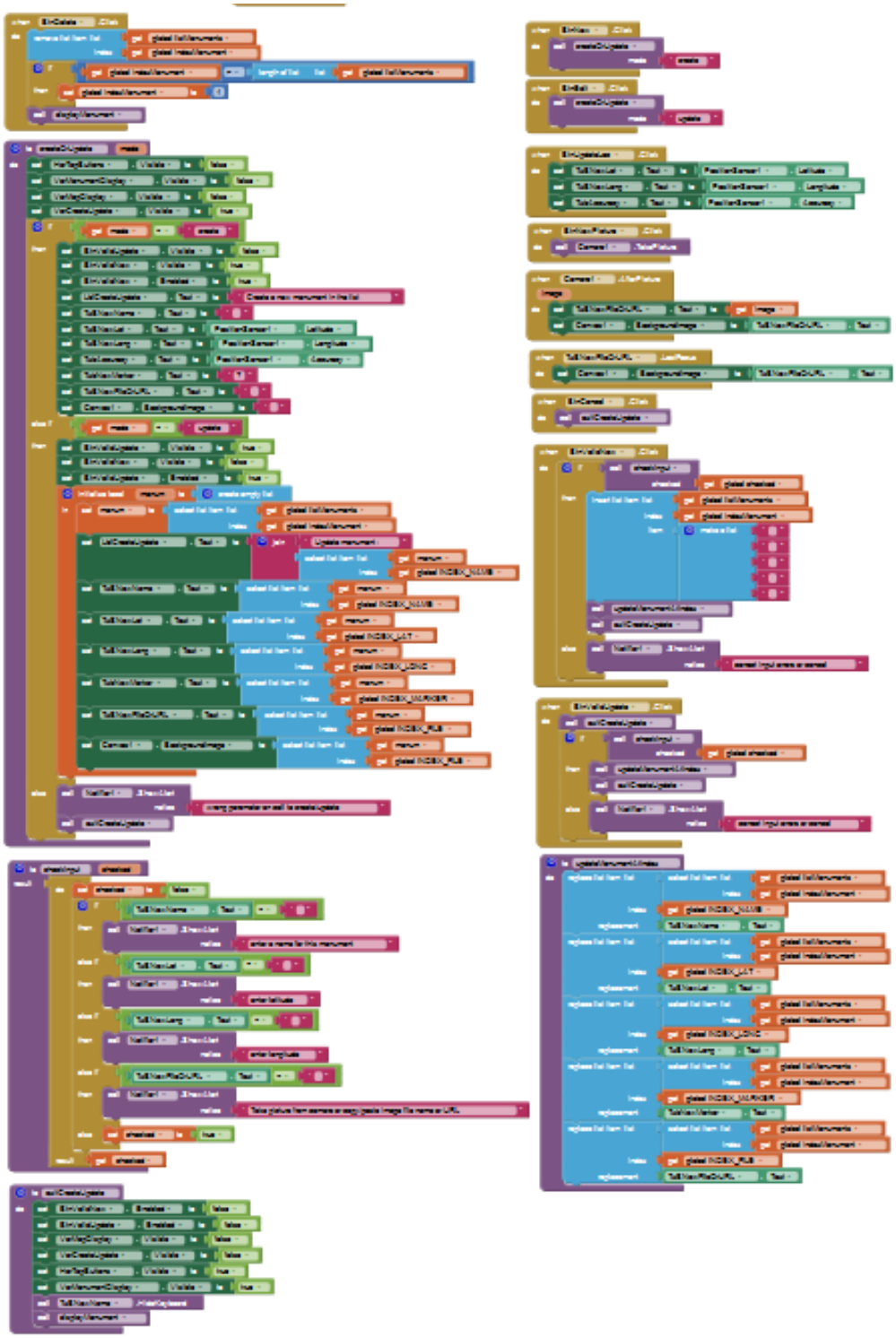


The ExitCreateUpdate procedure :

Resets the display to the monument configuration.



8.3. Chapter 2 completed



9. Chapter 3 : Display maps and satellite or aerial imagery

9.1. Introduction to static maps API and HTTP message

Open street map, Google, IGN and other organizations have setup web services to send maps on user's requests. We will use the "Google static map API" which returns an image file containing the map under jpeg, png of Gif format.

The request takes the form of an http message or URL (Resource locator) which contains the parameters that the server needs to build the map : latitude, longitude, zoom, map type, … We will use this URL exactly as we did with file name or http URL of the monument pictures.

What is new here is that we add parameters within our URL to specify the map content.

The way we specify the content in the http message is defined in the API (Application Programming Interface) of the Web service API we use. In our case the Google Static Maps Developer Guide, cf.

https://developers.google.com/maps/documentation/static-maps/intro#quick_example.

This http message is a single line of text starting with a base address followed by a question mark and the list of parameters are separated an ampersand '&'. Lists within a parameter use the '|' to separate items (ex : when you want to draw a line with several points).

Note : this implies that these characters : "&", "|",'%' and other such as white spaces are used as separators and forbidden in the rest of the message. In which case they must be replaced by their hexadecimal coding with %. For example : "|" must be replaced by "%7C". This called "html URL encoding" (App Inventor has a function that does it and you can also find tools online such as : http://www.w3schools.com/tags/ref_urlencode.asp)

After reading the API documentation (see above link), you will find that :

The HTTP base address is : https://maps.googleapis.com/maps/api/staticmap?

Then we have the following parameters (separated by '&'):

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

`size` : defines the rectangular dimensions of the map image (e.g. canvas size) with a string of the form *`{horizontal_value}x{vertical_value}`*.

Note : We must be careful that the size is not too big because it will generate data exchange on the web and delay times when internet bandwidth is low. It may be better to use an image size which is smaller than the canvas size, as long as the aspect ratio (e.g; height/width) is kept.

`zoom` : defines the display scale map with a numerical value : 0 for world coverage, 10 for cities and 21 for street level.

Note : zoom determines the pixel size : adding 1 to zoom level will decrease pixel size by a factor of 2. We will need pixel size when panning and an approximate evaluation of pixel size in degrees of latitude or longitude is

(this formula is OK for panning but requires checking for other purposes)

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

`markers` : defines markers overlaid on the map. We use them for user and monument's locations.

