B4X Booklets

BANano – Essentials

Creating lightweight and blistering fast Web Apps/PWAs with B4J!

Written by Alain Bailleul

Edition 1.0

Last update: 2022.02.26

Table of Contents

1 GETTING STARTED WITH B4J AND BANANO 6

1.1 BANano License 7
1.2 Installing BANano 9
1.3 Setting up Chrome with a Webserver 10

2 MY FIRST BANANO PROJECT 13

2.1 Looking into the BANano Layout files 17
2.2 A first look at the source code and structure of a BANano App 21

3 SUPPORT OF THE B4J LANGUAGE 23

4 THE WEB CONNECTION 25

5 BANANOOBJECT: THE JACK-OF-ALL-TRADES 28

6 BANANOELEMENT: TALKING TO THE DOM 34

6.1 Introduction 34
6.2 Using HTML tags, with style! 35
6.2.1 Getting existing tags 35
6.2.2 Creating new tags 36
6.2.3 Adding the tags to the DOM 37
6.2.4 Removing Tags (or only its children) 38
6.2.5 Looping through a multi-tag BANanoElement 40
6.2.6 Styling Tags 40
6.2.7 BANanoEvent: Working with Events 41
6.2.8 Adding Events 41
6.2.9 Removing Events 43
6.3 Loading Abstract Designer Layouts 44

7 BANANOPROMISE: GETTING AN ANSWER IN THE FUTURE 46

7.1 Making a promise 46
7.1.1 The structure of a promise 46
7.1.2 Breaking the code down 47
7.1.3 The .Then() can also be a .ThenWait() and the .Else() be a .ElseWait() 47
7.1.4 What is such a 'task'? 48
7.2 That was the long story. But BANano.Await! This can be simpler... 49
7.2.1 BANAno.Await to the rescue 49
7.2.2 Wait a minute: isn't that just B4Js Wait For? 50
7.3 Then why do these different systems exist? 51

8 BANANOFETCH: MAKING REQUESTS TO THE SERVER 52

8.1 GET 52
8.2 Handling the BANanoFetchResponse 53
8.3 POST/PUT/DELETE/... (using BANanoFetchOptions) 54
8.4 Shortcut methods 55

9 THE BANANO OBJECT: ONE OBJECT TO RULE THEM ALL! 56

9.1 Using BANano in AppStart 56
9.1.1 TranspilerOptions 58
9.1.2 BANanoHeader 61
9.1.2.1 Loading external CSS and JavaScript files 61

9.1.2.2 Loading assets… Later 62
9.1.2.3 Loading modern ES6 modules 63
9.1.2.4 Loading JavaScript files in the Service Worker of the PWA 65
9.1.2.5 PWA Specific Assets 66
9.1.3 Transpiling and Building 67
9.1.3.1 Building a PWA 67
9.1.3.2 Building a BANanoLibrary 69
9.1.3.3 Building a BANanoServer Websocket project 70
9.1.3.4 Building a BANanoLibrary for ABMaterial 70
9.1.3.5 Tree Shaking (removing dead code) 71
9.2 USING BANANO IN THE WEBAPP CODE 72
9.2.1 BANano Extended Property Objects 72

10 SAVING DATA IN THE BROWSER 74

10.1 COOKIES 74
10.2 LOCALSTORAGE AND SESSIONSTORAGE 75
10.3 CACHESTORAGE (BANANO V7.35+) 77
10.4 BANANOSQL 78
10.4.1 Creating the Database 78
10.4.2 INSERT new data 79
10.4.3 UPDATE existing data 79
10.4.4 DELETE data 79
10.4.5 SELECT data 80
10.4.6 Additional Remarks 80

11 COMPONENTS FOR THE ABSTRACT LAYOUT DESIGNER 82

11.1 CREATING A COMPONENT 82
11.2 A NOTE ON EXTRA ASSETS 84

12 BANANO LIBRARIES 85

13 INTRODUCING BANANOSERVER 87

13.1 WHAT IS BANANOSERVER? 88
13.2 CREATING A B4J APP USING THE BANANOSERVER LIBRARY 89
13.2.1 REST API BANanoServer 89
13.2.2 WebSockets BANanoServer 91
13.3 A REST API EXAMPLE 92
13.3.1 BROWSER side: PWA 92
13.3.2 SERVER side 94
13.4 A WEBSOCKETS EXAMPLE 99
13.4.1 BROWSER side: PWA 101
13.4.2 SERVER side 105

14 BACKGROUND WORKERS 108

15 CRON: AN ADVANCED TIMER 112

16 BANANOROUTER: MULTI PAGE PWA 114

16.1 WHAT IS A JAVASCRIPT ROUTER? 114
16.2 SETUP UP THE ROUTES 116
16.3 NAVIGATING BETWEEN PAGES 117
16.4 REMOVING A ROUTE 117

17 DEBUGGING 118

17.1 LIVE CODE SWAPPING 118
17.2 MAKING USE OF THE NEW B4J 'JUMP' FEATURE IN THE LOGS 118
17.3 JAVASCRIPT BREAKPOINTS 119
17.4 USING THE BROWSER DEVELOPER TOOLS 119
17.4.1 The Console Tab 120
17.4.2 The Network Tab 122
17.4.3 The Application Tab 125
17.4.4 The Security Tab 127
17.4.5 The Lighthouse Tab 128
17.4.6 Testing your PWA on emulated device sizes 130

18 QUICK REFERENCE 131

18.1 BANANO 131
18.2 BANANOCACHEREPORT 157
18.3 BANANOCONSOLE 158
18.4 BANANOELEMENT 160
18.5 BANANOEVENT 168
18.6 BANANOFETCH 169
18.7 BANANOFETCHOPTIONS 171
18.8 BANANOFETCHRESPONSE 173
18.9 BANANOGEOLOCATION 175
18.10 BANANOGEOPOSITION 176
18.11 BANANOHEADER 177
18.12 BANANOHISTORY 179
18.13 BANANOJSONGENERATOR (DEPRECIATED) 180
18.14 BANANOJSONPARSER (DEPRECIATED) 181
18.15 BANANOJSONQUERY 182
18.16 BANANOLOCATION 184
18.17 BANANOMQTTCLIENT (DEPRECIATED) 186
18.18 BANANOMQTTCONNECTOPTIONS (DEPRECIATED) 187
18.19 BANANOMEDIAQUERY 188
18.20 BANANOMUTATIONOBSERVER 189
18.21 BANANOMUTATIONRECORD 192
18.22 BANANONAVIGATOR 194
18.23 BANANOOBJECT 195
18.24 BANANOPROMISE 200
18.25 BANANOREGEX 203
18.26 BANANOROUTER 204
18.27 BANANOSQL 206
18.28 BANANOSCREEN 207
18.29 BANANOTRANSPILEROPTIONS 208
18.30 BANANOURL 212
18.31 BANANOWEBSOCKET 215
18.32 BANANOWINDOW 218
18.33 BANANOXMLHTTPREQUEST 222

Main contributors: Alain Bailleul (Alwaysbusy)

To search for a given word or sentence use the Search function in the Edit menu.

Updated for following versions:
B4J version 9.30

Other B4X Booklets by Klaus Christl (klaus), Erel Uziel (Erel):
B4X Getting Started
B4X Basic Language
B4X IDE Integrated Development Environment
B4X Visual Designer
B4X Help tools

B4XPages Cross-platform projects
B4X CustomViews
B4X Graphics
B4X XUI B4X User Interface
B4X SQLite Database
B4X JavaObject NativeObject

B4R Example Projects

You can consult these booklets online in this link [B4X] Documentation Booklets.
Be aware that external links don't work in the online display.

This booklet is a first introduction to BANano Web Apps, their structure and the special commands the BANano Core library has.

This booklet is not a full description of all the methods in BANano: that is why it is called Essentials. Some things can be done in several different ways, but I will not always go through all the possible ways and just mention the most common (and best) ways to do it.

Although this booklet goes in-depth on some core functionalities of BANano, I've tried to make is as accessible as possible for everyone.

If you understand what is in this manual, you have all the building blocks you need to get started with creating your own Web Apps and PWAs using BANano in B4J!

1 Getting started with B4J and BANano

B4J is a **100% free** development tool for desktop, server and IoT solutions
With B4J you can easily create desktop applications (UI), console programs (non-UI) and server solutions.
B4J apps can run on Windows, Mac, Linux and ARM boards (such as Raspberry Pi).
The compiled apps are standalone, without any external dependencies.

You can see all the libraries in the Documentation page in the forum or in the B4X Libraries Google Sheet.

BANano is a B4J library that Transpiles B4J source code to html/CSS and JavaScript. It is also **100% free**.

BANano is B4J's answer to the JavaScript frameworks like Angular, React, Vue, … including components, routers, etc.

It supports about **99% of the normal B4J keywords** and adds an additional set of keywords and methods to the B4J IDE, specifically focused on Web.

With BANano, you can create websites/webapps with (offline) Progressive Web App support. It does not rely on any particular framework like Materialize CSS or Bootstrap. You will have to write that part yourself, but on the other hand, you have the choice to pick which one. BANano does include a UI library already made to get you started: **BANanoSkeleton**. This manual will use this UI library in its examples.

Additional UI or JavaScript wrappers can be written by creating BANano .b4xlib Libraries.

Just like in B4J, you can use the **Abstract Designer** to design your views and layouts.

BANano does support **Live Code Swapping** and does optimize your code by removing all 'Dead Code' (code that is not used in the final project, also known as **Tree Shaking**).

1.1 BANano License

Freeware/Donationware License

B4J is Copyright (c) 2010 - 2022 by Anywhere Software All Rights Reserved.
LIBRARY (Library/library): B4J library files BANano.jar and BANano.xml (by Alain Bailleul)
SOFTWARE (Software/software): Computer Software
APPLICATION (Application/application): Any end product as the result of compiling with an Anywhere Software product
SOURCE CODE: human-readable program statements written by a programmer or developer in a high-level or assembly language that are not directly readable by a computer and that need to be compiled into object code before they can be executed by a computer

BY USING THIS LIBRARY, YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE.

1. THIS LIBRARY IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL ANY COPYRIGHT HOLDER/AUTHOR/DEVELOPER BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL,SPECIAL,INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY INCLUDING BUT NOT LIMITED TO LOSS OF DATA, FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER PROGRAMS OR LIBRARY, EVEN IF COPYRIGHT HOLDER/AUTHOR/DEVELOPER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

2. YOU MAY NOT COPY, SUB-LICENSE, REVERSE ENGINEER, DECOMPILE, DISASSEMBLE, OR MODIFY THIS LIBRARY IN ANY WAY.

3. YOU MAY NOT DISTRIBUTE THE LIBRARY ON ANY MEDIUM WITHOUT PRIOR NOTICE FROM ALAIN BAILLEUL (alain.bailleul@telenet.be). YOU HAVE TO ASK FOR PERMISSION IN ORDER TO MAKE THIS LIBRARY AVAILABLE FOR DISTRIBUTION OVER THE INTERNET OR ANY OTHER DISTRIBUTABLE MEDIUM.

4. YOU AGREE NOT TO DISTRIBUTE FOR A FEE AN APPLICATION USING THE LIBRARY THAT, AS ITS PRIMARY PURPOSE, IS DESIGNED TO BE AN AID IN THE DEVELOPMENT OF SOFTWARE FOR YOUR APPLICATION'S END USER. SUCH APPLICATION INCLUDES, BUT IS NOT LIMITED TO, A DEVELOPMENT IDE OR A B4J SOURCE CODE GENERATOR.

By possessing and/or using this library you are automatically agreeing to and show that you have read and understood the terms and conditions contained within this Freeware Software License Agreement. This Freeware Software License Agreement is then effective while you possess, use and continue to make use of these software products. If you do not agree with our Freeware Software License Agreement you must not possess or use our library products - this Freeware Software License Agreement will then not apply to you. This Freeware Software License Agreement is subject to change without notice.

Violators of this agreement will be prosecuted to the full extent of the law.

This library is free, however if you do enjoy it, please consider a donation to Alain Bailleul (alain.bailleul@telenet.be) for his time and efforts to make this library possible.

This license file (LICENSE.TXT) shall be included in all copies of the library or any distribution using the library in any form resulting from mechanical transformation or translation of the source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

If you have any questions regarding this license, please contact alain.bailleul@telenet.be

1.2 Installing BANano

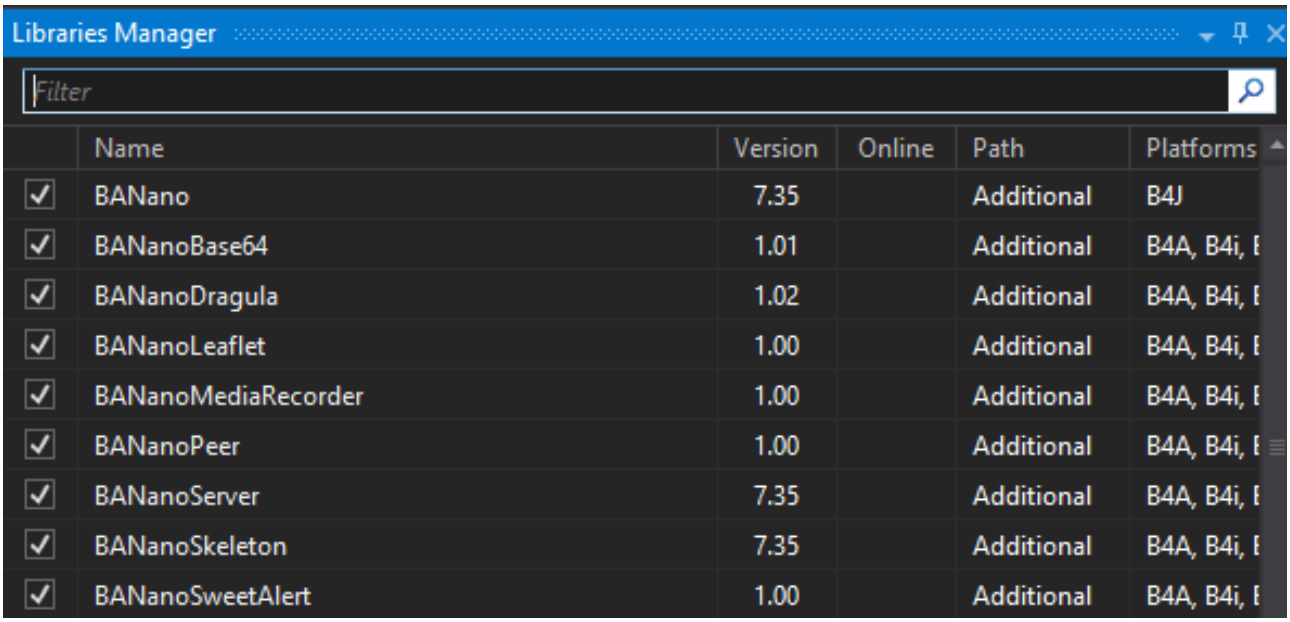
This manual assumes **you are familiar with B4X products & the B4J language**. If not, you will have to go through the other excellent booklets and videos Klaus and Erel have made first. It also assumes you have installed B4J with all its dependencies.

The most up to date installation instructions for B4J are in the forum at this link: https://www.b4x.com/b4j.html. Please, follow the instructions there if you have not installed B4J yet!

You can download the latest version of BANano from the forum https://www.b4x.com/android/forum/threads/banano-website-app-pwa-library-with-abstract-designer-support.99740/%23post-627764.

At the time of writing, the latest version of BANano is 7.35

1. Download the zip and unzip it.
2. Copy all files from the /Libraries folder to **your B4J Additional libraries folder**. You should see the BANano libraries in the Libraries Tab of the IDE. The versions may vary.



3. Copy the.b4xtemplate files from the /Templates folder to **your B4J Additional libraries folder**. You can now pick them in the B4J File - New menu.

Check if you do have any of these files in your B4J Libraries folder! If so, remove them.

Having the same library of .b4xtemplate in both your Libraries and Additional Libraries folders will cause problems!

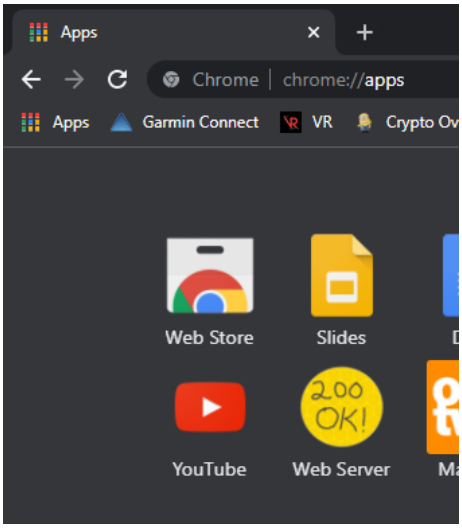
1.3 Setting up Chrome with a Webserver

When making Web Apps, it is very handy to use some kind of Web Server. Some functionalities of a Web App need this and cannot be used by just opening a .html file in the browser.

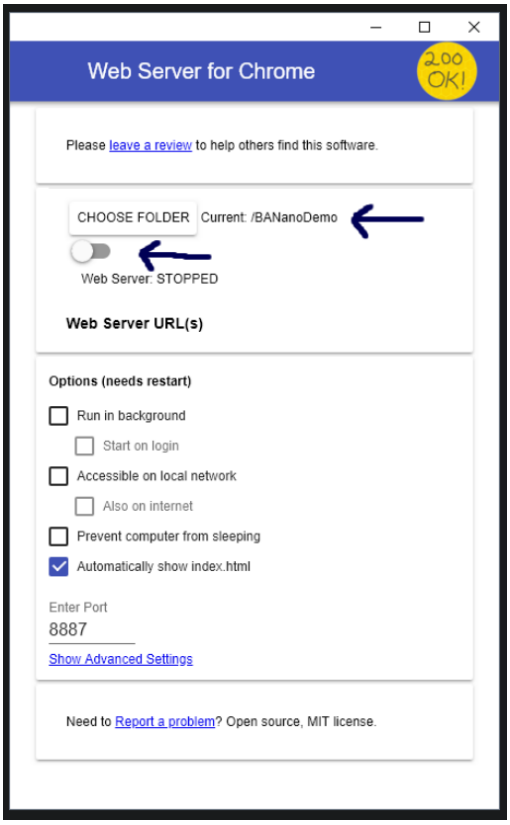
Chrome has a very nice little Web Server plugin you can use for free:

https://chrome.google.com/webstore/...chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en

After installing it, you will find it in your chrome apps. It can be opened like this:

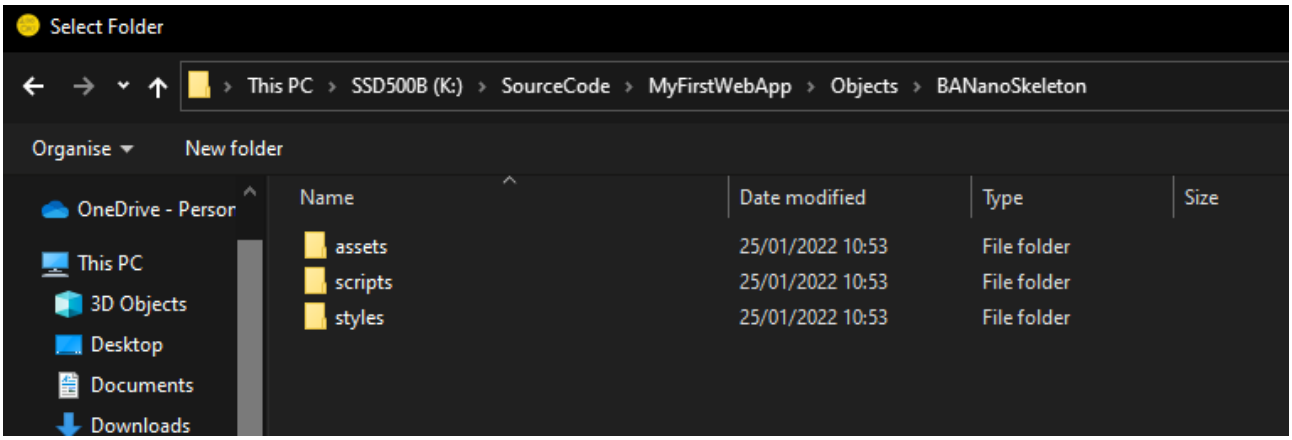


Just start the plugin and you will be presented a popup box:

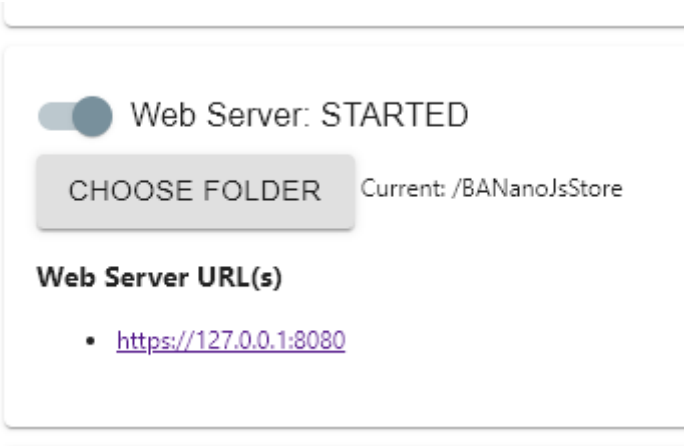


Now, all you have to do is select the folder where the BANano generated .html is located and start the plugin.

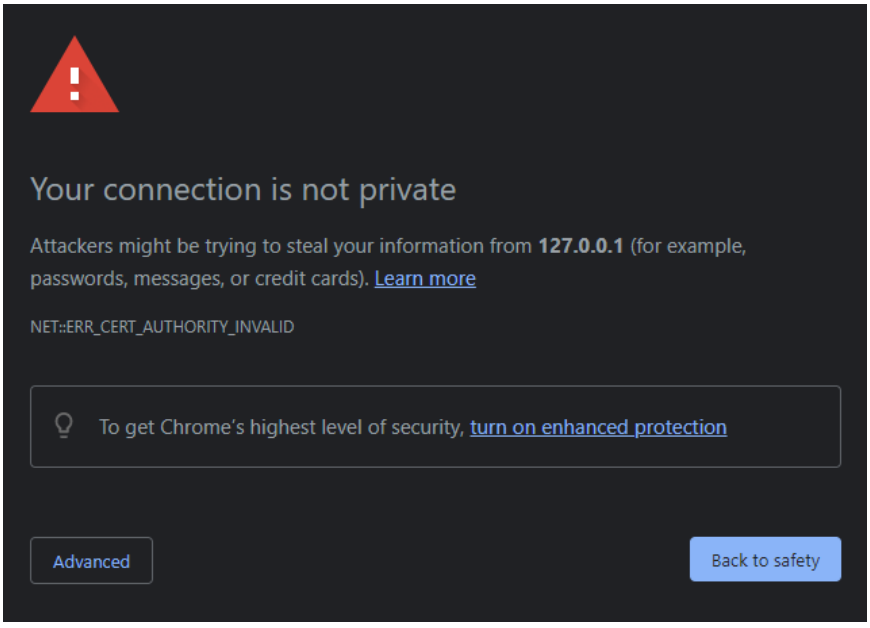
For example, here is the main folder that BANano generated (/Objects/NameOfYourWebApp)



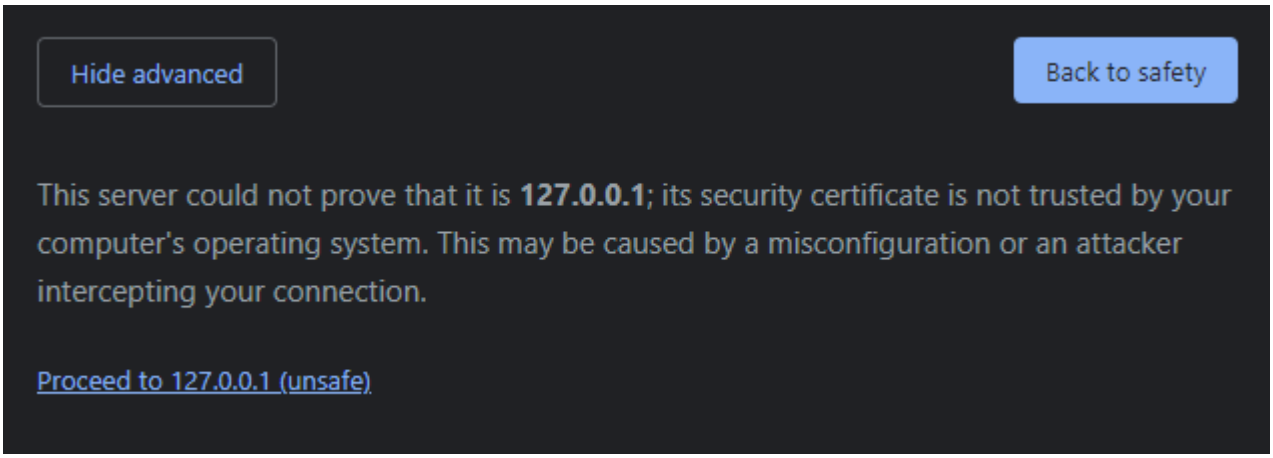
You can now browse to the URL the plugin is giving you (e.g., in this case https://127.0.0.1:8080):



Note: When opening a **https** Web App for the first time, the following Warning can be presented:



Just click on 'Advanced' and below the warning an additional text will appear:

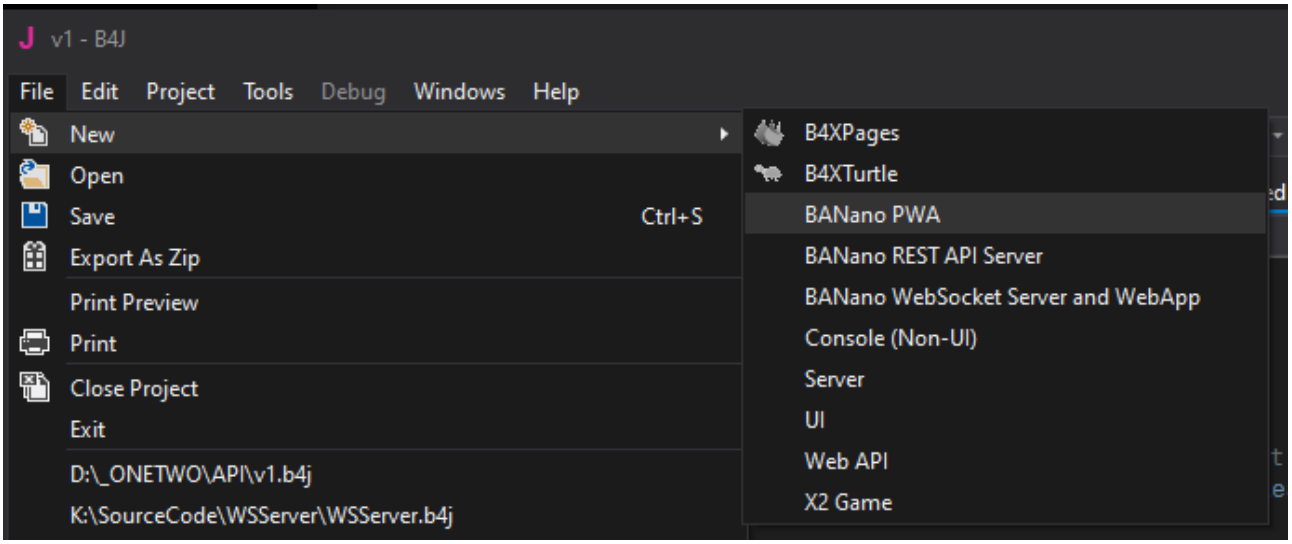


You can now click on '**Proceed to 127.0.0.1 (unsafe)**' to allow opening the Web App.

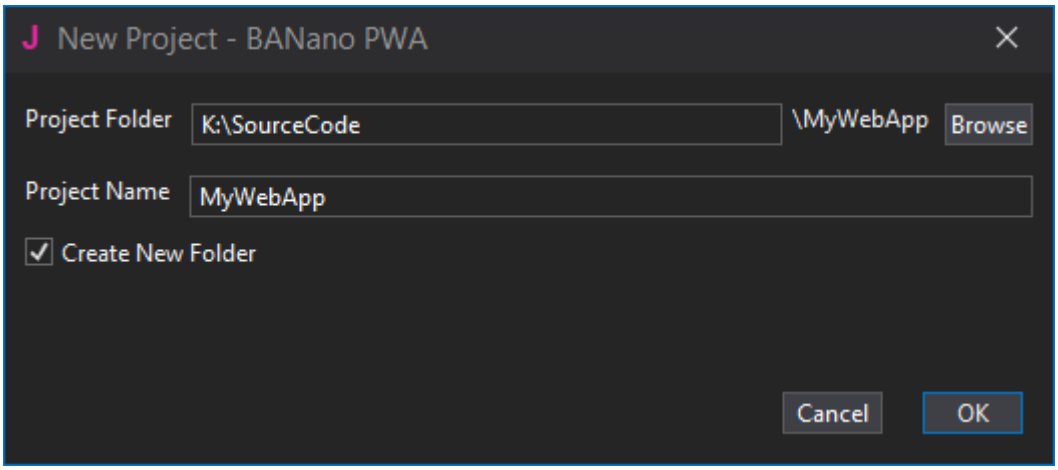
2 My first BANano Project

The installed BANano Skeleton template contains the basic code for a BANano Web App. The source code for this example as it will be automatically created if you make a new project.

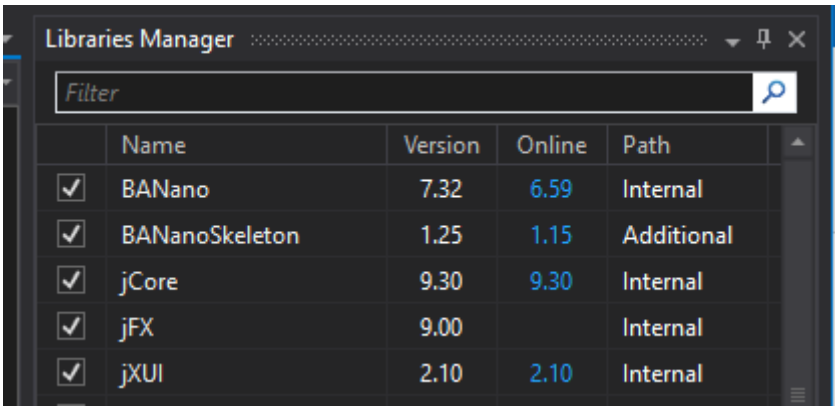
In B4J, make a new project: **File - New – BANano PWA**.



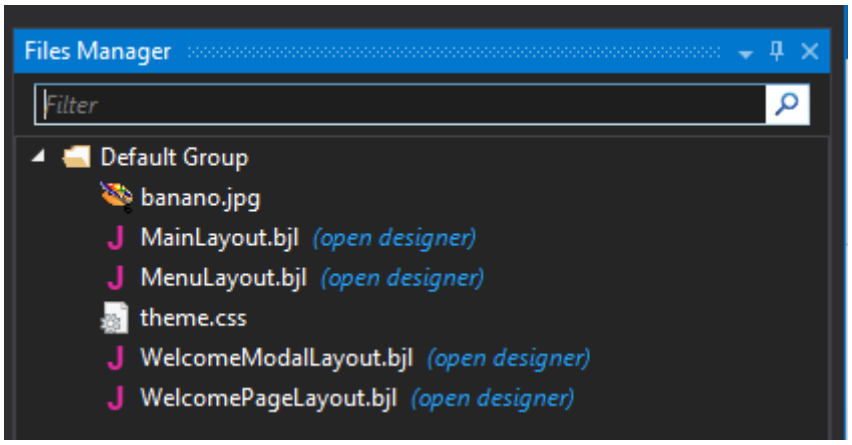
Give the project a name, e.g., MyWebApp and press OK:



You will see in the B4J Libraries Tab that the template has added the BANano and BANanoSkeleton libraries.



It has also generated some basic source code and a couple of Abstract Designer layouts (Files Tab in the IDE).



Just like in a normal B4J application, you can design you views and layouts in the Abstract Designer.

You can also add extra assets (like images, CSS, JavaScript, JSON, … files) like in a normal B4J project.

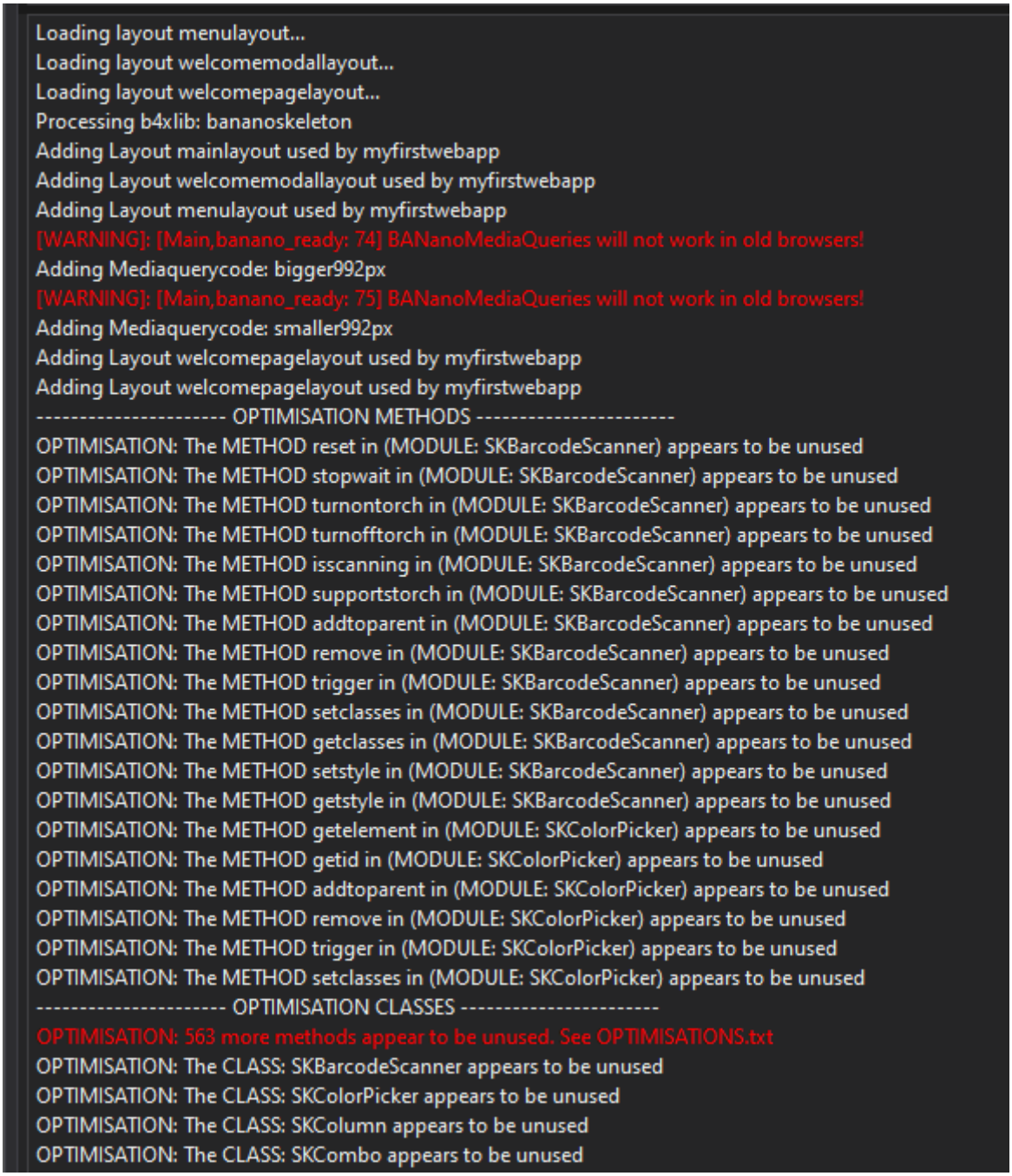
When adding new assets, make sure to Sync the folder before Building the Web App!

BANano Web Apps can be compiled just like a normal B4J application: in Debug or Release mode. Release (Obfuscated) does not do anything for a BANano project, as it has its own system that does a similar thing when compiling in normal Release mode.

When compiling (running) the project in Debug Mode, no optimizations (like removing Dead or not used Code) are done. In Release mode, this feature can be activated (see further in BANano Transpiler Options).

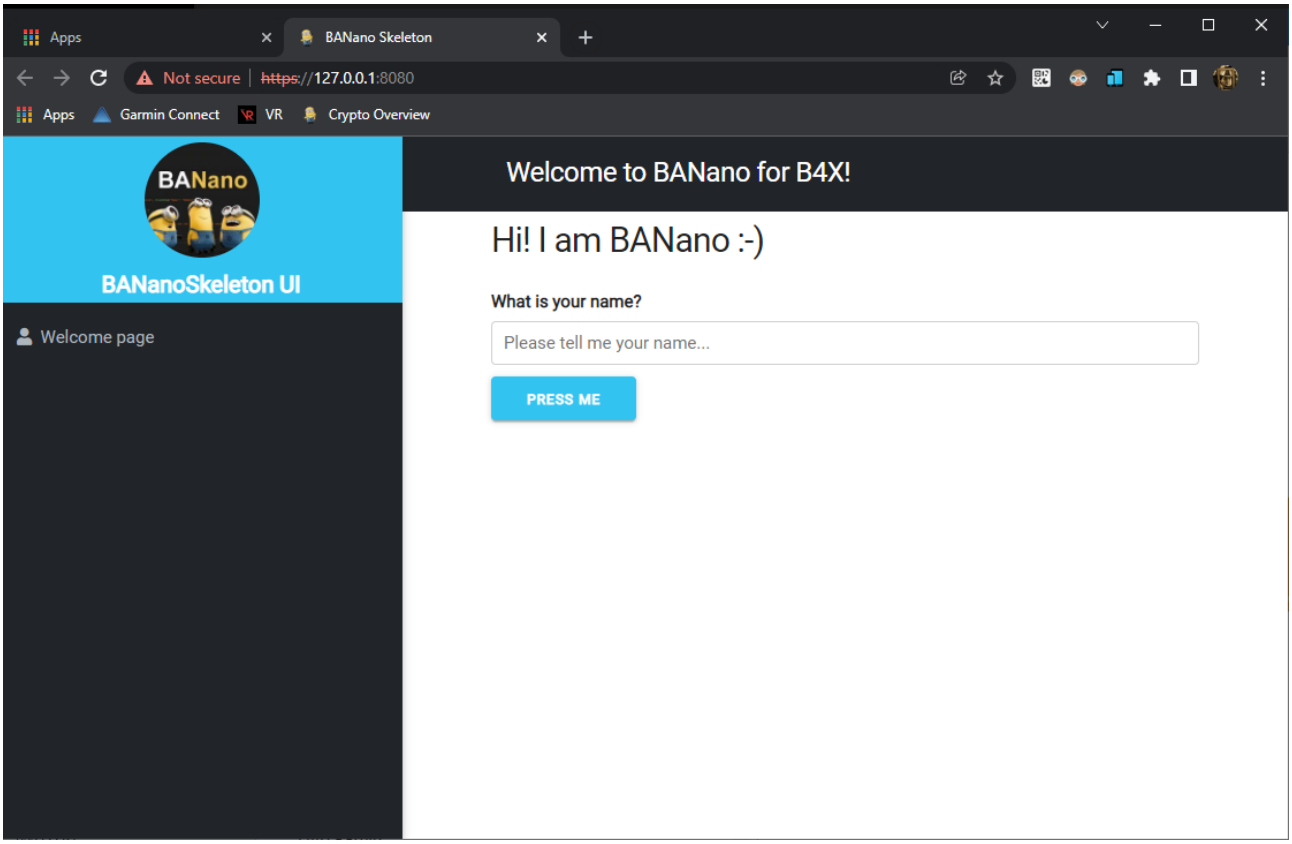
After we ran the project and selected the folder to the Chrome Web Server plugin, we can open the Transpiled project. You can follow the progress of the Transpiling in the B4J logs. It will show you if there are errors, warnings and if some optimizations can be done to make the project smaller. These optimizations will happen automatically when compiling in Release mode.

When something does not work, this is the first place to look what could've gone wrong!



When Transpiling, BANano will generate all the html, CSS and JavaScript code. This generated code can then be distributed without the need of the B4J code or any .jar file.

The result of this first project will look something like this when we open https://127.0.0.1:8080 (or whatever the Chrome Web Server plugin will show):

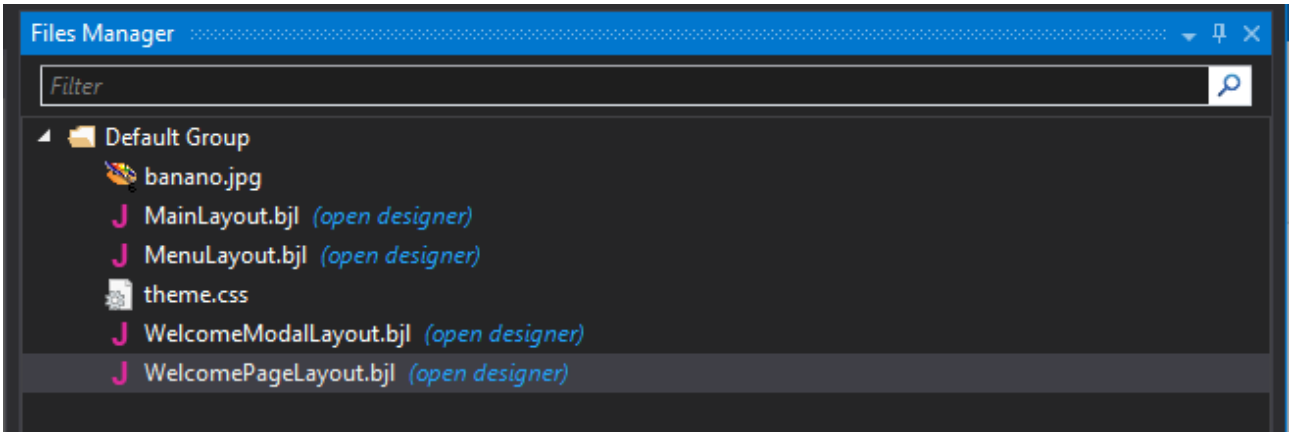


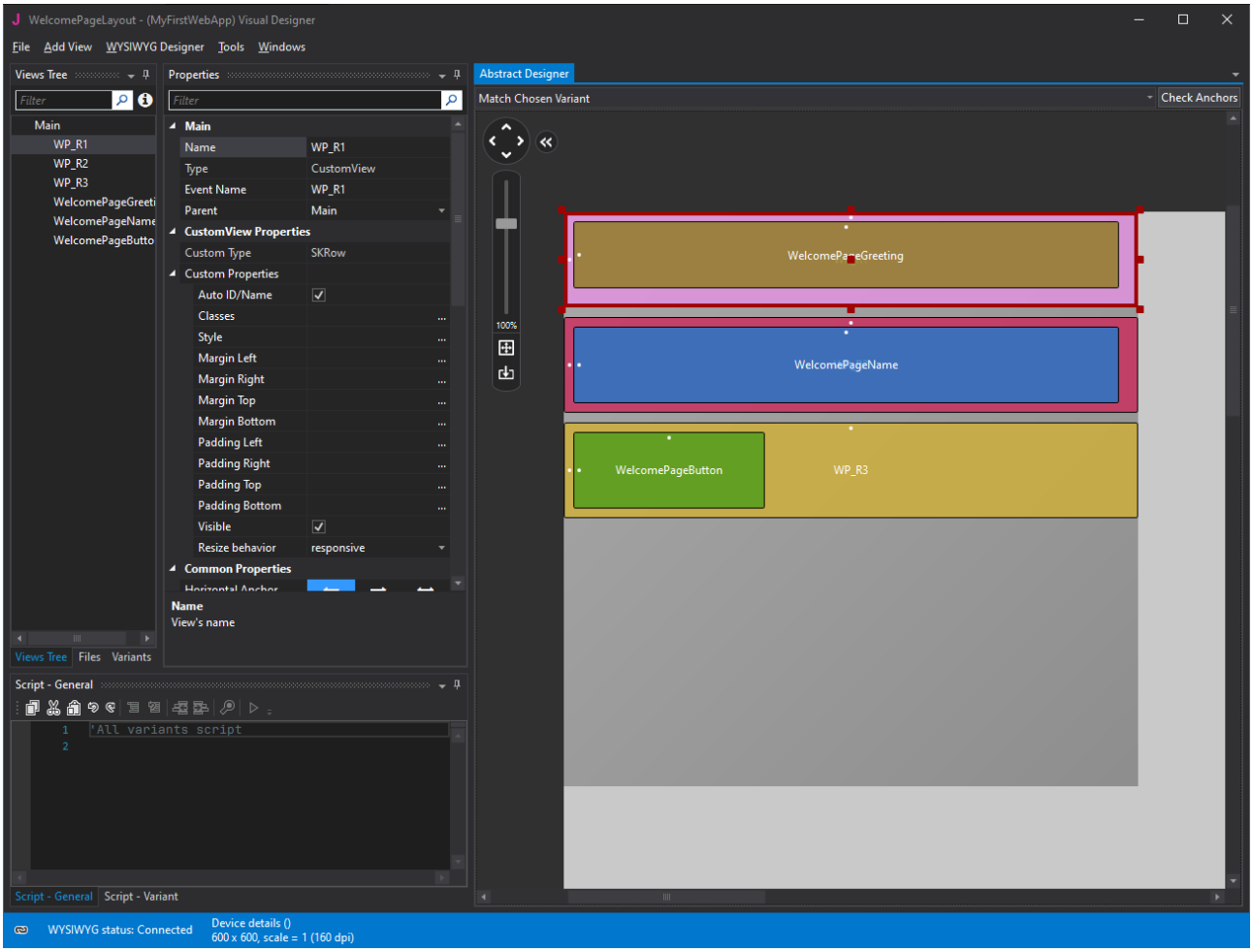
You have successfully made your first B4J BANano WebApp!