Note : Markers specification takes the form of `markers=markerStyles|markerLocation1|Location2| …` where we select color and label, then provide locations. As earlier reminded the '`|`' character is not allowed in HTTP requests and must be replaced by % 7C.

Note : Static Map API messages may contain other optional parameters :

`format` : defines the format of the resulting image with PNG as default.

`language` : defines the language to use for display of labels on map tiles.

`region` (*optional*) defines the appropriate borders to display.

Let us now build a sample request :
(where we have done URL encoding by replacing the forbiddet '|' by its hexadecimal value '%7C')

Base address	https://maps.googleapis.com/maps/api/staticmap?
Center	center = 48.85826, 2.294507
Zoom level	&zoom=10
Size	&size=320x320
Map type	&maptype=roadmap &maptype=satellite
Markers for monument	&markers=color:blue%7Clabel:M%7C48.85826, 2.294507
Markers for my position	&markers=icon:http://maps.google.com/mapfiles/ms/micons/man.png%7C

9.2.Display map procedure with HTTP message

We are now ready to build our http request on App inventor and to change the location, zoom factor, maptype, … as we want.

We have the following global variables :

center latitude initialized to 48.869

center longitude initialized to 2.434

zoom initialized to 10

mapType initialize to roadmap

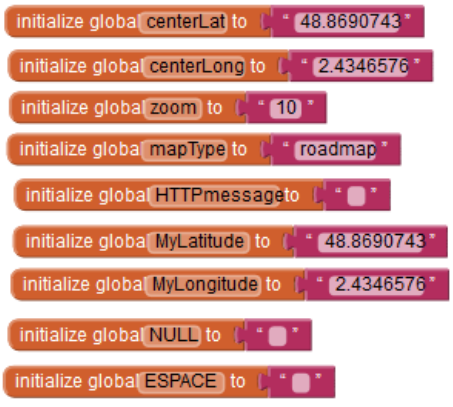
HTTP message initialize to null

MyLatitude initialized to 48.869

MyLongitude initialized to 2.434

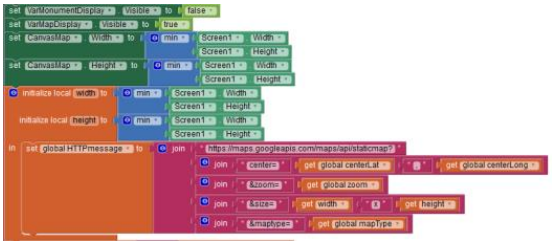
NULL constant initialized to null

ESPACE constant initialized to blank space



The map display procedure starts with making the **VarMonumentDisplay** hidden and the **Var MapDisplay** visible. Then it checks for the height and width of the map canvas, but if it is set to automatic, the first answer is 0 and we need this information as the size parameter in the HTTP query.

This is a very good subject to make you lose time … trying to adjust the automatic/fill parent component setup or the choix between "fixed or responsive" value of the Screen "Sizing" parameter.



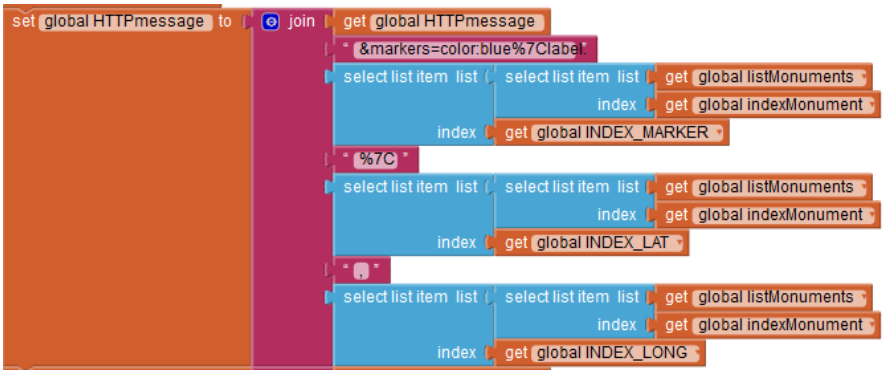
Read the doc and test … trying to anticipate wht will come out on phones that you do not know.

To make it short, we will ask for a square map with width = height = min (screen width, screen height) which should be OK in the portrait modethat we have chosen.

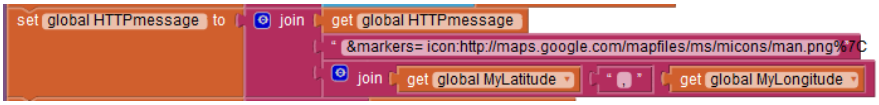
Note : set and get canvas display height and width report values which sometimes suprising and require more reading of the documentation

Following the HTTP request with center latitude, longitude, zoom, width, height and maptype

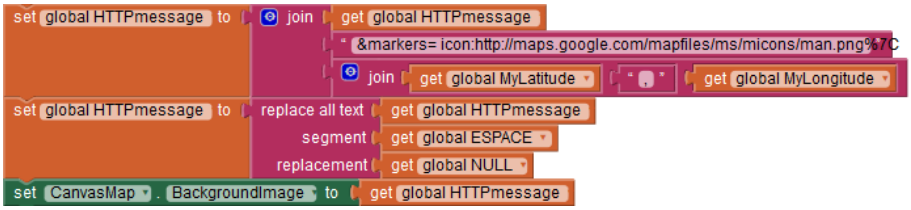
We add the marker for the current monument :



Then the marker for the user's (or smartphone) location :



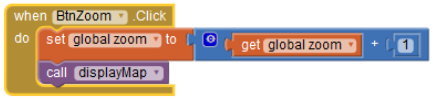
The message is complete but we will suppress possible blank spaces and set the canvasMap Image to this URL.That's it.

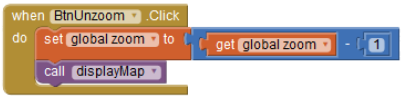


9.3. Map handling routines

Zoom and unzoom

The only thing to do is change the zoom value which is a global variable and call displayMap

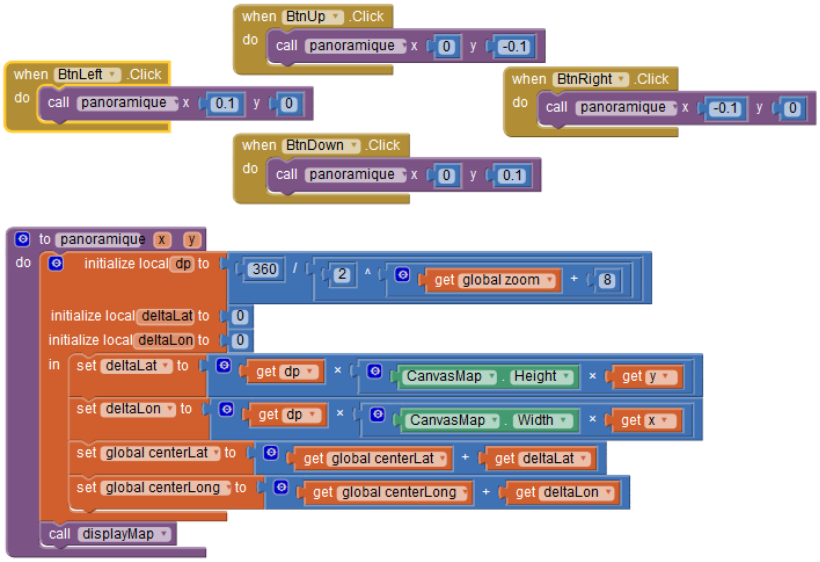




Pan up, down, left and right

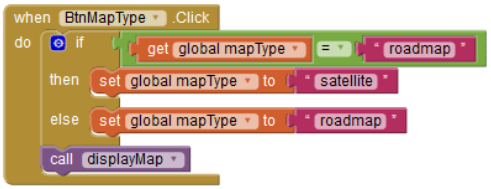
A panoramique function is written which computes the latitude and longitude range for the window. This range depends on the zoom factor and canvas height and width. It them changes the center lat and longitude for an amount which is the input x and y multiplied by the range.

Then the up, down, left and right panning functions are simple calls to this panoramique function.



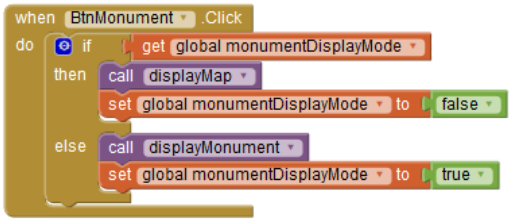
Maptype :

The only thing to do is to toggle maptype between "roadmap" and "satellite" (see API's documentation) then call displayMap.



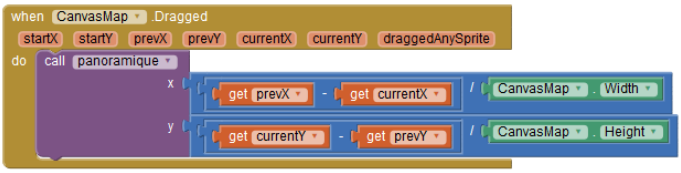
Toggle between map and picture display with  (BtnMonument) :

Call the adequate display procedure and update the MonumentDisplayMode accordingly.



Pan by dragging with finger on the map :

This with "finger dragging" panning mode may be more familiar. It may however general higher flow on the net with significant waiting time.



The thing to do is measure the difference of coordinates, divide by canvas size and call panoramique.

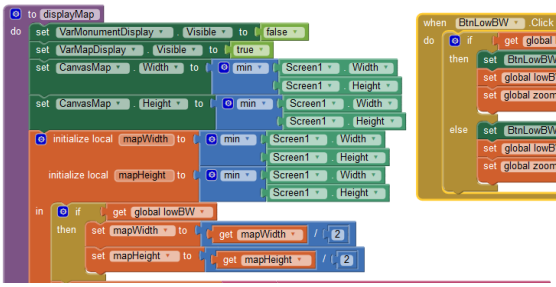
9.1. Tuning to Wifi Bandwidth

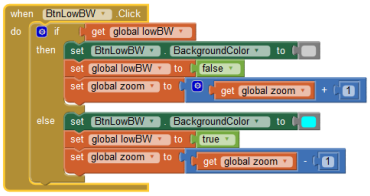
Recent smartphones and tablets have better and better resolution, which means that the size of maps and images to fill all these pixels will be bigger and bigger and therefore much longer to transmit through Wifi or cellular network.

One way app inventor deals with this is the screen sizing parameter which can be set to “Fixed” instead of “Responsive”.

We have not used this solution but added a global **lowBw** variable and a Low bandwidth button ”**btnLowBw**” which – when true - will let us divide by 2 the height and width requested from the map server (dividing by 4 the number of pixels transmitted).

We also need to divide pixel size by 2 in the panning procedure. (Not very clean way, but will do)