The following chapters will now break down how we got here and how you will be able to create your own functionalities in your Web Apps.

2.1 Looking into the BANano Layout files

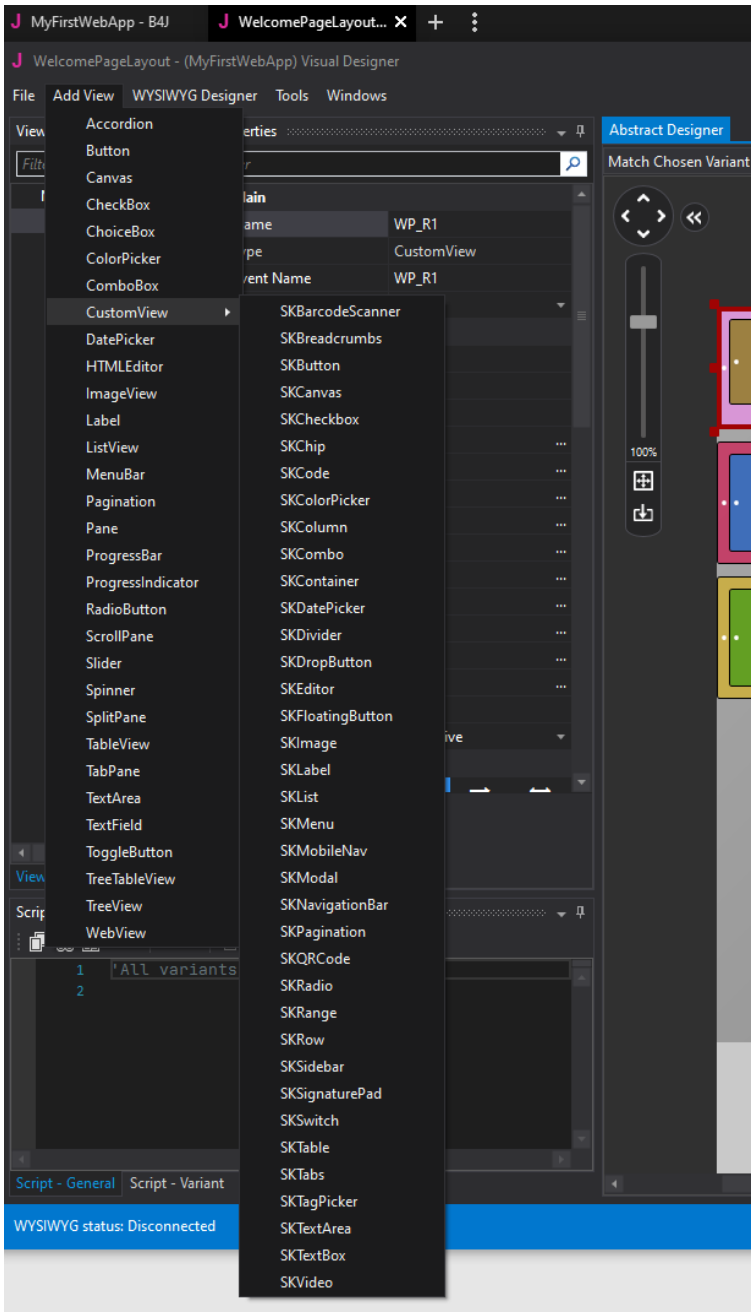
If you open the WelcomePageLayout.bjl for example in the Abstract Designer, you will see a very familiar B4J presentation.



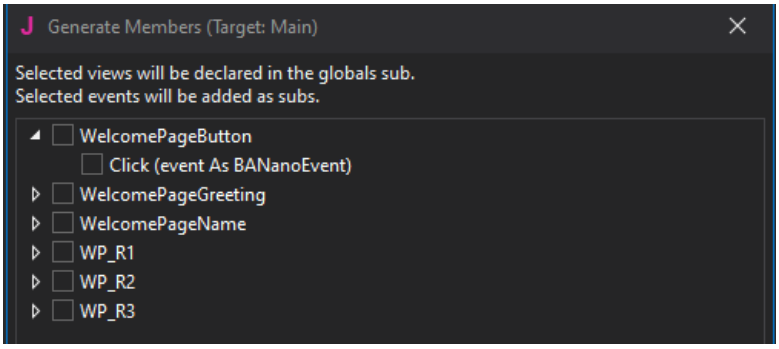


You can close the WYSIWYG form as it is not used by a BANano Project. You can use the live Browser instead.

BANano Layout files are made up **EXCLUSIVELY out of Custom Views!**



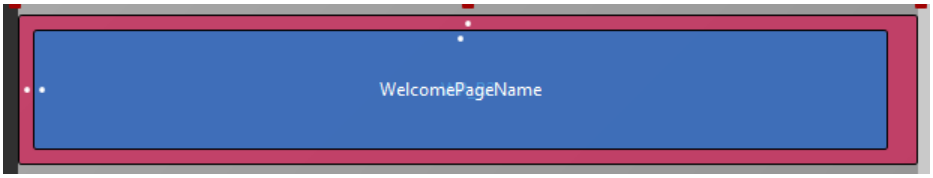
These views come from BANano Library files (like BANanoSkeleton) and are especially written to be used in BANano layouts. You can use the Generate Members to add the Components and Events to your code like in a normal B4J project.



There are a couple of rules that differ from a normal B4J Abstract Designer layout. This is because the BANano Library does not have access to some of the things in the B4J IDE like native B4J/B4A/B4i/B4R projects do.

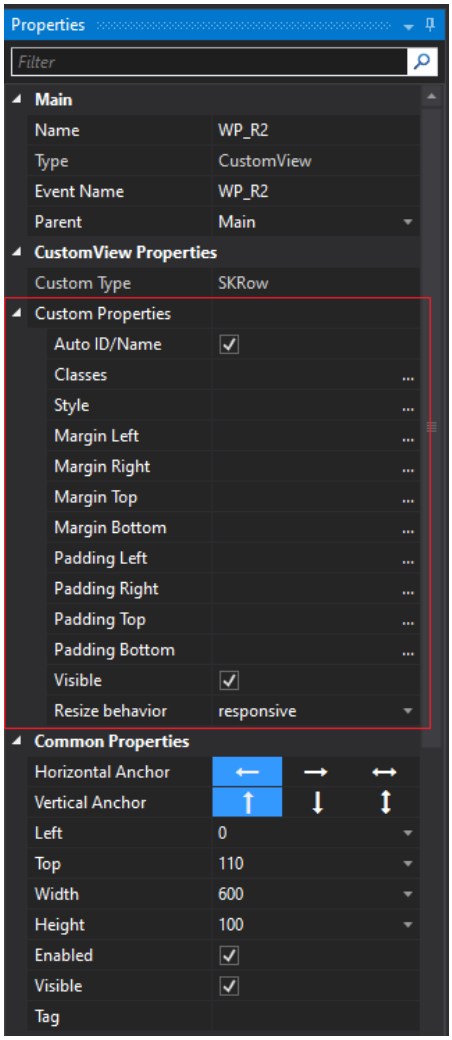
The BANano Layout Rules

1. Due to a current limitation in B4J, the Parent of a control cannot be changed for Custom Views. (It is always Main) A simple trick can be used to emulate this and the BANano Transpiler will automatically know which component is a child of another one. **All you have to do is keep some room around each control.**



Because the WelcomePageName control (blue) is smaller than the WP_R2 control (red), BANano will figure out that WelcomePageName is a child of WP_R2.

2. ONLY the properties **visible in this Screenshot (all of them)** can be used. the rest will be ignored by BANano. The ones with the red border can vary depending on the control.



The Common Properties will only be used if the builder of the BANanoLibrary uses them. In most cases, all useful properties will be in Custom Properties. The library writer will probably add an extra Left, Right, … property to the Custom Properties if they are relevant. See the documentation the specific BANano Library.

3. It is important that the Name of the controls you put on the layout are **unique across layouts if you plan to use them in your code**! This can be easily archived by giving those components a prefix: e.g., here the row control has been given the prefix WP_.

There is a special Property Auto ID/Name that can be used for controls that you do not plan to use in your B4J code. If checked, BANano will give this control a random name when transpiling so they do not interfere with other layouts.

2.2 A first look at the source code and structure of a BANano App

This is just a first glance on how the source code of a BANano Web App looks like. Following chapters will go deeper into the specifics of the BANano lingo.

A line that is always needed is **a declaration of BANano**. You can do this in every class or module as it is a singleton.

The BANano object is your main entry point to access JavaScript, HTML or CSS specific methods.

```
Sub Process_Globals
      Private BANano As BANano 'ignore
      …
End Sub
```

This is also the place where the Generated Members from the Abstract Layout Editor will add the controls.

The next important part is the **AppStart()** method. **This is the ONLY method that will NOT BE TRANSPILED to JavaScript!**

So, this is the place to set some BANano directives (like the name of your generated Web App, some Transpiler Options like remove dead code, doing some file copies and starting a BANano.Build. BANano.Build will start the Transpiler and generate everything needed for the stand-alone Web App.

```
Sub AppStart (Form1 As Form, Args() As String)
      ' you can change some output params here
      BANano.Initialize("BANano", "BANanoSkeleton",7)
      BANano.Header.Title="BANano Skeleton"
      BANano.JAVASCRIPT_NAME = "app" & DateTime.Now & ".js"

      BANano.TranspilerOptions.MergeAllCSSFiles = True
      BANano.TranspilerOptions.MergeAllJavascriptFiles = True
      BANano.TranspilerOptions.RemoveDeadCode = True
      BANano.TranspilerOptions.ShowWarningDeadCode = True

      ' write the theme
      SKTools.WriteTheme

      ' start the build
      BANano.Build(File.DirApp)

      #if release
            ExitApplication
      #end if
End Sub
```

The final **needed** method is **BANano_Ready()**. This is the main entry point to your WebApp when it is loaded. It means all needed assets (like CSS, JavaScript, etc…) is now loaded and you can start manipulating what you see in the Browser.

```
' HERE STARTS YOUR APP
Sub BANano_Ready()
	' get the body tag
	Private body As BANanoElement
	body.Initialize("#body")

	' append and load our main layout
	body.Append($"<div
id="mainHolder"></div>"$).Get("#mainHolder").LoadLayout("MainLayout")
	' append and load a modal sheet
	body.Append($"<div
id="modalHolder"></div>"$).Get("#modalHolder").LoadLayout("WelcomeModalLa
yout")

	' loading our menu in our sidebar
	MainSidebar.Element.LoadLayout("MenuLayout")

	' making the menu layout responsive: always open when screen size
is bigger than 992px
	Bigger992px.Initialize("(min-width: 992px)")
	Smaller992px.Initialize("(max-width: 991px)")

	' add our menu items
	MenuList.AddMenuItem("", "page1", "fas fa-user",
"{NBSP}{NBSP}Welcome page")
	MenuList.Start

	' load our first page
	MainPageHolder.Element.LoadLayout("WelcomePageLayout")
End Sub
```

3 Support of the B4J language

One of the major strengths of BANano is its support of the normal B4J Core Language. It covers about 99% of the B4J keywords that can be transpiled to pure JavaScript. For most of the keywords it does not support (either because they are not applicable to a WebApp or because they use some advanced B4J language feature), an alternative is provided through the BANano object.

This means you can re-use major chunks of B4X logic code directly into your BANano Web App projects!

Here is the list of keywords that are not (directly) transpilable by BANano:

```
Callsub, Callsub2, Callsub, Callsubdelayed, Callsubdelayed2,
Callsubdelayed3 ->  Use BANano.CallSub instead

Charstostring
Density
Diptocurrent
ExitApplication
Exitapplication2
File
Getenvironmentvariable
Getsystemproperty
Is -> Use BANano.IsMap, BANano.IsList, BANano.IsNull,… for more
possibilities
Isdevtool
Regex -> Use BANAnoRegEx instead
Setsystemproperty
Smartstringformatter
Stopmessageloop
Startmessageloop

(*) From BANano 7.35+, the normal CallSub/CallSubDelayed methods are also
supported
```

All variable Types (including Maps, Lists, Type, Arrays and StringBuilders) are supported. You can make Classes, use Modules, SmartStrings and all the powerful stuff Erel has provided in the B4J language. You make your normal For and While loops, your If and Select Case conditions, your variable declarations etc…

Limitations of the DateTime type. The following methods are supported:

```
Now,DateParse,TimeParse,Date,Time,Dateformat,TimeFormat,Add,GetYear,GetMonth,Get
DayOfMonth,GetDayOfYear,GetDayOfWeek,GetHour,GetMinute,GetSecond,GettimezoneOffs
etAt,TicksPerDay,TicksPerHour,TicksPerMinute,TicksPerSecond,TimezoneOffset,SetTi
meZone
```

For more advanced Date and Time functionalities you can use an external library like moment.js

Additional, BANano contains special objects that replace some of the other core libraries. They are not directly interchangeable with the B4J ones because they cannot exactly be matched. They are still very similar, but may have additional methods/properties available in JavaScript, or can be missing some typical B4J ones.

Some of those special objects available now are:

BANanoJSONGenerator <-> B4J JSONGenerator
BANanoJSONParser <-> B4J JSONParser
BANanoWebSocket <-> B4J WebSocket
BANanoMQTT <-> B4J jMQTT
BANanoRegEx <-> B4J RegEx
BANanoSQL <-> B4J jSQL

(*) DEPRECIATED: From BANano 7.35+, you can just use the normal B4J Json and jMQTT libraries!

More mappings of other B4J libraries may be added in future versions of BANano. E.g., when Web Apps fully support Bluetooth, a BANanoBluetooth object may become available.

The golden rules in BANano are:

1. Be as close to the B4X language as possible
2. Support as many browsers as possible (meaning if something is only available for e.g., Chrome but not in the other major browsers yet, it will not be part of the BANano Core library).

It should be the responsibility of BANano Library Builders to try to follow these rules also as close as possible.

This maximizes cross-platform **AND** cross-browser compatibility, without the programmer having to worry about it.

4 The Web Connection

Next to the core B4J language, BANano has a lot of objects and methods that make the link between B4J and HTML, CSS and JavaScript. They have both a 'B4J' and a 'Web' feel. They can use the fantastic B4X AutoComplete features (which speeds up programming dramatically!) and you can write/use them in a **very familiar B4J way**. But they are close enough to JavaScript so one can recognize how to write something that looks very similar. Comes in handy when you want to use some JavaScript snippet on the internet and you want to translate it to B4J code.

They are kind of 'The Best Of Both Worlds' objects!

Some of them are:

BANano Core Objects:
BANanoObject
BANanoElement
BANanoEvent
BANanoPromise
BANanoFetch
BANanoXMLHttpRequest (for legacy only, use the more modern BANanoFetch instead)

Cannot be initialized, but are properties of the main BANano Object
BANanoConsole
BANanoWindow
BANanoHistory
BANanoLocation
BANanoNavigator
BANanoScreen
BANanoGeoLocation (and the resulting BANanoGeoPosition)

JavaScript objects
BANanoURL
BANanoMediaQuery
BANanoMutationObserver

BANano Specials:
BANano Background Workers
BANanoRouter

We will go deeper into most of them in the next chapters, but it suffices for the moment that you are aware they exist.

Another important Web Connection in BANano is the ability to write **raw JavaScript and CSS right into your code**! Very similar to B4Js possibility to write raw Java in the code.

IMPORTANT! This can also be done with just B4J and BANano code. But they are available if you do know CSS or JavaScript for a quick fix, or if something would be missing in the BANano library. I will try to avoid to use this method as much as possible in the examples I will give in this manual and use pure B4J/BANano instead.

This is for example perfectly possible to write this CSS styling snippet anywhere in your code:

```
#If CSS
.hidelist {
      display: none;
}
.clock-loader {
  --clock-color: #FF8800;
  --clock-width: 2rem;
  --clock-radius: calc(var(--clock-width) / 2);
  --clock-minute-length: calc(var(--clock-width) * 0.4);
  --clock-hour-length: calc(var(--clock-width) * 0.25);
  --clock-thickness: 0.2rem;
  position: relative;
  display: flex;
  justify-content: center;
  align-items: center;
  width: var(--clock-width);
  height: var(--clock-width);
  border: 3px solid var(--clock-color);
  border-radius: 50%;
}
.clock-loader::before, .clock-loader::after {
  position: absolute;
  content: "";
  top: calc(var(--clock-radius) * 0.25);
  width: var(--clock-thickness);
  background: var(--clock-color);
  border-radius: 10px;
  transform-origin: center calc(100% - calc(var(--clock-thickness) / 2));
  animation: spin infinite linear;
}
.clock-loader::before {
  height: var(--clock-minute-length);
  animation-duration: 2s;
}
.clock-loader::after {
  top: calc(var(--clock-radius) * 0.125 + var(--clock-hour-length));
  height: var(--clock-hour-length);
  animation-duration: 15s;
}
#End If
```

BANano will recognize this code when transpiling and automatically add it to a .css file.

A similar example for JavaScript:

```
#if JAVASCRIPT
function evaluate(s) {
   // so we get back a string
   return '' + eval(s);
}
#End If
```

And we can call it in B4J like this:

```
Log(BANano.RunJavascriptMethod("evaluate", Array As String("10 * 20")))
```

The BANano.RunJavascriptMethod() can also be used in a similar way to call JavaScript methods from external JavaScript files.

You can also use #If SMARTJAVASCRIPT instead and use the ${var} of SmartStrings.

Example:

```
plot.Initialize("Plotly")

Dim body As BANanoElement
body.Initialize("#body")
body.Append($"<div id="chart" style="width:600px;height:250px;"></div>"$)

Dim element As String = "chart"
Dim coords As List
Dim margin As Map
Dim settings As Map

#If JavaScriptSmart
    ${coords} = [{x: [1, 2, 3, 4, 5], y: [1, 2, 4, 8, 16] }]
    ${margin} = {margin: { t: 0 } }
    ${settings} = {showSendToCloud:true}
#End If

plot.RunMethod("newPlot", Array(element, coords, margin, settings ))
```

BANano can even include PHP (with #If PHP … #End If) that will be added to a .php file, but it is limited as the preferred way is using B4J's **jServer** (or with the **BANanoServe**r, which is just an easy-to-use wrap around B4J's jServer), a full blown **embedded Jetty Web server**.

5 BANanoObject: The jack-of-all-trades

This is probably the toughest topic in BANano. I think it is important you understand what this object is and does, especially if you plan to use or wrap some existing JavaScript library or code. Even if you don't plan to do this, at least try to grasp the essence of what it is, as it can be at one point in your development become a life saver. The rest will be easier, I promise!

It is rather advanced BANano usage, but somehow I feel this story needs to be told before all the rest.

BANanoObject is probably the most powerful object in BANano. It can be or do about anything and is best comparable with an B4J Object on steroids. Many other objects like BANanoElement are children of the BANanoObject class and inherit many of its functionalities (like RunMethod or SetField).

Important note before we start: about parameters…

As B4J does not allow a variable number of in its methods, but this is very common in JavaScript, we have to use some trickery to do this anyway. For this, we use the B4J **Array** keyword.

For example, a method is declared as this and can have one or more parameters:

```
Sub MyFunction(params As Object)

End Sub
```

We can now pass one parameter:

```
MyFunction("Alain")
```

Or multiple parameters:

```
MyFunction(Array("Alain", 48, "Ieper"))
```

One of these parameters can be an Array in itself. How do we do that? By adding an extra Array.

```
MyFunction(Array(Array(48, 174)))
MyFunction(Array("Alain", Array(48, 174), "Ieper"))
```

Quiz: what is the different output between these two lines after transpiling?

```
MyFunction(Array(48, 174))
MyFunction(Array(Array(48, 174)))
```

Answer:

The first one will call the method as `MyFunction(48,174)` ' two sperate parameters
The Second one will call the method as `MyFunction(Array(48,174))` ' one parameter that is an array

Simple rule to remember: The outer Array will be removed when Transpiling.

OK, are you ready? Let's GO!

Because of its many possible appearances, it has **several initialization methods** depending on how you want to use it. I marked the important ones in Red. The others are really advanced ones you will probably never encounter when making a Web App. Just read them to know they exist in case you encounter a weird JavaScript library declaration.

Initialize(jsObject As Object)

This is its most basic form. It is the equivalent of =. Suppose you use some library like jQuery and you want to make a reference to it in B4J.

You would write:

```
Dim JQ as BANanoObject
JQ.Initialize("$") ' $ is in jQuery how you can access its methods
                   ' and properties.  You could compare it with
                   ' what the BANano object is in B4J.
```

Now you can use all the methods and properties of jQuery in B4J (with e.g. the RunMethod or GetField methods, see further)

Initialize2(jsObject As String, params As Object) As BANanoObject

Initializes the object with a New instance of a JavaScript class. While a library like jQuery is more like a 'Module' in B4J, some JavaScript libraries are more like a B4J 'Class' that need an .Initialize().

Take for example some JavaScript library called "When" (that is a DatePicker). The JavaScript documentation would show you would need to initialize an instance of the "When" class by writing this:

```
let datepicker = new When({input: '#datepicker', singleDate: true});
```

In BANano we would use the Initialize2() method:

```
Dim datepicker As BANanoObject
datepicker.Initialize2("When", CreateMap("input": "#datepicker",
"singleDate": True))
```

Initialize3(params As Object) As BANanoObject

This initialize is used to call some constructor (= an initialization method) of a JavaScript object. **It is rather rare that you will have to use this one.**

Let's take for example the RecordRTC (an audio/video library) JavaScript library. It acts like a module (see the normal Initialize() method), but you have to Initialize it differently (it needs other parameters) depending on what you want to do with it. Record audio, record video, ...).

So, if you want to record video, you would need to do this in JavaScript according to its documentation:

```
let recorder = RecordRTC(stream, {'type': 'video', 'mimeType':
'video/webm', 'videoBitsPerSecond' : 128000}
```

And for Audio:

```
let recorder = RecordRTC(stream, {'type': 'audio', 'mimeType':
'audio/webm', 'audioBitsPerSecond' : 128000}
```

As you can see, there is no New and RecordRTC acts as a module, not as a class.

So, in B4J we can use the Initialize3() method to get the Recorder:

```
' first we want to 'grab' the RecordRTC library module itself
' (using the normal Initialize)
Dim RecordRTC as BANanoObject
RecordRTC.Initialize("RecordRTC")

' and now 'run' its constructor method to get the Recorder.
Recorder = RecordRTC.Initialize3(Array(Stream, CreateMap("type": "video",
"mimeType": "video/webm", "videoBitsPerSecond" : mVideoBitsPerSecond)))
```

Similar for the Audio:

```
Dim RecordRTC as BANanoObject
RecordRTC.Initialize("RecordRTC")

Recorder = RecordRTC.Initialize3(Array(Stream, CreateMap("type": "audio",
"mimeType": "audio/webm", "audioBitsPerSecond" : mAudioBitsPerSecond)))
```

Initialize4(jsObject as String, params as Object) As BANanoObject

This is basically the same as .Initialize, but with parameters. It does NOT do a New like the Initialize2 method.

Initialize5() As BANanoObject

This Initializes the object to plain JavaScript Object. This is basically set the object in JavaScript to {}. You can use all the other BANanoObject methods like SetField on it.

Initialize6(javaScriptObject As String)

Initialize a BANanoObject from a JavaScript object, defined as a B4J SmartString. It is a shortcut method for Initialize5() where you would use the .SetField() method.

Example:

```
Dim b As BANanoObject
b.Initialize6($"{body: "myBody", name: "myName", city: "Ieper"}"$)
```

Initalize5() could be re-written as:

```
Dim b As BANanoObject
b.Initialize6("{}")
```

Initialize7(javaScriptObject As Object, constructor as String, params as Object)

Another advanced declaration you will probably never encounter.

Let's say in JavaScript you see something like this:

```
var innerConn = ...
var query = new innerConn.$sql.Query(SQL);
```

In B4J this would become:

```
Dim Query As BANanoObject
Query.Initialize7(innerConn, "$sql.Query", SQL)
```

Congratulations! You just made it through probably the hardest part of BANano!

Now we can start doing some fun stuff with the BANanoObject we created. I'm not going to go through all of the methods that can be used on this object, just the most commonly used ones. You can always check the quick reference if you would need one of the other ones. If you understand these couple of methods below, you're there!

RunMethod(methodName As String, params As Object) As BANanoObject

Runs a method of the BANanoObject. Suppose we have declared a BANanoObject like this:

```
Dim SomeLib As BANanoObject
SomeLib.Initialize("SomeJavaScriptLib") ' where "SomeJavaScriptLib" is
some JavaScript library.
```

Now, we know from the JavaScript documentation of this library that it **has a method Start.**

We can then run this on our BANanoObject:

```
SomeLib.RunMethod("Start", null) ' case sensitive, and pass null because
this function has no parameters.
```

Another example, if the JavaScript library has a method **Sum**(x,y).

This would be called as:

```
Dim mySum as Long = SomeLib.RunMethod("Sum", Array(10,20)) ' result is 30
```

Result() As Object

This method can be used if the result of e.g. RunMethod is something different than the B4J IDE expects. Internally it does nothing, but you get rid of the B4J error or warning.

Example (should for example the Sum method return something different than a long and the IDE gives an error):

```
Dim mySum as Long = SomeLib.RunMethod("Sum", Array(10,20)).Result
```

SetField(field As String, value As Object)

Sets a property on a BANanoObject.

Example:

```
Dim myObj as BANanoObject
myObj.Initialize5
myObj.SetField("prop1", value1)
myObj.SetField("prop2", "value2")
```

Getfield(field As String) As BANanoObject

Returns the value of a property.

Example:

```
Log(myObj.GetField("prop1"))
```

Delete(property As String)

Deletes a property from a BANanoObject.

Example:

```
myObj.Delete("prop1")
```

HasOwnProperty(property As String) as Boolean

Check if the property is native to the object, or inherited by a parent object.

ToString() As String

Converts the object to a string

A lot of the methods are chainable. This means you can call one after the other.

Example:

```
Dim myResult as String
myResult = myObj.GetField("prop").GetField("subprop").RunMethod("calc",
Array(10,20)).Result
```

As said, there are other methods available on this object to explore. Some of them will be explained further on in the examples as we go as they need a more extensive context.

6 BANanoElement: Talking to the DOM

6.1 Introduction

As shown in the previous chapter, we have learned how B4J can interact with all kind of JavaScript objects with BANanoObject. But how about the interaction with the UI? This is where BANanoElement comes in.

BANanoElement has some of the methods from the BANanoObject, like GetField() and RunMethod() etc. But its main purpose is **talking to the browser DOM**. It is a wrap of the well-used Umbrella framework, which is a very lightweight Vanilla JavaScript alternative to the maybe better known but bloated jQuery library.

This is the element you use to build HTML tags, set styles, add and remove CSS classes to change the appearance of the tag, add events like click or hover and a lot more.

In this chapter we will go over how Web UI controls can be created easily in B4J using BANano.

Learning to work with this framework has the huge advantage of speed, as you talk directly to the browser and don't have to go through other heavy JavaScript frameworks for example. **BANano is the native B4J answer to them!**

Speed is very important nowadays if you want your PWA to succeed. Users are becoming more are more demanding and BANano gives you the tools to give them what they want. The **BANanoSkeleton** library, which does talk directly to the browser, makes sure you get the maximum chance to do so and is still not very complicated to use, thanks to the B4X philosophy.

6.2 Using HTML tags, with style!

6.2.1 Getting existing tags

HTML Tags can be identified uniquely if they have an **id** property, or a bunch of them together, e.g. all the tags with **a certain style class** or from a **certain type** (div, button, input, …).

You can make this selection using the Initialize() method.

Initialize(target As Object)

To get a certain unique tag, use "#" + idName. **The idName is case sensitive**!

Example:

```
Dim element As BANanoElement
element.initialize("#myId")
```

To get a group of tags, you can either use a class (with the **.** **prefix**), or use the **tag name**.

Suppose this is or HTML:

```
<div>
      <button class="mybutton">Button 1</button>
      <button class="mybutton">Button 2</button>
      <button class="mybutton">Button 3</button>
</div>
<div>
      <button class="mybutton2 mycolor">Button 4</button>
      <button class="mybutton2 mycolor">Button 5</button>
      <button class="mybutton2 mycolor">Button 6</button>
</div>
```

We can grab all buttons now with:

```
Dim element As BANanoElement
element.Initialize("button") 'tag name button
```

The result will be that BANanoElement contains **ALL 6 buttons**.

If we want to grab only the buttons which have the **class "mybutton2"**, you can use:

```
Dim element As BANanoElement
element.Initialize(".mybutton2") ' notice the dot before mybutton2
```

This may look strange at first for a B4J programmer that one BANanoElement can be multiple Tags. But it has big advantages!

Instead of having to add a click event to every single button one by one, we can do them all at once:

```
Dim element As BANanoElement
element.Initialize("button")
element.on("click", Me, "handleClick")
```

Now, each time **one of these buttons is clicked**, it will call the handleClick method.

6.2.2 Creating new tags

This way of 'grabbing' one or more tags can of course only be done if the tags already exist. How about creating a brand new one?

For this we use the BANano.**CreateElement**() method

BANano.CreateElement(Tag as String)

This creates a new BANanoElement with the HTML tag "Tag". Note that it is NOT attached to anything (yet)!

```
Dim newElement As BANanoElement
newElement= BANano.CreateElement("div")
```

We can now do al kind of things with the 'virtual' tag, like adding styles, classes, even other BANanoElements.

```
Dim newElement As BANanoElement
newElement= BANano.CreateElement("div")
newElement.AddClass("myCSSclass1 mycolor2")
newElement.SetAttr ("id", "myDiv")
newElement.SetText("my text")
```

The result of this code will be in html:

```
<div id="myDiv" class="myCSSclass1 mycolor2">my text<div>
```

6.2.3 Adding the tags to the DOM

As explained before, it is not yet attached to anything in the browser. We can now attach this html to another BANanoElement with several methods like Append, Before, After, Replace and Prepend:

Append(htmlOrObject As Object) As BANanoElement

Add some html as a child at the end of each of the matched elements

```
Dim body as BANanoElement  ' note that the body tag does ALWAYS exist!
body.Initialize("body")

body.Append(newElement)
```

Before(htmlOrObject As Object) As BANanoElement

Add some html as a sibling before each of the matched elements

```
dim allWithClassMyButton as BANanoElement
allWithClassMyButton.initialize(".mybutton")

allWithClassMyButton.Before(newElement)
```

After(htmlOrObject As Object) As BANanoElement

Add some html as a sibling after each of the matched elements

```
dim allWithClassMyButton as BANanoElement
allWithClassMyButton.initialize(".mybutton")

allWithClassMyButton.After(newElement)
```

Replace(htmlOrObject As Object) As BANanoElement

Replace the matched elements with the passed elements

```
dim allWithClassMyButton as BANanoElement
allWithClassMyButton.initialize(".mybutton")

allWithClassMyButton.Replace(newElement)
```

Prepend(htmlOrObject As Object) As BANanoElement

Add some html as a child at the beginning of each of the matched elements

```
dim allWithClassMyButton as BANanoElement
allWithClassMyButton.initialize(".mybutton")

allWithClassMyButton.Prepend(newElement)
```

As you may have noticed, the parameter in these methods is called "htmlOrObject".

This means, next to appending another BANanoElement, we can also just add the HTML as a string.

Example:

```
Dim body as BANanoElement  ' note that the body tag does ALWAYS exist!
body.Initialize("body")

body.Append($"<div id="myDiv" class="myCSSclass1 mycolor2">my
text<div>$")
```

These methods return a BANanoElement. This is NOT the newly created one, but the original one.

If we want to return other tag, we can use for example the Get() method with the ID.

```
Dim myDiv as BANanoElement = body.Append($"<div id="myDiv"
class="myCSSclass1 mycolor2">my text<div>$").Get("#myDiv").Result
```

There are several other methods you can use to "get" other tags, like **.Find**(), .**Filter**(), .**Closest**(), .**Siblings**(), .**First**(), .**Last**(), … See the **quick reference** for more info on these methods.

6.2.4 Removing Tags (or only its children)

You can remove tags in two ways: the tag and all its children, or just its children.

Remove() As BANanoElement

Removes the matched elements. Example, suppose this is our html:

```
<div id="myDiv1">
      <button class="mybutton">Button 1</button>
      <button class="mybutton">Button 2</button>
      <button class="mybutton">Button 3</button>
</div>
<div id="myDiv2">
      <button class="mybutton2 mycolor">Button 4</button>
      <button class="mybutton2 mycolor">Button 5</button>
      <button class="mybutton2 mycolor">Button 6</button>
</div>
```

And we use this code:

```
Dim myDiv2 As BANanoElement
myDiv2.Initialize("#myDiv2")

myDiv2.Remove
```

Result:

```
<div id="myDiv1">
      <button class="mybutton">Button 1</button>
      <button class="mybutton">Button 2</button>
      <button class="mybutton">Button 3</button>
</div>
```

Empty() As BANanoElement

Remove all child nodes of the matched elements. Example, suppose this is our html:

```
<div id="myDiv1">
      <button class="mybutton">Button 1</button>
      <button class="mybutton">Button 2</button>
      <button class="mybutton">Button 3</button>
</div>
<div id="myDiv2">
      <button class="mybutton2 mycolor">Button 4</button>
      <button class="mybutton2 mycolor">Button 5</button>
      <button class="mybutton2 mycolor">Button 6</button>
</div>
```

And we use this code:

```
Dim myDiv2 As BANanoElement
myDiv2.Initialize("#myDiv2")

myDiv2.Empty
```

Result:

```
<div id="myDiv1">
      <button class="mybutton">Button 1</button>
      <button class="mybutton">Button 2</button>
      <button class="mybutton">Button 3</button>
</div>
<div id="myDiv2">
</div>
```

6.2.5 Looping through a multi-tag BANanoElement

As we have seen, BANanoElement can sometimes hold more than one tag (see the Button example)

So, suppose you want to do something with each of these tags separate. For this you can use the **EachStart** and **EachEnd** methods.

Those coupled methods are working in a special way in BANano. If works like an If Then – End If in B4J, meaning the code in between is executed as a whole.

Example:

```
Dim AllButtons As BANanoElement
AllButtons.Initialize("button")
```

Now we have all our buttons, we can now loop through them like this:

```
' some temporary variables the EachStart needs.
Dim OneButton as BANanoElement
Dim index as long

AllButtons.EachStart(OneButton, index)
      Log(index)
      Log(OneButton.GetAttr("id"))
AllButtons.EachEnd
```

6.2.6 Styling Tags

Styling Tags can be done with CSS. You can use the BANanoElement **AddClass / ToggleClass / RemoveClass** methods to assign a CSS class.

Example:

```
' hides the tag
#If CSS
.hidelist {
      display: none;
}
#End If

ListHolder.AddClass("hidelist")
```

It can also be done with the .**SetStyle**() method. You should always prefer CSS classes over using style. The latter should only be used if classes "can't handle it".

The parameter in .SetStyle is Json! So do mind the correct quotes and comma (instead ; in CSS)

```
ListHolder.SetStyle($"{"margin-right": "0px", "margin-top": "8px"}"$)
```

6.2.7 BANanoEvent: Working with Events

Time to put our BANanoElements to work!

6.2.8 Adding Events

We can simply add events to a BANanoElement using the .**On**() method, or by using an **AddEventListener**() or by using the **HandleEvents**(). They do approximately the same, it is a matter of preference. AddEventListener() is a bit more flexible and is the hardcore JavaScript way you will see used a lot in examples on the internet.

On(events As String, module as Object, method As String) As BANanoElement

Example:

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.On("click", Me, "handleClick") 'case sensitive!

Sub handleClick(event As BANanoEvent)
    BANano.Alert("Stop clicking me!")
End Sub
```

The method, here handleClick MUST have this signature:

```
Sub MethodName(event As BANanoElement)

End Sub
```

AddEventListener(eventName as String, callbackMethod As Object, useCapture as Boolean)

useCapture: A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.

true - The event handler is executed in the capturing phase
false - The event handler is executed in the bubbling phase

This is the alternative way to so the same:

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.AddEventListener("click", BANano.Callback(Me, "handleClick", Null),
true)

Sub handleClick(event As BANanoEvent)
    BANano.Alert("Stop clicking me!")
End Sub
```

HandleEvents(events As String, module As Object, method As String) As BANanoElement

Does exactly the same as the .On() method, except it will automatically do an event.PreventDefault.

The **preventDefault** method of the event tells the user agent that if the event does not get explicitly handled, its default action should not be taken as it normally would be.

The event continues to propagate as usual, unless one of its event listeners calls event.**StopPropagation** which terminates propagation at once.

Example:

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.HandleEvents("click", Me, "handleClick") 'case sensitive!

Sub handleClick(event As BANanoEvent)
    BANano.Alert("Stop clicking me!")
End Sub
```

This would be the same as:

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.On("click", Me, "handleClick") 'case sensitive!

Sub handleClick(event As BANanoEvent)
     Event.PreventDefault
     BANano.Alert("Stop clicking me!")
End Sub
```

6.2.9 Removing Events

This can be done using the **Off**() method or **RemoveEventListener**(). Again, it is a choice of preference.

So, suppose we want to remove our previously added click event.

Off(events as String)

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.Off("click")
```

RemoveEventListener(eventName As String, callbackMethod As Object, useCapture As Boolean)

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.RemoveEventListener("click", BANano.Callback(Me, "handleClick",
Null), true)
```

Off and On are often used together chained. This makes sure an event doesn't run twice.

Example:

```
Dim myDiv As BANanoElement
myDiv.Initialize("#myDiv")

myDiv.Off("click").On("click", Me, "handleClick") 'case sensitive!

Sub handleClick(event As BANanoEvent)
     Event.PreventDefault
     BANano.Alert("Stop clicking me!")
End Sub
```

6.3 Loading Abstract Designer Layouts

Of course, all the above is at its deepest level. Thankfully, we have B4Js Abstract Designer that can declare most of these things if we use an UI library like **BANanoSkeleton**. BANanoSkeleton has done all that already for you! But I find it important you are aware of how it all works internally.

Loading Layouts in BANano is almost identical to doing it in any other B4X product. It has some extra Load methods that will help you.

LoadLayout(layoutName As String)

```
' create a new BANanoElement pageHolder that can hold our layout
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")

pageHolder.LoadLayout("MainLayout")
```

LoadLayoutAppend(layoutName As String)

Same as LoadLayout but does not empty the BANanoElement that will hold the layout.

LoadLayoutArray(layoutName As String) As Long

This is useful if you want to load the same layout several times, e.g. to build some kind of list of items where each item is using the same layout. It does not empty the holding BANanoElement.

It will return a unique number (long) that has been added as suffix to every view in the layout.

Example situation:

Suppose we have a layout that has a button called btnStop on it. When we run to add the layout 3 times:

```
For i = 0 to 2
      Dim index as long = pageHolder.LoadLayoutArray("myLayout")
      Dim views As Map = BANano.GetAllViewsFromLayoutArray(Me,
"myLayout", index)
      Dim btnStop as SKButton = views.Get("btnstop")
      btnStop.Tag = index ' or whatever can identify which item in the
list this is.
Next
```

Each btnStop will now have a unique suffix number behind it. So instead of btnStop, in the HTML the ids will be btnstop_1, btnstop_2 and btnstop_3.

BUT: BANano is smart enough so you can still use ONE event to handle the click:

```
Private Sub btnStop_Click (event As BANanoEvent)
      Dim btnStop As SKButton = Sender
      Dim id As Int = btnStop.ID.Replace("btnstop_", "")

End Sub
```

Notice the use of B4Js Sender Object!

By just using this line, we got the exact btnStop the user clicked on.

```
Dim btnStop As SKButton = Sender
```

Loading a layout must be done directly on a BANanoElement, not via a method or chaining.

It is a Transpiler limitation.

Example:

Will work:

```
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")

pageHolder.LoadLayout("MainLayout")

' we put it first in a separate variable UserTab
Dim UserTab As BANanoElement = SKTabs1.GetTabContents(0)
' now we load the layout on this UserTab variable
UserTab.LoadLayout("Users")
```

Will **NOT** work:

```
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")

pageHolder.LoadLayout("MainLayout")

SKTabs1.GetTabContents(0).LoadLayout("Users")
```

As **SKTabs1.GetTabContents(0)** is a **method**, that although it returns a BANanoElement, it will **not work**. You will have to put **SKTabs1.GetTabContents(0)** in a separate BANanoElement variable first like in the above example**.**

7 BANanoPromise: Getting an answer in the future

What is a Promise?

This is probably the easiest way I found to explain what a Promise is:

> *"Imagine you are a kid. Your mom promises you that she'll get you a new phone next week."*
>
> *You don't know if you will get that phone until next week. Your mom can either really buy you a brand-new phone, or stand you up and withhold the phone if she is not happy .*
>
> *That is a promise. A promise has 3 states. They are:*
>
> ***Pending**: You don't know if you will get that phone*
> ***Fulfilled**: Mom is happy, she buys you a brand-new phone*
> ***Rejected**: Your mom is not happy, she withholds the phone*

Promises have a peculiar way of executing: Once you have given them a task, they start doing it and the code following its construction will execute immediately, not waiting for the Promise to end. This will become clear further in this chapter. **(*)**

I will first go through the classic way of using a promise. Then I will show you how to run them **async** with **BANano.Await** and end with how we can sometimes use the normal B4J **Wait For** too. The further you go into this chapter, the easier it will get. I promise!

7.1 Making a promise

7.1.1 The structure of a promise

```
' get all the files selected from the input #fu
Dim UploadedFiles() As String =
BANano.GetElement("#fu").GetField("files").Result

Dim Result as Map
Dim error as String

Log("Start") '(1)

Dim prom As BANanoPromise
Prom.CallSub(Me, "UploadAllFiles", Array(UploadedFiles))
Prom.Then(Result)
      Log("Success!") '(2)
       For each key as String in Result.Keys
           Log(key & "=" & Result.Get(key)) '(3)
       Next
Prom.Else(error)
      Log("Oops, something went wrong!") '(4)
Prom.Finally
     Log("Always runs, not matter if it was success or an error")'(5)
Prom.End

Log("This code will not wait until the Promise is fulfilled!") '(6)
```

see **(*) for the order of the logs:** So the output in the console will either be:

```
Start '(1)
This code will not wait until the Promise is fulfilled! '(6)
Success! '(2)
Key = value '(3)
Always runs, not matter if it was success or an error '(5)
```

Or if something went wrong:

```
Start '(1)
This code will not wait until the Promise is fulfilled! '(6)
Oeps, something went wrong! '(4)
Always runs, not matter if it was success or an error '(5)
```

7.1.2 Breaking the code down

We use the **.CallSub()** method to 'initialize' our BANanoPromise. Here will call some method that will upload the files to our server.

Such a method cannot simply Return a value because it can either be a success, or a failure. For this we use the BANano.ReturnThen() and the BANano.ReturnElse() methods. The ReturnThen will go back to the .Then branch of the Promise, The ReturnElse() will go to the .Else branch of the Promise.

If it was a success, we got back a map with all the URLs of the files we uploaded in the .**Then**() branch.

If it failed (network not connected for example), we got the error back in the **.Else**() branch.

In the **Finally** branch, we could do some clean-up for example..

7.1.3 The .Then() can also be a .ThenWait() and the .Else() be a .ElseWait().

What!?