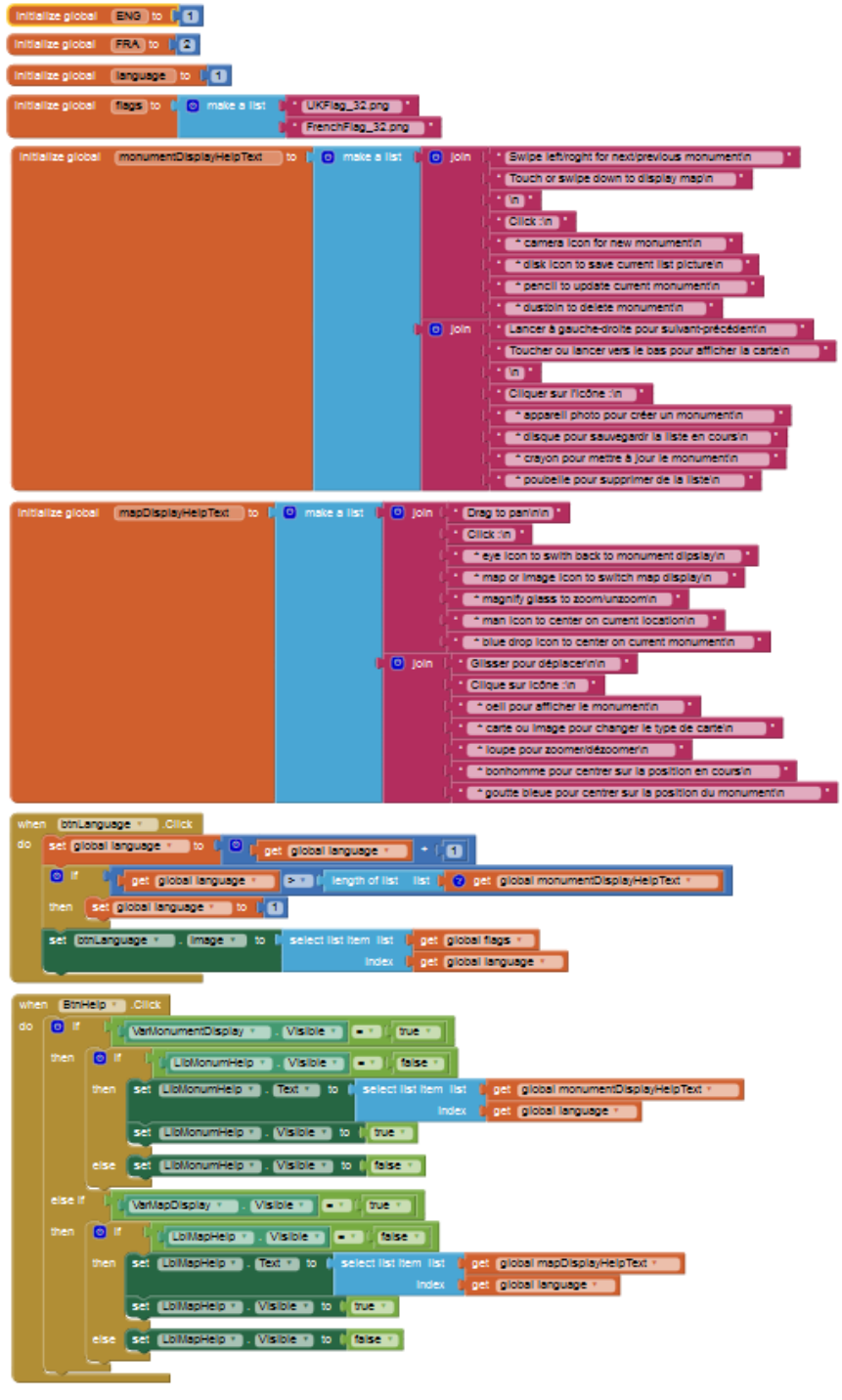
The button will be a toggle between the 2 size and you should see the difference (especially with the satellite image which does not allow high compression). You will also compare the quality to adjust to the need during tests.

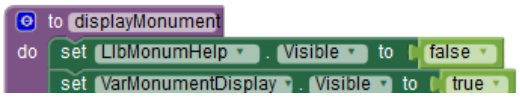
9.2.Add help button and choice of language

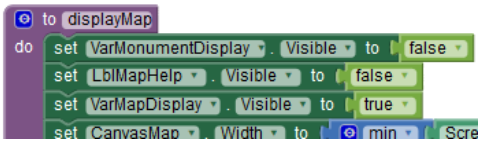
Add help text labels for monument display and map display and set them to non visible.

Toggle visible/invisible and click help button.

Set to invisible when map or monument display update







9.3.miscellaneous

Replace text by icons

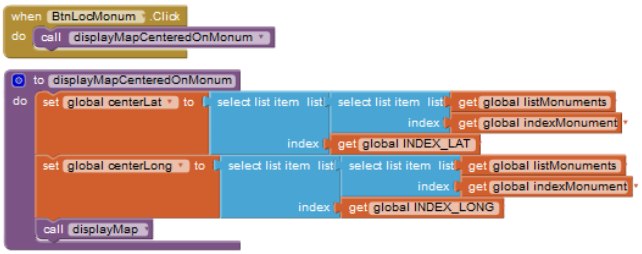
Map Icon 

Image Map icon 

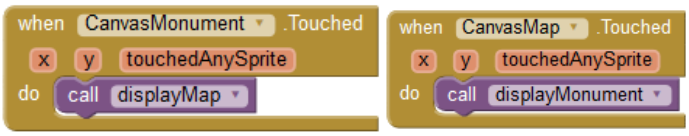
Hide pan buttons (not useful with drag) and center horizontal arrangement

Hide next/previous icons for monuments

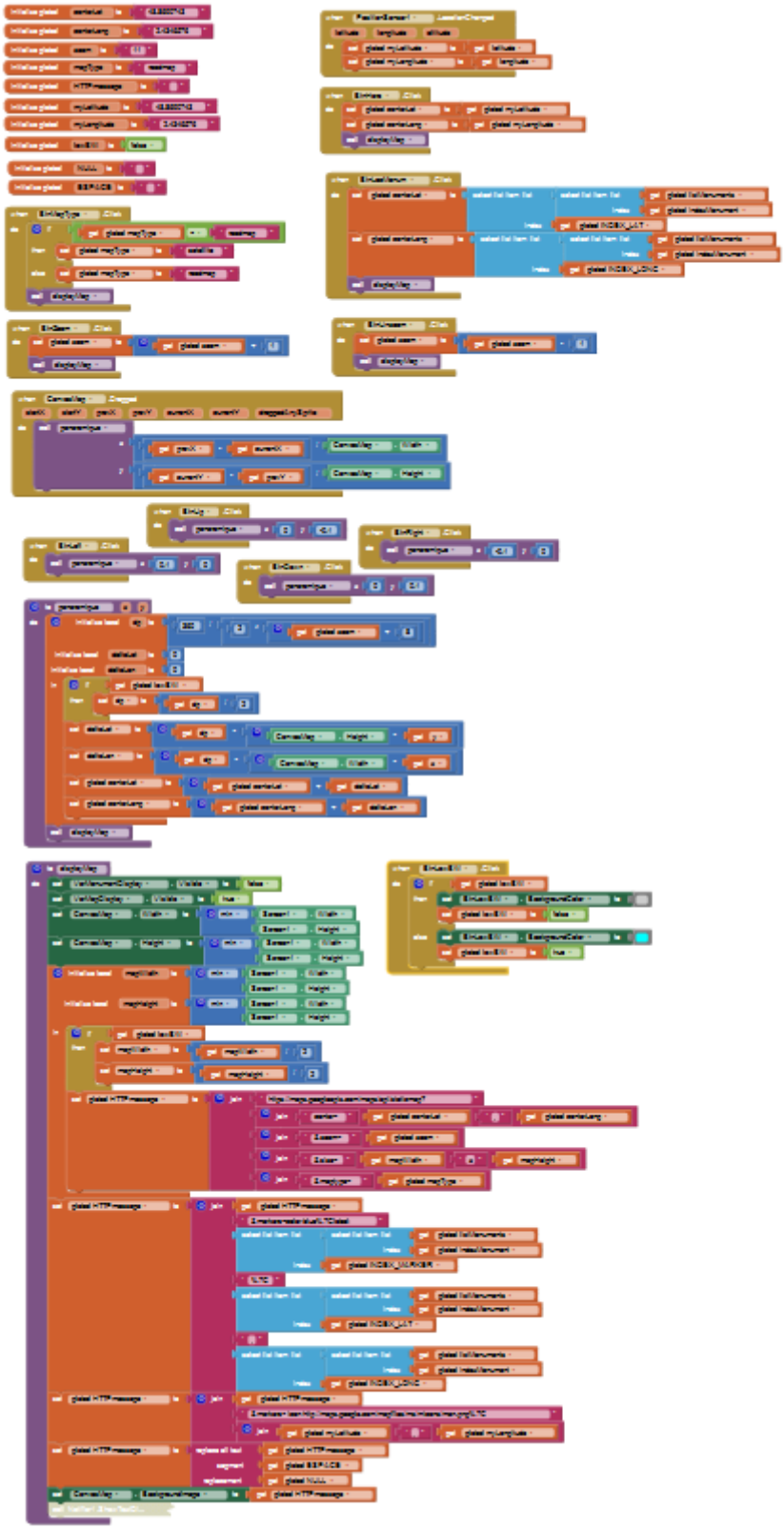
Use monument location as default map center with a diplayMapCenteredOnMonum procedure which is called instead from the monument display.