Methods ending with the word **Wait** are consider special in BANano. The word Wait indicates to the transpiler that **the called method is a Promise**. This is partially so due to legacy, when real BANanoPromises did not exist yet in BANano.

So, if we have a Method called `MyMethod`**`Wait()`**, then it is actually transpiled in JavaScript to:

```
this.MyMethodWait = async function() {}
```

Instead, without Wait it would simply be `MyMethod()`:

```
this.MyMethod = function() {}
```

The same goes for `.ThenWait()` and `.ElseWait()` transpiling to the JavaScript code:

```
.then(async function(param) {} // the ThenWait

.catch(async function() {} // the ElseWait
```

When do we use them?

Simply said, whenever we use a Wait inside the Then or Else branch. Such a promise call can be another **...Wait**() method, but also e.g. the **Sleep** method.

For Example:

```
Dim prom as BANanoPromise
prom.CallSub(Me, "MyMethodWait", Array("Alain"))
prom.thenWait(result) ' uses ThenWait because we use Sleep in the branch
     Sleep(1000)
     Log(result)
prom.end

Sub MyMethodWait(Name as String) ' ends with Wait because we use Sleep
    Sleep(2000)
    BANano.ReturnThen("Hello " & Name)
End Sub
```

7.1.4 What is such a 'task'?

A promise task can take many forms. It mostly is something that can take some to perform, like getting you current GPS position, or uploading some files to the server,...

Many of the Build-in Methods in BANano will return a BANanoPromise. Examples are:

```
BANano.GetFileAsDataURL
BANano.GetFileAsJson
BANano.GetFileAsText
BANano.GetGeoPosition
```

In this case you can simply do:

```
Dim dataUrl As String
Dim dataUrlProm As BANanoPromise =
BANano.GetFileAsDataURL("./assets/B4X.jpg", Null)
dataUrlProm.Then(dataUrl)
     Log(dataUrl)
dataUrlProm.end

Log("Done")
```

Output:

```
Done
Some dataUrl
```

7.2 That was the long story. But BANano.Await! This can be simpler…

As the chapters above describe, the flow of your code can get quite complex: when is what executed?

Comes in the magic word: **BANano.Await()**!

7.2.1 BANAno.Await to the rescue

Instead of using the .Then() and .Else() flows, we can just simply wait for the answer before we continue in our code.

Let's look back at our last example:

```
Dim dataUrl As String
Dim dataUrlProm As BANanoPromise =
BANano.GetFileAsDataURL("./assets/B4X.jpg", Null)
dataUrlProm.Then(dataUrl)
      Log(dataUrl)
dataUrlProm.end

Log("Done")
```

Output:

```
Done
Some dataUrl
```

That is not what one would expect when writing B4J code. We would like the 'Done' to come AFTER logging the dataUrl.

Well, let's wait for the dataUrlProm to finish its task:

```
' the code will Wait here until the file is fetched
Dim dataUrl as String = BANAno.Await(
BANano.GetFileAsDataURL("./assets/B4X.jpg", Null))

Log(dataUrl)
Log("Done")
```

Now our output will be:

```
Some dataUrl
Done
```

But how about the .Else() branch, I hear you say?

Indeed**, that information is lost**. Both BANano.ReturnThen() and BANano.ReturnElse() will be put in dataUrl.

We can resolve this by wrapping everything in a B4J Try .. Catch and **throw** an error:

```
Dim Division as double
Dim Error as String

Try
      Division = BANAno.Await(SomeMethod(10,0))
      Log(result)
Catch(Error)
      Log(Error)
End Try

Sub SomeMethod(a as int, b as int) As String
      If b = 0 then
            BANano.Throw("You cannot divide by zero!)
      Else
            Return a / b
      End if
End Sub
```

7.2.2 Wait a minute: isn't that just B4Js Wait For?

You are right! Since BANano version 7.35+ you can just sometimes use B4Js **`Wait For`** statement do this too. 😊

Example:

```
Wait For (DoSum(10,20)) Complete(Result As Int)
Log(Result)

Sub DoSum(a As Int, b As Int) As ResumableSub
    Return a + b
End Sub
```

When using BANano methods, you may have to write some **small wrapper** around something so it returns as **ResumableSub**

Example:

```
Wait For (GetFile("./assets/banano.jpg",Null)) complete (fileUrl As
String)
Log(fileUrl)

Sub GetFile(url As String, options As BANanoFetchOptions) As ResumableSub
      Return BANano.GetFileAsDataURL(url,options)
End Sub
```

So in such cases, it is probably easier to just use the BANano.Await method.

7.3 Then why do these different systems exist?

They all have their reasons to be available. The BANanoPromise with the .Then(), .Else() and .Finally() give a lot more information back than the Wait For and sometimes you just need that information. The nesting of BANanoPromises is also very powerful. E.g., sometimes a Promise will give back an object and you want to use it in the next chained .Then(). An example of this will be demonstrated in BANanoFetch, which is a special BANanoPromise.

Or you use BANanoPromise simply because it translates better from a JavaScript snippet you found on the internet to B4J code.

8 BANanoFetch: Making requests to the server

BANanoFetch is a special BANanoPromise with some handy methods to handle the received data.

JavaScript can send network requests to the server and load new information whenever it's needed. For example, we can use a network request to:

- Submit an order,
- Load user information,
- Receive latest updates from the server,
- ...etc.

...And all of that without reloading the page!

There's an umbrella term "AJAX" (abbreviated Asynchronous JavaScript And XML) for network requests from JavaScript. We don't have to use XML though: the term comes from old times, that's why that word is there. You may have heard that term already.

The basic syntax of a BANanoFetch is this:

```
Dim fetch As BANanoFetch
fetch.Initialize(URL, [BANanoFetchOptions]]
fetch.Then(BANanoFetchResponse)

fetch.Else(error)

fetch.End
```

URL: the URL to access
BANanoFetchOptions: optional parameters: method, headers, etc...
BANanoFetchResponse: the response from the Fetch call

8.1 GET

Without **BANanoFetchOptions**, this is a simple **GET** request, downloading the contents of the URL.

The browser starts the request right away and returns a BANanoPromise that the calling code should use to get the result.

Getting a response is usually a two-stage process.

First, the BANanoPromise, returned by the BANanoFetch, resolves with an object of the built-in **BANanoFetchResponse** class as soon as the server responds with headers.

At this stage we can check HTTP status, to see whether it is successful or not, check headers, but don't have the body yet.

The promise rejects if the fetch was unable to make HTTP-request, e.g. network problems, or there's no such site. Abnormal HTTP-statuses, such as 404 or 500 do not cause an error.

We can see the HTTP-status in **BANanoFetchResponse** properties:

.Status – HTTP status code, e.g. 200.
.OK – boolean, true if the HTTP status code is 200-299.

For example:

```
Dim fetch as BANanoFetch
Dim response as BANanoFetchResponse

fetch.Initialize(url, Null) ' a simple GET
fetch.Then(response)
      If response.OK then  'http status is 200-299
            Dim Json as Map = BANano.Await(response.Json)
            …
      Else
            Log("HTTP-Error: " & response.Status)
      End If
fetch.End
```

Now, in stage 2 we do something with the response we've received.

8.2 Handling the BANanoFetchResponse

BANanoFetchResponse provides multiple BANanoPromise-based methods to access the body in various formats:

- response**.Text** – read the response and return as text,
- response**.Json** – parse the response as JSON,
- response**.FormData** – return the response as FormData object,
- response**.Blob** – return the response as Blob (binary data with type),
- response**.arrayBuffer()** – return the response as ArrayBuffer (low-level representation of binary data),
- additionally, response**.body** is a Readable Stream object, it allows you to read the body chunk-by-chunk,

In the above example, we used the **.Json** method to receive the Json body which in this case we could simply put into a Map object and then work with it.

```
Json = BANano.Await(response.json)
Log(Json.Get("name"))
```

We can choose only one body-reading method.

If we've already got the response with `response.text`, then `response.json` won't work, as the body content has already been processed.

```
text = BANano.Await(response.Text) ' response body consumed
parsed = BANano.await(response.Json) ' fails (already consumed)
```

The **response headers** are available in a Map-like headers object in **response.headers**.

8.3 POST/PUT/DELETE/… (using BANanoFetchOptions)

We set the method in **BANanoFetchOptions.Method,** the body in **BANanoFetchOptions.Body.** To set a request header in BANanoFetch, we can use the **BANanoFetchOptions.Headers** option.

```
Dim fetch As BANanoFetch
Dim fetchOptions As BANanoFetchOptions
Dim fetchResponse As BANanoFetchResponse

Dim data As Map
Dim Error as String

fetchOptions.Initialize
fetchOptions.Method = "POST"
fetchOptions.Body = $"{"guid": ${GUID}}"$
fetchOptions.Headers = CreateMap("Content-type": "application/json;
charset=UTF-8", "api_key": APIKey)

fetch.Initialize(APIUrl & "/v1/activatepwa", fetchOptions)
fetch.Then(fetchResponse)
      Log(fetchResponse)
      fetch.Return(fetchResponse.Json) ' resolve it to the next .ThenWait
fetch.ThenWait(data)
      Log(data) 'ignore
      Sleep(1000)
      If data.get("status") = "OK" Then
            …
      End If
fetch.ElseWait(error)
      Log(error)
fetch.End
```

There's a list of **forbidden http headers** that we can't set:

- Accept-Charset, Accept-Encoding
- Access-Control-Request-Headers
- Access-Control-Request-Method
- Connection
- Content-Length
- Cookie, Cookie2
- Date
- DNT
- Expect
- Host
- Keep-Alive
- Origin
- Referer
- TE
- Trailer
- Transfer-Encoding

- `Upgrade`
- `Via`
- `Proxy-*`
- `Sec-*`

These headers ensure proper and safe HTTP, so they are controlled exclusively by the browser.

8.4 Shortcut methods

BANano has a couple of shortcut methods that can help you quickly to do some communication e.g. with your jServer.

```
BANano.GetFileAsArrayBuffer
BANano.GetFileAsDataURL
BANano.GetFileAsJson
BANano.GetFileAsText
```

These methods can be used as simple as:

```
Dim DataURL as String
DataURL = BANano.Await(BANano.GetFileAsDataURL("./assets/B4X.jpg", Null)
```

9 The BANano Object: One Object to rule them all!

Now that we understand de essential BANano objects like **BANanoObject to access JavaScript** and **BANanoElement to access the browser DOM HTML** it's time to talk about that master of this all: the BANano Object itself.

The BANano Object is a set of methods to assist in writing Web code in B4J. It also contains the Transpiler. Depending on where it used, it has another function.

9.1 Using BANano in AppStart

As said, this is the only method in your BANano Web App that is not transpiled and that will actually run in the B4J IDE as normal B4J code.

So this is the place where we give directions to the Transpiler on how to build our Web Project.

A **typical definition for a PWA** may look something like this:

```
Sub AppStart (Form1 As Form, Args() As String)
	' With this little snippet, the new B4J 9.30 logs with jump are
activated
	#if Debug
	' MUST be literally this line if you want to use the B4J Logs jump
to code feature!
		Log("BANanoLOGS")
	#End if

	' some general settings like the name of your PWA
	BANano.Initialize("BANano", "BANanoSkeleton",DateTime.Now)
	BANano.Header.Title="BANano Skeleton"
	' DateTime.Now is to make sure our app is reloaded on ReBuild
	BANano.JAVASCRIPT_NAME = "app" & DateTime.Now & ".js"
	' a PWA must have a service worker.  Will be built automatically
caching everything used in your Web App
	BANano.SERVICEWORKER_NAME = "service-worker.js"

	' some directives for the Transpiler
	BANano.TranspilerOptions.MergeAllCSSFiles = True
	BANano.TranspilerOptions.MergeAllJavascriptFiles = True
	BANano.TranspilerOptions.RemoveDeadCode = True
	BANano.TranspilerOptions.ShowWarningDeadCode = True
	BANano.TranspilerOptions.EnableLiveCodeSwapping = True

	' this line makes sure our Web App becomes a PWA
	#if Release
		BANano.TranspilerOptions.UseServiceWorkerWithUpdateMessage(Tru
	e, "#26AE60", "Update available", "Click here to update the app to
	the latest version")
	#end if

	' optional: if your WebApp is not in the root
	' BANano.TranspilerOptions.SetPWAStartUrl("myPWA/index.html")
	BANano.Header.BackgroundColor = "#1e1e1e"
```

```
	' additional JavaScript and CSS files we want to include
	' BANano.Header.AddJavascriptFile("jsstore.min.js")

	' settings needed for the PWA app icons, splash screens, etc…
	BANano.Header.AddMSTileIcon("mstile-150x150.png", "150x150")
	BANano.Header.MSTileColor = "#ffc40d"

	BANano.Header.AddManifestIcon("android-chrome-192x192.png",
"192x192")
	BANano.Header.AddManifestIcon("android-chrome-512x512.png",
"512x512")
	BANano.Header.SetAndroidMaskIcon("maskable_icon.png", "731x731")
	BANano.Header.MaskIconColor = "#1e1e1e"

	BANano.Header.AddAppleTouchIcon("apple-touch-icon.png", "")
	BANano.Header.SetAppleMaskIcon("safari.png")
	BANano.Header.AddAppleTouchStartupImage("iphone5_splash.png",
"320px", "568px", "2")
	BANano.Header.AddAppleTouchStartupImage("iphone6_splash.png",
"375px", "667px", "2")
	BANano.Header.AddAppleTouchStartupImage("iphoneplus_splash.png",
"621px", "1104px", "3")
	BANano.Header.AddAppleTouchStartupImage("iphonex_splash.png",
"375px", "812px", "3")
	BANano.Header.AddAppleTouchStartupImage("iphonexr_splash.png",
"414px", "896px", "2")
	BANano.Header.AddAppleTouchStartupImage("iphonexsmax_splash.png",
"414px", "896px", "3")
	BANano.Header.AddAppleTouchStartupImage("ipad_splash.png", "768px",
"1024px", "2")
	BANano.Header.AddAppleTouchStartupImage("ipadpro1_splash.png",
"834px", "1112px", "2")
	BANano.Header.AddAppleTouchStartupImage("ipadpro2_splash.png",
"834px", "1194px", "2")
	BANano.Header.AddAppleTouchStartupImage("ipadpro3_splash.png",
"1024px", "1366px", "2")

	BANano.Header.AddFavicon("favicon-16x16.png", "16x16")
	BANano.Header.AddFavicon("favicon-32x32.png", "32x32")

	' write the theme
	SKTools.WriteTheme

	' start the actual build
	BANano.Build(File.DirApp)

	' stop running. We do not need the .jar file running anymore
	' in release mode
	#if Release
		ExitApplication
	#End if
End Sub
```

9.1.1 TranspilerOptions

Transpiler Options are directives you can give the BANano Transpiler on how to build your final Web App.

You can find the full list of options in the **Quick Reference**, but here are some commonly used ones:

DoNotDeleteFileOnCompilation (fullPath As String)

Prevents the Transpiler from deleting this file. Useful e.g. for assets that are not in the /Files folder.

DoNotDeleteFolderOnCompilation (fullPath As String)

Prevents the Transpiler from deleting this folder. Useful e.g. for assets that are not in the /Files folder.

ExcludePWACachingUrlContaining (str As String)

URL containing the given string will not be cached by the PWA Service Worker. Case sensitive.

If for example you dynamically load some json files from the https://mydomain.com/pwalists path, then you can exclude them from being cached with:

```
BANano.TranspilerOptions.ExcludePWACachingUrlContaining("pwalists")
```

Of course be careful that is unique enough, so it does not interfere with files that need to be cached!

IgnoreB4JLibrary (libName As String)

A B4J library the BANano Transpiler should ignore. (the library itself, not the use of it!)

By default, the following are ignored:

```
BANano
BANanoServer
jCore
jFx
json
jMQTT
jServer
JavaObject
ABJJWT
```

RedirectOutput (dir As String, fileName As String)

Redirects the logs to a file. Must be set in AppStart

SetPWAStartUrl (StartURL As String)

Sets the Start URL in the manifest.json file for a PWA. Default is HTML_NAME

e.g. `BANano.TranspilerOptions.SetPWAStartUrl("PWA/index.html")`

UseServiceWorkerWithUpdateMessage (bool As Boolean, UpdateColor As String, UpdateTitle As String, UpdateMessage As String)

Use a service worker where an update toast is showed if an update is available.
The user can then click the toast to do the update.

This is a cool build-in system that will update the users PWA if a new version is uploaded to the server.

EnableLiveCodeSwapping As Boolean

Enable Live Code Swapping and watch live changes made in the B4J source code.
On Save, the changed B4J code is Transpiled again and reloaded by the browser.

Default = true

MergeAllCSSFiles As Boolean

Must be set before Build(). Only used when in Release mode.

MergeAllJavascriptFiles As Boolean

Must be set before Build(). Only used when in Release mode.

RemoveDeadCode As Boolean

Build-in Tree Shaking. Only works in Build

The transpiler does not **GENERATE** dead code (never used). It does **NOT remove the B4J code**!

Use ShowWarningDeadCode beforehand to check if the transpiler is correct.

Methods with **a _ in their name** are always considered to be needed.

It is advised to set this to TRUE! It can make your final app al lot smaller!

You can use `'ignoredeadcode` after a method name (like the `'ignore` in B4J) to tell the transpiler not remove a certain method.

ShowWarningDeadCode As Boolean

Only works in Build

Shows a warning in the log if the transpiler suspects some code is dead (never used).
This is handy, especially in the final stage of development to remove code that is never used.

Methods with **a _ in their name** are always considered to be needed.

9.1.2 BANanoHeader

Use the **BANano.Header** to set common html meta data and add css/javascript files.

Can only be used in AppStart()!

In HTML, the <head> element is a container for metadata (data about data) and is placed between the <html> tag and the <body> tag.

Metadata is data about the HTML document. Metadata is not displayed.

Metadata typically define the document title, character set, viewport, styles, scripts, and other meta information.

Examples of meta data:

```
BANano.Header.Title="Activity"
BANano.Header.Author = "Alain Bailleul"
BANano.Header.Description = "Activity OneTwo"
BANano.Header.Charset = "utf-8"
BANano.Header.Keywords = "HTML, CSS, JavaScript"
```

9.1.2.1 Loading external CSS and JavaScript files

Here we can also load additional Javascript and CSS files. We have a couple of methods we can use. Mostly you will only need these two:

BANano.Header.**AddJavascriptFile** (AssetFileNameOrURL As String)

Load an extra javascript file. If it is an asset file it will be copied to the scripts folder.
For locale files (not URLs), you can use the * wildcard

Examples:

```
' local asset
BANano.Header.AddJavascriptFile("BANanoSkeleton.datepicker.min.js")
' from an URL
BANano.Header.AddJavascriptFile("https://unpkg.com/leaflet@1.3.4/dist/leaflet.js
")
```

BANano.Header.**AddCSSFile** (AssetFileNameOrURL As String)

Load an extra css file. If it is an asset file it will be copied to the styles folder.
For locale files (not URLs), you can use the * wildcard

Examples:

```
' local asset
BANano.Header.AddCSSFile("BANanoSkeleton.datepicker.min.css")
' from an URL
BANano.Header.AddCSSFile("https://unpkg.com/leaflet@1.3.4/dist/leaflet.css")
```

9.1.2.2 Loading assets… Later

Sometimes these files need to be loaded differently. For example, we have some Javascript or CSS file that aren't needed when loading the Web App as it is only used further into your program. We will only load them when needed with the BANano.**AssetsLoad**... methods.

Note: You will not often need the below methods. They are only used in specific situations, e.g. you do only want to load certain assets if the user requests them. The AssetsLoadWait method can be handy for loading a bundle of images later, but are not immediately needed.

For this, you can use the ...**ForLater** versions of the above methods:

BANano.Header.**AddJavascriptFileForLater** (AssetFileNameOrURL As String)

BANano.Header.**AddCSSFileForLater** (AssetFileNameOrURL As String)

Example:

```
' in Sub AppStart()
BANano.Header.AddCSSFileForLater("mini-nord.min.css")
...

' in Sub BANano_Ready()
Dim pathsNotFound() as String
If BANano.AssetsIsDefined("Loader") = False then
    pathsNotFound = BANano.AssetsLoadWait("Loader", Array("./assets/1.jpg",
"./styles/mini-nord.min.css"))
    If BANano.IsNull(pathsNotFound) = False Then
       Log("Doh! Loader has not been loaded completely!")
       For Each path As String In pathsNotFound
          Log(path)
       Next
    Else
       Log("Loader has been loaded!")
    End If
end if
```

Here we have defined an **Asset bundle** "Loader". It contains an image and a css file. CSS and JavaScript files need to be added in appStart with the ...**ForLater** methods. Other assets, like images, do not have to be defined in the Header.

When we need them, we check if we haven't already loaded the bundle with the `AssetsIsDefined` method. If not, then we load them with the `AssetsLoadWait` method. This method will return an Array of Strings containing the paths of the assets it could not load, of Null if they are all loaded.

9.1.2.3 Loading modern ES6 modules

Another special Load system is for **ES6 modules**.

JavaScript programs started off pretty small — most of its usage in the early days was to do isolated scripting tasks, providing a bit of interactivity to your web pages where needed, so large scripts were generally not needed. Fast forward a few years and we now have complete applications being run in browsers with a lot of JavaScript, as well as JavaScript being used in other contexts (Node.js, for example).

It has therefore made sense in recent years to start thinking about providing mechanisms for splitting JavaScript programs up into separate modules that can be imported when needed. Node.js has had this ability for a long time, and there are a number of JavaScript libraries and frameworks that enable module usage (for example, other CommonJS and AMD-based module systems like RequireJS.

The good news is that modern browsers have started to support module functionality natively. This can only be a good thing — browsers can optimize loading of modules, making it more efficient than having to use a library and do all of that extra client-side processing and extra round trips.

To define and load such modules, BANano provides you with some easy to use methods to **define** such modules. They are similar to the ones described above, but have **ES6** in their name:

BANano.Header.**AddJavascriptES6File** (AssetFileName As String)

Load an extra ES6 javascript file. **It must be an asset file and cannot be an URL**.

You can use the * wildcard

BANano.Header.**AddJavascriptES6FileForLater** (AssetFileName As String)

Load an extra ES6 javascript file. **It must be an asset file and cannot be an URL.**

This asset will not be loaded at loading the html file, but you will have to do it 'Later' using the BANano.AssetsLoad... methods

You can use the * wildcard

To actually use them, you will need to **import** those files with:

BANano.**Import** (moduleName As String) As BANanoPromise

Loads the complete file. Used to import bindings which are exported by another module.

Example:

```
Dim res as BANanoObject
Dim prom as BANanoPromise = BANano.Import("myES6Module")
prom.Then(res)
      prom.RunMethod("ES6Method", Array(1,2))
prom.End
```

BANano.**ImportWait** (moduleName As String) As BANanoObject

Same as above, but the promise is already resolved.

BANano.**ImportRaw** (importStatement As String)

Literally takes over the importStatement. This can be used to only load certain methods from a module.

Example:

```
BANano.ImportRaw("import { export1 , export2 as alias2} from 'module-name'")
```

Will be Transpiled in JavaScript to:

```
import { export1 , export2 as alias2} from 'module-name'
```

See for more info on raw imports:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

9.1.2.4 Loading JavaScript files in the Service Worker of the PWA

In some cases you need some extra JavaScript files that will be used in the PWA's service worker **only**.

IMPORTANT: such a javascript file can NOT use window or document or any other reference to the DOM as a Service Worker cannot access this!

These have to be defined specifically with the **SW** methods:

BANano.Header.**AddJavascriptFileSW** (AssetFileNameOrURL As String)

Does the same a AddJavascriptFile() but will write it also in the ImportScripts() method in the Service Worker file.

If it is a javascript file used in a BANanoLibrary, it **MUST** be added in the final app explicitly! It cannot be done in the library.

These javascript files will NOT be merged!

BANano.Header.**AddJavascriptFileForLaterSW** (AssetFileNameOrURL As String)

Does the same a AddJavascriptFileForLater() but will write it also in the ImportScripts() method in the Service Worker file.

If it is a javascript file used in a BANanoLibrary, it **MUST** be added in the final app explicitly! It cannot be done in the library.

These javascript files will NOT be merged!

The JavaScript `importScripts` method synchronously imports one or more scripts into the worker's scope.

9.1.2.5 PWA Specific Assets

It is also the place where you define PWA info, like the manifest and images that need to be used.

Example of PWA specific meta data:

```
BANano.Header.BackgroundColor = "#ff8800"

BANano.Header.AddMSTileIcon("mstile-150x150.png", "150x150")
BANano.Header.MSTileColor = "#ff8800"

BANano.Header.AddManifestIcon("android-chrome-192x192.png", "192x192")
BANano.Header.AddManifestIcon("android-chrome-512x512.png", "512x512")
BANano.Header.SetAndroidMaskIcon("maskable_icon3.png", "731x731")
BANano.Header.MaskIconColor = "#ff8800"

BANano.Header.AddAppleTouchIcon("apple-touch-icon.png", "")
BANano.Header.SetAppleMaskIcon("safari.svg")
BANano.Header.AddAppleTouchStartupImage("iphone5_splash.png", "320px", "568px",
"2")
BANano.Header.AddAppleTouchStartupImage("iphone6_splash.png", "375px", "667px",
"2")
BANano.Header.AddAppleTouchStartupImage("iphoneplus_splash.png", "621px",
"1104px", "3")
BANano.Header.AddAppleTouchStartupImage("iphonex_splash.png", "375px", "812px",
"3")
BANano.Header.AddAppleTouchStartupImage("iphonexr_splash.png", "414px", "896px",
"2")
BANano.Header.AddAppleTouchStartupImage("iphonexsmax_splash.png", "414px",
"896px", "3")
BANano.Header.AddAppleTouchStartupImage("ipad_splash.png", "768px", "1024px",
"2")
BANano.Header.AddAppleTouchStartupImage("ipadpro1_splash.png", "834px",
"1112px", "2")
BANano.Header.AddAppleTouchStartupImage("ipadpro2_splash.png", "834px",
"1194px", "2")
BANano.Header.AddAppleTouchStartupImage("ipadpro3_splash.png", "1024px",
"1366px", "2")

BANano.Header.AddFavicon("favicon-16x16.png", "16x16")
BANano.Header.AddFavicon("favicon-32x32.png", "32x32")
```

Some websites to help you create these assets:

https://realfavicongenerator.net

https://favicon.io/favicon-converter/

https://appsco.pe/developer/splash-screens

9.1.3 Transpiling and Building

BANano can build (Transpile) for several different purposes: it can generate the final PWA, make a BANanoLibrary or be part of a BANano Websockets project. It can even be used to make BANanoLibraries for my other library ABMaterial.

These commands can only be done in the AppStart method!

9.1.3.1 Building a PWA

This can be a stand-alone PWA, or a PWA using BANanoFetch calls to a BANanoServer REST API project.

EVERYTHING, except what is in the AppStart() method will be transpiled to JavaScript, including the BANanoLibraries you used in the project

To do this, we use the **BANano.Build** method. The parameter is the full path where it has to be Transpiled to.

```
' start the actual build
BANano.Build(File.DirApp)
```

In the log, you will see BANano at work, and will give feedback, like mistakes in the code or where optimizations can be done. **Check this log carefully**!

This is an example of a typical one in Debug Mode:

```
Waiting for debugger to connect...
Program started.
BANanoLOGS
Reading B4J INI in C:\Users\pi\AppData\Roaming\Anywhere Software\B4J to find Additional Libraries folder...
Found Additional Libraries folder: K:\B4J\AddLibraries
Starting to transpile...
Building K:\SourceCode\testPWA\Objects\BANanoSkeleton\scripts\app1645613014830.js
[WARNING]: RemoveDeadCode is disabled if EnableLiveCodeSwapping = true
Merging CSS files ignored.  Only applicable for Build in Release mode.
Merging Javascript files ignored.  Only applicable for Build in Release mode.
Loading layout mainlayout...
Loading layout menulayout...
Loading layout welcomemodallayout...
Loading layout welcomepagelayout...
Processing b4xlib: bananoskeleton
Jump Logs activated: 7
The method Await will not work in old browsers!
The method Await will not work in old browsers!
The method Await will not work in old browsers!
The method Await will not work in old browsers!
The method GetGeoPosition will not work in old browsers!
The method OpenWait will not work in old browsers!
The method ExecuteWait will not work in old browsers!
The method OpenWait will not work in old browsers!
```

```
The method ExecuteWait will not work in old browsers!
The method Await will not work in old browsers!
The method Await will not work in old browsers!
The method ExecuteWait will not work in old browsers!
Adding Layout mainlayout used by testpwa
Adding Layout welcomemodallayout used by testpwa
Adding Layout menulayout used by testpwa
BANanoMediaQueries will not work in old browsers!
Adding Mediaquerycode: bigger992px
BANanoMediaQueries will not work in old browsers!
Adding Mediaquerycode: smaller992px
Adding Layout welcomepagelayout used by testpwa
The method Await will not work in old browsers!
The method ExecuteWait will not work in old browsers!
Adding Layout welcomepagelayout used by testpwa
---------- OPTIMISATION METHODS ----------
OPTIMISATION: The METHOD reset in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD stopwait in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD turnontorch in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD turnofftorch in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD isscanning in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD supportstorch in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD addtoparent in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD remove in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD trigger in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD setclasses in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD getclasses in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD setstyle in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD getstyle in (MODULE: SKBarcodeScanner) appears to be unused
OPTIMISATION: The METHOD getelement in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: The METHOD getid in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: The METHOD addtoparent in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: The METHOD remove in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: The METHOD trigger in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: The METHOD setclasses in (MODULE: SKColorPicker) appears to be unused
OPTIMISATION: 583 more methods appear to be unused. See OPTIMISATIONS.txt
---------- OPTIMISATION CLASSES ----------
OPTIMISATION: The CLASS: SKBarcodeScanner appears to be unused
OPTIMISATION: The CLASS: SKColorPicker appears to be unused
OPTIMISATION: The CLASS: SKColumn appears to be unused
OPTIMISATION: The CLASS: SKCombo appears to be unused
OPTIMISATION: The CLASS: SKDatePicker appears to be unused
OPTIMISATION: The CLASS: SKDivider appears to be unused
OPTIMISATION: The CLASS: SKDropButton appears to be unused
OPTIMISATION: The CLASS: SKEditor appears to be unused
OPTIMISATION: The CLASS: SKFloatingButton appears to be unused
OPTIMISATION: The CLASS: SKBreadcrumbs appears to be unused
OPTIMISATION: The CLASS: SKList appears to be unused
OPTIMISATION: The CLASS: SKMobileNav appears to be unused
OPTIMISATION: The CLASS: SKNavigationBar appears to be unused
OPTIMISATION: The CLASS: SKPagination appears to be unused
OPTIMISATION: The CLASS: SKQRCode appears to be unused
OPTIMISATION: The CLASS: SKRadio appears to be unused
OPTIMISATION: The CLASS: SKRange appears to be unused
OPTIMISATION: The CLASS: SKSignaturePad appears to be unused
```

```
OPTIMISATION: The CLASS: SKSwitch appears to be unused
OPTIMISATION: 11 more classes appear to be unused. See OPTIMISATIONS.txt
The method startwait will not work in old browsers!
The method stopwait will not work in old browsers!
The method startwait will not work in old browsers!
The method stopwait will not work in old browsers!
The method getlocation will not work in old browsers!
The method insertwait will not work in old browsers!
The method welcomepagebutton_click will not work in old browsers!
Copying CSS files to WebApp assets...
Copying Javascript files to WebApp assets...
Building K:\SourceCode\testPWA\Objects\BANanoSkeleton\index.html
Done! Live Code Swapping is active...
```

9.1.3.2 Building a BANanoLibrary

You can easily split up your project by using BANanoLibraries. A BANanoLibrary is just a simple .b4xlib file that you can include in your projects.

EVERYTHING, except what is in the Main module will added to the .b4xlib.

This is especially handy to test some things in your library. So you can use the BANano_Ready() method to run some tests in your library and you can leave it in there, as this whole module will be ignored when making the final library.

To make a BANanoLibrary, simply call:

```
' start the build
#if release
	BANano.BuildAsB4Xlib("7.35")
#else
	BANano.Build(File.DirApp)
#end if
```

We use the normal Build() to test our library as this command will run the BANano_Ready method.

The log will look somewhat different to a PWA in Release Mode:

```
Reading B4J INI in C:\Users\pi\AppData\Roaming\Anywhere Software\B4J to find Additional Libraries folder...
Found Additional Libraries folder: K:\B4J\AddLibraries
Starting to transpile...
Building K:\B4J\AddLibraries\BANanoSkeleton.b4xlib
[WARNING]: [SKBarcodeScanner,startwait: 151] The method Await will not work in old browsers!
[WARNING]: [SKBarcodeScanner,startwait: 152] The method Await will not work in old browsers!
[WARNING]: [SKTakePicture,startwait: 216] The method Await will not work in old browsers!
[WARNING]: [SKTools,getlocation: 94] The method Await will not work in old browsers!
[WARNING]: [SKTools,getlocation: 94] The method GetGeoPosition will not work in old browsers!
Building K:\B4J\AddLibraries\BANanoSkeleton\BANanoSkeleton.dependsOn
----------------------------------------------------------
K:\B4J\AddLibraries/BANanoSkeleton.b4xlib created!
----------------------------------------------------------
```

9.1.3.3 Building a BANanoServer Websocket project

See the chapters on BANanoServer on how they are made. It is enough to know here that a BANanoServer WebSocket project consists of SERVER, BROWSER and SHARED classes.

EVERY class with its name starting with BROWSER or SHARED will be transpiled to JavaScript, including all BANanoLibraries used in the project.

To build such a project use:

```
' transpile all the BANano b4J code to javascript
Server.BANano.BuildForServer(Server.OutputFolder)
```

This will do two things: it will generate the PWA, but also the server .jar file. You can the upload this .jar file and its www folder to your VPN like a normal B4J jServer project.

For more info on how to deploy a jServer .jar file, see the B4J forum.

As a BANanoServer REST API project is just a normal jServer project, the same applies.

9.1.3.4 Building a BANanoLibrary for ABMaterial

This is the same as a normal BANanoLibrary, but the .b4xlib file has to build slightly different.

You use the BANano.BuildAsB4XLibForABM method instead.

```
BANano.BuildAsB4XlibForABM("D:\MyProject\MyABMProject\Objects\www","1.15")
```

The first parameter is the www folder of your ABMaterial project.

Will Build the transpiled files to your Additional Libraries folder as a B4XLib for ABMaterial (**prefix: ABMBanano**).

You do not need to compile your Library with the B4J IDE.

If ABMStaticFilesFolder in ABMaterial (the /www folder) is set, then the assets will be automatically unzipped in this folder.

9.1.3.5 Tree Shaking (removing dead code)

BANano has the ability to remove dead (unused) code from your **final PWA**. It goes a lot further than other packing tools, as it can even **remove single methods** within a module/class that is not used in the final PWA. Most packing tools can only go to the level of removing a full class.

This means as soon as you use one method from a class, those packagers will include the complete class. BANano will strip those not used methods from the class and will only include a small fraction of the total class in your PWA.

The transpiler does not **GENERATE** dead code (never used). It does **NOT remove the B4J code**!

Methods with **a _ in their name** are always considered to be needed.

This can be done by the following Transpiler Options switches:

```
BANano.TranspilerOptions.ShowWarningDeadCode = True
BANano.TranspilerOptions.RemoveDeadCode = True
```

The first one will only log what could be removed as it wasn't used in the final PWA. The second one will actually remove all methods and classes you did not use in your project.

9.2 Using BANano in the WebApp code

The BANano object contains a lot of methods that are typical for JavaScript and have no real equivalent in B4J.

We have already met such a method, the BANano.Await method. **The full list is in the quick reference.**

If you are searching for a typical JavaScript command, it is almost certain you will find its B4J equivalent in the BANano object.

9.2.1 BANano Extended Property Objects

These are property objects that **cannot be initialized**, but that provide additional methods and properties typical for a certain object.

BANanoConsole
BANanoWindow
BANanoHistory
BANanoLocation
BANanoNavigator
BANanoScreen

You can work with those by accessing their property on the BANano object. E.g.

```
BANano.Console.Info("myMessage")
Log(BANano.Window.InnerWidth)
Log(BANano.Screen.Height)

' this is NOT a Geo location, but the structure of the URL!
Log(BANano.Location.Host)
```

For a **Geo** Location and Position you can use the **BANanoGeoLocation** object:

Example:

```
Dim pos As BANanoGeoPosition
Dim error As Int

BANano.GeoLocation.GetCurrentPosition(BANano.CallBack(Me,
"HandleGotPosition", Array(pos)), BANano.CallBack(Me,
"HandleErrorPosition", Array(error)))
…

Sub HandleGotPostion(pos As BANanoGeoPosition)
     Log(pos.Latitude & "-" & pos.Longitude)
End Sub

Sub HandleErrorPosition(error As Int)
     Log(error)
End Sub
```

You can get the current position easier with the shortcut

```
Dim pos as BANanoGeoPosition =
BANano.Await(BANano.GetGeoPosition(CreateMap("enableHighAccuracy": true,
"timeout": 5000, "maximumAge": 0))

Log(pos.Latitude & "-" & pos.Longitude)
```

10 Saving data in the browser

We can use several ways to save data in the user's browser. How you use it largely depends on its purpose. It can be done with the classic Cookies, in LocalStorage or SessionStorage and with the build-in BANanoSQL that mimics a real Database with SQL Queries!

A special case is the CacheStorage.

10.1 Cookies

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

Cookies are saved in name-value pairs like: activeUser = Alain Bailleul

When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

In BANano you have a couple of methods to assist you in managing the cookies:

BANano.**SetCookie** (name As String, value As String, jsonOptions As String)

jsonOptions: expires, path, domain, secure

example: expires 7 days from now

```
BANano.SetCookie("mycookie", "myvalue", "{expires: 7, path: '', domain:
'mydomain.com', secure: 'true'}")
```

BANano.**GetCookie** (name As String) As String

Returns a the value of the cookie

BANano.**RemoveCookie** (name As String, jsonOptions As String)

Deletes a cookie.

IMPORTANT! When deleting a cookie and you're not relying on the default attributes, you must pass the exact same path and domain attributes that were used to set the cookie

```
BANano.RemoveCookie("mycookie", "{path: '', domain: 'mydomain.com'}")
```

10.2 LocalStorage and SessionStorage

The **localStorage** and **sessionStorage** objects, part of the web storage API, are two great tools for saving key/value pairs locally. Using localStorage and sessionStorage for storage is an alternative to using cookies and there are some advantages:

- The data is saved locally only and can't be read by the server, which eliminates the security issue that cookies present.
- It allows for much more data to be saved (10mb for most browsers).
- The syntax is straightforward.

It's also supported in all modern browsers, so you can use it today without an issue. Cookies are still useful, especially when it comes to authentication, but there are times when using localStorage or sessionStorage may be a better alternative.

localStorage and **sessionStorage** are almost identical and have the same API. The difference is that with sessionStorage, the data is persisted only until the window or tab is closed. With localStorage, the data is persisted until the user manually clears the browser cache or until your web app clears the data.

With this knowledge, you can now create, read, and update key/value pairs in localStorage.

BANano has two pairs of methods. The original ones (without a 2 at the end) are still in there and use the JavaScript library Vault. The ones with a 2 at the end are the native ones and it is preferable to use those. I will only explain the ones with a 2 at the end.

The syntax for both Local as SessionStorage are the same. Just replace the word **Local** by the word **Session**.

BANano.**SetLocalStorage2** (key As String, value As Object)

You can create entries for the localStorage object by using the SetLocalStorage2() method. The SetLocalStorage2() method takes two arguments, the key and corresponding value:

```
BANano.SetLocalStorage2("otwprojectnew", "Last Project")
```

BANano.**GetLocalStorage2** (key As String) As Object

To read entries, use the GetLocalStorage2() method. The GetLocalStorage2 () method takes one argument which must be the key. This function will return the corresponding value as a string:

```
Dim LastProject as String = BANano.GetLocalStorage2("otwprojectnew")
```

BANano.**RemoveLocalStorage2** (key As String)

You can delete an entry with the RemoveLocalStorage2() method.
The RemoveLocalStorage2() method takes one argument which will be a key of the localStorage object:

```
BANano.RemoveLocalStorage2("otwprojectnew")
```

BANano.**EmptyLocalStorage2** (key As String)

You can also clear all items in localStorage. This can be done with the EmptyLocalStorage2() method. Here's how to clear everything that's stored in localStorage:

```
BANano.EmptyLocalStorage2
```

LocalStorage can only store string values. If you want to store objects or arrays as values in localStorage, you can use BANano.toJson() method to convert them into strings and BANano.FromJson() to convert it back.

10.3 CacheStorage (BANano v7.35+)

This is a special kind of storage that cashes files (URLs). You can use it for files that are not by default cached if a Service Worker is used. If a Service Worker is used, BANano will cache all the files in the B4J /Files folder and all requests made to the server automatically.

This feature is only available if **HTTPS** is used! URL must be a valid http or https!

This is added to complete the storage possibilities, but the below methods are very rarely used in BANano Web App.

BANano.**SetCacheStorage2**(url as String)

Native to set URL with parameters into the cacheStorage RUNTIME.

```
BANano.SetCacheStorage2("https://mydomain.com/image.png?param=Alain")
```

BANano.**GetCacheStorage2**(url as String) As String

Native returns the full URL (with parameters) if the URL is in the cacheStorage RUNTIME
The URL will be searched without parameters.

```
BANano.GetCacheStorage2("https://mydomain.com/image.png")
```

will return: https://mydomain.com/image.png?param=Alain

BANano.**RemoveCacheStorage2**(url as String)

Native deletes a key from the cacheStorage RUNTIME.
The URL will be searched without parameters.

```
BANano.RemoveCacheStorage2("https://mydomain.com/image.png")
```

10.4 BANanoSQL

BANanoSQL is an easy-to-use wrap around the alaSql library. It allows using normal SQL queries (to a certain point) on the IndexedDb. **The library is not flawless**: e.g. the functionalities to update the database structure (e.g. adding a column) or setting indexes do not work on an IndexedDb. These are known problems to the developers of the alaSQL library and hopefully one day they will be resolved.

Despite these problems, I have found that it works very well in our PWAs and it is just something you have to take into account.

Also, keep your Queries as simple as possible.

Again, there are multiple ways to do things with the object. I will go over the methods I find most useful and easy to use.

Note: I will use both examples with the variables parameter and without. In a PWA in a local DB that is not as important as on a server. But on a server **ALWAYS** use variables to avoid SQL injections!

10.4.1 Creating the Database

Add an instance of the BANanoSQL object to your apps Globals:

```
Public SQL As BANanoSQL
```

You create/open a database with SQL.OpenWait. For the first parameter, use the name of your SQL variable (here SQL) and for the second one the name of your Database.

VERY IMPORTANT: the name of the database can NOT be a variable: it must be a literal String!

Next you run some normal CREATE queries. We use IF NOT EXISTS so if we open the database, the queries will be skipped if the tables already exist.

```
SQL.OpenWait("SQL", "PWAMatData1")

SQL.ExecuteWait("CREATE TABLE IF NOT EXISTS tProject (prjid INT, prjtype
INT, prjparent INT, prjcode STRING, prjdesc STRING, prjiden STRING,
prjunit STRING, prjpar1 STRING, prjpar2 STRING)", Null)

SQL.ExecuteWait("CREATE TABLE IF NOT EXISTS tItem (itid INT, ittype INT,
itparent INT, itcode STRING, itdesc STRING, itiden STRING, itunit STRING,
itpar1 STRING, itpar2 STRING)", Null)

SQL.ExecuteWait("CREATE TABLE IF NOT EXISTS tData (dtid INT
AUTOINCREMENT, dttype INT, dtstatus INT, dtdate STRING, dtdatetime
STRING, dtstart INT, dtstop INT, dtvalue REAL, dtex STRING, dtgrp STRING,
dtit STRING, dtexscan STRING, dtgrpscan STRING, dtitscan STRING, dtexid
INT, dtgrpid INT, dtitid INT, dtextra INT, dtoms STRING, dtunit STRING)",
Null)
```

10.4.2 INSERT new data

Very familiar looking if you are used to inserting data in e.g. MySQL in B4J:

```
Dim now As Long = DateTime.Now

Dim Variables As List
Variables.Initialize

Variables.Add(CurrentItemList.RegistrationType)
Variables.Add(2)
Variables.Add(JavaDateTimeToStr(now))
Variables.Add(JavaDateToStr(now))
Variables.Add(0)
Variables.Add(0)
Variables.Add(0)
Variables.Add(Login)
Variables.Add(prjItem.Oms)
Variables.Add(lstOms)
Variables.Add("")
Variables.Add(prjItem.Iden)
Variables.Add(lstIden)
Variables.Add(0)
Variables.Add(prjItem.id)
Variables.Add(lstID)
Variables.Add(0)
Variables.Add("")
Variables.Add(lstUnit)

SQL.ExecuteWait($"INSERT INTO tData (dttype, dtstatus, dtdatetime,
dtdate, dtstart, dtstop, dtvalue, dtex, dtgrp, dtit, dtexscan, dtgrpscan,
dtitscan, dtexid, dtgrpid, dtitid, dtextra, dtoms, dtunit) VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"$,Variables)
```

10.4.3 UPDATE existing data

Again, nothing special about it:

```
SQL.ExecuteWait($"UPDATE tData SET dtextra=${dtExtra} WHERE dtid=${id}"$,
Null)
```

10.4.4 DELETE data

Some SQL syntaxes would use DELETE * FROM, in alaSQL you have to remove the * in a DELETE.

```
SQL.ExecuteWait($"DELETE FROM tData WHERE dtid=${CurrentToDelete.ID}"$,
Null)
```

I have also noticed that the delete does not always work in alaSQL without a WHERE clause. If you do not have one, use something like WHERE 1=1

10.4.5 SELECT data

Retrieving some records from the database.

```
Dim results As List

Results = SQL.executeWait($"SELECT dtid, dtvalue, dtit, dtstatus,
dtextra, dtunit, dtoms, dttype FROM tData WHERE (${RegistrationTypes})
AND dtgrpid=${lstItem.ID} AND dtdate='${tmpDate}' ORDER BY dtid DESC"$,
Null)

For i = 0 To results.Size - 1
      Dim m As Map = results.Get(i)
      Dim reg As Double = m.Get("dtvalue")
      …
Next
```

10.4.6 Additional Remarks

Here are some things I did encounter myself when using BANanoSQL in my own PWAs.

1. **Every PWA needs to have its own domain**. When PWAs share a domain name, alaSQL seems to have trouble and shares instances of objects.

   So use e.g.

   ```
   app1.mydomain.com
   app2.mydomain.com
   ```

   Instead of:

   ```
   mydomain.com/app1
   mydomain.com/app2
   ```

2. On bigger record sets, it is most of the time faster to load them once and **keep the result in a list.** It is faster to go a couple of times through the list than it is to re-query the database.

3. A handy way to initially insert a whole bunch of data (= json where each item has **exactly** the same property names as the field names!) is this:

   Json:

   ```
   [{"itid": 1,"ittype": 1000, "itparent": 2, "itcode": "code1", "itdesc":
   "desc1", "itiden": "iden1", "itunit": "unit1", "itpar1":  "", "itpar2":
   ""},{"itid": 2,"ittype": 1001, "itparent": 3, "itcode": "code2", "itdesc":
   "desc2", "itiden": "iden2", "itunit": "unit2", "itpar1":  "", "itpar2":
   ""},…]
   ```

   Queries:

   ```
   SQL.ExecuteWait($"DELETE FROM tItem WHERE 1=1"$, Null)
   SQL.ExecuteWait($"SELECT * INTO [tItem] FROM ?"$, Array(data))
   ```

4. BANanoSQL is **not a real variable but kind of an atom object** (there is no 'new'). You do not have to pass it to a sub. Just like the BANano Object, you just define it in the Globals of the class and 'assign' your database to it with the OpenWait() command.

   `SQL.OpenWait("SQL", "MyDB")` just 'assigns' the MyDB database to the B4J SQL variable.

   So, you can have in your **Main**:

```
Sub Process_Globals
   Public SQL As BANanoSQL
   Public myDB As MyDBFuncs
   ...
End Sub

Sub BANano_Ready()
    ' Initialize your local browser database
    SQL.OpenWait("SQL", "MyDB")
    SQL.ExecuteWait("CREATE TABLE IF NOT EXISTS tTable (tblid INT, tblcode
STRING, tbldesc STRING)", Null)

    myDB.Initialize

    ' insert some record via our MyDB class
    BANano.Await(myDB.InsertWait(1, "A", "Alain"))
    BANano.Await(myDB.InsertWait(2, "J", "Jos"))

    Dim Results As List = SQL.ExecuteWait("SELECT tblcode, tbldesc FROM
tTable", Null)
    Log(Results)
    ...
End Sub
```

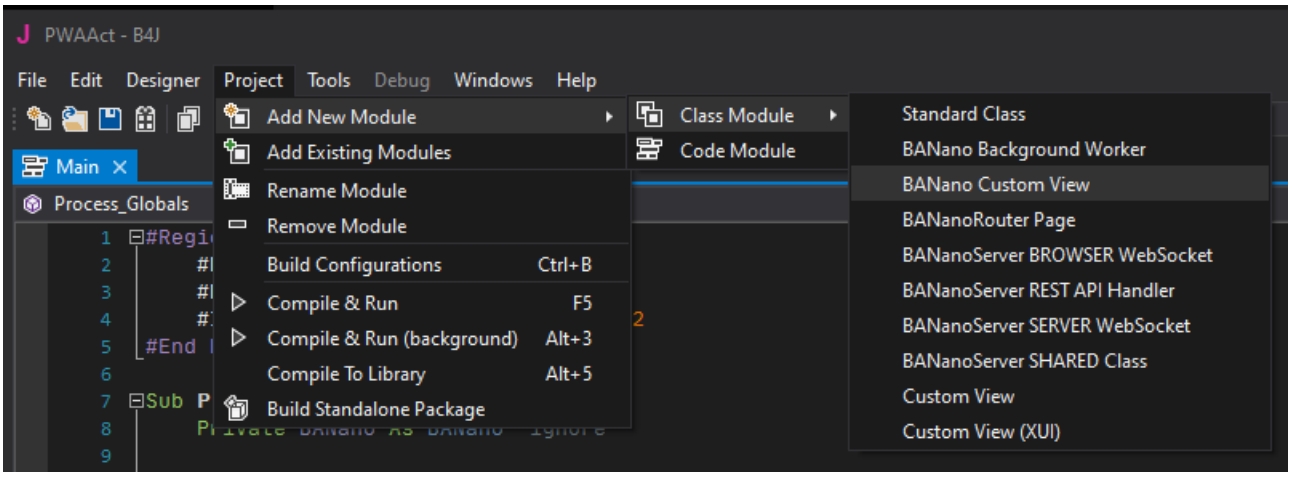
   And a class **MyDBFuncs**:

```
Sub Class_Globals
    Private BANano As BANano 'ignore
    Private SQL As BANanoSQL
End Sub

Public Sub Initialize

End Sub

public Sub InsertWait(id As Long, code As String, desc As String)
    Dim Vars As List
    Vars.Initialize
    Vars.Add(id)
    Vars.Add(code)
    Vars.Add(desc)

    ' just 're-assign' MyDB to the local SQL variable
    SQL.OpenWait("SQL", "MyDB")
    SQL.ExecuteWait("INSERT INTO tTable (tblid, tblcode, tbldesc) VALUES
(?, ?, ?)", Vars)
End Sub
```

11 Components for the Abstract Layout Designer

11.1 Creating a Component

The easiest way to add a BANano Custom View is by using the menu:



This will create the base code for a BANano Custom View, which is very similar to a normal B4J Custom View.

The main difference if the syntax of the DesignerCreateView method. Instead of a Panel of Pane, it is a **BANanoElement**.

```
Public Sub DesignerCreateView (Target As BANanoElement, Props As Map)
      mTarget = Target ' IMPORTANT
      …
End Sub
```

Some default events will also be generated (uncomment to activate which ones you use, or write new ones)

```
' Uncomment the events you want to show to the user and implement the
HandleEvents in DesignerCreateView
'#Event: Focus (event As BANanoEvent)
'#Event: Blur (event As BANanoEvent)
'#Event: Resize (event As BANanoEvent)
'#Event: Scroll (event As BANanoEvent)
'#Event: Keydown (event As BANanoEvent)
'#Event: KeyPress (event As BANanoEvent)
…
```

As well as some default much used properties:

```
' Properties that will be show in the ABStract Designer.  They will be
passed in the props map in DesignerCreateView (Case Sensitive!)
#DesignerProperty: Key: Classes, DisplayName: Classes, FieldType: String,
DefaultValue: , Description: Classes added to the HTML tag.
#DesignerProperty: Key: Style, DisplayName: Style, FieldType: String,
DefaultValue: , Description: Styles added to the HTML tag. Must be a json
String.
```

```
#DesignerProperty: Key: MarginLeft, DisplayName: Margin Left, FieldType:
String, DefaultValue: , Description: Margin Left
#DesignerProperty: Key: MarginRight, DisplayName: Margin Right,
FieldType: String, DefaultValue: , Description: Margin Right
#DesignerProperty: Key: MarginTop, DisplayName: Margin Top, FieldType:
String, DefaultValue: , Description: Margin Top
#DesignerProperty: Key: MarginBottom, DisplayName: Margin Bottom,
FieldType: String, DefaultValue: , Description: Margin Bottom
#DesignerProperty: Key: PaddingLeft, DisplayName: Padding Left,
FieldType: String, DefaultValue: , Description: Padding Left
#DesignerProperty: Key: PaddingRight, DisplayName: Padding Right,
FieldType: String, DefaultValue: , Description: Padding Right
#DesignerProperty: Key: PaddingTop, DisplayName: Padding Top, FieldType:
String, DefaultValue: , Description: Padding Top
#DesignerProperty: Key: PaddingBottom, DisplayName: Padding Bottom,
FieldType: String, DefaultValue: , Description: Padding Bottom
```

TIP: do not over-create properties! They carry quite some weight, and properties that are not used are still passed through the LoadLayout method. They can blow-up the size for your Web App or .b4xlib fast.

Think which ones are really useful and are almost always needed when designing a layout. Make methods for properties that are only used in specific cases. They will be removed in Release time if they are not used in the final code.

You can read these properties back in the DesignerCreateView **Props** parameter:

```
If Props <> Null Then
     mFlavor = Props.Get("Flavor")
     mText = Props.Get("Text")
     …
End If
```

Adding events to your component is done as in the BANanoEvent chapter. In this example, we do not have to do anything special so we can just pass it through to the WebApp:

```
' defining events is very simple. Note that it has to be run AFTER adding
it to the HTML DOM! eventName must be lowercase!
mElement.HandleEvents("click", mCallBack, mEventName & "_click")
```

And on top we uncomment the Click event:

```
#Event: Click (event As BANanoEvent)
```

Now we can use this event in our WebApp as (suppose we have an SKButton called myButton on our layout):

```
Sub myButton_Click(event As BANanoEvent)
     Dim myButton as SKButton = Sender
     …
End Sub
```

11.2 A note on extra Assets

Sometimes you will have a Component that requires extra assets (like JavaScript or CSS files).

You add these files in the **Files Tab** in the IDE first. **Do not forget to Sync!**

Then, in AppStart, you have to define them (example for the DatePicker in BANanoSkeleton):

```
BANano.Header.AddCSSFile("BANanoSkeleton.datepicker.min.css")
BANano.Header.AddJavascriptFile("BANanoSkeleton.datepicker.min.js")
```

Finally, we add in the Initialize() method of the Component that this component requires them:

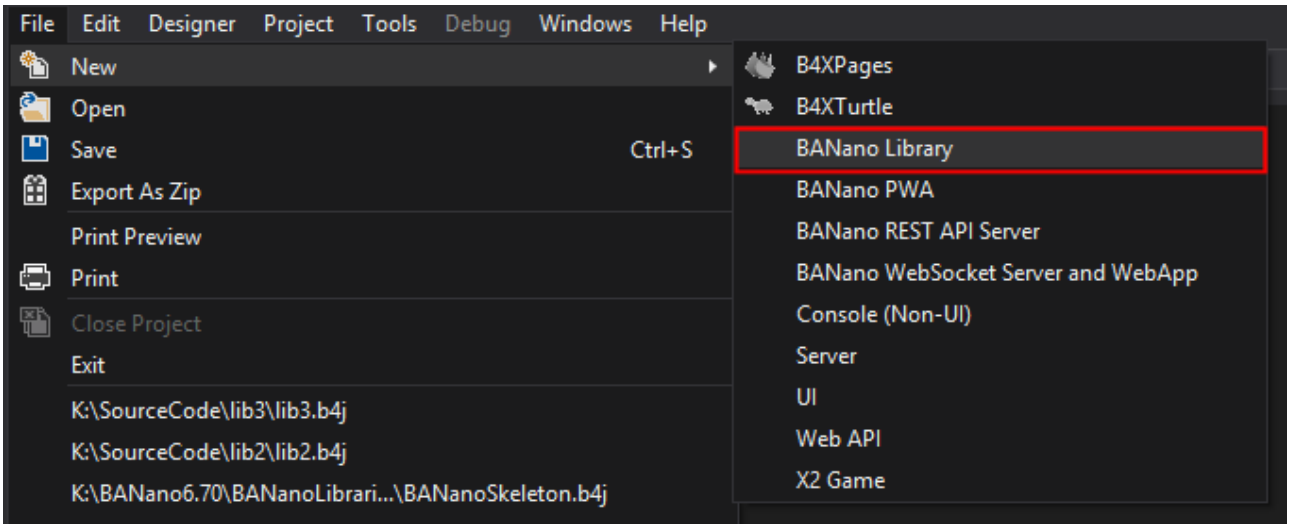
```
Public Sub Initialize (CallBack As Object, Name As String, EventName As
String)
      mName = Name.ToLowerCase
      mEventName = EventName.ToLowerCase
      mCallBack = CallBack
      S.Initialize(Me)

      BANano.DependsOnAsset("BANanoSkeleton.datepicker.min.css")
      BANano.DependsOnAsset("BANanoSkeleton.datepicker.min.js")
End Sub
```

Why these two last lines?

The BANano Transpiler can use these directives and **ONLY** add them to your final Web App **IF** you do use a SKDatePicker in your Web App! So if you do not use it, those Assets will not be loaded, hence making your final Web App a lot lighter.

12 BANano libraries

With the .b4xlib format of B4J, it is very easy to reate BANanoLibraries.
You can use the '**BANano Library**' Template to get started.



This will generate some basic code.

Main:

```
#Region Project Attributes
	#MainFormWidth: 600
	#MainFormHeight: 600
	#IgnoreWarnings: 16, 10, 14, 15
	#LibraryAuthor: Alain Bailleul (AlwaysBusy)
	#LibraryVersion: 1.01
#End Region

Sub Process_Globals
	Private BANano As BANano 'ignore
End Sub

Sub AppStart (Form1 As Form, Args() As String)
	' The name of your library. Strongly recommanded to let it begin
with BANano!
	BANano.Initialize("BANano", "BANanoMyLibName", DateTime.Now)

	' add any extra files you need for your library.  Files must be in
the /Files folder of the project
	'BANano.Header.AddCSSFile("extra.min.css")
	'BANano.Header.AddJavascriptFile("extra.min.js")

	' start the build
	#if release
		BANano.BuildAsB4Xlib("1.01") 'version
	#else
		BANano.Build(File.DirApp)
	#end if

	ExitApplication
End Sub
```

```
'Return true to allow the default exceptions handler to handle the
uncaught exception.
Sub Application_Error (Error As Exception, StackTrace As String) As
Boolean
     Return True
End Sub

' HERE STARTS YOUR APP
' you can use the Main (only) to do some tests on your library
Sub BANano_Ready()

End Sub
```

MyLibName (which is just a normal B4J class):

```
' Your BANano library
Sub Class_Globals
     Private BANano As BANano 'ignore
End Sub

'Initializes the object. You can add parameters to this method if needed.
Public Sub Initialize

End Sub
```

BANano.BuildAsB4Xlib will add EVERYTHING in the project, except what is in Main.

If you have extra assets that this library needs (like JavaScript files, CSS files or images), you have to add them in the projects /Files folder. **Do not forget to sync!**

It is strongly recommended to let the name of your library start with the prefix BANano. This to make it easier for everyone to recognize which library is for BANano, and which for a normal B4J project.

Next to logic code, or a wrapper for an existing JavaScript library, you can also add BANano Custom Views in a BANanoLibrary.

13 Introducing BANanoServer

B4J has great build-in Server capabilities with the **jServer** library based on **Jetty**! It has several advantages for a B4J programmer over for example PHP. More and more companies are moving away from PHP to the much more secure and faster Java implementation. And you can use your beloved B4X syntax with its extensive libraries!

When it comes to the PHP vs **Java performance** comparison, Java is the winner. Java is precompiled, and PHP needs time to comply with bytecode on each request. The optimization work in both Java and PHP has been done but Java seems to put more work into that.

Java is considered to be **more stable** than PHP. It requires a longer code which takes time. At the same time, a well-written longer code becomes a more stable application with fewer crashes. It has become the reason why banks and fintech brands pick Java without any further considerations.

Java is also considered to be a **more secure language**, compared to PHP. It has more built-in security features while PHP developers have to opt for other frameworks. However, in terms of security, Java works better for complex projects because it can block some features in low-level programming to protect the PC.

Moreover, a B4J Server can do **WebSockets** (Yeah!). With Websockets we can do bi-directional communication: not only can de Web App ask something to the Server and get a Response, as long as the Web App is connected, the server can push things to the Web App too!

13.1 What is BANanoServer?

Plain and simple: **BANanoServer = a pre-configured jServer!**

Because of that, this booklet will NOT go into everything a jServer can do! All this information can be found on the B4X website and forum. There are plenty of tutorials and examples to be found there.

This booklet will cover the BANano side of making a Request and Receiving a response back, only showing the B4J jServer things needed for the examples.

The BANanoServer library has pre-configured a lot of stuff for you, but as it is a just a B4J .b4xlib library, you can make any changes to it to match with your needs.

We use BANanoServer to make Requests from our BANano Web App to the server, and handle the Response back. This can be for example because we want to retrieve some data from a MySQL database. Most commonly, this response is some Json.

Most common use of the BANanoServer is as a REST API server. B4X has a brilliant way to do this using Handlers!

There are many ways to talk to a REST API, but the easiest and most modern way is by making a **BANanoFetch** call.

In a second example I also will go into WebSockets and we will see we can write the **SERVER** and **BROWSER** side all in **ONE B4J** app!

First we will create both types of BANano servers in B4J, then we go deeper into an example of them.

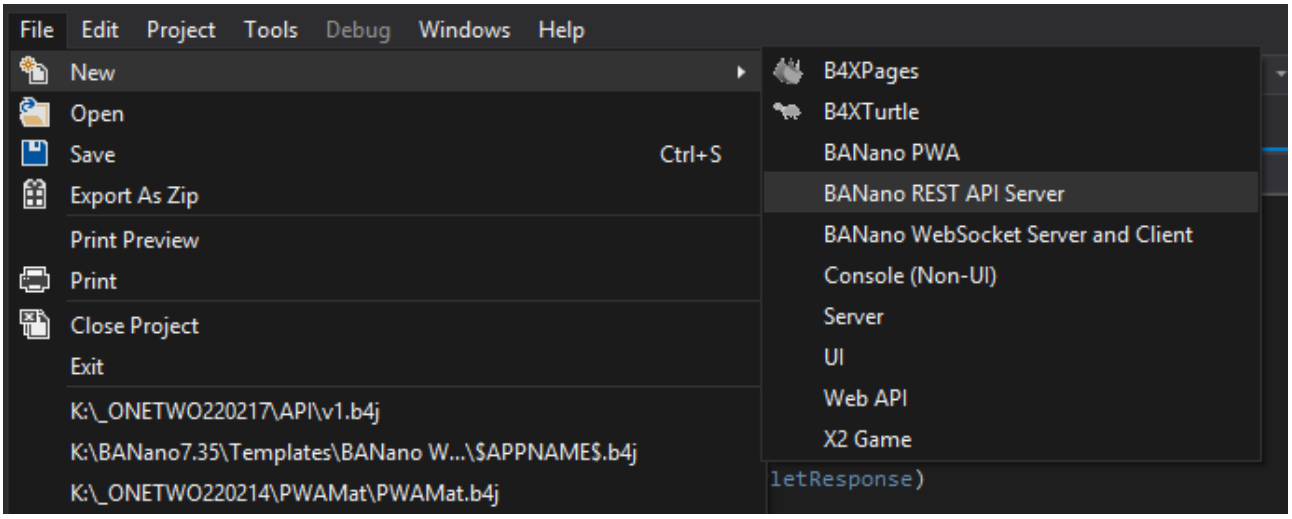
13.2 Creating a B4J App using the BANanoServer library

There are two ways to do this. One with REST API server and one with a WebSockets server.

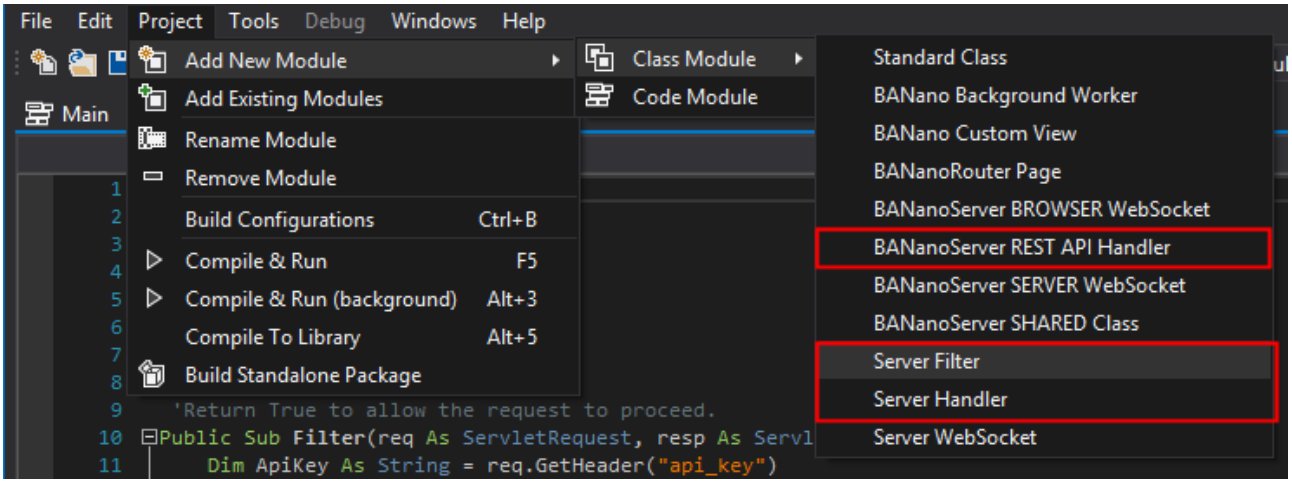
13.2.1 REST API BANanoServer

Such a project consists **of one REST API Java app** and (**at least) one BANano PWA app**. The PWA app is stand-alone and communicates with the server using a REST API and with BANanoFetch calls.

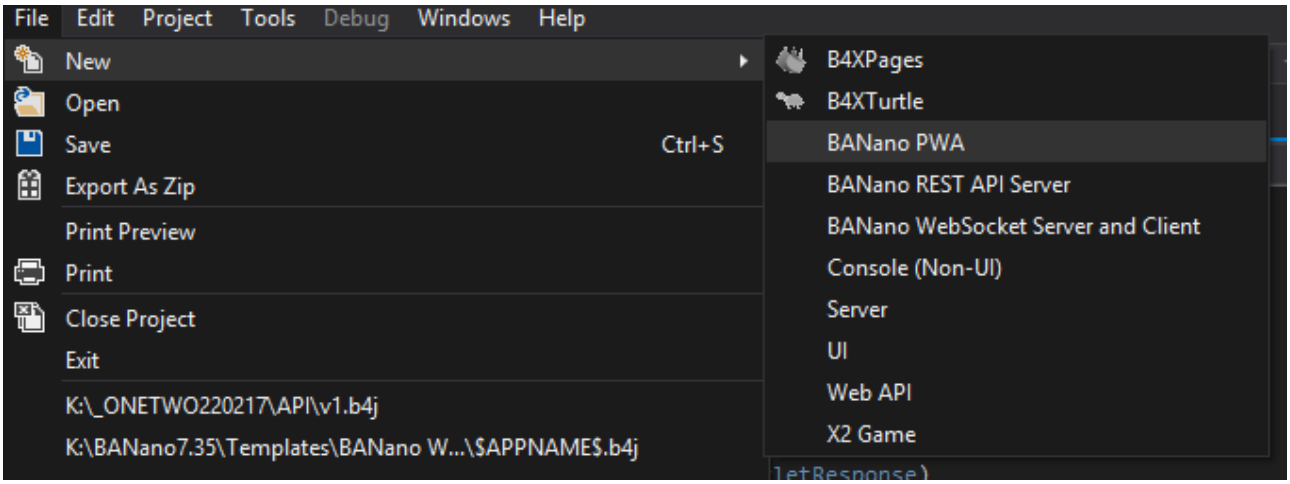
1. To create the REST API server, use the '**BANano REST API Server**' template from the menu:



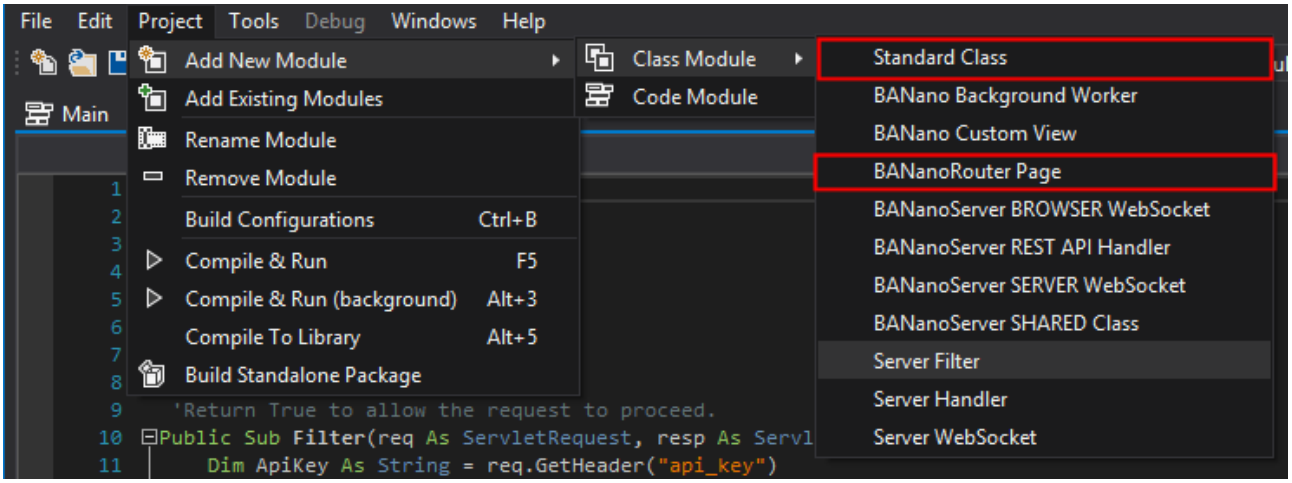
You can use these template classes:



2. The PWA is created as we have seen before using the '**BANano PWA**' template:

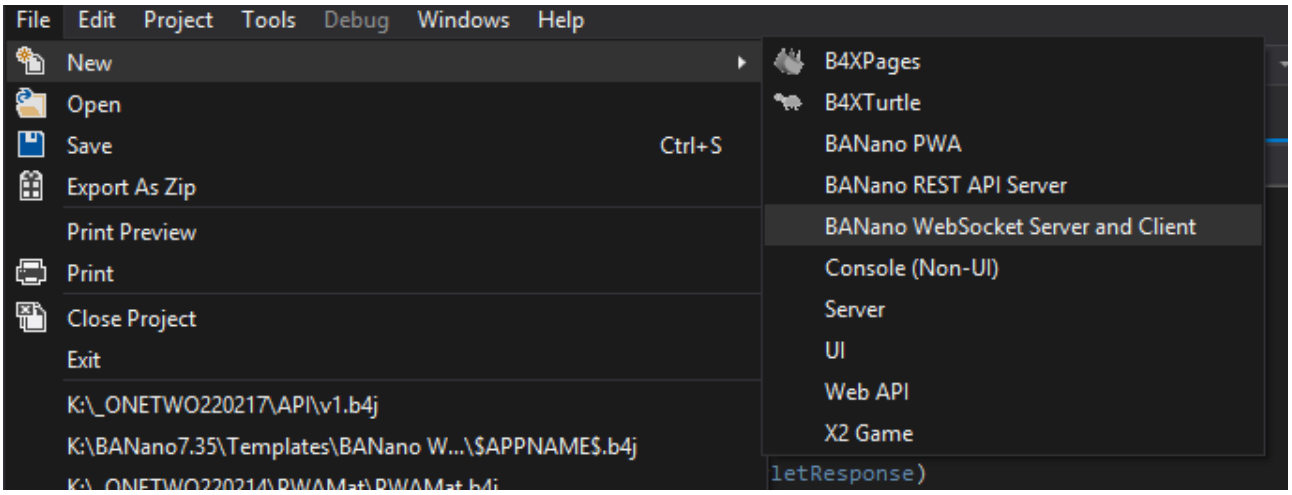


For the PWA you can use all normal classes and modules, and also a special one BANano Router Page that will be explained in a separate chapter.



13.2.2 WebSockets BANanoServer

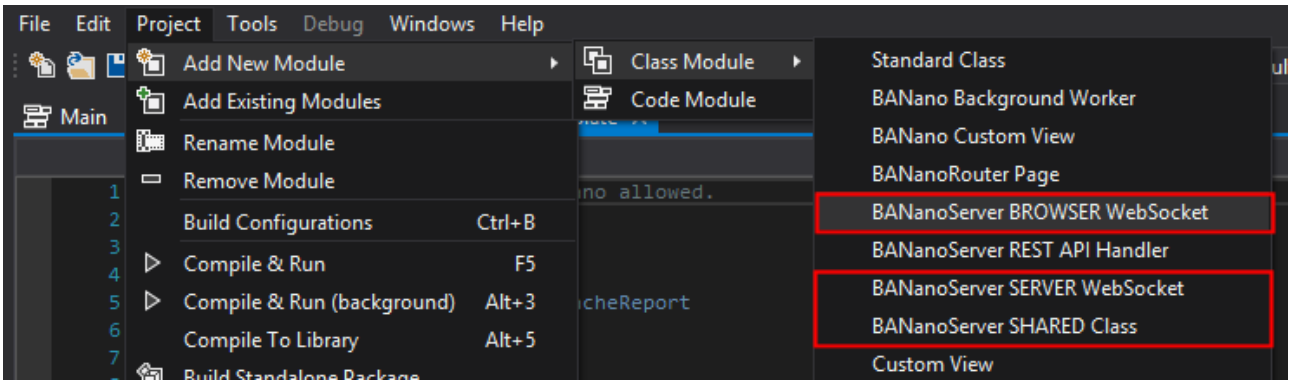
Such a Web App has **both the server code and the browser code in one B4J project**, so we only have to create one project using the '**BANano WebSocket Server and Client**' template.



When you open the project, you will notice that some files have a **special uppercase prefix**. **This is very important** in a BANanoServer WebSockets project!

SERVER: code for the SERVER ONLY
BROWSER: code for the PWA ONLY
SHARED: code that will be used by the SERVER and will be TRANSPILED for the PWA

You can easily add a template page from the menu to add such a class:



WebSockets classes generally have both a SERVER and a BROWSER part. E.g.,

SERVERTemplate and **BROWSERTemplate**. These two will be automatically connected with each other via the WebSocket. They do work as a pair.

You can of course also **add normal classes and modules** (not WebSockets ones). Just use the same prefixes: SERVER if it is for the server, BROWSER if it needs to be transpiled for the PWA, SHARED if it is used in both.

13.3 A REST API Example

In this example we will make a Request to a BANanoServer where we want to retrieve some data. E.g. the tProjects table from a MySQL database and put the result in our BANanoSQL database on the browsers side. So if we go offline with our PWA, we can still show all the info on those projects if we need to.

13.3.1 BROWSER side: PWA

Let's assume we created the database and the table tProject as in the BANanoSQL chapter:

```
SQL.OpenWait("SQL", "PWAMatData1")

SQL.ExecuteWait("CREATE TABLE IF NOT EXISTS tProject (prjid INT, prjtype INT,
prjparent INT, prjcode STRING, prjdesc STRING, prjiden STRING, prjunit STRING,
prjpar1 STRING, prjpar2 STRING)", Null)
```

What we need is a BANanoFetch to get all the projects from a certain type (GroupType here) from the Server so we can add them to the local BANanoSQL database. This could be done with a GET, but in this case I will use a POST because I want to send an ApiKey (saved as a cookie for example) with every call.

We are also going to make a json file on the Server side that we then can retrieve very quickly. In many cases you just can return the Json directly, but for this example, I want the initial fetch only to return a small Json file containing the URL the next step has to retrieve with a simple GET.

The code is commented so you can follow what happens in each step.

```
public Sub GetProjectsWait(GroupType as Long)
      ' Declaring our Fetch objects
      Dim fetch As BANanoFetch
      Dim fetchOptions As BANanoFetchOptions
      Dim fetchResponse As BANanoFetchResponse

      ' Some helper variables for the fetch to receive the responses
      Dim data As Map
      Dim Error As String

      ' Setting up our Request
      fetchOptions.Initialize
      ' it is a POST
      fetchOptions.Method = "POST"
      ' in the body of the Request, we add some Json containing the GroupType we
      want to receive
      fetchOptions.Body = $"{"type": ${GroupType}}"$
      ' in the Headers we say it is Json UTF-8 and we also add our ApiKey so we
      can check on the server side if this user is allowed to make this call
      fetchOptions.Headers = CreateMap("Content-type": "application/json;
charset=UTF-8", "api_key": "myAPIKey")
```

```
	' Let's make the initial POST Request!
	fetch.Initialize(APIUrl & "/v1/group/getgroups", fetchOptions)
	fetch.Then(fetchResponse)
		' we got a response, but as the Json() method returns a Promise, we
		will need to process it in the next 'then' so we return it to this
		Fetch
		fetch.Return(fetchResponse.Json)
	fetch.ThenWait(data)
		' here we got the actual Json back and get the url property
		Dim url As String = data.Get("url")

		' We build a second GET Fetch to retrieve the file from json the
		server
		Dim fetch2 As BANanoFetch
		fetch2.Initialize(url, Null)
		fetch2.Thenwait(fetchResponse)
			' same here, we get the actual Json and process it in the
			second .ThenWait
			fetch2.Return(fetchResponse.Json)
		fetch2.Thenwait(data)
			' we use the very fast Trick we learned from the BANanoSQL
			chapter to add the data
			SQL.ExecuteWait($"DELETE FROM tProject WHERE 1=1"$, Null)
			SQL.ExecuteWait($"SELECT * INTO [tProject] FROM ?"$,
Array(data))
			' show a toast to the user that we got the new projects
			SKTools.ShowToast("Projects Received!", "info", 3000, True)
		fetch2.ElseWait(Error)
			' We got an error!
			SKTools.ShowToast(Error, "info", 3000, True)
		fetch2.End
	fetch.ElseWait(data)
		' We got an error!
		SKTools.ShowToast(Error, "info", 3000, True)
	fetch.End
End Sub
```

Now we can have an SKButton btnSync for example where, if the user clicks on it, we call the above method:

```
Private Sub btnSync_Click (event As BANanoEvent)
	BANano.Await(GetProjectsWait(100110))
End Sub
```

13.3.2 SERVER side

So what would we need to handle the request on the server side?

After we created the Server with the BANano REST API Server' template, let's take a look at the Apps entry point:

```
Sub AppStart (Args() As String)
	' initialize the database
	Dim DBUrl As String ' =
"jdbc:mysql://127.0.0.1:3306/DATABASENAME?characterEncoding=utf8&zeroDateTimeBeh
avior=convertToNull&allowLoadLocalInfile=True"
	Dim DBLogin As String ' = "DATABASELOGIN"
	Dim DBPassword As String ' = "DATABASEPASSWORD"
	Dim MaxConnections As Long = 25

	' initialize the BANano Server
	If File.Exists(File.DirApp, "server.ini") = False Then
		Dim txtOUT As TextWriter
		txtOUT.Initialize(File.OpenOutput(File.DirApp, "server.ini", False))
		txtOUT.WriteLine("Host=localhost")
		txtOUT.WriteLine("Port=55056")
		txtOUT.WriteLine("PortSSL=0")
		txtOUT.WriteLine("CacheScavengePeriodSeconds=900")
		txtOUT.WriteLine("SessionMaxInactiveIntervalSeconds=900")
		txtOUT.Close
	End If
	Server.Initialize("server.ini") ' in Objects path defined

	' for our upload
	Server.UploadAllowedFileTypes = "ZIP;JPG"
	Server.UploadMaxSize = 1024*1024*5 ' 5 MB

	Server.AddAuthFilter("/*", "AuthFilter", False)

	' add your RESt API handlers
	Server.AddHandler("/v1/template/*", "HandlerTemplate", False)

	' cors configuration
	Server.SetCORSFilter("/*", "*", "*", "*")

	Log("http://localhost:" & Server.Port & "/" & Server.StartPage)

	' your database
	If DBUrl <> "" Then
		SERVERDBM.InitializeMySQL(DBUrl, DBLogin, DBPassword,
MaxConnections)
	End If

	' lets start the B4J server
	If Server.PortSSL <> 0 Then
		Server.StartServerHTTP2("keystore.jks", "SSLKeyStorePassword",
"SSLKeyManagerPassword")
	Else
		Server.StartServer
	End If

	StartMessageLoop
End Sub
```

As you see, this is all normal B4J jServer code. With the BANanoServer wrapper, we just have some easy to use methods like the **AddAuthFilter** and **SetCORSFilter**.

In the above BROWSER Side example (12.3.1), we need a REST API Handler in the server that can receive this:

Path: `"/v1/group/getgroups"`
Header: `api_key`
Body:

```
{
      "groupType": number
}
```

The first thing we will do in our BANanoServer B4J App is checking if this user is Authorized to use our REST API by checking the api_key.

For that, we can use a **B4J jServer Filter** that we can use for all our REST API calls. If you used the '**BANano REST API Server**' template, there is already a **AuthFilter** class build in:

```
'Return True to allow the request to proceed.
Public Sub Filter(req As ServletRequest, resp As ServletResponse) As Boolean
      ' Get the api_key from the header
      Dim ApiKey As String = req.GetHeader("api_key")
      ' Set some header options on the response
      resp.ContentType = "application/json"
      resp.SetHeader("X-Frame-Options", "DENY")
      resp.SetHeader("X-XSS-Protection", "1;mode=block")
      resp.SetHeader("Strict-Transport-Security", "max-
age=31536000;includeSubDomains;preload")
      resp.SetHeader("X-Content-Type-Options", "nosniff")
      resp.SetHeader("Referrer-Policy", "no-referrer-when-downgrade")
      resp.SetHeader("Content-Security-Policy", "script-src
https://yourdomain.com")
      resp.SetHeader("Feature-Policy", "microphone 'none'")
      ' Check the api key, if not valid, return False
      If ApiKey <> "myAPIKey" Then
            resp.Status = 401
            resp.Write("Unauthorized")
            Return False
      End If
      ' if OK, set some additional response headers and return True
      resp.SetHeader("Access-Control-Allow-Origin","*")
      resp.SetHeader("Access-Control-Allow-Methods" ,"GET, POST, UPDATE, DELETE,
OPTIONS")
      resp.SetHeader("Access-Control-Allow-Headers", "Access-Control-Allow-
Headers, Origin, Accept, X-Requested-With, Content-Type, Access-Control-Request-
Method, Access-Control-Request-Headers, Authorization")
      Return True
End Sub
```

In a real-life app, you would of course have smarter code to check e.g. if the api_key is in a database.

Next we need a handler to process our incoming BANanoFetch call. We can use a normal B4J Server handler, or the **BANanoServer REST API Handler**, which has already some code to get started in it.