9.4.Switch to map/monument when canvas touched



9.5. Chapter 3 completed



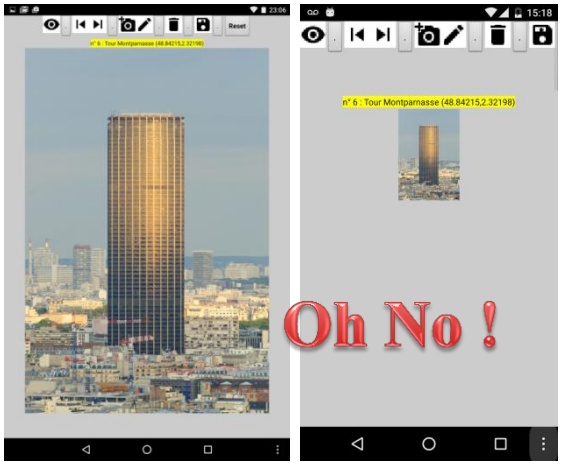
10. Debug/Tune /Tests /Doc/terrain feedback

To be completed.

You are certainly aware that programmers – for most of their time - are debuggers rather than developers! (That's life folks and they come home late!)

Note : Once you have completed your first working app, look back on how much time you thought it would take and how much it really did !!! and possible reasons ?
Can you find your own "good practices" and what's your idea on existing ones (MVP, paper prototyping, initial conception, …).

Image to canvas mapping :



After completing development and testing on a Nexus 7 (left) here is what comes out on a Nexus 5 (right) with exactly the same android version. …

Surprise, surprise !

How can this be ? A large image on a screen that has more pixels and a small image on a screen that has less : it should be the contrary …

Well… as I am writing, I don't know

What do you think ?

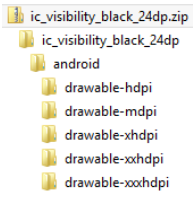
I don't feel like changing the canvas height and width from "automatic" to "fill parent" because this led to bad result with image aspect ratio (e.g. height/width ratio). ….. And I do not see how to cattch image dimensions, ……Well, maybe I will try with the enclosing Vertical arrangement ? …does not seem to work either. ……

OK, if we can't understand **why** let's try to see **when** this happens: review format and size of pictures … it then appears that images with both dimensions below screen size are displayed OK, images with both dimensions above screen size are reduced when displayed and pictures with height below screen height and width beyond screen width are distorted on display …

That's too bad, there is not much we can do to adjust multiple phones,

The best we found for now is to set the screen1 "sizing" attribute to "fixed" rather than "resizing". It will be less pretty with new phones, but will be OK until correction of what seems to be a bug (we should however advise users that there is image distortion for landscape style images – and they should use portrait mode for now).

Icon size : fingers too fat for large screens / icons too fat for small screens !



Icons were too small on your first tablet so you increased pixel size. Now they are too big and some hidden on the phone … All these pixels, percents, automatic choice, … multiple screen sizes and resolutions (dpi : dots per inch) make it a little hard.

There is a solution for this. Go back to the google design icons library where each icon is available under different sizes.

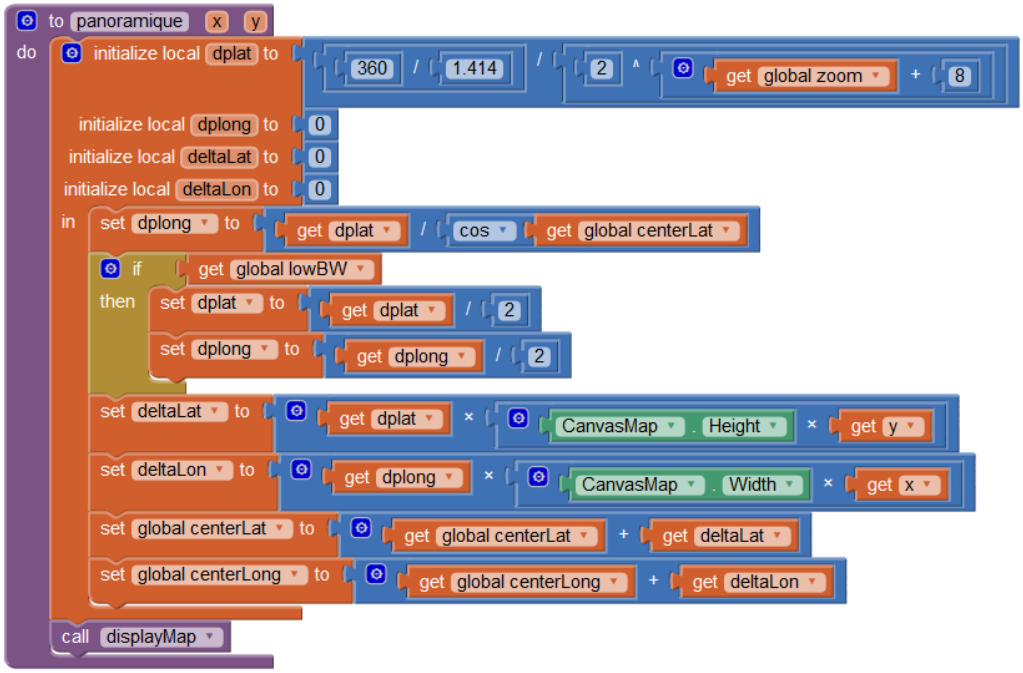
mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi

Android sometimes manages the choice for you, but her you have to do it yourself : Go ahead and do it …

Map aspect ratio and pixel size

Pixel size is computed in degrees but images are displayed in pixels which have the same width and height in meters. The map would otherwise look distorted to us. But if one degree of latitude is always 110 km (e.g. 40 000 km / 360°) one degree of longitude is less, between 110 km at equator and 0 km at the poles. Its value is around 110 km * cosine(latitude)

Let us suppose that the average pixel size has been computed at 45° of latitude (e.g. cosine = 0,707) the pan function is then rewritten with different values for geographic (lat, long) pixel size.



"Global variables pointing at lists or stuctures" : a dangerous cocktail !

To be adjusted / TBD

Documentation

Good practices have not been applied here … in block documentation should have been used.

11. Chapter 5 : feedback from terrain and updates

To be completed / "Design thinking" principle must be reminded before going further.

Pair reviews should preferably be done at the end of each chapter and user review at latest at the end of chapter 3. Addons or improvements should mainly come from user feedback.

People (including yourself) never use your app the way you thought they would !

Don't take it bad (unless you are their boss), find what they liked and begin to surf.

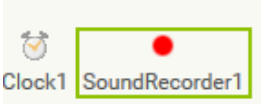
OK I used it myself, but in fact I don't really care about monuments (nor does anyone else), but I went to an art exhibit and I never remember the name of the painters, I just have pictures with the date. So – this time - I used my brand new app, put the name of the artist as the title and took a picture. Then I got my database with picture, painter, location but – oups - no date !!!

- … I should add the date,
- besides this the save icon is very close to the reset button (and I goofed it twice),
- and there is far too much to validate for each new item. All this should be much more simple : click new, take picture and input name (don't tell me of the rest unless it went wrong)

So let's get back to work :

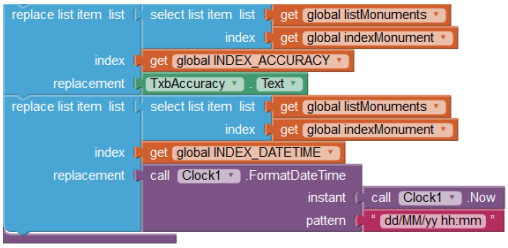
- add a date-time parameter in the list
- add a location accuracy parameter in the list
- update date and time when creating new object (with a clock component)
- save list of objects each time a new item validated
- an may I would like to say a little word on what I see, whether it is a painting or monument, …

So :

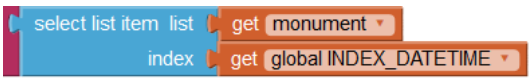


- add a clock and a sound recorder to components,
- add accuracy and date-time and soundFile indexes to the monument list,

- add accuracy and date updates in the **udpateMonumentsAtIndex** procedure,



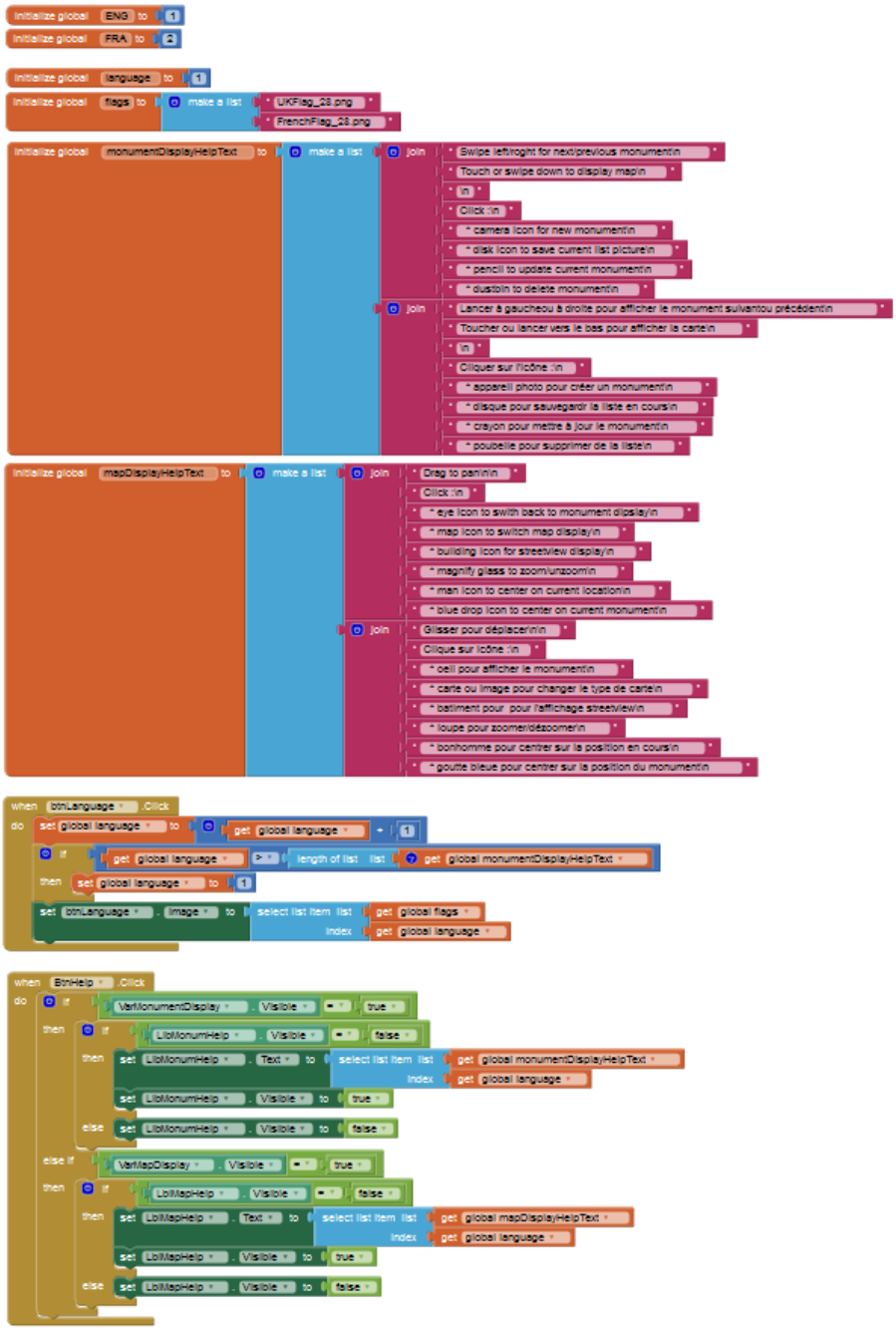
- add date and time display in monument text label in the **displayMonument** procedure,