We pick such a **BANanoServer REST API Handler** from the menu, give it a name e.g. **HandlerGroup.**

First we change our HandlerPath (mind the / at the end!):

```
Dim HandlerPath As String = "/v1/group/"
```

In Main, we have also have to add our new path, with the * to handle all sub paths:

```
Server.AddHandler("/v1/group/*", "HandlerTemplate", False)
```

This will catch all the calls that start with the path **/v1/group/**

Back in our HandlerGroup class, we will process the POST from the BANanoFetch. After removing some not needed code and processing the POST call, we may have something like this:

Note: SERVERDBM is a build-in wrapper in BANanoServer to handle the Database.

```
Sub Class_Globals
	' CHANGE THIS MATCHING YOUR API
	' Also add in Main the Server.AddHandler(): path with *
	Dim HandlerPath As String = "/v1/group/"
End Sub

Public Sub Initialize

End Sub

Sub Handle(req As ServletRequest, resp As ServletResponse)
	resp.ContentType = "application/json"

	Dim Response As String
	Dim bodyCode As String
	Dim body As TextReader

	Select Case req.Method
	Case "POST"
		If req.RequestURI.Length + 1 <= HandlerPath.Length Then
			SendError(resp, 404, "Invalid call")
			Return
		End If
		Dim TypePost As String
		TypePost = req.RequestURI.SubString(HandlerPath.Length)

		Select Case TypePost
		Case "getgroups"
			body.Initialize(req.InputStream)
			bodyCode = body.ReadAll
			If bodyCode.StartsWith("{") Then
				Dim jsonP As JSONParser
				jsonP.Initialize(bodyCode)
				Dim m As Map = jsonP.NextObject
				Dim GroupType As Long = m.GetDefault("type", 0)

				Dim SQL As SQL = SERVERDBM.GetSQL
				' query that gets the results you want to return
				Dim SQL_str As String = $"SELECT grpID AS lstid,
grpGrpTypID As lsttype, grpCode as lstcode, grpDescription as lstdesc,
```

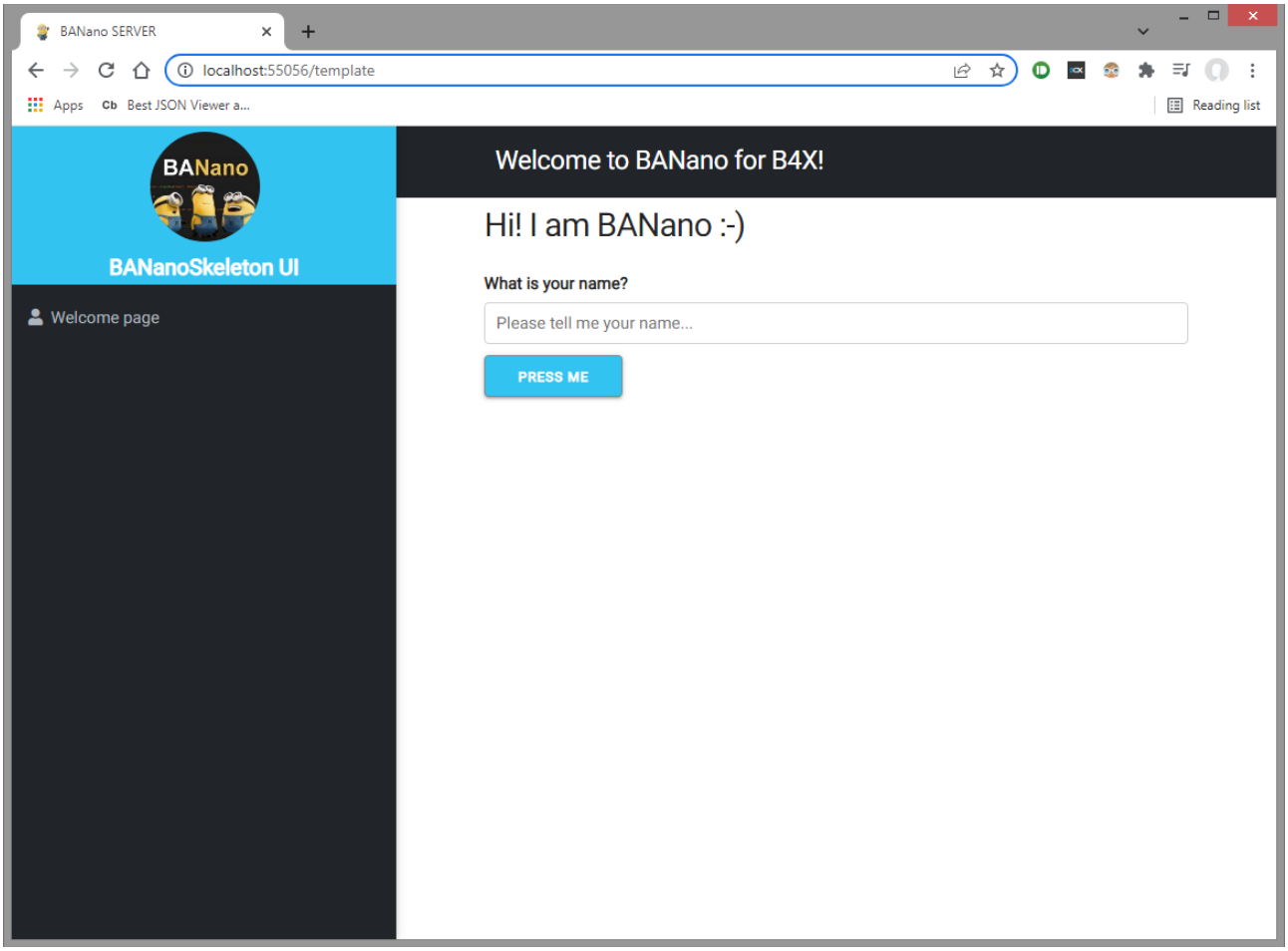
```
grpIdenCode as lstiden, grpParentID as lstparent, "" as lstunit, "" as lstpar1,
"" as lstpar2 FROM tGroup grp WHERE grp.grpGrpTypID = ${GroupType}"$

                        Dim jsonAllStr As String = "[]"
                        Dim founds As List = SQLSelectToJson(SQL, SQL_str, Null)
                        SERVERDBM.CloseSQL(SQL)

                        If founds.Size > 0 Then
                              Dim jsonG As JSONGenerator
                              jsonG.Initialize2(founds)
                              jsonAllStr = jsonG.ToString
                        End If

                        ' should be a folder in your www folder,
                        ' will be different if running in debug or release mode!
                        Dim LiteDir As String = File.GetFileParent(File.DirApp)
& "/myApp/www" & "/myApp/PWALists"

                        DateTime.DateFormat = "MMddHHmm"
                        Dim FileName As String
                        FileName = DateTime.Date(DateTime.Now) & ".json"

                        ' write the json in a file
                        Dim txtOUT As TextWriter
                        txtOUT.Initialize(File.OpenOutput(LiteDir, FileName,
False))
                        txtOUT.Write(jsonAllStr)
                        txtOUT.Close

                        ' return an url to the new file
                        Dim mOUT As Map
                        mOUT.Initialize
                        mOUT.Put("status", 1)
                        mOUT.Put("url", "http://localhost:" & Main.Server.Port &
"/" & Main.Server.StartPage & "/PWALists/" & FileName)

                        Dim jsonG As JSONGenerator
                        jsonG.Initialize(mOUT)
                        Response = jsonG.ToString
                  Else
                        SendError(resp, 404, "Invalid call")
                        Return
                  End If
            Case Else
                  SendError(resp, 404, "Invalid call")
                  Return
            End Select
      Case Else
            SendError(resp, 404, "Invalid call")
            Return
      End Select

      If Response <> "" Then
            resp.write(Response)
      End If
End Sub

' helper method to send error
public Sub SendError(resp As ServletResponse, code As Int, msg As String)
      resp.Status = code
      resp.Write(msg)
End Sub
```

```
' helper method to get the SQL results in Json format (Array of records)
Sub SQLSelectToJson(SQL As SQL, Query As String, args As List) As List
	Dim l As List
	l.Initialize
	Dim cur As ResultSet
	Try
		cur = SQL.ExecQuery2(Query, args)
	Catch
		Log(LastException)
		Return l
	End Try
	Dim first As Boolean = True
	Dim ColTypes() As Int
	Do While cur.NextRow
		If first Then
			ColTypes = GetMetaTypes(cur)
		End If
		Dim res As Map
		res.Initialize
		For i = 0 To cur.ColumnCount - 1
			Dim colName as String = cur.GetColumnName(i))
			Select Case ColTypes(i)
			Case 4
				res.Put(colName, cur.GetInt2(i))
			Case 3,8,6,7
				res.Put(colName, cur.GetDouble2(i))
			Case Else
				res.Put(colName, NullSafe(cur.GetString2(i), ""))
			End Select
		Next
		l.Add(res)
		first = False
	Loop
	cur.Close
	Return l
End Sub

Sub NullSafe(inp As Object, default As Object) As Object
	If inp = Null Or inp = "null" Then
		Return default
	End If
	Return inp
End Sub

Sub GetMetaTypes(rs As ResultSet) As Int()
	Dim JO As JavaObject = rs
	Dim rsmd As JavaObject = JO.RunMethod("getMetaData", Null )
	Dim colTypes(rs.ColumnCount) As Int

	For i = 0 To rs.ColumnCount - 1
		colTypes(i) = rsmd.RunMethod("getColumnType", Array(i+1))
	Next
	Return colTypes

	#if java
	import java.sql.ResultSet;
	import java.sql.SQLException;
	import java.sql.ResultSetMetaData;

	public ResultSetMetaData  getMeta(ResultSet rs) throws SQLException {
	    ResultSetMetaData rsmd = rs.getMetaData();
	    return rsmd;
	}
	#End If
End Sub
```

13.4 A WebSockets Example

Using WebSockets is very similar to using normal B4J WebSockets. The server side is the same, but now we can also write the Client side (as in default B4J WebApps, we had to write any JavaScript, HTML or CSS manually ourselves).

If you used the '**BANano WebSocket Server and Client**' template, it will contain the familiar normal PWA example, but now using WebSockets:



Unlike the REST API example, we have here both the server and the client code in one B4J project.

Let's have a look how the Apps entry point looks like compared to the REST API one:

```
Sub AppStart (Args() As String)
	' initialize the database
	Dim DBUrl As String ' =
"jdbc:mysql://127.0.0.1:3306/DATABASENAME?characterEncoding=utf8&zeroDateTimeBeh
avior=convertToNull&allowLoadLocalInfile=True"
	Dim DBLogin As String ' = "DATABASELOGIN"
	Dim DBPassword As String ' = "DATABASEPASSWORD"
	Dim MaxConnections As Long = 25

	' initialize the BANano Server
	If File.Exists(File.DirApp, "server.ini") = False Then
		Dim txtOUT As TextWriter
		txtOUT.Initialize(File.OpenOutput(File.DirApp, "server.ini", False))
		txtOUT.WriteLine("Host=localhost")
		txtOUT.WriteLine("Port=55056")
```

```
            txtOUT.WriteLine("PortSSL=0")
            txtOUT.WriteLine("CacheScavengePeriodSeconds=900")
            txtOUT.WriteLine("SessionMaxInactiveIntervalSeconds=900")
            txtOUT.Close
      End If
      Server.Initialize("server.ini") ' in Objects path defined

      ' for our upload
      Server.UploadAllowedFileTypes = "ZIP;JPG"
      Server.UploadMaxSize = 1024*1024*5 ' 5 MB

      ' OPTIONAL: the prefix of our BROWSER (BANano only code) classes that
miror their SERVER counterpart (default is "BROWSER")
      ' IMPORTANT to set this one if you do not use this default Prefix! (not
advised.
      Server.BROWSERPrefix = "BROWSER"

      ' initialize BANano
      Server.BANano.Initialize("BANano", "BANanoServer" ,1)
      Server.BANano.TranspilerOptions.SetStaticFolder("www")
      Server.BANano.Header.Title="BANano SERVER"

      ' enable/disable live code swapping
      Server.BANano.TranspilerOptions.EnableLiveCodeSwapping = False

      ' some B4J typical libs we want to be ignored by the Transpiler
      Server.BANano.TranspilerOptions.IgnoreB4JLibrary("Json")

      ' write the theme
      SKTools.WriteTheme

      ' transpile all the BANano b4J code to javascript
      Server.BANano.BuildForServer(Server.OutputFolder)

      ' add your SERVER classes, not the BROWSER parts:
      Server.AddWebSocket("/ws/" & Server.BANano.StaticFolder & "/template" ,
"SERVERTemplate")

      ' set the start page one will go to if they enter the site by the root
      Server.StartPage = "template"

      ' cors configuration
      Server.SetCORSFilter("/*", "*", "*", "*")

      Log("http://localhost:" & Server.Port & "/" & Server.StartPage)

      ' your database
      If DBUrl <> "" Then
            SERVERDBM.InitializeMySQL(DBUrl, DBLogin, DBPassword,
MaxConnections)
      End If

      ' lets start the B4J server
      If Server.PortSSL <> 0 Then
            Server.StartServerHTTP2("keystore.jks", "SSLKeyStorePassword",
"SSLKeyManagerPassword")
      Else
            Server.StartServer
      End If

      StartMessageLoop
End Sub
```

The big picture is very similar to the REST API one, except here we are also using **BANano for the Client side in the same project**, instead of in a separate PWA.

For this, we have to incorporate also the same things we do in a BANano PWA StartApp(): Initializing BANano, Setting some Transpiler Options, Writing the theme.

Differences with the normal PWA are:

```
Server.BROWSERPrefix = "BROWSER"  ' optional
Server.BANano.TranspilerOptions.SetStaticFolder("www")
```

And especially:

```
Server.BANano.BuildForServer(Server.OutputFolder)
```

In a normal BANano PWA, we use .**Build**(), but for a BANanoServer WebSocket project, we do need to use the .**BuildForServer**() method.

Furthermore (but this is also normal B4J jServer code), we add our pages using the `Server.AddWebSocket` method.

ONLY Add the SERVER versions of the SERVER/BROWSER pair classes!

Finally, we have to set our first page when the user first enters our WebApp:

```
Server.StartPage = "template"
```

13.4.1 BROWSER side: PWA

As said before, in such a project we have to use a prefix on our classes to indicate if it is for the BROWSER, for the SERVER, or shared code for both.

Looking at the BROWSER Template class, you will see this all looks very familiar to a normal PWA (differences in red, some of it just to demonstrate something):

```
'BANano compatible ONLY code. You cannot use typical B4J libraries here.  Use
their BANano version (if it exists)
'Making changes in this module/class in B4J debug mode will NOT have any effect
until recompiled!
Sub Class_Globals
	Private BANano As BANano 'ignore
	Private ws As BANanoWebSocket

	' from the MainLayout
	Private MainHamburgerMenu As SKLabel 'ignore
	Private MainSidebar As SKSidebar 'ignore
	Private MainPageHolder As SKContainer 'ignore

	' from the WelcomeModalLayout
	Private WelcomeModal As SKModal 'ignore
	Private WelcomeModalMessage As SKLabel 'ignore

	' from the WelcomePageLayout
```

```
	Private WelcomePageName As SKTextBox 'ignore
	Private WelcomePageButton As SKButton 'ignore

	' from the MenuLayout
	Private MenuList As SKMenu 'ignore

	' some media queries for our responsive menu
	Private Bigger992px As BANanoMediaQuery
	Private Smaller992px As BANanoMediaQuery

	Private Counter As Long
End Sub

'Initializes the object. You can NOT add extra parameters!
Public Sub Initialize
	' does the browser support websockets?
	If ws.IsSupported Then
		' here we connect to our SERVERTemplate websocket class using the
'classic' B4J Websocket events WebSocket_Connected and WebSocket_Disconnected
		' this must match with the first parameter of .AddWebSocket in Main!
		' Server.AddWebSocket("/ws/" & Server.BANano.StaticFolder &
"/template" , "SERVERTemplate")
		ws.Initialize("ws://" & BANano.Location.GetHost & "/ws/" &
BANano.StaticFolder & "/template")
	End If
End Sub

' Server says socket is ready
Sub WebSocket_Connected()
	Log("Connected ===> My B4J PageId: " & BANano.GetPageID)

End Sub

Sub WebSocket_Disconnected(event As BANanoEvent)
	Log("Websocket closed")
End Sub

public Sub BANano_Ready()
	Private body As BANanoElement
	body.Initialize("#body")

	' append and load our main layout
	body.Append($"<div
id="mainHolder"></div>"$).Get("#mainHolder").LoadLayout("MainLayout")
	' append and load a modal sheet
	body.Append($"<div
id="modalHolder"></div>"$).Get("#modalHolder").LoadLayout("WelcomeModalLayout")

	' loading our menu in our sidebar
	MainSidebar.Element.LoadLayout("MenuLayout")

	' making the menu layout responsive: always open when screen size is
bigger than 992px
	Bigger992px.Initialize("(min-width: 992px)")
	Smaller992px.Initialize("(max-width: 991px)")

	' add our menu items
	MenuList.AddMenuItem("", "page1", "fas fa-user", "{NBSP}{NBSP}Welcome
page")
	MenuList.Start

	' load our first page
	MainPageHolder.Element.LoadLayout("WelcomePageLayout")
End Sub
```

```
Sub Bigger992px_Matched()
	MainSidebar.AlwaysOpen = True
	' and hide the hamburger button
	MainHamburgerMenu.Element.SetStyle($"{"visibility": "hidden"}"$)
End Sub

Sub Smaller992px_Matched()
	MainSidebar.AlwaysOpen = False
	' and show the hamburger button
	MainHamburgerMenu.Element.SetStyle($"{"visibility": "unset"}"$)
End Sub

Sub WelcomePageButton_Click (event As BANanoEvent)
	If WelcomePageName.Text = "" Then
		Counter = Counter + 1
		' must end with _BAN and have only one parameter (a map)
		ws.B4JSend("SERVERForgotHisName_BAN", CreateMap("counter": Counter))
		SKTools.ShowToast("Please enter your name!", "info", 3000, True)
		Return
	End If
	WelcomeModalMessage.Text = "Welcome " & WelcomePageName.Text

	WelcomeModal.Open
End Sub

Sub MenuList_Click (returnName As String)
	SKTools.ShowToast("Clicked on " & returnName & "!", "info", 3000, True)
	' here we can load the layout of the menu item we clicked
	Select Case returnName
		Case "page1"
			MainPageHolder.Element.Empty
			MainPageHolder.Element.LoadLayout("WelcomePageLayout")
	End Select
	' and close the menu, if not always open
	If MainSidebar.AlwaysOpen = False Then
		MainSidebar.Close
	End If
End Sub

Sub MainHamburgerMenu_Click (event As BANanoEvent)
	MainSidebar.Open
End Sub

' the SERVER counterpart of this BROWSER page can ask what the current value of
counter is
public Sub BROWSERAskForCounter() As Long
	Return Counter
End Sub
```

In the **Initialize** method, we have to define the WebSocket connection. This maps with the SERVER one we defined in AppStart:

AppStart:

```
Server.AddWebSocket("/ws/" & Server.BANano.StaticFolder & "/template" ,
"SERVERTemplate")
```

BROWSERTemplate:

```
ws.Initialize("ws://" & BANano.Location.GetHost & "/ws/" & BANano.StaticFolder &
"/template")
```

We also have two new events:

```
Sub WebSocket_Connected()
      Log("Connected ===> My B4J PageId: " & BANano.GetPageID)

End Sub
```

The BANano.**GetPageID** is a handy method to find out which BROWSERClass is connected with which SERVERClass.

```
Sub WebSocket_Disconnected(event As BANanoEvent)
      Log("Websocket closed")
End Sub
```

In the `WelcomePageButton_Click` method, you see an example of calling a method in the matching SERVERClass (see further for its definition, but important to remember here is that such methods **must have a suffix _BAN**)

```
ws.B4JSend("SERVERForgotHisName_BAN", CreateMap("counter": Counter))
```

Here we have also written a method that we will later call from the SERVER side (see further):

```
public Sub BROWSERAskForCounter() As Long
      Return Counter
End Sub
```

13.4.2 SERVER side

On the SERVER side of this class pair, we have the following code:

```
' B4J compatible ONLY code, no BANano allowed.
'WebSocket class
Sub Class_Globals
	Private ws As WebSocket
	Private CacheReport As BANanoCacheReport

End Sub

Public Sub Initialize

End Sub

Private Sub WebSocket_Connected (WebSocket1 As WebSocket)
	Log("Connected")

	ws = WebSocket1

	' Lets update the cache with this class
	CacheReport = Main.Server.UpdateFromCache(Me, ws)
	Log("PageID: " & CacheReport.BANPageID)
	Log("Comes From Cache:" & CacheReport.ComesFromCache)
	Log("Is a reconnecting socket: " & CacheReport.IsReconnected)

	' IMPORTANT lets tell the browser we are ready to receive call from the
browser
	' Uses the classic WebSocket_Connected and WebSocket_DisConnected events
on the browser size
	' Use Main.Server.SendReady(ws, "ws") if you use the advanced events
OnOpen, OnMessage, OnServerReady, ...
	Main.server.SendConnected(ws)
End Sub

Private Sub WebSocket_Disconnected
	Log("disconnected")
End Sub

' event raised to distribute incoming events coming from the BROWSER
public Sub BANano_ParseEvent(params As Map)
	Main.Server.ParseEvent(Me, ws, CacheReport.BANPageID,
CacheReport.BANSessionID, params)
End Sub

' event raised when a file has been uploaded
public Sub BANano_Uploaded(status As Int, fileName As String)
	Log(fileName & " = " & status)
	Select Case status
		Case 200 ' OK
		Case 500 ' was not a POST call
		Case 501 ' file to big
		Case 502 ' file type not allowed
	End Select
End Sub
```

```
' a method that can be called by the BROWSER class mathing this SERVER class
' must have the suffix _BAN and have only ONE parameter: params As Map
public Sub SERVERForgotHisName_BAN(params As Map)
	Dim counter1 As Long = params.Get("counter")

	' ask for the counter from the server side
	Dim fut As Future = ws.RunFunctionWithResult("BROWSERAskForCounter", Null)
	Dim counter2 As Long = fut.Value

	If counter1 = counter2 Then
		Log("They are the same!")
	Else
		Log("They are different!")
	End If
End Sub
```

Let's break the code down.

First we declare a **BANanoCacheReport** object:

```
Private CacheReport As BANanoCacheReport
```

This is an object that holds some valuable information like:

```
CacheReport = Main.Server.UpdateFromCache(Me, ws)
Log("PageID: " & CacheReport.BANPageID)
Log("Comes From Cache:" & CacheReport.ComesFromCache)
Log("Is a reconnecting socket: " & CacheReport.IsReconnected)
```

A BANanoServer holds a **cache** of the class into memory, in case you somehow get disconnected. This is done by the `Main.Server.UpdateFromCache(Me, ws)` method.

If now for example `CacheReport.IsReconnected` is **true**, you can take a different action then if it is a brand-new connection. As the class was cached, some variables the user did give a value will still be retained as we the class has been 'restored' by the `UpdateFromCache()` method.

And remember the BANano.**GetPageID** method we used in the previous chapter about the BROWSER side? Well here we got the same value on the SERVER side with `CacheReport.BANPageID`!

ALWAYS end the WebSocket_Connected() method with:

`Main.server.SendConnected(ws)`

This lets the BROWSER side know we are connected and we can start communicating.

We also have here two special events:

```
public Sub BANano_ParseEvent(params As Map)
      Main.Server.ParseEvent(Me, ws, CacheReport.BANPageID,
CacheReport.BANSessionID, params)
End Sub
```

This event will parse any call from the BROWSER side and delegate it to the corresponding **_BAN** method here on the SERVER side.

In this case this call in the BROWSER:

```
ws.B4JSend("SERVERForgotHisName_BAN", CreateMap("counter": Counter))
```

will pass through the `BANano_ParseEvent` method and then will be delegated to the final `SERVERForgotHisName_BAN` method.

Another event we have here is the Uploaded event. In case we upload some file from the BROWSER side (using a POST call), this event will be called.

```
public Sub BANano_Uploaded(status As Int, fileName As String)
      Log(fileName & " = " & status)
      Select Case status
            Case 200 ' OK
            Case 500 ' was not a POST call
            Case 501 ' file to big
            Case 502 ' file type not allowed
      End Select
End Sub
```

Just like we could call the **_BAN** method here in our SERVER class from the BROWSER class with the **ws.B4Jsend** method, we can also call a method in the BROWSER class from the SERVER class. For this, we use **the normal B4J** method **ws.RunFunctionWithResult**.

```
Dim fut As Future = ws.RunFunctionWithResult("BROWSERAskForCounter", Null)
Dim counter2 As Long = fut.Value
```

14 Background Workers

Background Workers are a simple means for web content to run scripts in background threads. The worker thread can perform tasks without interfering with the user interface. In addition, they can perform I/O using BANanoFetch.

Once created, a worker can send messages to the main thread code (that created it) by posting messages with **BANano.SendFromBackgroundWorker()** to a **BANano_MessageFromBackgroundWorker()** event handler specified by that code (and vice versa by using the **BANano.RunBackgroundWorkerMethod()**).

They **cannot be added in a BANanoLibrary**, so it must be done in your final project. Background workers are actually Web Workers, but I prefer using the term 'Background Worker' as it is familiar to the B4J language.

IMPORTANT NOTES:

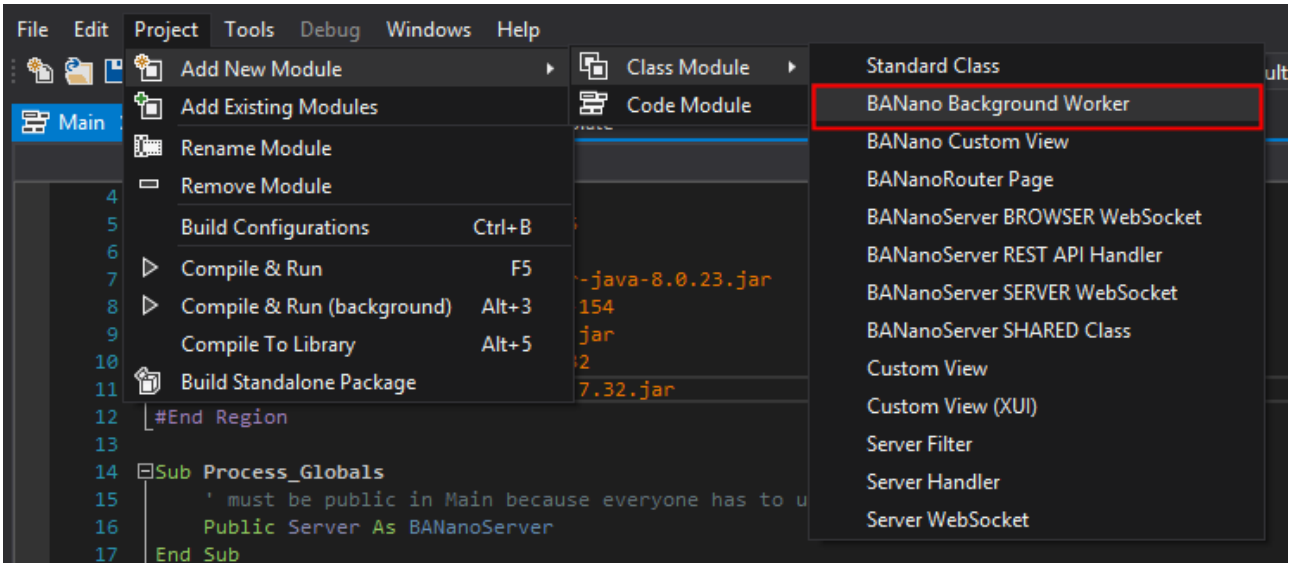
1. Background Workers can **NOT** access:
- The DOM: they cannot read or modify the HTML document. In addition, you cannot access global variables or JavaScript functions within the page
- The window, the document and the parent objects

2. Data send to Background Workers is **copied, NOT shared**. So changing the value of a variable (that you passed with **RunBackgroundWorkerMethod())** in a Background Worker will NOT be changed in the caller class. You will have to pass on the new value to the caller class with **SendFromBackgroundWorker().**

3. A worker cannot be run directly from the filesystem. It can only be run via a server.

An example

1. Add a new BANano Background Worker Class, let's call it MyBackgroundWorker:



Some code will be generated:

```
'This is a BANano Background worker template class
Sub Class_Globals
    Private BANano As BANano 'ignore
    Private mTimer As Timer
    Private mTimerTickMs As Int = 1000
End Sub

' can have additional parameters
Public Sub Initialize(TicksMs As Int)
    ' additional javscript needed in the Worker
    ' THESE CAN NOT CONTAIN JAVASCRIPT CODE THAT MANUPULATE THE DOM
    ' BANano.DependsOnAsset("myCode.js")

    mTimerTickMs = TicksMs
    mTimer.Initialize("Timer", mTimerTickMs)
    mTimer.Enabled = True

End Sub

Public Sub BANano_StopBackgroundWorker()
    mTimer.Enabled = False

End Sub

Sub Timer_Tick
    'do the work required


    ' Send something back to the calling class
    ' BANano.SendFromBackgroundWorker("SomeTag", Array(SomeValues), Null)

End Sub
```

Now we can make it do something, e.g. we want it do run in the background every second and add +1 to a counter.

When it reaches *counter mod 10,* send the current value back to the calling class.

We also add a method `AddToCounter` to immediately add some value to the counter variable from the calling class. The Initialize method has been changed to accept an additional parameter Title too.

```
'This is a BANano Background worker class
Sub Class_Globals
    Private BANano As BANano 'ignore
    Private mTimer As Timer
    Private mTimerTickMs As Int = 1000
    Private mCounter As Int = 0
    Private mTitle As String
End Sub

' can have additional parameters
Public Sub Initialize(Title As String, TicksMs As Int)
    mTitle = Title
    mTimerTickMs = TicksMs

    mTimer.Initialize("Timer", mTimerTickMs)
```

```
    mCounter = 0
    mTimer.Enabled = True
End Sub

Public Sub BANano_StopBackgroundWorker()
    mTimer.Enabled = False
End Sub

Sub Timer_Tick
    'do the work required
    mCounter = mCounter + 1
    Log(mTitle & ": every " & mTimerTickMs & ", Counter: " & mCounter)
    If mCounter Mod 10 = 0 Then
        ' can only be used in a BackgroundWorker!
        BANano.SendFromBackgroundWorker("Mod10", Array(mCounter), Null)
    End If
End Sub

public Sub AddToCounter(value As Int)
    mCounter = mCounter + value
End Sub
```

Now, how to use our Background Worker?

First me must create a couple of instances of the worker. This **MUST** be predefined in the **AppStart()** method. E.g. here, we are going to use two instances of our MyBackgroundWorker somewhere in our code later.

```
Sub AppStart (Form1 As Form, Args() As String)
...
    BANano.AddBackgroundWorker("worker1", "MyBackgroundWorker")
    BANano.AddBackgroundWorker("worker2", "MyBackgroundWorker")
...
End Sub
```

When we need our Background Workers, we can start them like this, e.g. in BANano_Ready()

```
' run de Start method of the Background Workers
' start Worker1, and in our Initialize of our MyBackgroundWorker we need two
parameters (Title and TicksMs)
BANano.StartBackgroundWorker("worker1", Array("From Worker 1", 1000))
BANano.StartBackgroundWorker("worker2", Array("From Worker 2", 2000))
```

Now all we have to do is add the event **BANano_MessageFromBackgroundWorker**() to receive messages from our Background Workers. e.g. In MyBackgroundWorker, we do use a BANano.**SendFromBackgroundWorker**() call to send the current counter back to our calling class with a Tag "Mod10".

```
public Sub BANano_MessageFromBackgroundWorker(WorkerName As String, Tag As
String, Value As Object, Error As Object)
    ' the Tag will define the type of message send by the Background Worker
    If Tag = "Mod10" Then
        Log(WorkerName)
        Log("Current Error: " & Error)
        Log("Current Value: " & Value)
    End If
End Sub
```

Somewhere else in our code, e.g. by pressing a button, we can call the AddToCounter method from our class that initialized the workers with the **RunBackgroundWorkerMethod()** to immediately add 1000 to the counter.

```
Sub BtnAdd_Click (event As BANanoEvent)
      BANano.RunBackgroundWorkerMethod("Worker1", "", "AddToCounter",
Array(1000))
End Sub
```

We can stop the Background Workers with **StopBackgroundWorker()**

```
' calls the BANano_StopBackgroundWorker event in the BackgroundWorker Class
BANano.StopBackgroundWorker("worker1", Null)
BANano.StopBackgroundWorker("worker2", Null)
```

So, a BANano Background Worker is like an advanced timer. It can for example be useful for fetching a big JSON file, while the user can still interact with the WebApp.

Helpful for such a case is the use of a **CRON job** instead of a Timer, discussed in the next chapter.

15 CRON: an Advanced Timer

A Cron is Timer like functionality that runs a certain job **automatically at a specified time following a predefined pattern**. A Cron Job is the scheduled task itself. Cron jobs can be very useful to automate repetitive tasks.

You could for example schedule in a Background Worker to sync your data every hour on a weekday.

cronName: Name of the Cron job. This cannot be a variable and must be a literal String and cannot contain spaces or special characters!

maxRuns: you can set a maximum number of times the Cron job should run (0 = indefinite, until you Stop it)

pattern: Cron jobs use a special **Pattern** format to define them:

```
* * * * * *
S M H D m d
```

S: second (0 - 59)
M: minute (0 - 59)
H: hour (0 - 23)
D: day of month (1 - 31)
m: month (1 - 12)
d: day of week (0 - 6), 0 to 6 are Sunday to Saturday; 7 is Sunday, the same as 0

Ranges:
Ranges are two numbers separated with a "-", and they indicate all numbers from one to the other. e.g. 10-30 would indicate all numbers between and including 10 to 30.

Interval:
A interval is a range and a number separated by "/". The range specifies the group of values, and number specifies every nth value to take from that range.
e.g. 0-10/2 would indicate every 2nd number from 0 to 10, therefore [0,2,4,6,8,10]

```
' at 00:00:00 on every weekday run, for a total of 15 times, then stop this Cron
BANano.CronStart("myCron", 15, "0 0 0 * * 2-6")

Public Sub MyCron_Run()
   ' do something, like a sync of your data to a server
End Sub

Public Sub btnPause_Click(event as BANanoEvent)
     BANano.CronPause("myCron")
End Sub

Public Sub btnResume_Click(event as BANanoEvent)
     BANano.CronResume("myCron")
End Sub

Public Sub btnStop_Click(event as BANanoEvent)
     BANano.CronStop("myCron")
End Sub
```

So we could use such a CRON job to sync for example our data from our PWA to the server in a Background Worker that runs every hour on week days.

```
Sub Class_Globals
    Private BANano As BANano 'ignore
    Private SQL As BANanoSQL
End Sub

Public Sub Initialize()
    ' every weekday, every hour run this CRON job
    BANano.CronStart("myCron", 0, "0 * * * * 2-6")
End Sub

Public Sub MyCron_Run()
   BANAno.Await(SendDataWait)
End Sub

Public Sub BANano_StopBackgroundWorker()
    ' Stop the CRON job
    BANano.CronStop("myCron")
End Sub

public Sub SendDataWait()
      ' re-initialize the database in this local class
      SQL.OpenWait("SQL", "MyDB")

      Dim SQL_str as String
      Dim Results As List
      ' get all the records on status 1 (not yet send)
      SQL_str = $"SELECT * FROM tData WHERE dtstatus=1"$
      Results = SQL.ExecuteWait(SQL_str, Null)

      ' buiding our POST BANanoFetch
      Dim fetch As BANanoFetch
      Dim fetchOptions As BANanoFetchOptions
      Dim fetchResponse As BANanoFetchResponse

      Dim Data As Map
      Dim Error As String

      ' Make json string from the list of records
      Dim JsonG As JSONGenerator
      JsonG.Initialize(Results)

      ' buiding our POST BANanoFetch to a REST API "/v1/data/upload"
      fetchOptions.Initialize
      fetchOptions.Method = "POST"
      fetchOptions.Body = JsonG.ToString
      fetchOptions.Headers = CreateMap("Content-type": "application/json;
charset=UTF-8", "api_key": APIKey)

      fetch.Initialize(APIUrl & "/v1/data/upload", fetchOptions)
      fetch.Then(fetchResponse)
            fetch.Return(fetchResponse.Json)
      fetch.ThenWait(Data)
            If data.get("status") = "OK" Then
                  ' if OK, set all our records to status 2 (send)
                  SQL_str = $"UPDATE tData SET dtstatus=2 WHERE dtstatus=1"$
                  SQL.ExecuteWait(SQL_str, Null)
            End If
      fetch.ElseWait(Error)
            Log(Error)
      fetch.End
End Sub
```

16 BANanoRouter: multi page PWA

BANanoRouter is a full-blown Router with paths. It allows you to organize your code in a PWA WebApp. It is based on the Navigo project, tuned for B4J.

I personally do not use this in my PWA's. B4J has already so much organization of your code build-in with its classes and modules that I do not really need it. I simply put the code in a normal B4J class per page and load them when needed. By emptying the <body> tag and loading the new layout, I have the same effect.

16.1 What is a Javascript Router?

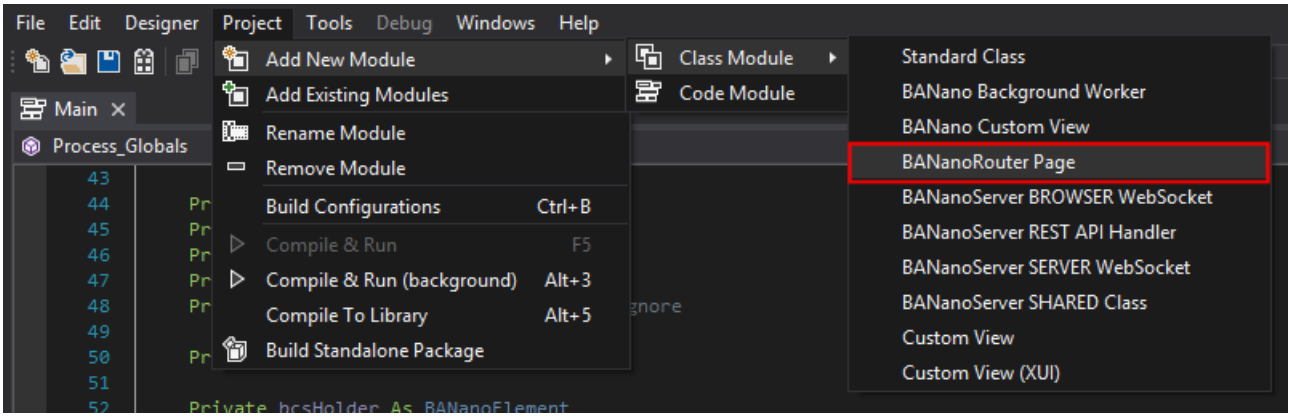
A Javascript router is a key component in many frontend frameworks (e.g. Angular, Vue, ...). It is the piece of software in charge to organize the states of the application, switching between different views. For example, the router will render the login screen initially, and when the login is successful it will perform the transition to the user's welcome screen.

So 'logically', you web app could be something like:

```
App
--- Page1
--- Page2
--- Page3
```

Every page will have its own path, with its own variables and query parameters. The router is built so the URL in the navigation bar of the browser does not change. However, it one can still call a certain page with certain parameters by entering it in the navigation bar if one wants to.

We can add a new '**BANanoRouter Page**' from the menu:



This will generate some basic structure of a page.

```
'This class is router page template class
Sub Class_Globals
    Private BANano As BANano 'ignore

End Sub

'Initializes the object. You can add parameters to this method if needed.
Public Sub Initialize()

End Sub

' router path /testPage1
Sub BANano_RouterHandle(url As String, data As Map, params As Map)
    Log(url)
    Log(data)
    Log(params)

    ' navigating to some other page
    ' Main.router.Navigate("/testPage2/carine/?id=10&lastName=Bailleul")
End Sub

Sub BANano_RouterLeaving() As Boolean
    Log("Do some checks...")

    Return True ' (Or False If the navigation from this page is not allowed)
End Sub
```

It contains two special methods:

BANano_RouterHandle(url As String, data As Map, params As Map)

In this method you can do all your nice BANano stuff (like loading a Layout, getting data from your database etc)

BANano_RouterLeaving() As Boolean

This method is optional, and allows you to e.g. do some checks (is every field filled in?) before someone can leave the page and navigate to another one. If it returns **True** then it will go further, if **False** it will not.

16.2 Setup up the routes

Add a BANanoRouter to the Process_Globals of your Main:

```
Public router As BANanoRouter
```

In BANano_Ready():

1. Initialize the router

```
router.Initialize("/",False)
```

rootPath: the root path of your application. For example, if you are hosting the application at https://site.com/my/project you have to specify the following:

matchAll: default false, meaning that when a match is found the router stops resolving other routes. If set true, it will continue searching for other matches

e.g. `Router.AddRoute("/foo/:id/?", "FooClass")` matches "/foo/20/save" and also "/foo/20"

2. Now we can add our routes, for example

```
' here our initialize method in our page requires an extra parameter
router.AddRoute("/testPage1", "Page1", Array("Something extra"))

' e.g. handle /testPage2/carine (carine will be in the Data map in
BANano_RouterHandle of Page2)
router.AddRoute("/testPage2/:name", "Page2", Null)

' e.g. handles /testPage2/alain/test,  /testPage2/jos/test (alain or jos will be
in the Data map in BANano_RouterHandle of Page3)
router.AddRoute("/testPage3/:name/test", "Page3", Null)
```

Some more advanced examples of paths:

```
' matches "/about-page"
Router.AddRoute(":page", "FooClass")

' matches "/foo/a/b/c"
Router.AddRoute("/foo/*", "FooClass")

' matches "/foo/bar/moo"
Router.AddRoute("*", "FooClass")

' matches "/foo/20/save" and also "/foo/20"
Router.AddRoute("/foo/:id/?", "FooClass")
```

3. A special one can be added if there is no match found for the path.

```
router.NotFound("NotFound", Null)
```

4. And finally, we start our router, going to our first page

```
router.Start("/testPage1")
```

16.3 Navigating between pages

This can be done in two ways:

By code

```
' will go to the Page2 class, with the variable "name" set to carine and the
parameters id=10 and lastName="Bailleul"
Main.router.Navigate("/testPage2/carine/?id=10&lastName=Bailleul")

' will go to the Page3 class with the variable "name" set to jos
Main.router.Navigate("/testPage3/jos/test")

' will e.g. go to the NotFound class because it does not exist
Main.router.Navigate("/testPage4")
```

So, although you internally change to another URL, the text in the Navigation Bar in the browser will still be https://mydomain.com

By entering an URL in the Browsers Navigation Bar

The router does use a hash system (#), so by just entering your path with the prefix /#/, it will be handled by the router.

Example will do exactly the same as the first example here above:

https://mydomain.com/#/testPage2/carine/?id=10&lastName=Bailleul"

If you use the second method (entering in the Browser Navigation Bar) you will not be able to use the `GetURLParamDefault` method directly as the # (hash) will interfere with this method.

You can use this small trick to work around this problem.

```
Dim Token As String =
BANano.GetURLParamDefault(BANano.Location.GetHref.Replace("#/",""), "token", "")
Log(Token)
```

16.4 Removing a route

Just call RemoveRoute with the original path you used to add it.

```
Router.RemoveRoute("/testPage2/:name")
```

17 Debugging

17.1 Live Code Swapping

BANano has some nice features to debug projects. It can for example make use of B4Js **Live Code Swapping** feature!

Live Code Swapping is only available in the final PWA project, not in BANanoLibraries or BANanoServer projects

To activate this feature, you just have to set this parameter in Appstart:

```
' enable live code swapping
BANano.TranspilerOptions.EnableLiveCodeSwapping = True
```

Now you can run de project in debug mode and make live changes to the B4J code. When you press Save, BANano will try to make the changes to the transpiled JavaScript code. By pressing F5 in the browser, the new code will be loaded.

You can even make changes in the Abstract Designer and on save the new layouts will be used.

Live Code Swapping is much faster than completely recompile your code, as it will only transpile the changes and e.g. not the BANanoLibraries you used in the project.

17.2 Making use of the new B4J 'jump' feature in the logs

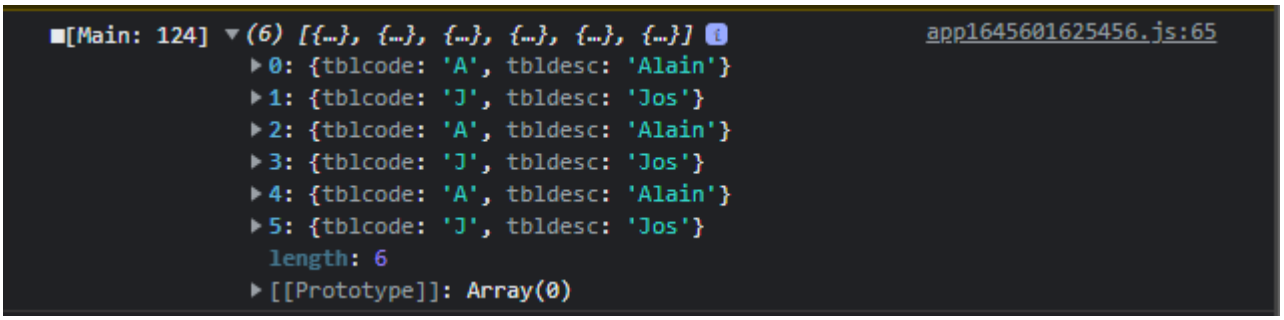
Since B4J v9.30, you can click into the log and it will jump to the line where the error (or log line) is done.

BANano v7.35+ can also use this feature. As it doesn't have access to the IDE, a little trick has to be used.

By including the following snippet (**must be exactly this!**) on top of your AppStart() method, the jumps will also work with BANano.

```
#if Debug
      ' MUST be literally this line if you want to use the B4J Logs jump to code
feature!
      Log("BANanoLOGS")
#End if
```

In the browsers log, you will also see on which line something happened (e.g. a log)



This is the result of Log(myList) in the Main module at line 124.

17.3 JavaScript Breakpoints

You can add a breakpoint in the JavaScript code by using the **BANano**.BP method.

It stops the execution of JavaScript in Debug Mode. This command is ignored if in release mode.

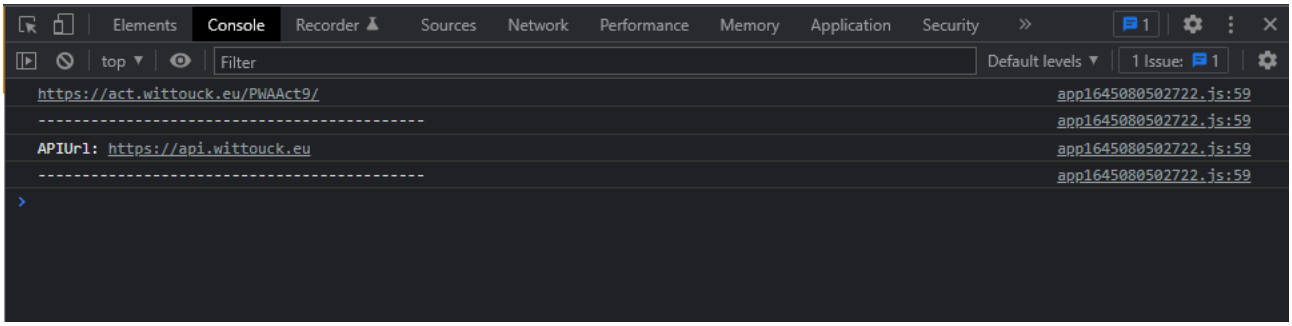
Use the Developer Tools in the browser to inspect e.g. variable values

17.4 Using the Browser Developer Tools

Meet your next Best Friend when developing Web apps: **The Chrome Devtools**!

Every browser has some tools to help the developer in debugging their apps. I will go into the ones in Chrome that are important for debugging (or resetting) a BANano PWA app.

To open up de Chrome Developer Tools, press F12 in the browser. A new panel will open up:



Some of the important tabs and functionalities are Console, Network, Application and Lighthouse.

17.4.1 The Console Tab

The Console has two main uses: **viewing logged messages** and **running JavaScript**.

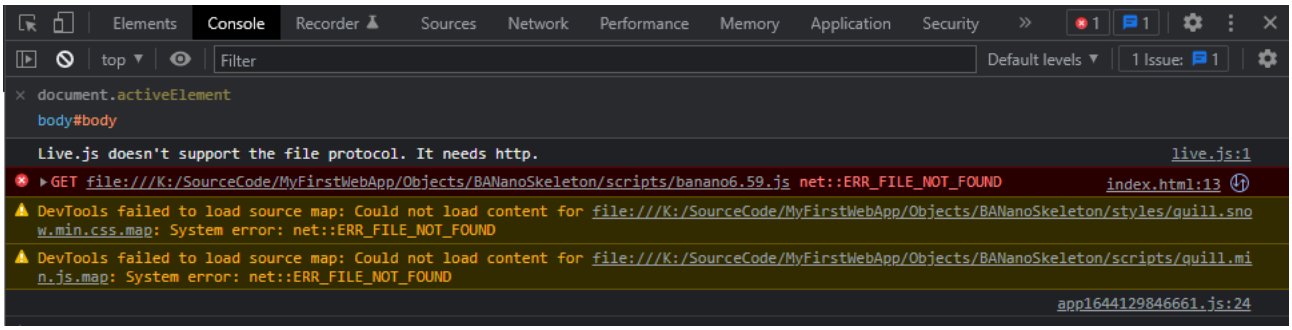
Viewing logged messages

Web developers often log messages to the Console to make sure that their code is working as expected. To log a message, you insert an expression like `Log("Hello, Console!")` into your B4J code. When the browser executes the BANano transpiled JavaScript and sees an expression like that, it knows that it's supposed to log the message to the Console.

Web developers log messages for 2 general reasons:

- Making sure that code is executing in the right order.
- Inspecting the values of variables at a certain moment in time.

This is also the place where you will see warnings and errors in the Transpiled JavaScript.

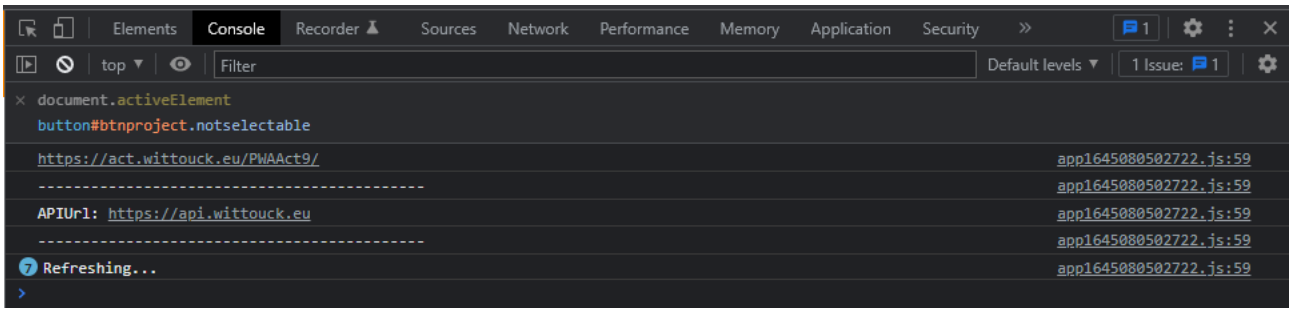


Here we see a warning the Live Code Swapping does not work if we don't run the Web App from a real Web Server and also and error that a the file banano6.59.js was not found.

DevTools also shows some warnings that .map files are missing. These warnings can be ignored.

You can view **Live Expressions** by clicking on the little eye icon  . The Live Expression text box will appear.

For example, we can follow the active element in our Web App by typing `document.activeElement`



When we now use the Web App and we click on something, we can see which element is active. In this case it is `button#btnproject.notselectable`, so we know it is on the SKButton with the id **btnproject** we clicked.

Running JavaScript

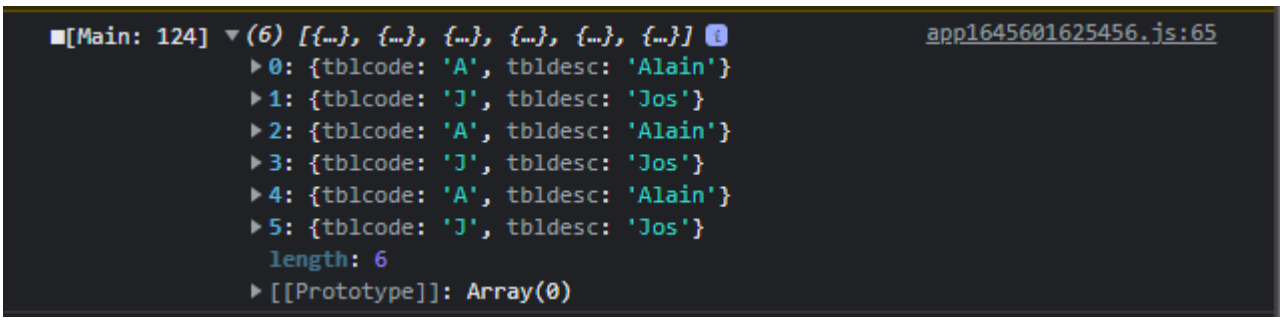
The Console is also a realtime Javascript Evaluation tool. You can run JavaScript in the Console to interact with the page that you're inspecting. For example, you can type:

```
Document.querySelector('h1').textContent = 'My New Document Title';
```

in the console to change the page's title.

If you activated the **'jump' feature snippet**, the logs are formatted in such a way that you can see the original B4J module and line where the log was done. The logged expressions are also **real JavaScript objects**.

For example if you logged a B4J list, the browser will let you inspect the content of that list as an object:

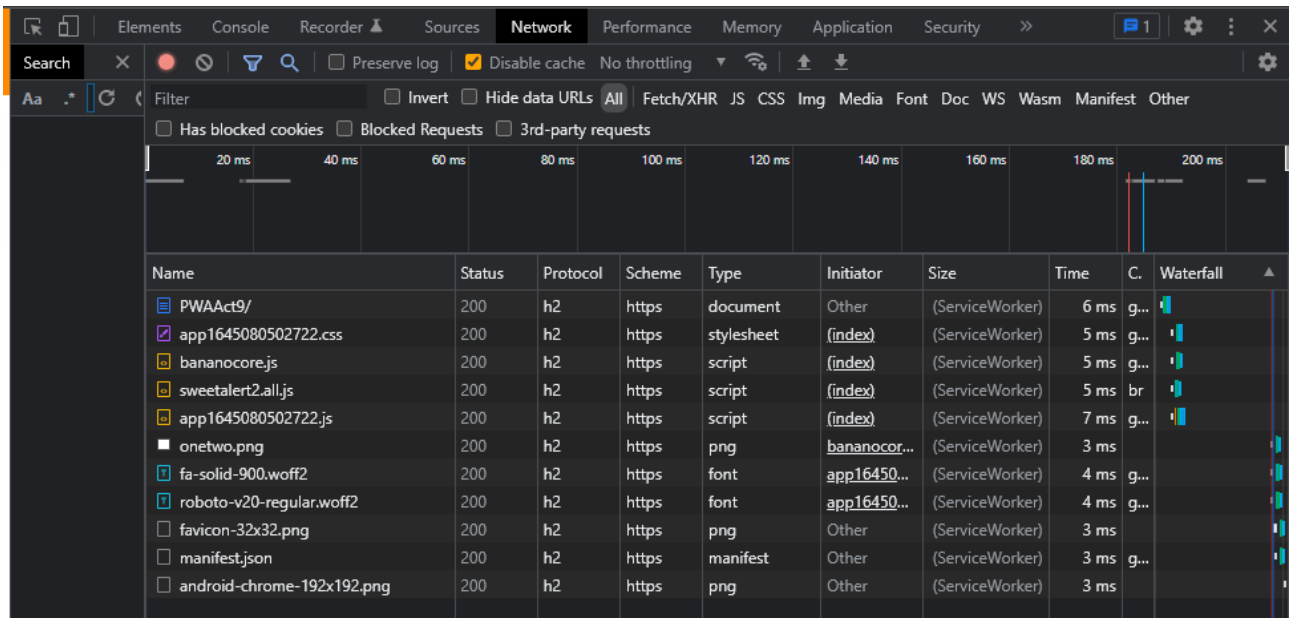


Here is the result of a `Log(results)` where results is a list containing 6 items. You can open up this object by clicking on the little arrow next to [Main: 124] and inspecting its contents.

17.4.2 The Network Tab

In general, use the Network panel when you need to make sure that resources are being downloaded or uploaded as expected. The most common use cases for the Network panel are:

- Making sure that resources are actually being uploaded or downloaded at all.
- Inspecting the properties of an individual resource, such as its HTTP headers, content, size, and so on.



Here we can see some very useful information:

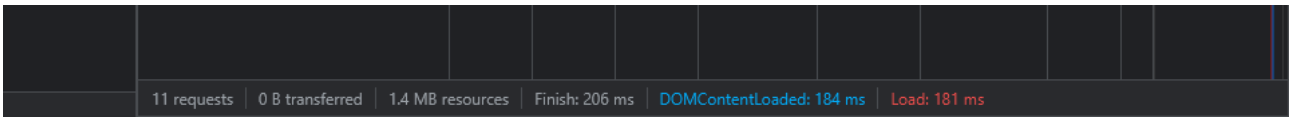
Each row of the **Network Log** represents a resource. By default the resources are listed chronologically. The top resource is usually the main HTML document. The bottom resource is whatever was requested last.

Each column represents information about a resource.

- **Status**. The HTTP response code.
- **Type**. The resource type.
- **Initiator**. What caused a resource to be requested. Clicking a link in the Initiator column takes you to the source code that caused the request.
- **Time**. How long the request took.
- **Waterfall**. A graphical representation of the different stages of the request. Hover over a Waterfall to see a breakdown.

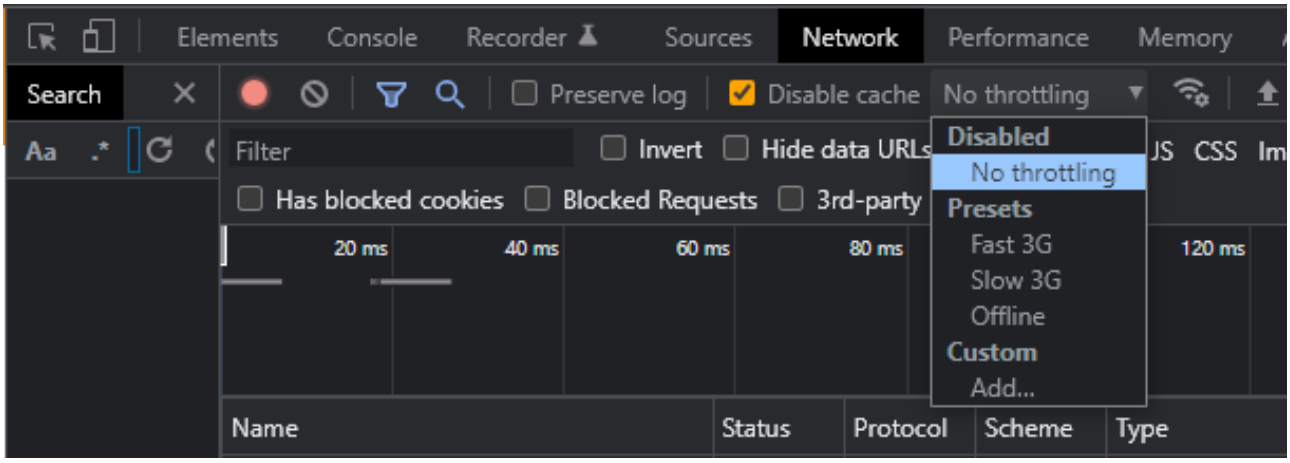
In the example above, we can clearly see that everyting is running on HTTP/2 (h2) and that our PWA is working correctly because all resources where loaded from the Service Worker.

On the bottom we can also so see our Web App is nicely optimized!



Load time is about 200ms (everything above 1 second is considered bad. My thumb of rule is trying to get it under 500ms) and we also made only 11 requests to the server.

This is also the place where we can simulate a slower connection, or even if there is no internet connection.



If we set it to offline, we can test if our PWA will keep working, or how it will behave on a slow 3G connection.

When testing a Web App, I mostly check the **Disable cache** checkbox. This will force the browser to reload all assets on refresh.

If running in PWA mode (with Service Worker), this will not always reload all assets as they will keep being retrieved from the PWA's cache! See the next chapter (The Application Tab) on how to resolve this.

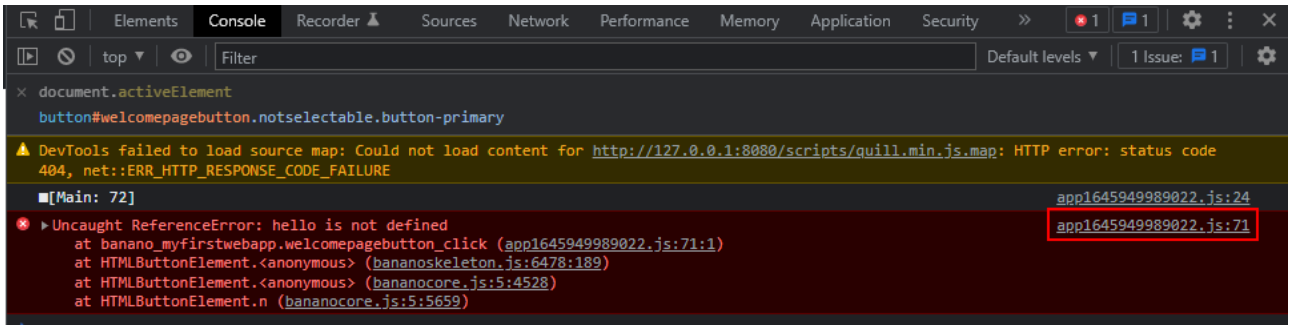
Because of this 'caching' done by the Service Worker, it is easier to run in B4J Debug Mode while developing your Web App. In that case the Service Worker is disabled.

Additional, when running in B4J Debug Mode, BANano will add your original B4J code as comments in the transpiled JavaScript file. This makes it much easier to locate a problem.

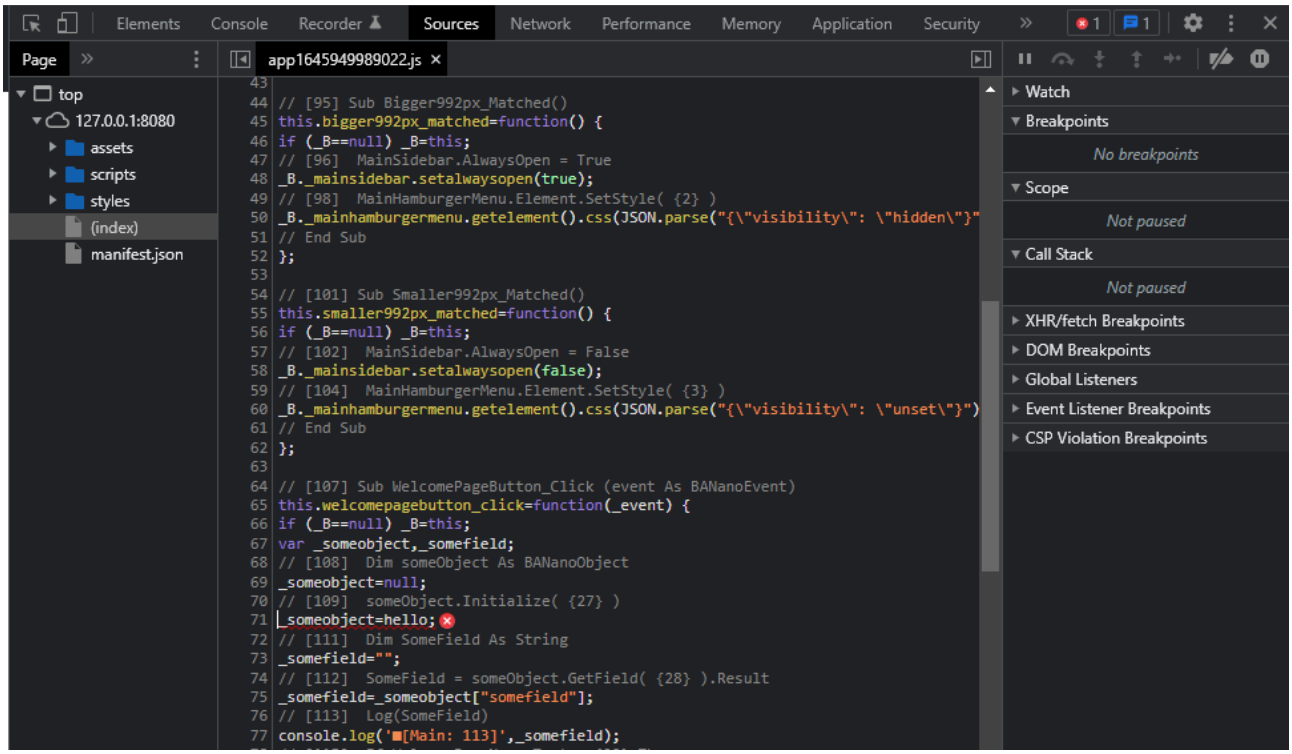
Suppose we have this code in our Web App (where hello does not exist):

```
108 Dim someObject As BANanoObject
109 someObject.Initialize("hello")
110
111 Dim SomeField As String
112 SomeField = someObject.GetField("somefield").Result
113 Log(SomeField)
```

We will get an error in the browser:



By clicking on the link on the right, we will jump to the transpiled JavaScript code:



As you can see, your original B4J code (and its line number) will be show on top of where the error occurred we be visible.

In this case the error is on line 109: `someObject.Initialize( {27} )`

Note: Strings will not appear, but will show something like {27}. This is a transpiler limitation.

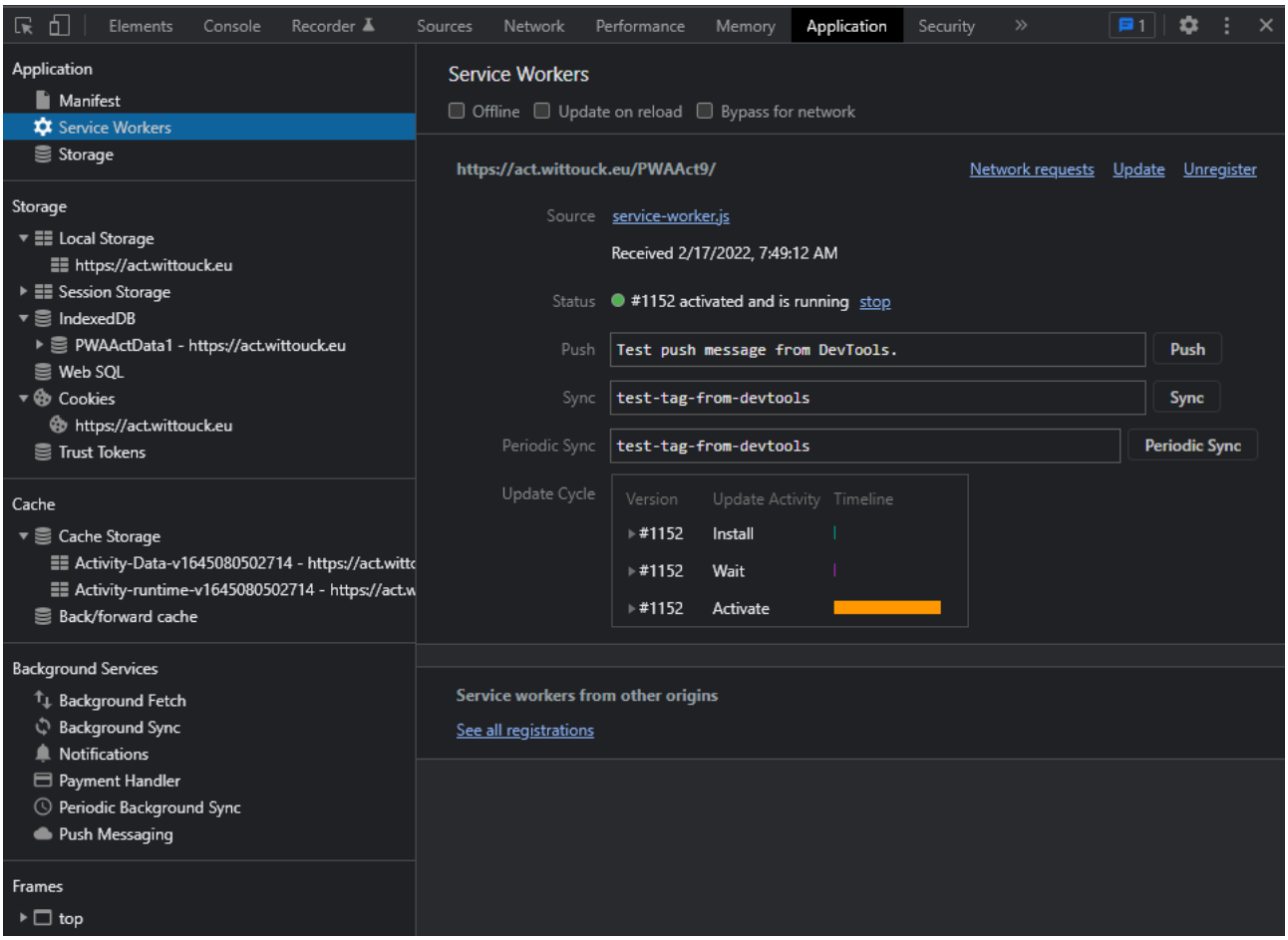
We we go back to our B4J code and check line number 109, we will see the same line causing the error:

```
109 someObject.Initialize("hello")
```

If the TranspilerOption **EnableLiveCodeSwapping = true,** you can now simply make the correction in your code, press Save and reload the page in the browser without having to recompile the whole thing!

17.4.3 The Application Tab

The Application tab is especially useful in PWA modus. It is the place where you inspect Cookies, Storages, Databases, Caches and the Service Worker.



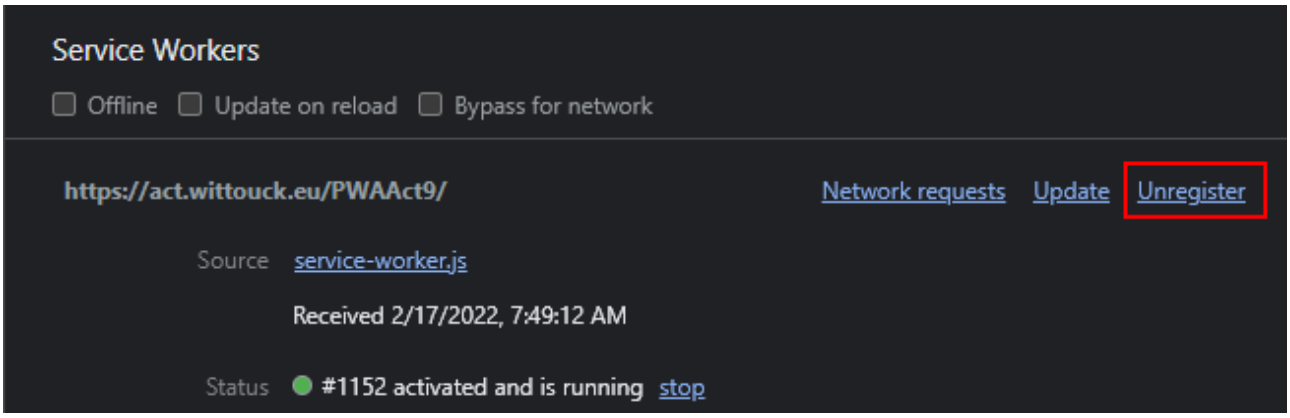
Service Workers

This is the place where you check if your Service worker is installed and running. If the Status is anything else than **green** (activated and is running), something went wrong. You will then have to check the console and network tabs to try to fgind out what.

As soon as there is some error in your code or a file is missing, the PWA Service Worker installation will fail!

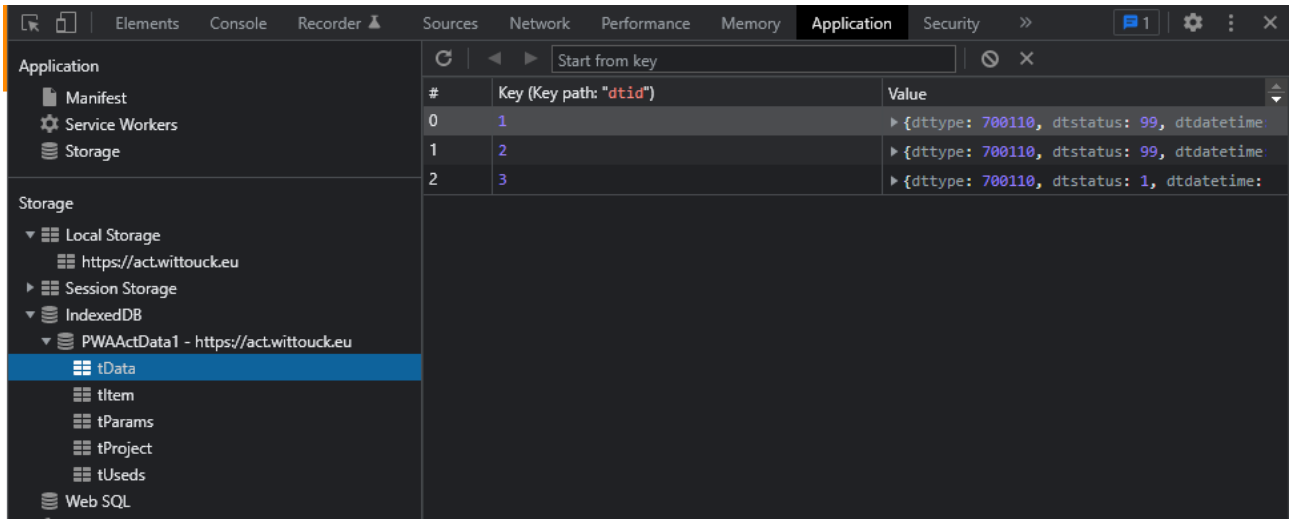
It is not a guarantee that you have already a full PWA (see further), but it is a first indication at least your assets are cached for offline modus.

This is also the place to **reset our Web App's caches**. This is for example needed if we have added new assets (like a JavaScript file or an image). We can do that by **unregistering** the Service Worker.



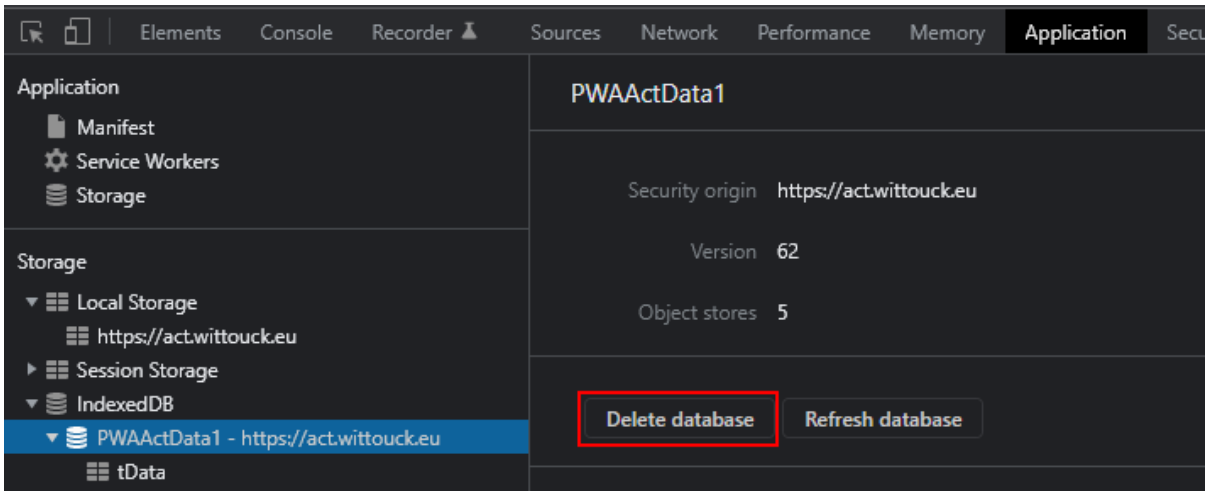
Databases

If we are using BANanoSQL, we can look at the data saved in the local database:

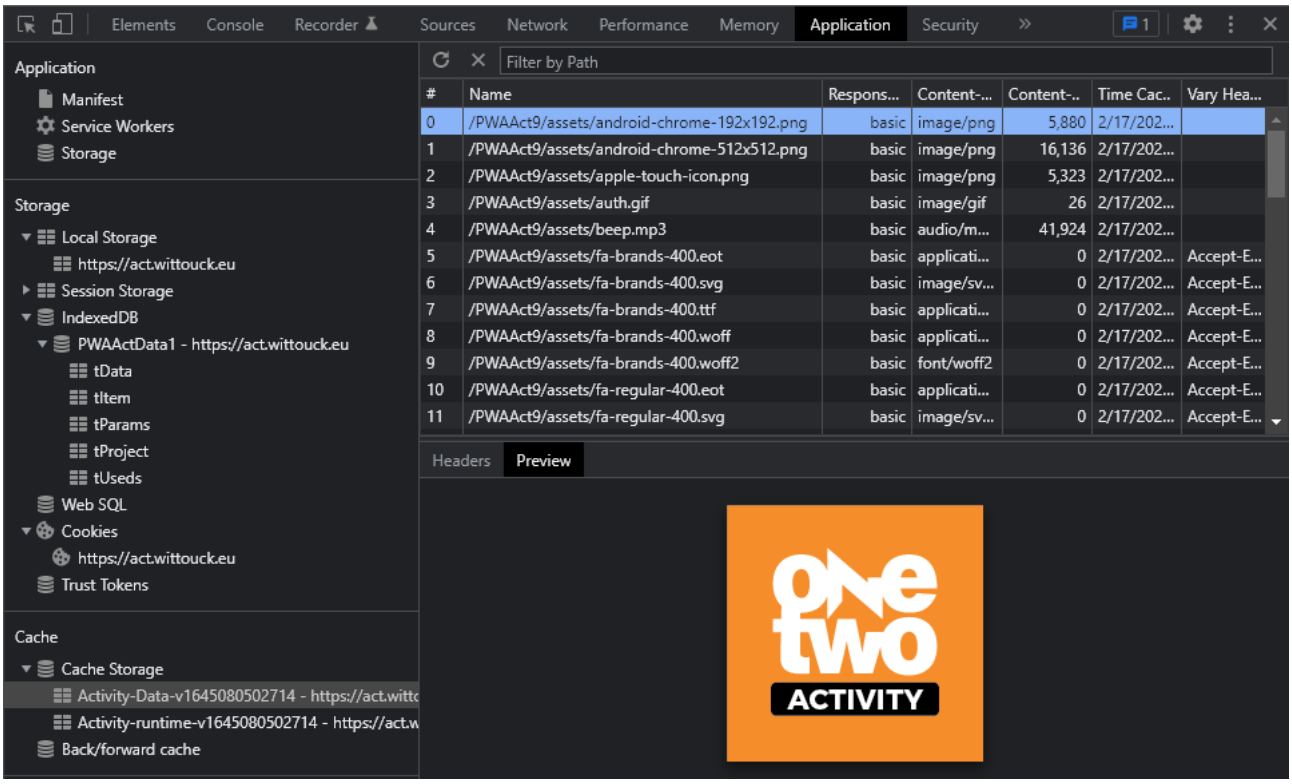


On the left are the tables we created and on the right its contents.

Sometimes, it my be needed to **remove the database**. You can do this by clicking '**Delete database**'.



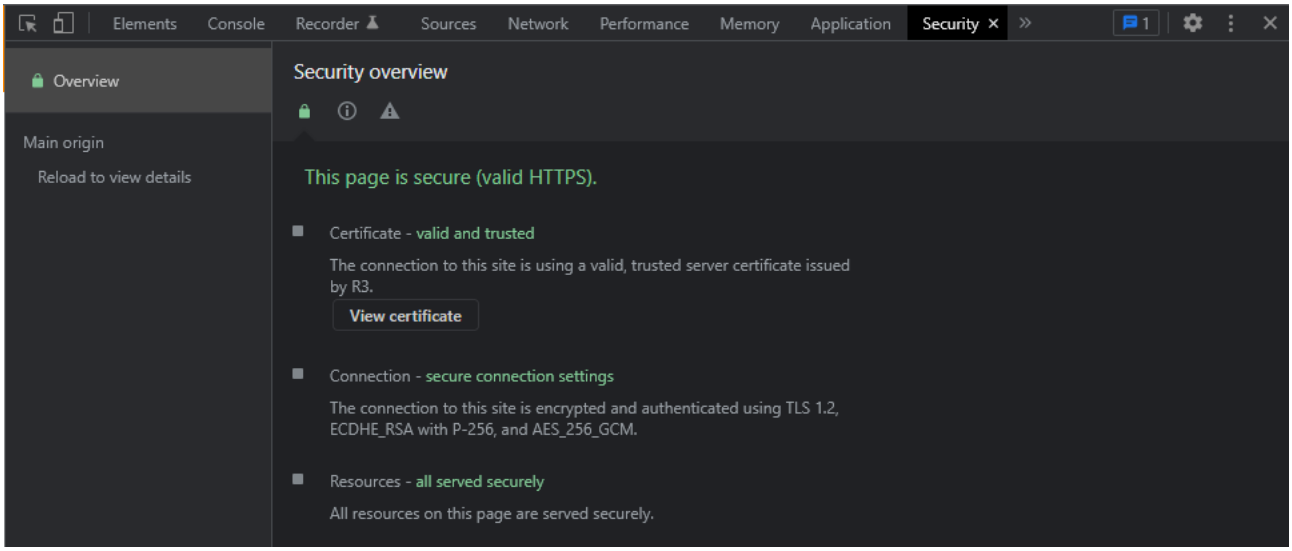
An interesting item is also the **Cache Storage**. Here we can see which files are actually cached by our PWA:



17.4.4 The Security Tab

You can check here if your PWA is running secure (this means your domain has a valid Certificate). **PWA's must run on a valid HTTPS domain!**

Example of a certificate installed correctly:

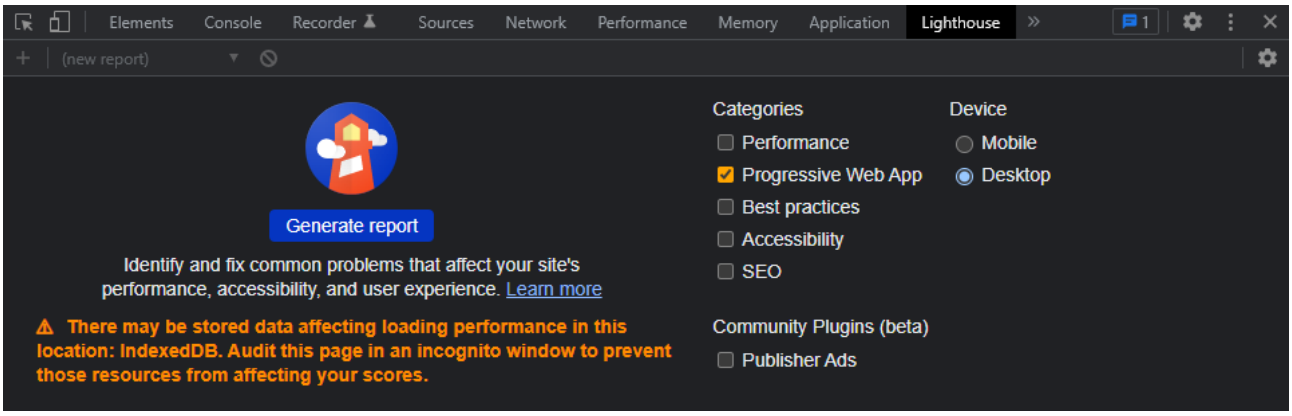


Note: this manual will not go into creating and installing Certifcates. It is strongly advised to talk to an expert in this field (which I am not). For our Web Apps, we have an external company taking care of this.

17.4.5 The Lighthouse Tab

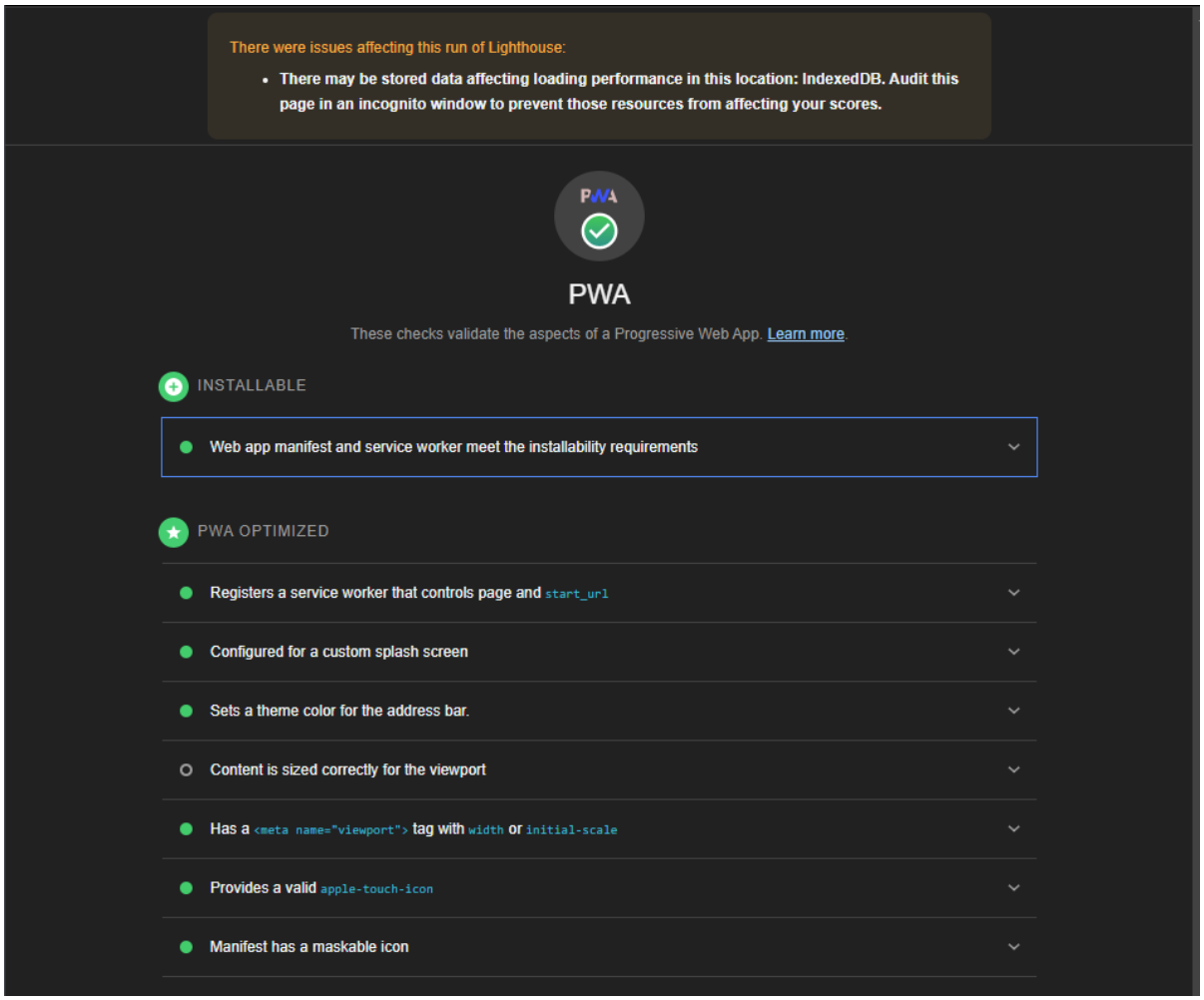
This is the tab that will tell you if you have done everything right to make this Web App a real installable PWA.

Check the **Progressive Web App checkbox** first:



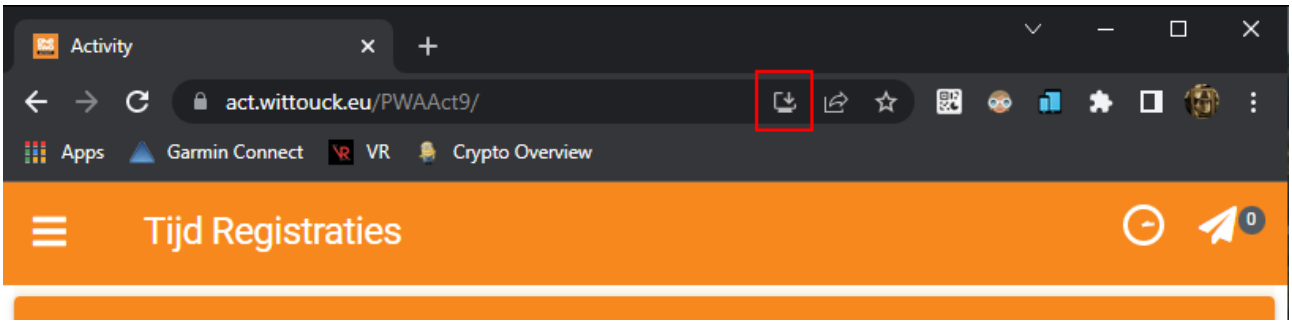
Now click on '**Generate report**'. The Chrome DevTools will start to analyse your PWA and give you a report.

If you have followed everything in this manual, created the correct icons, splash screens and activated the Service Worker, your report will look something like this:



A final check to do:

For your Web App to be a truly installable PWA, a new 'install' icon should appear in the browsers naviagation bar in Chrome:



This allows the user to install the PWA on their Desktop, or as an App icon on their Mobile Device.

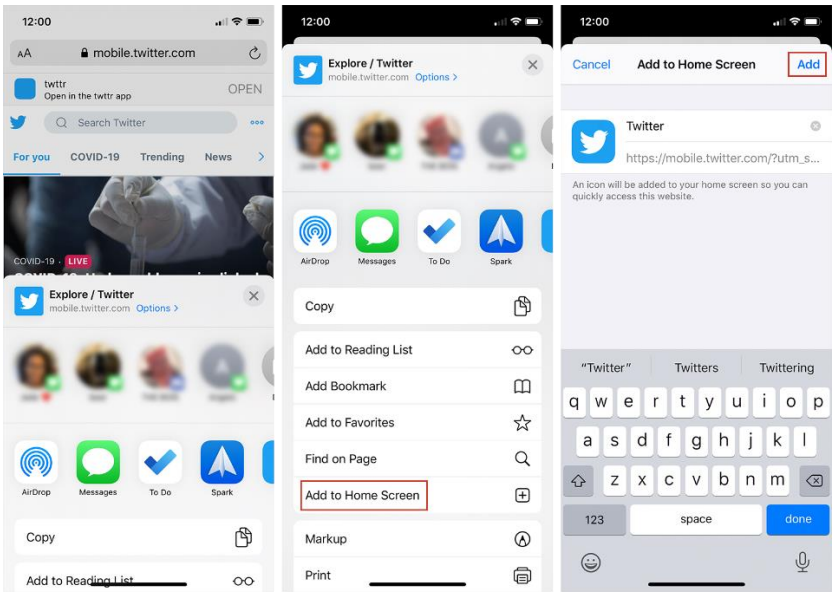
Note:

Not all browsers or OS's allow PWA's, especially Apple is very peculiar about that. Here are the combos that do work:

Windows + Chrome (or Edge) MacOS + Chrome Android + Chrome iOS + Safari

Apple does allow Chrome on Desktop to install a PWA (but not Safari), but on iOS they do not allow Chrome and require Safari! Go figure…

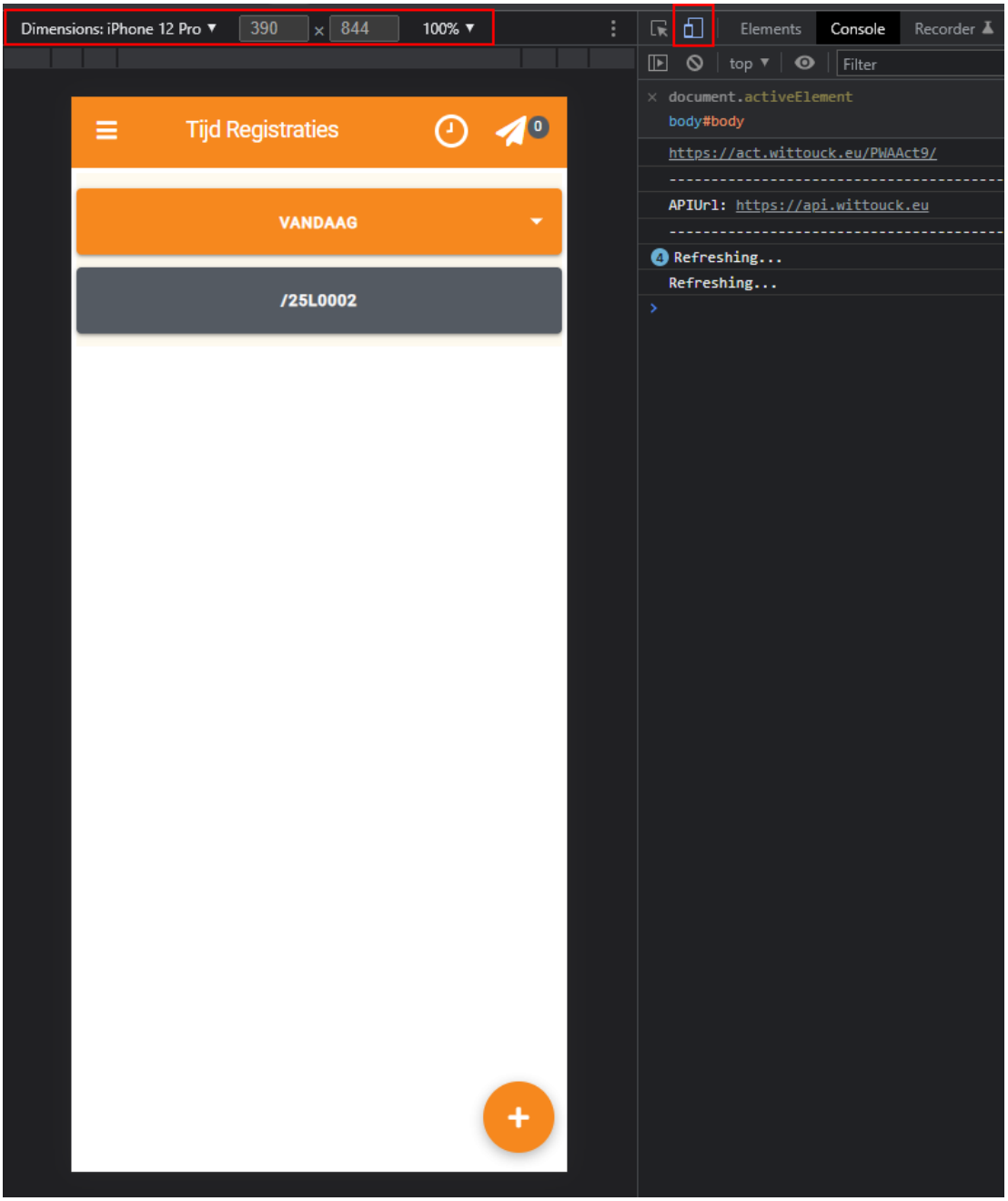
Installing on Safari on iOS does require some steps to go through. Navigate to the website you want to add as a PWA in Safari. Then tap the '**Share**' button, scroll down and tap '**Add to Home Screen**.' Enter the name for the app then tap **'Add'**. The PWA will show up on your home screen like a native iOS app.