12. Chapter 4 : other Addons - improvements

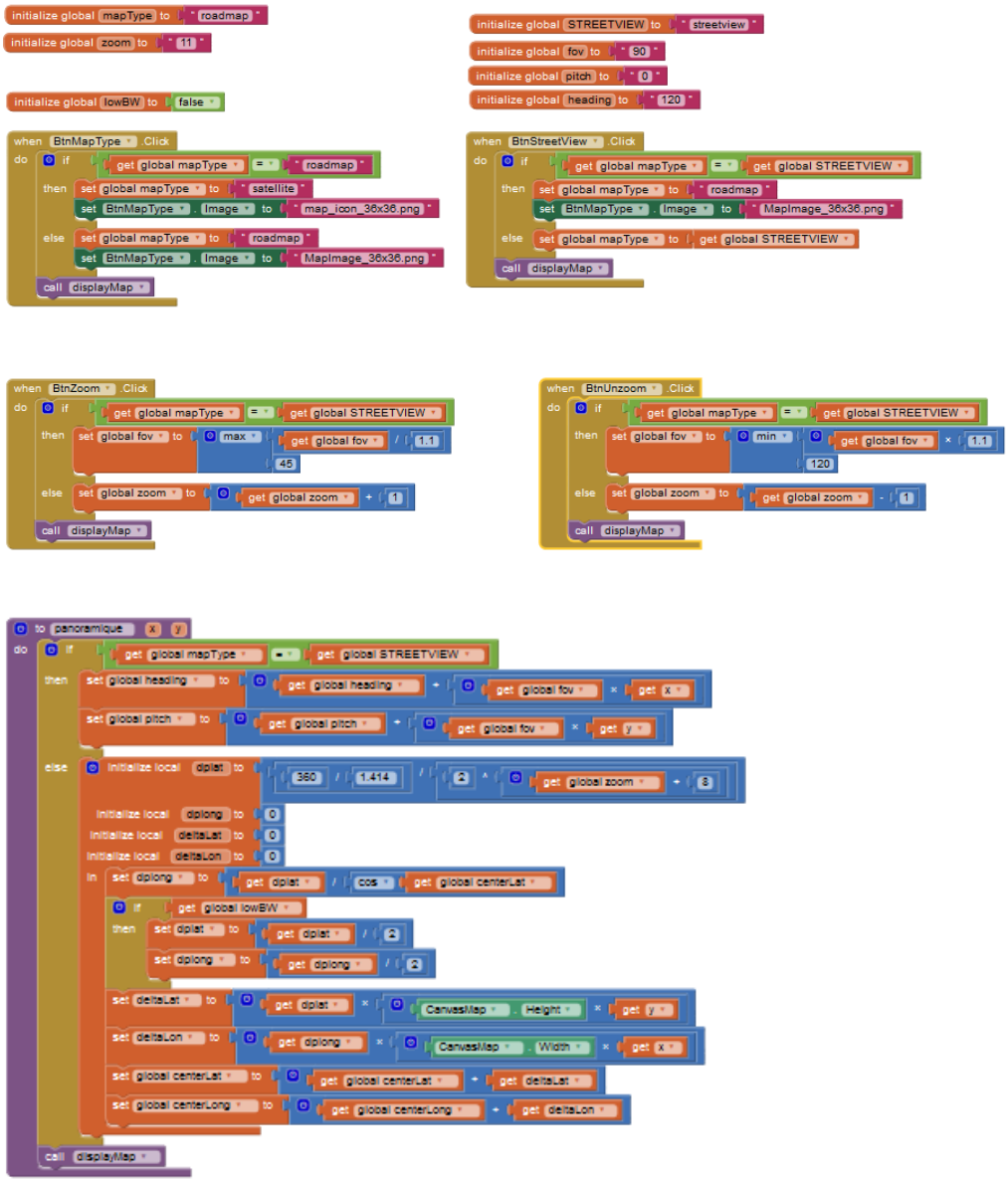
12.1. multilingual user manual

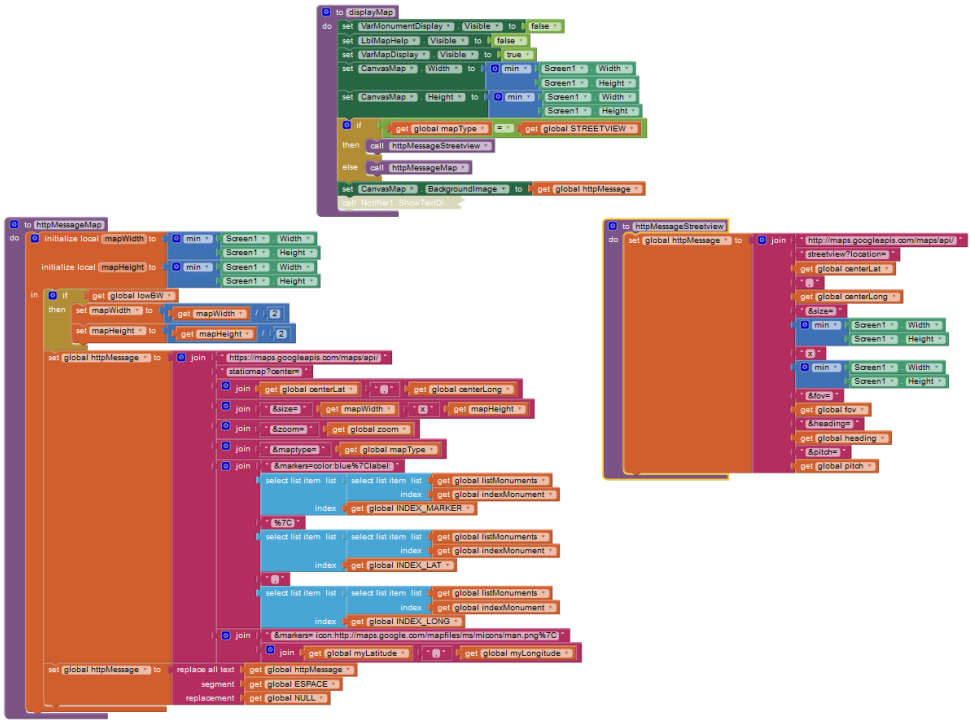
To be completed. use ISO 639-2 Code : eng, fra



12.2. Chapter 4 : Adding streetView

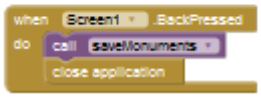
To be completed.



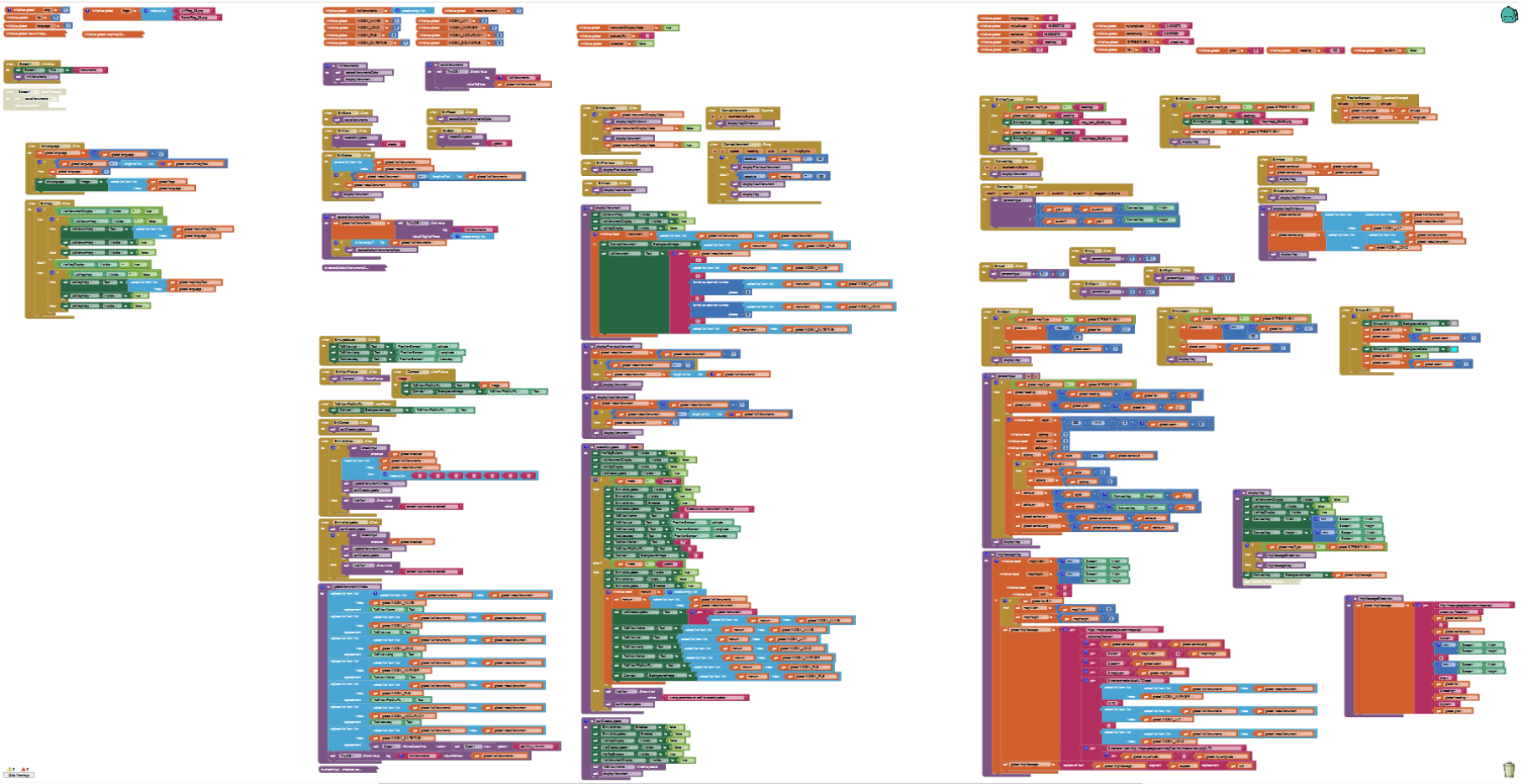


12.3. Re-organize init and exit

To be completed.



13. Global software View



14. Bibliography, Acronyms, Definitions, Acknowledgments

14.1. Bibliography

14.2. Acronyms

API	Application Programming Interface	
DPI	Dots Per Inch	
GUI	Graphic User Interface	
HTTP	HyperText Transfer Protocol	
MVP	Minimal Viable Product	
TBD	To be Defined	Used when work has not been completed , (but should be sometimes …)
UI	User Interface	
URI	Uniform Resource Identifier	
URL	Uniform Resource Locator	

14.3. Definitions

Event handler	

14.4. Acknowledgments

Tutorial canvas has been built from Dave Wolber's scheme ref. TBD

Icons from google design … TBD