17.4.6 Testing your PWA on emulated device sizes

It is always good to test your PWA's on **real devices**, but Chrome Devtools give you some help that can be useful in development.

You can **emulate** all kind of **device sizes** in the Chrome browser.



You can activate this view by clicking on the little  icon and pick (or create) Device Sizes from the dropdown box.

18 Quick Reference

18.1 BANano

Events

- **AssetsLoaded** (pathsNotFound() As String)
- **CallAjaxResult** (Success As Boolean, UniqueID As String, Result As String)
- **CallInlinePHPResult** (Success As Boolean, UniqueID As String, Result As String)
- **CronRun()**
- **EmailSent** (Tag As String, Message As String)
- **IsConnectedResult** (Tag As String, Result As Boolean)
- **MessageFromBackgroundWorker** (WorkerName As String, Tag As String, Value As Object, Error As Object)
- **OffLine()**
- **OnLine()**
- **ParseEvent** (params As Map)
- **Ready()**
- **Resized()**
- **RouterHandle** (url As String, data As Map, params As Map)
- **RouterLeaving** As Boolean
- **ScrollSpyEnter** (element As BANanoElement)
- **ScrollSpyExit** (element As BANanoElement)
- **StopBackgroundWorker()**
- **VisibilityChanged** (visible As Boolean)

Fields

- **ASSETS_FOLDER** As String
- **Header** As BANanoHeader
- **HTML_NAME** As String
- **IsForBANanoServer** As Boolean
- **JAVASCRIPT_NAME** As String
- **MANIFEST_NAME** As String
- **PHP_NAME** As String
- **PHPHost** As String
- **SCRIPTS_FOLDER** As String
- **SERVICEWORKER_NAME** As String
- **ShowDebugEveryLine** As Boolean
- **STYLES_FOLDER** As String
- **TranspilerOptions** As BANanoTranspilerOptions
- **UploadHandlerPath** As String
- **Version** As String
- **Version735** As String

- **VersionName** As String

Functions

- **AddBackgroundWorker** (name As String, className As String)
  Adds a Background Worker. Can only be used in AppStart and with Build(). BuildForServer is not yet supported!

  Name should be unique.

  Tip: make the name lowercase so it is easier to process the results back reported in the BANano_MessageFromBackgroundWorker event.
- **Alert** (text As String)
  Shows an alert box, same as BANano.Msgbox
- **AssetsIsDefined** (bundleName As String) As Boolean
  Check if bundle has already been defined
- **AssetsLoad** (bundleName As String, assets As List) As BANanoPromise
  Loads a bundle of assets and returns a promise.
  If the asset is a CSS or JS file, it must have been Added with the BANano.header.Add...ForLater() methods

```
' in Sub AppStart()
BANano.Header.AddCSSFileForLater("mini-nord.min.css")
...

' in Sub BANano_Ready()
Dim pathsNotFound() as String
If BANano.AssetsIsDefined("Loader") = False then
   Dim prom as BANanoPromise = BANano.AssetsLoad("Loader",
Array("./assets/1.jpg", "./styles/mini-nord.min.css"))
   prom.Then(Null)
     Log("Loader has been loaded!")
   prom.Else(pathsNotFound)
     Log("Doh! Loader has not been loaded completely!")
     For Each path As String In pathsNotFound
        Log(path)
     Next
   prom.End
End if
```

- **AssetsLoadEvent** (module As Object, bundleAndEventName As String, assets As List)
  Loads a bundle of assets and uses the BANano bundleEventName_AssetsLoaded() event

  If the asset is a CSS or JS file, it must have been Added with the BANano.header.Add...ForLater() methods

  The bundle name and eventName is the same.

```
' in Sub AppStart()
BANano.Header.AddCSSFileForLater("mini-nord.min.css")
...

' in Sub BANano_Ready()
BANano.AssetsLoadEvent(Me, "Loader", Array("./assets/1.jpg",
"./styles/mini-nord.min.css"))
...

Sub loader_AssetsLoaded(pathsNotFound() As String)
   If BANano.IsNull(pathsNotFound) = False Then
      Log("Doh! Loader has not been loaded completely!")
      For Each path As String In pathsNotFound
```

```
            Log(path)
         Next
      Else
         Log("Loader has been loaded!")
      End If
   End Sub
```

- **AssetsLoadWait** (bundleName As String, assets As List) As Object
  Loads a bundle of assets and waits until it is loaded. Returns a String array containing the paths that failed.

  If the asset is a CSS or JS file, it must have been Added with the BANano.header.Add...ForLater() methods

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.

```
' in Sub AppStart()
BANano.Header.AddCSSFileForLater("mini-nord.min.css")
...

' in Sub BANano_Ready()
Dim pathsNotFound() as String
If BANano.AssetsIsDefined("Loader") = False then
   pathsNotFound = BANano.AssetsLoadWait("Loader",
Array("./assets/1.jpg", "./styles/mini-nord.min.css"))
   If BANano.IsNull(pathsNotFound) = False Then
      Log("Doh! Loader has not been loaded completely!")
      For Each path As String In pathsNotFound
         Log(path)
      Next
   Else
      Log("Loader has been loaded!")
   End If
End if
```

- **AssetsReset**
  Reset the dependency trackers
- **Atob** (base64String As String) As String
  Decodes a base-64 encoded string
- **Await** (promise As Object) As Object
- **B4JCallSubX** (Component As Object, Sub As String, Arguments As Object()) As Object
  Similar to CallSub, but with unlimited arguments. Arguments must be passed as an Array.

  Can only be used in pure B4J, not in a BANano module!
- **B4JRemoveMeFromCache** (cachedPages As Map, pageID As String)
  Removes a page from the BANanoServer cache. See the BANanoServer.b4xlib: BANanoServer class.
- **B4JScavengeCache** (cachedPages As Map)
  Runs the Cache Scavenger. See the BANanoServer.b4xlib: BANanoCacheScavenger class.
- **B4JUpdateFromCache** (me As B4AClass, cachedPages As Map, ws As WebSocket, ba As BA) As BANanoCacheReport
  Add or Update a SERVERPage to the BANanoServer cache.
- **BigInt** (value As Object) As BANanoObject
  Creates a BigInt (64 bit value). You can only calculate with other BigInt numbers!

Only Applicable for BANano code.

- **BN** (B4JName As String) As String
  Returns the full BANano name that will be used in the Transpiled javascript

  B4JName MUST be a quoted string:

  Will work:

```
log(BANano.BN("myvar")
' returns e.g. _banano_mylib_myvar
```

  Will not work:

```
dim tmp as String = "myvar"
log(BANano.BN(tmp)
```

- **BP**
  BreakPoint.
  Stops the execution of JavaScript. Is ignored if in release mode.
  Use the Developer Tools in the browser to inspect e.g. variable values
- **Btoa** (string As String) As String
  Encodes a string in base-64
- **Build** (outputDir As String)
  Should be called in AppStart() in the Main module.
- **BuildAsB4XLib** (LibraryVersion As String)
  Should be called in AppStart() in the Main module.

  Will Build the transpiled files to your Additional Libraries folder as a B4XLib.

  You do not need to compile your Library with the B4J IDE
- **BuildAsB4XlibForABM** (ABMStaticFilesFolder As String, LibraryVersion As String)
  Should be called in AppStart() in the Main module.

  Will Build the transpiled files to your Additional Libraries folder as a B4XLib for ABMaterial (prefix: ABMBanano).

  You do not need to compile your Library with the B4J IDE

  If ABMStaticFilesFolder (the /www folder) is set, then the assets will be automatically unzipped in this folder.

  e.g.
  `BANano.BuildAsB4XlibForABM("D:\MyProject\MyABMProject\Objects\www","1.15")`
- **BuildAsLibrary**
  DEPRECIATED: You should use the BuildAsB4XLib instead.

  Should be called in AppStart() in the Main module.

  Will Build the transpiled files to your Additional Libraries folder.

Do not forget to compile your Library with the B4J IDE: Project - Compile To Library to generate the .jar and .xml files.

- **BuildForServer** (outputDir As String)
  Should be called in AppStart() in the Main module.
  Builds all the css/html/javascript files from the B4J source code.
- **CallAjax** (url As String, type As String, dataType As String, data As Object, uniqueId As String, withCredentials As Boolean, headers As Map)
  Makes an ajax call. Returns the result of the call to BANano_CallAjaxResult()

  Example:
```
dim headers as Map
headers.initialize
headers.put("Content-Type", "application/json")
BANano.CallAjax("https://reqres.in/api/users?page=2","GET","jsonp",
"","ID0001", false, headers)

Sub BANano_CallAjaxResult(Success As Boolean, UniqueID As String, Result
As String)
   Log(Success)
   Log(UniqueID)
   Log(Result)
End Sub
```
- **CallAjaxWait** (url As String, type As String, dataType As String, data As Object, withCredentials As Boolean, headers As Map) As Object
  Makes an ajax call. Returns the result of the call

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.
- **CallBack** (module As Object, methodName As String, params As List) As Object
  Useful where a library you are wrapping needs a function() {} as parameter.
- **CallBackExtra** (module As Object, methodName As String, params As List, extraParams As List) As Object
  Useful where a library you are wrapping needs a function() {} as parameter.

  The params are the 'event' params like in the normal CallBack.
  The extraParams are extra parameters that the callback method takes, not default to the callback

  Example:

```
Sub GetFileFromServer(FileName As String)
   Dim Response As BANanoFetchResponse
   Dim Blob As BANanoObject

   ' list (GET is default, and we do not need extra options so we pass
Null for the options)
   Dim fetch1 As BANanoFetch
   fetch1.Initialize(FileName, Null)
   fetch1.Then(Response)
      ' we got the response promise, so resolve it to a blob
      fetch1.Return(Response.Blob)
   fetch1.Then(Blob)
      ' we got the blob, read it in a FileReader
      Dim FileReader As BANanoObject
      FileReader.Initialize2("FileReader", Null)
      Dim event As BANanoEvent
```

```
      ' the CallBackExtra, which next to the normal event, also we like
to pass the filename
      FileReader.SetField("onload", BANano.CallBackExtra(Me, "ReadData",
Array(event), Array(FileName)))
      ' get the DataURL
      FileReader.RunMethod("readAsDataURL", Blob)
   fetch1.End ' always end a fetch with this!
End Sub

Sub ReadData(event As BANanoEvent, FileName As String) 'ignore
   ' get the data
   Dim Target As BANanoObject = event.OtherField("target")
   Dim DataUrl As String = Target.GetField("result").Result
   Log(FileName)
   log(DataURL)
End Sub
```

- **CallBackMethod** (module As Object, methodName As String) As Object
  Get the BANano name of a method, to be used in e.g. AddEventListener and RemoveEventListener.
- **CallDebugger**
  DEPRECIATED: Use BANano.BP instead.

  Stops the execution of JavaScript. Is ignored if in release mode.
  *Use the Developer Tools in the browser to inspect e.g. variable values **
- **CallInlinePHP** (methodName As String, methodParams As Map, uniqueId As String)
  Makes a php call to an inline php method. Returns the result of the call to BANano_CallInlinePHPResult()

  Example:

```
BANano.CallInlinePHP("SayHello", CreateMap("Name": "BANano"), "ID0001")

#if PHP
   function SayHello($Name) {
   $ret = Array("answer" => "Hello " .$Name. "!");
   echo json_encode($ret);
}
#End If

Sub BANano_CallInlinePHPResult(Success As Boolean, UniqueID As String,
Result As String)
   Log(Success)
   Log(UniqueID)
   Log(Result)
End Sub
```

- **CallInlinePHPWait** (methodName As String, methodParams As Map) As Object
  Makes a php call to an inline php method. Returns the result of the call.

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.

  Example:

```
Dim res as String = BANano.CallInlinePHPWait("SayHello",
CreateMap("Name": "BANano"))
log(res)

#if PHP
function SayHello($Name) {
   $ret = Array("answer" => "Hello " .$Name. "!");
   echo json_encode($ret);
```

```
}
#End If
```

- **CallSub** (module As Object, methodName As String, params As List) As Object
  Calls a method from another module/class with unlimited parameters
- **CheckInternetConnection** (tag As String)
  Checks if the app can reach the internet

  Will raise the banano IsConnected(Tag as String, Result as boolean) event

  you can then use the tag to see who was the caller and act accordingly
- **CheckInternetConnectionWait** As Boolean
  Checks if the app can reach the internet

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.
- **Concat** (arr As List, otherArray As List) As Object
  The concat method creates a new array by merging (concatenating) existing arrays.
  The concat method does not change the existing arrays. It always returns a new array.
- **Console** As BANanoConsole
  Returns the Console Object as a BANanoConsole.
- **CreateElement** (Tag As String) As BANanoElement
  Creates a BANanoElement, not attached to something
- **CreateObjectUrl** (object As Object) As BANanoURL
  The URL.createObjectURL() static method creates a DOMString containing a URL representing the object given in the parameter.
  The URL lifetime is tied to the document in the window on which it was created. The new object URL represents the specified File object or Blob object.
  To release an object URL, call revokeObjectURL().
- **CronPause** (cronName As String)
  Pauses the Cron job previously started with BANano.CronStart
- **CronResume** (cronName As String)
  Pauses the Cron job previously paused with BANano.CronPause
- **CronStart** (cronName As String, pattern As String, maxRuns As Int)
  Starts a Cron job

  cronName: Name of the Cron job. This cannot be a variable and must be a literal String and cannot contain spaces or special characters!

  Pattern:

  S M H D m d
  * * * * * *

  S: second (0 - 59)
  M: minute (0 - 59)
  H: hour (0 - 23)
  D: day of month (1 - 31)
  m: month (1 - 12)
  d: day of week (0 - 6), 0 to 6 are Sunday to Saturday; 7 is Sunday, the same as 0

maxRuns: maximum number of runs, 0 = indefinite

Ranges:
Ranges are two numbers separated with a "-", and they indicate all numbers from one to the other. e.g. 10-30 would indicate all numbers between and including 10 to 30.

Interval:
A interval is a range and a number separated by "/". The range specifies the group of values, and number specifies every nth value to take from that range.
e.g. 0-10/2 would indicate every 2nd number from 0 to 10, therefore [0,2,4,6,8,10]

Will raise the event: cronName_CronRun() in the calling class

Example:

```
BANano.CronStart("myCron", 15, "0 0 0 * * 2-6") ' at 00:00:00 on every
weekday run, for a total of 15 times, then stop this Cron

Public Sub MyCron_Run()
   ' do something
End Sub

Public Sub btnPause_Click(event as BANanoEvent)
   BANano.CronPause("myCron")
End Sub

Public Sub btnResume_Click(event as BANanoEvent)
   BANano.CronResume("myCron")
End Sub

Public Sub btnStop_Click(event as BANanoEvent)
   BANano.CronStop("myCron")
End Sub
```

- **CronStop** (cronName As String)
  Stops the Cron job previously started with BANano.CronStart
- **DebugTrackLine** (moduleName As String, virtualLineNumber As Int)
  DEPRECIATED: Use BANano.BP instead

  When running in debug mode, you can get some extra debug information by tracking some line.
  In the generated javascript file, some comment lines showing the B4J code have a virtual number prefix: [number]

  You can use this number to track the transpiling of that line.

  Tracks in the B4J log
- **DebugTrackMethod** (moduleName As String, methodName As String)
  DEPRECIATED: Use BANano.TM or BANano.TMC instead

  Will track this method in the Browsers log
- **DecodeURI** (o As Object) As String
  The decodeURI() function is used to decode a URI.
- **DecodeURIComponent** (o As Object) As String
  The decodeURIComponent() function decodes a URI component.

- **DeepClone** (obj As Object) As Object
  Deep Clones an object (e.g. a map)
- **DeepMerge** (obj1 As Object, obj2 As Object) As Object
  Merges two object into one (e.g. two maps into one)
- **DependsOnAsset** (AssetFileNameOrURL As String)
  ONLY works for Builds, using b4xlibs, not for .jar libs.

  This should NOT be used in AppStart, but is mainly for a library where e.g. for a Custom View.
  Should be placed in the Initialize method.

  Assets defined with header.AddCSSFile, header.AddJavascriptFile or other assets (images, fonts) are ONLY loaded
  if the class where the DependsOnAsset is defined, is actually used in the final app.

  NOTE: AssetFileNameOrUrl MUST be a String, NOT a variable! (case sensitive)
- **EmptyLocalStorage**
  Empty the LocalStorage for this domain
- **EmptyLocalStorage2**
  Native Empty JavaScript the LocalStorage for this domain
- **EmptySessionStorage** (key As String)
  Empty the SessionStorage for this domain
- **EmptySessionStorage2** (key As String)
  Native Empty JavaScript the SessionStorage for this domain
- **EncodeURI** (o As Object) As String
  The encodeURI() function is used to encode a URI.
- **EncodeURIComponent** (o As Object) As String
  The encodeURIComponent() function encodes a URI component.
- **Eval** (o As Object) As Object
  The eval() function evaluates or executes an argument.

  If the argument is an expression, eval() evaluates the expression. If the argument is one or more JavaScript statements, eval() executes the statements.
- **Every** (arr As List, callbackMethod As String) As Object
  Every method checks if all array values pass a test.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, array as List)
- **Exists** (target As String) As Boolean
  Checks if an element(s) exists

  Target: and ID (use #), class (use ."), tag etc...
- **Existy** (var As Object) As Boolean
  Test if the object is existy (not null or undefined)
- **ExternalHTMLToHTMLBlocks** (name As String, fullPath As String)
  Can only be used in AppStart()

  Will extract all html blocks where the tag has the class BANANO

  if the class = BANANO, then the body HTML String will not contain these elements.

you can later get their HTML by using the GetHTMLBlock() method.

the Body HTML can be returned with the GetHTMLBody() method.

- **Filter** (arr As List, callbackMethod As String) As Object
  The filter method creates a new array with array elements that passes a test.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, array as List)
- **Finally**
  A try catch in B4J does not have a Finally statement.

  e.g.
```
Dim num As Int = 10
Dim divider As Int = 0
Try
   If divider = 0 Then
      BANano.Throw("You can not divide by 0!")
   Else
      Log(num/divider)
   End If
Catch
   Log(LastException)
   ' will still do the Finally part, but not the "After the Try" log.
   Return
BANano.Finally 'ignore
   Log("Always doing this")
End Try

Log("After the Try")
```
- **Find** (arr As List, callbackMethod As String) As Object
  The find method returns the value of the first array element that passes a test function.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, items as List)
- **FindIndex** (arr As List, callbackMethod As String) As Int
  The findIndex method returns the index of the first array element that passes a test function.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, items as List)
- **ForEach** (arr As List, callbackMethod As String)
  The forEach method calls a function (a callback function) once for each array element.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, array as List)
- **FromBase64** (s As String) As String
  Converts a base 64 string back to a normal string
- **FromBinary** (binaryStr As String) As String
  Reverse conversion from ToBinary
- **FromJson** (object As Object) As Object
  Shortcut function to make an object of JSON
- **GenerateUniqueID** As Object
  Generate a unique ID (64-bit long). This id is sortable in time (bigger = later)

  Can be used both in B4J as in BANano code.

  Note: The BANano version returns a BigInt. You can only calculate with other BigInt numbers!

```
Dim big64 as Long = BANano.BigInt(0)
log(big64)
Dim big64 as BANanoObject = BANano.BigInt(0)
log(big64.ToString(10))
```

- **GenerateUUID** As String
  Generates a UUID. Works both in B4J as in BANano code
- **GeoLocation** As BANanoGeoLocation
  Returns the Navigator GeoLocation Object as a BANanoGeoLocation.
- **GetAllViewsFromLayoutArray** (module As Object, layoutName As String, unqiueIndex As Long) As Map
  Returns a map with all the custom views that are in the layout, with unique index

  The unique index was returned by the LoadLayoutArray() method.

  All keys in the map are lowercased!
- **GetAsset** (url As String) As Object
  returns the object previously loaded with AssetsLoad/AssetsLoadEvent/AssetsLoadWait

  needs the exact url path used in the Load methods.
- **GetCacheStorage2** (url As String) As String
  Native returns the full url (with parameters) if the url is in the cacheStorage RUNTIME
  The url will be searched without parameters.

  URL must be a valid http or https!

  e.g.

  ```
  BANano.GetCacheStorage2("https://mydomain.com/image.png")
  ```

  will return: "https://mydomain.com/image.png?param=Alain"
- **GetCookie** (name As String) As String
  Returns the value of the cookie
- **GetCurrentUrl** As String
  Returns the full URL of the current page
- **GetElement** (target As String) As BANanoElement
  Target: and ID (use #), class (use ."), tag etc...
- **GetElements** (target As String) As BANanoElement()
  Target: and ID (use #), class (use ."), tag etc...
- **GetFileAsArrayBuffer** (fileURL As String, options As BANanoFetchOptions) As BANanoPromise
  Does a Fetch with a resource (e.g. url) and an optional Request options object
  Pass null if no request options need to be set

  Returns a promise holding the ArrayBuffer
- **GetFileAsDataURL** (fileURL As String, options As BANanoFetchOptions) As BANanoPromise
  Does a Fetch with a resource (e.g. url) and an optional Request options object
  Pass null if no request options need to be set

Returns a promise holding the DataURL

```
Dim dataUrl As String
Dim dataUrlProm As BANanoPromise =
BANano.GetFileAsDataURL("./assets/B4X.jpg", Null)
dataUrlProm.Then(dataUrl)
   Log(dataUrl)
dataUrlProm.end
```

- **GetFileAsJSON** (fileURL As String, options As BANanoFetchOptions) As BANanoPromise
  Does a Fetch with a resource (e.g. url) and an optional Request options object
  Pass null if no request options need to be set

  Returns a promise holding the JSON
- **GetFileAsText** (fileURL As String, options As BANanoFetchOptions, encoding As String) As BANanoPromise
  Does a Fetch with a resource (e.g. url) and an optional Request options object
  Pass null if no request options need to be set

  Returns a promise holding the text
- **GetFirebaseToken** As String
  Get the Firebase Messaging token after the user gave permission
- **GetGeoPosition** (options As Object) As BANanoPromise
  Shortcut method to get the users current BANanoGeoPosition

  options: {"enableHighAccuracy": true, "timeout": 5000, maximmumAge: 0}
  Use Null for defaults: false, Infinity, 0

  Usage:

```
Dim POS as BANanoGeoPosition =
BANano.Await(BANano.GetGeoPosition(CreateMap("enableHighAccuracy": true,
"timeout": 5000, "maximumAge": 0))
```

- **GetHTMLBlock** (name As String, origId As String, idPostFix As String) As String
  Returns the HTML String from a removed HTML Block (using the BANANO class) previously added with the ExternalHTMLToHTMLBlocks() method.

  The idPostFix will be added to the original Id as _idPostFix. Is normally a number.
- **getHTMLBody** (name As String) As String
  returns the HTML String with all HTML Blocks having the BANano class removed.
- **GetLocalStorage** (key As String) As Object
  Returns the saved json from the key in the LocalStorage
- **GetLocalStorage2** (key As String) As Object
  Native Get JavaScript LocalStorage
- **GetP** (Class As Object, propName As String) As Object
  Method to get a property from any class
- **GetPageHTML** (B4JClassName As String) As String
  BANanoServer only. Returns the generated page HTML in Release Mode where /Files folder does not exist.
- **GetPageID** As String
  In case you are connected to a BANanoServer, you can here retrieve the Page ID.
  *This Page ID can then be used in the request to a B4J handler class to identify where the class came from. **

- **GetSessionStorage** (key As String) As Object
  Returns the saved json from the key in the SessionStorage
- **GetSessionStorage2** (key As String) As Object
  Native Get JavaScript SessionStorage
- **GetSuffixFromID** (id As String) As Long
  Extracts the last number from an id

  e.g. if the id = mybutton_1 then 1 is returned
- **GetType** (var As Object) As Object
  Will be depreciated in the future. Use the normal B4J GetType() instead!

  to get the type of the object (same as BANano.TypeOf)
- **GetURLParamDefault** (url As String, key As String, Default As Object) As Object
  Gets all the URL parameter. If it does not exist it returns the passed default value.
- **GetURLParams** (url As String) As Map
  Gets all the URL parameters as a map.
- **GetViewFromLayout** (module As Object, id As String) As Object
  Returns the Custom view from a layout loaded with LoadLayout
- **GetViewFromLayoutArray** (module As Object, layoutName As String, id As String, uniqueIndex As Long) As Object
  Returns the Custom view from a layout loaded with LoadLayoutArray, with unique index

  The unique index was returned by the LoadLayoutArray() method.
- **GZipGeneratedWebsite** (minSizeKB As Double)
  DEPRECIATED: Use BANano.TranspilerOptions.GZipGeneratedWebsite instead

  Will GZip your html/css/js/json files on compilation. Set a minimum filesize so small files are not compressed

  This is ONLY useful if you use NGinx with gzip_static set to 'on'
- **History** As BANanoHistory
  Returns the History Object as a BANanoHistory.
- **IIf** (condition As Object, returnTrue As Object, returnFalse As Object) As Object
  Shortcut method to do an 'if then else'
  DEPRECIATED: Use the build-in B4J IIf instead.

  e.g.

  ```
  mRoot = BANano.IIf(Root = "/", "/", "/" & TrimSlashes(Root) & "/")
  ```
- **Import** (moduleName As String) As BANanoPromise
- **ImportRaw** (importStatement As String)
  Literaly takes over the importStatement

  e.g. BANano.ImportRaw("import { export1 , export2 as alias2} from 'module-name'")
- **ImportWait** (moduleName As String) As BANanoObject
- **IndexOf** (arr As List, searchValue As Object) As Int
  The indexOf method searches an array for an element value and returns its position.
  Note: The first item has position 0, the second item has position 1, and so on.

- **IndexOf2** (arr As List, searchValue As Object, start As Int) As Int
  The indexOf2 method searches an array for an element value and returns its position.
  Note: The first item has position 0, the second item has position 1, and so on.

  start: Where to start the search. Negative values will start at the given position counting from the end, and search to the end.
- **INFINITY** As Object
  transpiles as 'Infinity'
- **Initialize** (eventName As String, appShortName As String, appVersion As Long)
  Do not uses spaces in the appShortName!
- **Initialize2** (eventName As String, appShortName As String, appVersion As Long, B4JAdditionalLibrariesPath As String)
  Do not uses spaces in the appShortName!
- **IsArray** (var As Object) As Boolean
  Test if the object is an array
- **IsBoolean** (var As Object) As Boolean
  Test if the object is a boolean
- **IsCapitalized** (var As String) As Boolean
  Test if the string is capitalized
- **IsClass** (var As Object, className As String) As Boolean
  Test if the given value is a certain B4J class
  className must be a String, not a variable.

  e.g. `BANano.IsClass(myObj, "MyB4JClassName")`
- **IsDate** (var As Object) As Boolean
  Test if the object is a date
- **IsDecimal** (var As Object) As Boolean
  Test if the given value is decimal
- **IsDomNode** (var As Object) As Boolean
  Test if the object is a DOM Node
- **IsEmpty** (var As Object) As Boolean
  Test if the object is an empty
- **IsError** (var As Object) As Boolean
  Test if the object is an error
- **IsFinite** (o As Object) As Boolean
  The isFinite() function determines whether a number is a finite, legal number.
- **IsFunction** (var As Object) As Boolean
  Test if the object is a function
- **IsInteger** (var As Object) As Boolean
  Test if the given value is integer
- **IsJson** (var As Object) As Boolean
  Test if the object is a Json object
- **IsList** (var As Object) As Boolean
  Test if the object is a List
- **IsMap** (var As Object) As Boolean
  Test if the object is a Map
- **IsNaN** (o As Object) As Boolean
  The isNaN() function determines whether a value is an illegal number (Not-a-Number).

- **IsNull** (var As Object) As Boolean
  Test if the object is null
- **IsNumber** (var As Object) As Boolean
  Test if the object is a number
- **IsObject** (var As Object) As Boolean
  Test if the object is an object
- **IsPhone** As Boolean
  Returns if the browser is running on a phone
- **IsString** (var As Object) As Boolean
  Test if the object is a string
- **IsTablet** As Boolean
  Returns if the browser is running on a tablet
- **IsUndefined** (var As Object) As Boolean
  Test if the object is undefined
- **Join** (listOfStrings As List, delimiter As String) As String
  Makes a new string from the list where all items are seperated by the delimiter
- **LastIndexOf** (arr As List, searchValue As Object) As Int
  lastIndexOf is the same as indexOf, but returns the position of the last occurrence of the specified element.
  Note: The first item has position 0, the second item has position 1, and so on.
- **LastIndexOf2** (arr As List, searchValue As Object, start As Int) As Int
  lastIndexOf2 is the same as indexOf2, but returns the position of the last occurrence of the specified element.
  Note: The first item has position 0, the second item has position 1, and so on.

  start: Where to start the search. Negative values will start at the given position counting from the end, and search to the beginning.
- **LoadLayout** (target As String, layoutName As String)
  Loads a .bjl layout (using ONLY BANano Custom views).

  You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view).
  Only these properties apply:
  Name = HTML ID
  EventName:
  + All CustomView Properties

  Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

  layoutName must be a string. It cannot be a variable.
- **LoadLayoutAppend** (target As String, layoutName As String)
  Loads a .bjl layout (using ONLY BANano Custom views). Does not empty the Target first.

  You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view).
  Only these properties apply:
  Name = HTML ID
  EventName:
  + All CustomView Properties

Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

layoutName must be a string. It cannot be a variable.

- **LoadLayoutArray** (target As String, layoutName As String, emptyTargetFirst As Boolean) As Long
  Loads a .bjl layout (using ONLY BANano Custom views) as an array. You can NOT Dim one of the views in such a layout in Globals!

  You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view).
  Only these properties apply:
  Name = HTML ID
  EventName:
  + All CustomView Properties

  Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

  layoutName must be a string. It cannot be a variable.

  Return: will return a unique number that has been added as suffix to every view in the layout.
- **Location** As BANanoLocation
  Returns the Location Object as a BANanoLoation.
- **Map** (arr As List, callbackMethod As String) As Object
  The map method creates a new array by performing a function on each array element.
  The map method does not execute the function for array elements without values.
  The map method does not change the original array

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, array as List)
- **MethodVarsToMap** (includeSubName As Boolean) As Map
  Creates a Map of all the parameters past in the current method.
  If includeSubName = true, then the subs name is added with key "subname"
- **Msgbox** (text As String)
  Shows an alert box, same as BANano.Alert
- **NAN** As Object
  transpiles as 'NaN' (Not-a-Number)
- **Navigator** As BANanoNavigator
  Returns the Navigator Object as a BANanoNavigator.
- **New** (B4JClassName As String) As Object
  Creates a new instance of a B4J Class, based on its name

  e.g.

```
Dim P1 As Page1 = BANano.New("Page1")
Log("I'm " & P1.Name)
```

  Page1 class:

```
Sub Class_Globals
Public Name As String = "Page 1"
```

```
End Sub

'Initializes the object. You can add parameters to this method if needed.
Public Sub Initialize

End Sub
```

Note: if you want to use a variable for "Page1" in BANano.New()
make sure you use BANano.BN() when you SET the variable!

see BANano.BN() for more info.

e.g.

```
dim myClass as String = BANano.BN("Page1")

'later in the code you can then use:
Dim P1 As Page1 = BANano.New(myClass)
Log("I'm " & P1.Name)
```

- **ObjectToNumber** (object As Object) As Int
  The + operator used as a unary operator, converts its operand to a number
- **OLDBROWSER** As Boolean
  Is a TRANSPILER CONTROL WORD to write code depending on the browser.
  Older browsers which cannot use ES6 keywords like Wait methods can have an alternative this way.

  MUST be used like this and the if can NOT contain additional conditions!
  Do NOT put this value in a variable.

  e.g.
```
If BANano.OLDBROWSER Then
' code for old browsers
Else
' code for new browsers
End If
```
- **parseFloat** (o As Object) As Double
  The parseFloat() function parses a string and returns a floating point number.
- **parseInt** (o As Object) As Int
  The parseInt() function parses a string and returns an integer.
- **PHPAddHeader** (header As String)
  Add a header in the generated php file

  e.g.

```
BANano.PHPAddHeader("Access-Control-Allow-Origin: *")
```
- **PingServer** (host As String, port As Int, timeoutMs As Int) As Boolean
  Pings a host to check if it is reachable.
- **Pop** (arr As List) As Object
  The pop method removes the last element from an array.
  The pop method returns the value that was "popped out".
- **PromiseAll** (promises As List) As BANanoPromise
  Returns a single Promise that resolves when all of the promises passed as a list have resolved or when the list contains no promises.
  It rejects with the reason of the first promise that rejects. There is no implied ordering in the

execution of the array of Promises given.
On some computers, they may be executed in parallel, or in some sense concurrently, while on others they may be executed serially.

For this reason, there must be no dependency in any Promise on the order of execution of the Promises

- **PromiseAllSettled** (promises As List) As BANanoPromise
  Returns a promise that resolves after all of the given promises have either resolved or rejected, with an array of objects that each describes the outcome of each promise.

  NOT supported in old browsers yet!
- **PromiseAny** (promises As List) As BANanoPromise
  as soon as one of the promises in the iterable fulfils, returns a single promise that resolves with the value from that promise.
  If no promises in the iterable fulfil (if all of the given promises are rejected), then the returned promise is rejected.

  Essentially, this method is the opposite of PromiseAll().
- **PromiseRace** (promises As List) As BANanoPromise
  Returns a promise that fulfils or rejects as soon as one of the promises in the list fulfils or rejects, with the value or reason from that promise.
- **Push** (arr As List, newObj As Object) As Int
  The push method adds a new element to an array (at the end).
  The push method returns the new array length.
- **RaiseEventToABM** (eventName As String, eventParamNames As List, eventParamValues As List, Description As String)
  Method to raise an event to an ABM, maximum two params because B4J only supports this maximum. Use a Map is more are needed.

  The Description will be added to the generated .bas file for ABM as comment.
- **Reduce** (arr As List, callbackMethod As String) As Object
  The reduce method runs a function on each array element to produce (reduce it to) a single value.
  The reduce method works from left-to-right in the array. See also reduceRight.
  The reduce method does not reduce the original array

  callbackMethod should look as: Sub MyCallback(total as int, value as Object, index as int, array as List)

  The total = the initial value / previously returned value
- **ReduceRight** (arr As List, callbackMethod As String) As Object
  The reduceRight method runs a function on each array element to produce (reduce it to) a single value.
  The reduceRight works from right-to-left in the array. See also reduce.
  The reduceRight method does not reduce the original array.

  callbackMethod should look as: Sub MyCallback(total as int, value as Object, index as int, array as List)

The total = the initial value / previously returned value

- **RemoveCacheStorage2** (url As String)
  Native deletes a key from the cacheStorage RUNTIME.
  The url will be searched without parameters.

  URL must be a valid http or https!

  e.g.

  BANano.RemoveCacheStorage2("https://mydomain.com/image.png")
- **RemoveCookie** (name As String, jsonOptions As String)
  Deletes a cookie.

  IMPORTANT! When deleting a cookie and you're not relying on the default attributes, you must pass the exact same path and domain attributes that were used to set the cookie

  example:

  ```
  RemoveCookie("mycookie", "{path: '', domain: 'mydomain.com'}")
  ```
- **RemoveLocalStorage** (key As String)
  Deletes the key from the LocalStorage
- **RemoveLocalStorage2** (key As String)
  Native Deletes JavaScript the key from the LocalStorage
- **RemoveSessionStorage** (key As String)
  Deletes the key from the SessionStorage
- **RemoveSessionStorage2** (key As String)
  Native Deletes JavaScript the key from the SessionStorage
- **ReplaceRegEx** (s As String, regEx As String, replacement As String) As String
  The ReplaceRegEx() method returns a new string with some or all matches of a pattern replaced by a replacement
- **Resolve** (returnPromise As Object)
  This method is called in a ...Wait() method with the signature funcNameWAIT(Resolve as Object)
  see also BANano.WaitFor()
- **ReturnElse** (returnPromise As Object) As Object
  This method is called in a function that is run by promise.CallSub()
  It returns the value of the returnValue in promise.Else(returnValue)

  Use Null is no returnValue is passed

  Is the Reject() method in javascript
- **ReturnThen** (returnPromise As Object) As Object
  This method is called in a function that is run by promise.CallSub()
  It returns the value of the returnValue in promise.Then(returnValue)

  Use Null is no returnValue is passed

  Is the Resolve() method in javascript

- **Reverse** (arr As List)
  The reverse method reverses the elements in an array.
  You can use it to sort an array in descending order.
- **RevokeObjectURL** (url As BANanoURL)
  The URL.revokeObjectURL() static method releases an existing object URL which was previously created by calling URL.createObjectURL().
  Call this method when you've finished using an object URL to let the browser know not to keep the reference to the file any longer.
- **RunBackgroundWorkerMethod** (name As String, tag As String, methodName As String, params As List)
  Runs a method on the instance (name) of the Background Worker previously added with AddBackGroundWorker.

  The Result will be returned to the main thread in the calling Class via the BANano_MessageFromBackgroundWorker event.
- **RunInlineJavascriptMethod** (methodName As String, Params As List) As Object
  Will be depreciated. Use RunJavascriptMethod() instead.
- **RunJavascriptMethod** (methodName As String, Params As List) As Object
  Method to call a Javascript method. The methodName is Case Sensitive!

  For inline javascript, use #If JAVASCRIPT and #End If
  Note: it does not matter where you put inline javascript, all of it is global.

  Example: *
```
Log(BANano.RunInlineJavascriptMethod("evaluate", Array As String("10 * 20")))

#if JAVASCRIPT
function evaluate(s) {
   // so we get back a string
   return '' + eval(s);
}
#End If
```
- **RunThenCatchJavascriptMethod** (methodName As String, Params As List, thenCallBack As Object, catchCallBack As Object)
  Method to call a Javascript method with then/catch callbacks. The methodName is Case Sensitive!

  Use BANano.CallBack to build the callbacks or pass null if not used
- **Screen** As BANanoScreen
  Returns the Screen Object as a BANanoScreen.
- **ScrollSpy** (selector As String, offset As Double, runOnce As Boolean)
  selector: ID (use #), class (use ."), tag etc...
  offset: A value from 0 to 1 of how far from the bottom of the viewport to offset the trigger by.
  0 = top of element crosses bottom of viewport (enters screen from bottom)
  1 = top of element crosses top of viewport (exits screen top).
  runOnce: Whether or not to trigger the callback just once.

  Events:
```
BANano_ScrollSpyEnter(element As BANanoElement)
BANano_ScrollSpyExit(element As BANanoElement)
```

- **SendEmail** (token As String, tag As String, from As String, to As String, subject As String, body As String)
  Sends a simple email

  Use https://www.smtpjs.com/ to encrypt your credentials and generate the token.

  It will raise the _EmailSent() event, returning the tag and a message. Can be OK or an error message.
- **SendFromBackgroundWorker** (tag As String, value As Object, error As Object)
  Directly send something from a Background Worker class to the main Thread via the BANano_MessageFromBackgroundWorker event.
- **SetCacheStorage2** (url As String)
  Native to set url with parameters into the cacheStorage RUNTIME.

  URL must be a valid http or https!

  e.g.

  ```
  BANano.SetCacheStorage2("https://mydomain.com/image.png?param=Alain")
  ```
- **SetCookie** (name As String, value As String, jsonOptions As String)
  jsonOptions: expires, path, domain, secure

  example: expires 7 days from now

  ```
  SetCookie("mycookie", "myvalue", "{expires: 7, path: '', domain:
  'mydomain.com', secure: 'true'}")
  ```
- **SetLocalStorage** (key As String, json As Object)
  To set data into localStorage, you must use the SetLocalStorage API. There are two arguments:
  key for the Object's key, and json for the key value

  example:

  ```
  dim json as JSONGenerator
  json.initialize("{ founded: '1992', formed: 'California', members: ['Tom
  Delonge', 'Mark Hoppus', 'Travis Barker']}")
  SetLocalStorage("someband", json)
  ```
- **SetLocalStorage2** (key As String, value As Object)
  Native Set JavaScript LocalStorage
- **SetMeToNull**
  This would be the same as typing Me = Null in B4J, but this is not possible in the IDE
- **SetP** (Class As Object, propName As String, value As Object)
  Method to set a property from any class
- **SetSessionStorage** (key As String, json As Object)
  To set data into sessionStorage, you must use the SetSessionStorage API. There are two arguments:
  key for the Object's key, and json for the key value

  example:

  ```
  dim json as JSONGenerator
  json.initialize("{ founded: '1992', formed: 'California', members: ['Tom
  ```

```
Delonge', 'Mark Hoppus', 'Travis Barker']}")
SetSessionStorage("someband", json)
```

- **SetSessionStorage2** (key As String, value As Object)
  Native Set JavaScript SessionStorage
- **SetTabNotification** (Number As Int)
  Adds a notification number to the browsers tab. e.g. '(2) My Website'
- **SF** (smartFormattedText As String) As String
  Returns the html conversion of a string using Smart Formatting tags.

  Smart formatting Tags:

  {B}{/B}: Bold
  {I}{/I}: Italic
  {U}{/U}: Underline
  {SUB}{/SUB}: Subscript
  {SUP}{/SUP}: Superscript
  {BR}: Line break
  {WBR}: Word break opportunity
  {NBSP}: Non-breakable space
  {AL}http://...{AT}text{/AL}: Link, opening a new tab
  {AS}http://...{AT}text{/AS}: Link, not opening a new tab
  {C:#RRGGBB}{/C}: Color
  {ST:styles}{/ST}: Add specific styles e.g. {ST:font-size:0.9rem;color:#2B485C}My text in font-size 0.9rem{/ST}
  {IC:#RRGGBB}{/IC}: Icons (if the correct .css or font is loaded) e.g. {IC:#FFFFFF}fa fa-refresh{/IC}
- **Shift** (arr As List) As Object
  The shift method removes the first array element and "shifts" all other elements to a lower index.
  The shift method returns the string that was "shifted out".
- **Sleep** (milliseconds As Int)
  Can only be used in a ...Wait() method.
- **Slice** (arr As List, start As Int) As Object
  The slice method slices out a piece of an array into a new array from the start until the end.
  The slice method creates a new array. It does not remove any elements from the source array.
  It is like the B4J SubString for arrays.
- **Slice2** (arr As List, start As Int, endNotIncluded As Int) As Object
  The slice2 method slices out a piece of an array into a new array.
  The slice3 method creates a new array. It does not remove any elements from the source array.
  It is like the B4J SubString2 for arrays.
- **Some** (arr As List, callbackMethod As String) As Object
  Some method checks if some array values pass a test.

  callbackMethod should look as: Sub MyCallback(value as Object, index as int, array as List)
- **Sort** (arr As List)
  The sort method sorts an array alphabetically
- **Sort2** (arr As List, callbackMethod As String)
  By default, the sort function sorts values as strings.

You can fix this by providing a compare function.
The compare function should return a negative, zero, or positive value, depending on the arguments.

callbackMethod should look as: Sub MyCompare(a as object, b as object) as int

If the result is negative a is sorted before b.
If the result is positive b is sorted before a.
If the result is 0 no changes are done with the sort order of the two values.

- **Splice** (arr As List, start As Int, length As Int, newObjs As List) As Object
  The splice method can be used to add new items to an array.
  The second parameter defines the position where new elements should be added (spliced in).
  The third parameter defines how many elements should be removed.
  The fourth parameter is an array of new elements to be added.
  The splice method returns an array with the deleted items.
- **Split** (pattern As String, text As String) As String()
  Same a B4Js Regex.Split()
- **Spread** (variable As Object) As Object
  Adds the spread operator (three dots) before the variable.

  e.g. BANano.Spread(myVar)

  becomes: ...myVar
- **StartBackgroundWorker** (name As String, params As List)
  Start a previously added with AddBackgroundWorker worker. (Runs the Initialize method with its parameters)
- **StaticFolder** As String
  Gets the static folder name. Is the appShortVersion if not set by BANano.TranspilerOptions.SetStaticFolder.

  Can be called from within a BANano script.
- **StopBackgroundWorker** (name As String, params As List)
  Stops a previously added with AddBackgroundWorker worker. (Runs the BANano_StopBackgroundWorker method with its parameters)
- **SubExists** (module As Object, methodName As String) As Boolean
  Will be depreciated in the future. Use the normal B4J SubExists instead!

  Checks if a method exists in a module.

  Note: for CallBacks, Events or CallSub, this is already done automatically
- **Throw** (error As Object)
  The throw statement allows you to create a custom error.

  Technically you can throw an exception (throw an error).

  e.g. BANano.Throw("This number is not valid")
  BANano.Throw(500)
- **TM**
  Is ignored in Release mode.

Track Method. Use it in a Sub to trace some info after it ran in the browsers log.

- **TMC**
  Is ignored in Release mode.

  Same as TM (TrackMethod), but initially collapsed
- **ToBase64** (s As String) As String
  Converts a string to a base 64 string.
- **ToBinary** (str As String) As String
  Convert a Unicode string to a string in which each 16-bit unit occupies only one byte
- **ToElement** (obj As BANanoObject) As BANanoElement
  Converts the BANanoObject to a BANAnoElement
- **ToJson** (object As Object) As Object
  Shortcut function to make JSON of an object
- **ToObject** (elem As BANanoElement) As BANanoObject
  Converts the BANanoElement to a BANanoObject
- **ToString** (o As Object) As String
  The ToString() function converts the value of an object to a string.

  Note: The ToString() function returns the same value as toString() of the individual objects.
- **TypeOf** (var As Object) As Object
  Will be depreciated in the future. Use the normal B4J GetType() instead!

  to get the type of the object (same as BANano.GetType)
- **UNDEFINED** As Object
  transpiles as 'undefined' (without quotes)
- **Unshift** (arr As List, newObj As Object) As Int
  The unshift method adds a new element to an array (at the beginning), and "unshifts" older elements.
  The unshift method returns the new array length.
- **UploadFile** (file As Object) As BANanoPromise
  Uploads a File object to the BANanoServer UploadHandler
  the following request properties are added:

  pageId - the unique page id
  fileName - the file name

  This calls a POST using the Fetch API
- **UrlBase64ToUint8Array** (base64String As String) As Int()
  Converts a Base64 String to an unsigned int array
- **WaitFor** (result As Object, module As Object, methodName As String, params As List)
  DEPRECIATED: using the normal BANano.Await() wrapper with promises is easier to use.

  Waits for the methodName to be resolved
  This method MUST have this signature (name must end with 'Wait' and param name must be Resolve as Object!):

  e.g. `methodNameWAIT(Resolve as Object, otherParam as int, ...)`

Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.

- **Window** As BANanoWindow
  Returns the Window Object as a BANanoWindow.

Properties

- **BROWSERPrefix** As String [write only]
  Internally used by the BANanoServer lib
- **ExternalTestConnectionServer** As String [write only]
  DEPRECIATED: Use BANano.TranspilerOptions.ExternalTestConnectionServer instead

  By default the connection to the internet is tested by checking if donotdelete.gif can be retrieved
  from the assets folder where the app is hosted.

  However, if you do not put it on a host (e.g. just by opening the .html file from disk),
  You can upload the donotdelete.gif to some host on the internet to test for an internet connection.
- **Initialbody** As String [write only]
  Can ONLY be used in AppStart(). It writes the string directly as the innerHTML of the body tag.
- **MinifyOnline** As Boolean [write only]
  DEPRECIATED: Use BANano.TranspilerOptions.MinifyOnline instead

  Using the API of:

  https://javascript-minifier.com, the generated Javascript file will be minified
  https://cssminifier.com, the CSS files will be minified
- **SHAREDPrefix** As String [write only]
  Internally used by the BANanoServer lib
- **UseServiceWorker** As Boolean [write only]
  DEPRECIATED: Use BANano.TranspilerOptions.UseServiceWorker instead

  Can ONLY be used in AppStart(). Set this param to false if you do not want to use a ServiceWorker
  Default true

18.2 BANanoCacheReport

Properties

- **BANPageID** As String [read only]
  Returns the current unique page ID

  Only useful with a B4J server, using the BANano.B4JUpdateFromCache() method
- **BANSessionID** As String [read only]
  Returns the current session ID

  Only useful with a B4J server, using the BANano.B4JUpdateFromCache() method
- **ComesFromCache** As Boolean [read only]
  Returns if BANano could recover the B4J class from its cache

  Only useful with a B4J server, using the BANano.B4JUpdateFromCache() method
- **IsReconnected** As Boolean [read only]
  Returns if the WebSocket was reconnected

  Only useful with a B4J server, using the BANano.B4JUpdateFromCache() method

18.3 BANanoConsole

Functions

- **Assert** (expression As Object, message As Object)
  Write a message to the console, only if the first argument is false:
- **Clear**
  Clear all messages in the console
- **Count**
  Writes to the console the number of times that particular Count() is called.
- **Error** (message As Object)
  This method writes an error message to the console.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Group** (label As Object)
  Creates a new inline group in the console. These indents following console messages by an additional level, until console.groupEnd() is called
- **GroupCollapsed** (label As Object)
  Creates a new inline group in the console. However, the new group is created collapsed. The user will need to use the disclosure button to expand it
- **GroupEnd**
  Exits the current inline group in the console
- **Info** (message As Object)
  This method writes an info message to the console.
- **Log** (message As Object)
  This method writes a message to the console.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **Table** (tableData As Object, tablecolumns As Object)
  The console.table() method writes a table in the console view.
- **Time** (label As String)
  Starts a timer in the console view.

- **TimeEnd** (label As String)
  Ends a timer, and writes the result in the console view.
- **Trace**
  Outputs a stack trace to the console
- **Warn** (message As Object)
  This method writes an warning message to the console.

18.4 BANanoElement

Functions

- **AddClass** (Class As String) As BANanoElement
  Add html class(es) to all of the matched elements
- **AddEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)
  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.

  true - The event handler is executed in the capturing phase
  false - The event handler is executed in the bubbling phase
- **AddEventListenerOpen** (eventName As String, params As Object)
  All the code between AddEventListenerOpen and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpen MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpen("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = function(aEvt) {
...
};
```

- **AddEventListenerOpenAsync** (eventName As String, params As Object)
  All the code between AddEventListenerOpenAsync and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpenAsync MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpenAsync("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = async function(aEvt) {
```

```
...
};
```

- **After** (htmlOrElement As Object) As BANanoElement
  Add some html as a sibling after each of the matched elements
- **Append** (htmlOrElement As Object) As BANanoElement
  Add some html as a child at the end of each of the matched elements
- **Attributes** As List
  Returns a list with the Attributes
- **Before** (htmlOrElement As Object) As BANanoElement
  Add some html as a sibling before each of the matched elements
- **Children** (filter As String) As BANanoElement()
  Get the direct children of all of the nodes with a filter
- **ClientHeight** As Double
  Returns the height of an element, including padding
- **ClientLeft** As Double
  Returns the width of the left border of an element
- **ClientTop** As Double
  Returns the width of the top border of an element
- **ClientWidth** As Double
  Returns the width of an element, including padding
- **CloseEventListener**
  Closes an AddEventListenerOpen or AddEventListenerOpenAsync method
- **Closest** (filter As String) As BANanoElement()
  Find the first ancestor that matches the selector for each node
- **EachEnd**
  Close a EachStart
- **EachStart** (element As BANanoElement, index As Int)
  Loop through all of the elements and execute the code between EachStart and EachEnd for each element.

  e.g.

```
Dim element as BANanoElement
Dim index as Int

allelements.EachStart(element, index)
   log(element)
allelements.EachEnd
```

- **Empty** As BANanoElement
  Remove all child nodes of the matched elements.
- **Filter** (filter As String) As BANanoElement()
  Get the direct children of the nodes with a filter
- **Find** (filter As String) As BANanoElement()
  Get all of the descendants of the nodes with a filter
- **First** As BANanoElement
  Retrieve the first of the matched nodes
- **Get** (target As Object) As BANanoElement
  Gets the BANanoElement with the given ID (use '#')
- **GetAttr** (name As String) As String
  Handle attributes for the matched elements

- **GetChecked** As Boolean
  Retrieve the checked value of matched elements
- **GetData** (name As String) As String
  Handle data- attributes for the matched elements*
- **GetField** (field As String) As BANanoObject
  Gets a field value

  Shortcut method for `.ToObject.GetField()`
- **GetHTML** As String
  Retrieve the html of the elements
- **GetScrollLeft** As Double
  Sets or returns the number of pixels an element's content is scrolled horizontally
- **GetScrollTop** As Double
  Sets or returns the number of pixels an element's content is scrolled vertically
- **GetStyle** (property As String) As String
  Returns the property value
- **GetText** As String
  Retrieve the text content of matched elements
- **GetValue** As String
  Retrieve the value content of matched elements
- **HandleEvents** (events As String, module As Object,
  method As String) As BANanoElement
  module: the module or class where the method is defined
  method: the method you want to call

  This function is the same as on(), but it executes the e.preventDefault()

  The method MUST be defined like this:

```
sub methodName(event As BANanoEvent)
   log(event.ID)
end sub
```

- **HasAttr** (name As String) As Boolean
  Find if any of the matched elements contains the attribute passed
- **HasClass** (Class As String) As Boolean
  Find if any of the matched elements contains the class passed
- **Initialize** (target As Object)
  Target: ID (use #), class (use ."), tag etc...
- **IsInitialized** As Boolean
- **Last** As BANanoElement
  Retrieve the last of the matched nodes
- **Length** As Int
  You can check how many elements are matched with .Length
- **LoadLayout** (layoutName As String)
  Loads a .bjl layout (using ONLY BANano Custom views).

  You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view). Only these properties apply:
  Name = HTML ID

EventName:
+ All CustomView Properties

Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

layoutName must be a string. It cannot be a variable.

Note: Must be done directly on a BANanoElement, not via a method or chaining

e.g.
Will Work:

```
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")
pageHolder.LoadLayout("MainLayout")

Dim UserTab As BANanoElement = SKTabs1.GetTabContents(0) ' returns a
BANanoElement
UserTab.LoadLayout("Users")
```

Will NOT work:

```
body.Append(html).Get("#pageHolder").LoadLayout("MainLayout")
SKTabs1.GetTabContents(0).LoadLayout("Users")
```

- **LoadLayoutAppend** (layoutName As String)
Loads a .bjl layout (using ONLY BANano Custom views). Does not empty the Target first.

You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view). Only these properties apply:
Name = HTML ID
EventName:
+ All CustomView Properties

Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

layoutName must be a string. It cannot be a variable.

Note: Must be done directly on a BANanoElement, not via a method or chaining

e.g.
Will Work:

```
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")
pageHolder.LoadLayoutAppend("MainLayout")

Dim UserTab As BANanoElement = SKTabs1.GetTabContents(0) ' returns a
BANanoElement
UserTab.LoadLayoutAppend("Users")
```

Will NOT work:

```
body.Append(html).Get("#pageHolder").LoadLayoutAppend("MainLayout")
SKTabs1.GetTabContents(0).LoadLayoutAppend("Users")
```

- **LoadLayoutArray** (layoutName As String, emptyTargetFirst As Boolean) As Long
  Loads a .bjl layout (using ONLY BANano Custom views) as an array. You can NOT Dim one of the views in such a layout in Globals!

  You add/set BANano Custom Views in the Abstract Designer (Add View -> Custom view). Only these properties apply:
  Name = HTML ID
  EventName:
  + All CustomView Properties

  Note: As B4J custom components currently can't set the Parent property except to main, an own algorithm tries to determine it.

  layoutName must be a string. It cannot be a variable.

  Note: Must be done directly on a BANanoElement, not via a method or chaining

  e.g.
  Will Work:

```
Dim pageHolder As BANanoElement = body.Append(html).Get("#pageHolder")
pageHolder.LoadLayout("MainLayout")

Dim UserTab As BANanoElement = SKTabs1.GetTabContents(0) ' returns a
BANanoElement
UserTab.LoadLayout("Users")
```

  Will NOT work:

```
body.Append(html).Get("#pageHolder").LoadLayout("MainLayout")
SKTabs1.GetTabContents(0).LoadLayout("Users")
```

  Return: will return a unique number that has been added as suffix to every view in the layout.
- **Name** As String
- **Not** (filter As String) As BANanoElement()
  Remove known nodes from nodes
- **Off** (events As String) As BANanoElement
  Remove event handler from matched nodes
- **OffsetHeight** As Double
  Returns the height of an element, including padding, border and scrollbar
- **OffsetLeft** As Double
  Returns the horizontal offset position of an element
- **OffsetTop** As Double
  Returns the vertical offset position of an element
- **OffsetWidth** As Double
  Returns the width of an element, including padding, border and scrollbar
- **On** (events As String, module As Object, method As String) As BANanoElement
  module: the module or class where the method is defined
  method: the method you want to call

The method MUST be defined like this:

```
sub methodName(event As BANanoEvent)
   log(event.ID)
end sub
```

- **Parent** (filter As String) As BANanoElement()
  Retrieve each parent of the matched nodes, optionally filtered by a selector
- **Prepend** (htmlOrElement As Object) As BANanoElement
  Add some html as a child at the beginning of each of the matched elements
- **Remove** As BANanoElement
  Removes the matched elements.
- **RemoveAttr** (name As String) As BANanoElement
  Handle removing attributes for the matched elements
- **RemoveClass** (Class As String) As BANanoElement
  Remove html class(es) to all of the matched elements.
- **RemoveEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  Removes event handlers that have been attached with the addEventListener() method

  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies the event phase to remove the event handler from.

  true - Removes the event handler from the capturing phase
  false - Removes the event handler from the bubbling phase
- **Render** (htmlTemplate As String, jsonData As String) As BANanoElement
  Sets the innerHTML of the target using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.
- **RenderAfter** (htmlTemplate As String, jsonData As String) As BANanoElement
  Add some html as a sibling after each of the matched elements using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.
- **RenderAppend** (htmlTemplate As String, jsonData As String) As BANanoElement
  Add some html as a child at the end of each of the matched elements using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.
- **RenderBefore** (htmlTemplate As String, jsonData As String) As BANanoElement
  Add some html as a sibling before each of the matched elements using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.
- **RenderPrepend** (htmlTemplate As String) As BANanoElement
  Add some html as a child at the beginning of each of the matched elements using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.

- **RenderReplace** (htmlTemplate As String, jsonData As String) As BANanoElement
  Replaces the target using the htmlTemplate and the provided jsonData

  The htmlTemplate is a Mustache template.
- **Replace** (htmlOrElement As Object) As BANanoElement
  Replace the matched elements with the passed elements
- **Result** As Object
  Gets the result

  Shortcut method for .ToObject.Result()
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **Scroll**
  Scroll to the first matched element, smoothly if supported.
- **ScrollHeight** As Double
  Returns the entire height of an element, including padding
- **ScrollWidth** As Double
  Returns the entire width of an element, including padding
- **SetAttr** (name As String, value As String) As BANanoElement
  Handle attributes for the matched elements
- **SetChecked** (checked As Boolean) As BANanoElement
  Set the checked value of matched elements
- **SetData** (name As String, value As String) As BANanoElement
  Handle data- attributes for the matched elements*
- **SetField** (field As String, value As Object)
  Sets a field value

  Shortcut method for .ToObject.SetField()
- **SetHTML** (html As String) As BANanoElement
  Set the html of the elements.

  Note: it is better to use the Render method, as it is much more powerful
- **SetScrollLeft** (x As Double)
  Sets or returns the number of pixels an element's content is scrolled horizontally
- **SetScrollTop** (y As Double)
  Sets or returns the number of pixels an element's content is scrolled vertically

- **SetStyle** (jsonString As String)
  Sets the style of the target BANanoElement.

  example:

```
' must be valid Json!
BANano.GetElement("#someid").SetStyle($"{ "width": "200px", "height":
"200px", "background": "green", "border-radius": "5px" }"$)
```

- **SetText** (text As String) As BANanoElement
  Set the text content of matched elements
- **SetValue** (text As String) As BANanoElement
  Set the value content of matched elements
- **Siblings** (filter As String) As BANanoElement()
  Get the siblings of all of the nodes with a filter
- **ToggleClass** (Class As String) As BANanoElement
  Toggles the class of matched elements
- **ToObject** As BANanoObject
  Returns a BANanoObject: nodes[0], the native html object
- **Trigger** (event As String, params As String()) As BANanoElement
  Triggers an event.

  Params: MUST be defined as Array("", 0, ...)
  Will be returned in the event.detail property
- **Wrap** (html As String) As BANanoElement
  Wraps the matched element(s) with the passed argument. It accepts an html tag like .wrap(' ')

18.5 BANanoEvent

Fields

- **ReturnValue** As Boolean

Functions

- **AltKey** As Boolean
- **Buttons** As Int
- **Char** As String
- **CharCode** As String
- **ClientX** As Int
- **ClientY** As Int
- **CtrlKey** As Boolean
- **CurrentTarget** As Object
- **Data** As Object
- **DeltaMode** As Int
- **DeltaX** As Int
- **DeltaY** As Int
- **DeltaZ** As Int
- **Detail** As Object()
- **ID** As String
- **Key** As String
- **KeyCode** As String
- **MetaKey** As Boolean
- **OffsetX** As Int
- **OffsetY** As Int
- **OtherField** (field As String) As BANanoObject
  Gets another field value
- **PageX** As Int
- **PageY** As Int
- **PreventDefault** As Object
- **Reason** As Object
- **RelatedTarget** As Object
- **ScreenX** As Int
- **ScreenY** As Int
- **ShiftKey** As Boolean
- **StopPropagation** As Object
- **Target** As Object
- **TimeStamp** As Int
- **Type** As String
- **Value** As Object

Properties

- **Tag** As Object

18.6 BANanoFetch

Functions

- **Else** (error As Object)
  Continues here after the Fetch if an error occurs.
- **ElseWait** (returnValue As Object)
  Is the same as .Else, except the function will be async.

  This can be used if the code in the .ElseWait clause contains ...Wait functions or Sleep
- **End**
  Terminates the promise Then/Else/Finally
- **Finally**
  Will always run at the end
- **FinallyWait**
  Is the same as .Finally, except the function will be async.

  This can be used if the code in the .FinallyWait clause contains ...Wait functions or Sleep
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Initialize** (resource As String, init As BANanoFetchOptions) As BANanoPromise
  Does a Fetch with a resource (e.g. url) and an optional Request options object
  Pass null if no request options need to be set
- **Result** As Object
  Gets the result
- **Return** (data As Object)
  Returns something in a then part. Can be passed on the next then.
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **Then** (response As Object)
  Continues here after the Fetch or another then/else
- **ThenWait** (returnValue As Object)
  Is the same as .Then, except the function will be async.

This can be used if the code in the .ThenWait clause contains ...Wait functions or Sleep

18.7 BANanoFetchOptions

Functions

- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Initialize**
  Creates a new options object.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value

Properties

- **Body** As Object
  Any body that you want to add to your request: this can be a Blob, BufferSource, FormData,
  URLSearchParams, or USVString object.

  Note that a request using the GET or HEAD method cannot have a body.
- **Cache** As String
  The cache mode you want to use for the request.
- **Credentials** As Object
  The request credentials you want to use for the request: omit, same-origin, or include.
  To automatically send cookies for the current domain, this option must be provided.

  Starting with Chrome 50, this property also takes a FederatedCredential instance or a PasswordCredential instance.
- **Headers** As Object
  Any headers you want to add to your request, contained within a Headers object or an object literal with ByteString values.
  Note that some names are forbidden.

- **Integrity** As String
  Contains the subresource integrity value of the request (e.g., sha256-BpfBw7ivV8q2jLiT13fxDYAe2tJllusRSZ273h2nFSE=).
- **KeepAlive** As Boolean
  The keepalive option can be used to allow the request to outlive the page.
- **Method** As String
  The request method, e.g., GET, POST.

  Default: GET
- **Mode** As String
  The mode you want to use for the request, e.g., cors, no-cors, or same-origin.
- **Redirect** As String
  The redirect mode to use: follow (automatically follow redirects), error (abort with an error if a redirect occurs), or manual (handle redirects manually). In Chrome the default is follow (before Chrome 47 it defaulted to manual).
- **Referrer** As String
  A USVString specifying no-referrer, client, or a URL. The default is client.
- **ReferrerPolicy** As String
  Specifies the value of the referred HTTP header. May be one of no-referrer, no-referrer-when-downgrade, origin, origin-when-cross-origin, unsafe-url.
- **Signal** As Object
  An AbortSignal object instance; allows you to communicate with a fetch request and abort it if desired via an AbortController.

18.8 BANanoFetchResponse

Functions

- **ArrayBuffer** As Object
  Takes a Response stream and reads it to completion. It returns a promise that resolves with an ArrayBuffer.
- **Blob** As Object
  Takes a Response stream and reads it to completion. It returns a promise that resolves with a Blob.
- **Body** As Object
  A simple getter used to expose a ReadableStream of the body contents.
- **BodyUsed** As Boolean
  Stores a Boolean that declares whether the body has been used in a response yet.
- **Clone** As BANanoFetchResponse
  Creates a clone of a Response object.
- **Error** As BANanoFetchResponse
  Returns a new Response object associated with a network error.
- **FormData** As Object
  Takes a Response stream and reads it to completion. It returns a promise that resolves with a FormData object.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Headers** As Object
  Contains the Headers object associated with the response.
- **Json** As Object
  Takes a Response stream and reads it to completion. It returns a promise that resolves with the result of parsing the body text as JSON.
- **OK** As Boolean
  Contains a boolean stating whether the response was successful (status in the range 200-299) or not.
- **Redirect** (url As String, Status As Int) As BANanoFetchResponse
  The redirect() method returns a Response resulting in a redirect to the specified URL. Status is optional, pass null if not used.
- **Redirected** As Boolean
  Indicates whether or not the response is the result of a redirect; that is, its URL list has more than one entry.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

If only one, non-Array param is passed, you can ignore this.

e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **SetField** (field As String, value As Object)
  Sets a field value
- **Status** As Int
  Contains the status code of the response (e.g., 200 for a success).
- **StatusText** As String
  Contains the status message corresponding to the status code (e.g., OK for 200).
- **Text** As Object
  Takes a Response stream and reads it to completion. It returns a promise that resolves with a USVString (text).
- **Type** As String
  Contains the type of the response (e.g., basic, cors).
- **Url** As String
  Contains the URL of the response.

Properties

- **UseFinalUrl** As Boolean
  State whether this is the final URL of the response.

18.9 BANanoGeoLocation

Functions

- **ClearWatch** (watchID As Int)
  Stops the watchPosition() method.
- **GetCurrentPosition** (PositionCallback As Object, ErrorCallback As Object)
  Only works with https!

  Return the user's position

  PositionCallback: MUST look like functionName(position AS BANanoGeoPosition)
  ErrorCallback: MUST look like functionName(error As int):

  1: Permission Denied
  2: Position Unavailable
  3: Timeout
  Else: Unknown error
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **WatchPosition** (PositionCallback As Object, ErrorCallback As Object) As Int
  Returns the current position of the user and continues to return updated position as the user moves (like the GPS in a car).

  PositionCallback: MUST look like functionName(position AS BANanoGeoPosition)
  ErrorCallback: MUST look like functionName(error As BANanoGeoError)

  Returns a watch id that then can be used to unregister the handler by passing it to the Geolocation.clearWatch() method

18.10 BANanoGeoPosition

Functions

- **Accuracy** As Double
  The accuracy of position (always returned)
- **Altitude** As Double
  The altitude in meters above the mean sea level (returned if available)
- **AltitudeAccuracy** As Double
  The altitude accuracy of position (returned if available)
- **Heading** As Double
  The heading as degrees clockwise from North (returned if available)
- **Latitude** As Double
  The latitude as a decimal number (always returned)
- **Longitude** As Double
  The longitude as a decimal number (always returned)
- **Speed** As Double
  The speed in meters per second (returned if available)
- **Timestamp** As Double
  The date/time of the response (returned if available)

18.11 BANanoHeader

Fields

- **Author** As String
- **BackgroundColor** As String
- **BaseTarget** As String
- **BaseURL** As String
- **Charset** As String
- **Description** As String
- **Expires** As Long
- **Keywords** As String
- **Language** As String
- **Manifest** As String
- **MaskIconColor** As String
- **MSTileColor** As String
- **OnDOMContentLoaded** As String
- **ThemeColor** As String
- **Title** As String
- **Viewport** As String

Functions

- **AddAppleTouchIcon** (AssetFileNameOrURL As String, size As String)
  home screen icons for Safari and iOS. Size e.g. 16x16.
- **AddAppleTouchStartupImage** (AssetFileNameOrURL As String, deviceWidth As String, deviceHeight As String, devicePixelRatio As String)
  startup screen for Safari on iOS. e.g. ("myimage", "320px", "568px", "2")
- **AddCSSFile** (AssetFileNameOrURL As String)
  Load an extra css file. If it is an asset file it will be copied to the styles folder

  *For locale files (not URLs), you can use the * wildcard*
- **AddCSSFileForLater** (AssetFileNameOrURL As String)
  Load an extra css file. If it is an asset file it will be copied to the styles folder
  This asset will not be loaded at loading the html file, but you will have to do it 'Later' using the BANano.AssetsLoad... methods

  *For locale files (not URLs), you can use the * wildcard*
- **AddFavicon** (AssetFileNameOrURL As String, size As String)
  Add additional fav icons. Size e.g. 16x16.
- **AddJavascriptES6File** (AssetFileName As String)
  Load an extra ES6 javascript file. It must be an asset file and cannot be an url.

  *You can use the * wildcard*
- **AddJavascriptES6FileForLater** (AssetFileName As String)
  Load an extra ES6 javascript file. It must be an asset file and cannot be an url.
  This asset will not be loaded at loading the html file, but you will have to do it 'Later' using the BANano.AssetsLoad... methods

*You can use the * wildcard*

- **AddJavascriptFile** (AssetFileNameOrURL As String)
  Load an extra javascript file. If it is an asset file it will be copied to the scripts folder

  *For locale files (not URLs), you can use the * wildcard*
- **AddJavascriptFileForLater** (AssetFileNameOrURL As String)
  Load an extra javascript file. If it is an asset file it will be copied to the scripts folder
  This asset will not be loaded at loading the html file, but you will have to do it 'Later' using the BANano.AssetsLoad... methods

  *For locale files (not URLs), you can use the * wildcard*
- **AddJavascriptFileForLaterSW** (AssetFileNameOrURL As String)
  Does the same a AddJavascriptFileForLater() but will write in also in the ImportScripts() method in the Service Worker file.

  NOTE: such a javascript file can NOT use window or document or any other reference to the DOM as a Service Worker cannot access this!

  If it is a javascript file used in a BANanoLibrary, it MUST be added in the app explicitly!

  These javascript files will NOT be merged!
- **AddJavascriptFileSW** (AssetFileNameOrURL As String)
  Does the same a AddJavascriptFile() but will write it also in the ImportScripts() method in the Service Worker file.

  NOTE: such a javascript file can NOT use window or document or any other reference to the DOM as a Service Worker cannot access this!

  If it is a javascript file used in a BANanoLibrary, it MUST be added in the app explicitly!

  These javascript files will NOT be merged!
- **AddManifestIcon** (AssetFileNameOrURL As String, size As String)
  PWA Icon used in the manifest.json file. Size e.g. 16x16.
- **AddMeta** (metaTag As String)
  must be a full html meta tag. Use smartstrings! e.g.

  ```
  $"<meta name="keywords" content="HTML,CSS,XML,JavaScript">"$
  ```
- **AddMSTileIcon** (AssetFileNameOrURL As String, size As String)
  home screen icons for Microsoft. Size e.g. 16x16.
- **SetAndroidMaskIcon** (AssetFileNameOrURL As String, size As String)
  Set the Mask Icon for Android devices. Size e.g. 731x731.
- **SetAppleMaskIcon** (AssetFileNameOrURL As String)
  Set the Mask Icon for Apple devices
- **UnzipAdditionalAssets** (fileName As String)
  Unzips a .zip file (with folders) in the root folder of your app on Build()

  The .css and .js files will be loaded when the App initializes.
  Additionally, if using a Service Worker, all the files in the zip will also be cached.

18.12 BANanoHistory

Functions

- **Back**
  Go back to the previous page
- **Forward**
  Go to the next page
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Go** (step As Int)
  Navigate back or forward multiple levels deep
- **Length** As Int
  Number of entries in the history
- **PushState** (state As Object, url As Object)
  Create a new history entry programmatically
  State must be serializable and is limited in size.

  Note that the URL needs to belong to the same origin domain of the current URL.
- **ReplaceState** (state As Object, url As Object)
  Allows you to edit the current history state.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value

Properties

- **State** As Object [read only]
  Returns the current state object

18.13 BANanoJSONGenerator (DEPRECIATED)

Use the normal B4J Json library instead.

Functions

- **Initialize** (Map As Map)
  Initializes the object with the given Map.
- **Initialize2** (List As List)
  Initializes the object with the given List.
- **ToPrettyString** (Indent As Int) As String
  Creates a JSON string from the initialized object.
  The string will be indented and easier for reading.
  Note that the string created is a valid JSON string.
  Indent - Number of spaces to add to each level.
- **ToString** As String
  Creates a JSON string from the initialized object.
  This string does not include any extra whitespace.

18.14 BANanoJSONParser (DEPRECIATED)

Use the normal B4J Json library instead.

Functions

- **Initialize** (Text As String)
  Initializes the object and sets the text that will be parsed.
- **NextArray** As List
- **NextObject** As Map
  Parses the text assuming that the top level value is an object.
- **NextValue** As String

18.15 BANanoJSONQuery

Functions

- **All** As List
  Returns all records.
- **Count** As Int
  Returns the number of records.
- **Find** (field As String, value As Object) As Map
  Returns the first record, matching the field with the passed value.
- **First** As Map
  Returns the first record.
- **GroupBy** (field As String) As BANanoJSONQuery
  Returns a Map where all records are grouped by field.

  The key in the map is the group field, the value is a list of records matching the group field.
- **Initialize** (json As Object)
  Initializes with a Json Object (must be an array of records).
- **Initialize2** (jsonString As String)
  Initializes with a Json String.
- **Last** As Map
  Returns the last record.
- **Limit** (limit As Int) As BANanoJSONQuery
  Limits the number of records returned. Often used together with Offset.

  e.g. `.Limit(5).Offset(20).All` *returns records 21,22,23,24,25*
- **Offset** (offset As Int) As BANanoJSONQuery
  Offset in the recordset. Often used together with Limit.

  e.g. `.Limit(5).Offset(20).All` *returns records 21,22,23,24,25*
- **Order** (jsonOrder As String) As BANanoJSONQuery
  Orders the result of the query.

  "{'FieldName' : 'desc/asc', ...}"
- **OrWhere** (jsonSelector As String) As BANanoJSONQuery
  Must be used after a Where.

  A selector is a string in this format:

  ```
  "{'FieldName.Operator' : value, ...}"
  ```

  e.g. `.Where(...).OrWhere("{'rating.$eq': 8.4, 'name.$li': /braveheart/i"}")`

  if the operator is omitted, .$eq is used.

  Possible operators:

  ```
  .$eq: Equal
  ```

```
.$ne: Not Equal
.$lt: Less than
.$gt: Greater than
.$bt: Between (expects an Array, e.g. [4, 6])
.$in: In (expects an Array, e.g. [6,15])
.$ni: Not in (expects an Array, e.g. [6,15])
.$li: Like (expects a String or a regex expression)
```

- **Pluck** (field As String) As List
  Returns an array containing all values of field.
- **SelectFields** (fields As List) As BANanoJSONQuery
  Returns an array of json objects with a subset of only the fields past .
- **ToJQ** As BANanoJSONQuery
  Returns a subset of the records, using the query settings.
  The result is a new BANanoJSONQuery object.

  e.g.

  ```
  Dim resultJQ as BANanoJSONQuery
  resultJQ = placesJQ.Where("'name': 'Bayview'").ToJQ

  resultJQ.Where("'types.$eq': 'polictical'").All
  ```

- **Unique** (field As String) As BANanoJSONQuery
  Returns a List with the unique values of a field.
- **Where** (jsonSelector As String) As BANanoJSONQuery
  A selector is a string in this format:

  ```
  "{'FieldName.Operator' : value, ...}"
  ```

  e.g. `.Where("{'rating.$eq': 8.4, 'name.$li': /braveheart/i}")`

  if the operator is omitted, .$eq is used.

  Possible operators:

  ```
  .$eq: Equal
  .$ne: Not Equal
  .$lt: Less than
  .$gt: Greater than
  .$bt: Between (expects an Array, e.g. [4,6])
  .$in: In (expects an Array, e.g. [6,15])
  .$ni: Not in (expects an Array, e.g. [6,15])
  .$li: Like (expects a String or a regex expression)
  ```

18.16 BANanoLocation

Functions

- **Assign** (URL As String)
  The assign() method loads a new document.
  The difference between this method and replace(), is that replace() removes the URL of the current document from the document history,
  meaning that it is not possible to use the 'back' button to navigate back to the original document.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **GetHash** As String
  Sets or returns the anchor part (#) of a URL
- **GetHost** As String
  Sets or returns the hostname and port number of a URL
- **GetHostName** As String
  Sets or returns the hostname of a URL
- **GetHref** As String
  Sets or returns the entire URL
- **GetPathName** As String
  Sets or returns the path name of a URL
- **GetPort** As Int
  Sets or returns the port number of a URL
- **GetProtocol** As String
  Sets or returns the protocol of a URL
- **GetSearch** As String
  Sets or returns the querystring part of a URL
- **Reload** (forceGet As Boolean)
  The reload() method is used to reload the current document.
  The reload() method does the same as the reload button in your browser.

  By default, the reload() method reloads the page from the cache, but you can force it to reload the page
  from the server by setting the forceGet parameter to true
- **Replace** (newURL As String)
  The replace() method replaces the current document with a new one.
  The difference between this method and assign(), is that replace() removes the URL of the current document from the document history,
  meaning that it is not possible to use the 'back' button to navigate back to the original document.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

If only one, non-Array param is passed, you can ignore this.

e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **SetField** (field As String, value As Object)
  Sets a field value
- **SetHash** (hash As String)
  Sets or returns the anchor part (#) of a URL
- **SetHost** (host As String)
  Sets or returns the hostname and port number of a URL
- **SetHostName** (hostName As String)
  Sets or returns the hostname of a URL
- **SetHref** (href As String)
  Sets or returns the entire URL
- **SetPathName** (pathName As String)
  Sets or returns the path name of a URL
- **SetPort** (port As Int)
  Sets or returns the port number of a URL
- **SetProtocol** (protocol As String)
  Sets or returns the protocol of a URL
- **SetSearch** (search As String)
  Sets or returns the querystring part of a URL

Properties

- **Origin** As String [read only]
  Returns the protocol, hostname and port number of a URL

18.17 BANanoMQTTClient (DEPRECIATED)

Use the normal B4J jMQTT library instead.

Events

- **Connected** (Success As Boolean)
- **Disconnected**
- **MessageArrived** (Topic As String, Payload() As Byte)

Functions

- **Close**
- **Connect**
- **Connect2** (Options As BANanoMQTTConnectOptions)
- **Initialize** (EventName As String, Server As String, port As Int, path As String, isSecure As Boolean, ClientID As String)
  Only Websockets are supported
- **IsInitialized** As Boolean
- **Publish** (Topic As String, Payload As Byte())
- **Publish2** (Topic As String, Payload As Byte(), QOS As Int, Retained As Boolean)
- **Subscribe** (Topic As String, QOS As Int)
- **Unsubscribe** (Topic As String)

Properties

- **ClientID** As String [read only]
- **QOS_0_MOST_ONCE** As Int [read only]
- **QOS_1_LEAST_ONCE** As Int [read only]
- **QOS_2_EXACTLY_ONCE** As Int [read only]

18.18 BANanoMQTTConnectOptions (DEPRECIATED)

Use the normal B4J jMQTT library instead.

Functions

- **Initialize** (UserName As String, Password As String)
  Initializes the object and sets the username and password.
  Pass empty strings if username or password are not required.
- **SetLastWill** (Topic As String, Payload As Byte(), QOS As Int, Retained As Boolean)

Properties

- **CleanSession** As Boolean [write only]
  If set to true (default value) then the state will not be preserved in the case of client restarts.
- **Password** As String
  Gets or sets the connection password.
- **UserName** As String
  Gets or sets the connection user name.

18.19 BANanoMediaQuery

Events

- **Matched()**

Functions

- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Initialize** (mediaQueryList As String)
  mediaQueryList examples:

```
(max-width: 400px)
(min-width: 401px) and (max-width: 600px)
(min-width: 601px) and (max-width: 800px)
(min-width: 801px)
(orientation: portrait)
(orientation: landscape)
```

- **IsInitialized** As Boolean
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **SetField** (field As String, value As Object)
  Sets a field value

18.20 BANanoMutationObserver

Events

- **CallBack** (records() As BANanoMutationRecord,
  observer As BANanoMutationObserver)

Functions

- **Disconnect**
  Stops the MutationObserver instance from receiving further notifications until and unless Observe() is called again.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Initialize** (callbackEventName As String)
  callbackEventName: the eventName of the CallBack event.

```
' record: A BANanoMutationRecord represents an individual DOM
mutation.
' observer: this observer
Sub EventName_CallBack(record As BANanoMutationRecord, observer As
BANanoMutationObserver)

End Sub
```

- **Observe** (target As Object)
  target: Which node (or parent node) to observe
  Will use the settings on this object.

  At least one of the following has to be set to true for it to work.

  .Attributes
  .ChildList
  .CharacterData
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **SetField** (field As String, value As Object)
  Sets a field value

- **TakeRecords** As BANanoMutationRecord()
  returns an array of all matching DOM changes that have been detected but not yet processed by the observer's callback function,
  leaving the mutation queue empty. The most common use case for this is to immediately fetch all pending mutation records
  immediately prior to disconnecting the observer, so that any pending mutations can be processed when stopping down the observer.

Properties

- **AttributeFilter** As List [write only]
  An array of DOMString objects, each specifying the name of one attribute whose value is to be monitored for changes.
  There is no default value.
- **AttributeOldValue** As Boolean [write only]
  A Boolean value indicating whether or not the prior value of a changed attribute should be included in the MutationObserver.oldValue property
  when reporting attribute value changes. If true, oldValue is set accordingly. If false, it is not.

  When using attributeOldValue, setting the attributes option to true is optional.
- **Attributes** As Boolean [write only]
  A Boolean value indicating whether or not to report through the callback any changes to the values of attributes on the node or nodes being monitored.
  The default value is false.

  If true, the callback specified when observe() was used to start observing the node or subtree will be called any time one or more attributes have changed on observed nodes.

  You can expand the capabilities of attribute mutation monitoring using other options:

  attributeFilter lets you specify specific attribute names to monitor instead of monitoring all attributes.
  attributeOldValue lets you specify whether or not you want the previous value of changed attributes to be included in the MutationRecord's oldValue property.
  subtree lets you specify whether to watch the target node and all of its descendants (true), or just the target node (false).

  If you set either attributeFilter or attributeOldValue to true, attributes is automatically assumed to be true, even if you don't expressly set it as such.
- **CharacterData** As Boolean [write only]
  A Boolean value indicating whether or not to call the observer's callback function when textual nodes' values change.

  If true, the callback specified when observe() was used to start observing the node or subtree is called any time the contents of a text node are changed.

  You can expand the capabilities of attribute mutation monitoring using other options:

  characterDataOldValue lets you specify whether or not you want the previous value of changed text nodes to be provided using the MutationRecord's oldValue property.

subtree lets you specify whether to watch the target node and all of its descendants (true), or just the target node (false).

If you set characterDataOldValue to true, characterData is automatically assumed to be true, even if you don't expressly set it as such.

- **CharacterDataOldValue** As Boolean [write only]
  A Boolean value indicating whether or not to set the MutationRecord's oldValue property to be a string containing the value of
  the character node's contents prior to the change represented by the mutation record.

  By default, only changes to the text of the node specified as the target parameter when you called observe() are monitored.
  To watch for changes to the text contents of all descendants of target, set the subtree option to true.

  If you set characterDataOldValue to true, characterData is automatically assumed to be true, even if you don't expressly set it as such.
- **ChildList** As Boolean [write only]
  A Boolean value indicating whether or not to invoke the callback function when new nodes are added to or removed from the section of the DOM being monitored.
  If subtree is false, only the node indicated by the observer's target node is monitored for changes. Setting subtree to true causes addition or removal of nodes
  anywhere within the subtree rooted at target to be reported.
- **SubTree** As Boolean [write only]
  Set to true to extend monitoring to the entire subtree of nodes rooted at target. All of the other MutationObserverInit properties are then extended to all of the nodes
  in the subtree instead of applying solely to the target node.

  The default value is false.

18.21 BANanoMutationRecord

Functions

- **AddedNodes** As BANanoObject()
  Return the nodes added. Will be empty if no nodes were added.
- **AttributeName** As String
  Returns the local name of the changed attribute, or null.
- **AttributeNamespace** As String
  Returns the namespace of the changed attribute, or null.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **NextSibling** As BANanoObject
  Return the next sibling of the added or removed nodes, or null.
- **OldValue** As String
  The return value depends on the MutationRecord.type.

  For attributes, it is the value of the changed attribute before the change.
  For characterData, it is the data of the changed node before the change.
  For childList, it is null.
- **PreviousSibling** As BANanoObject
  Return the previous sibling of the added or removed nodes, or null.
- **RemovedNodes** As BANanoObject()
  Return the nodes removed. Will be empty if no nodes were removed.
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **Target** As BANanoObject
  Returns the node the mutation affected, depending on the Type.

  For attributes, it is the element whose attribute changed.
  For characterData, it is the CharacterData node.
  For childList, it is the node whose children changed.

- **TypeRecord** As String
  Returns
  "attributes" if the mutation was an attribute mutation,
  "characterData" if it was a mutation to a CharacterData node,
  "childList" if it was a mutation to the tree of nodes.

18.22 BANanoNavigator

Functions

- **AppCodeName** As String
  Returns the code name of the browser
- **AppName** As String
  Returns the name of the browser
- **AppVersion** As String
  Returns the version information of the browser
- **CookieEnabled** As Boolean
  Determines whether cookies are enabled in the browser
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **JavaEnabled** As Boolean
  Specifies whether or not the browser has Java enabled
- **Language** As String
  Returns the language of the browser
- **OnLine** As Boolean
  Determines whether the browser is online
- **Platform** As String
  Returns for which platform the browser is compiled
- **Product** As String
  Returns the engine name of the browser
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **UserAgent** As String
  Returns the user-agent header sent by the browser to the server

18.23 BANanoObject

Functions

- **AddEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)
  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.

  true - The event handler is executed in the capturing phase
  false - The event handler is executed in the bubbling phase
- **AddEventListenerOpen** (eventName As String, params As Object)
  All the code between AddEventListenerOpen and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpen MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpen("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = function(aEvt) {
...
};
```

- **AddEventListenerOpenAsync** (eventName As String, params As Object)
  All the code between AddEventListenerOpenAsync and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpenAsync MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpenAsync("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = async function(aEvt) {
...
};
```

- **ClientHeight** As Double
  Returns the height of an element, including padding
- **ClientLeft** As Double
  Returns the width of the left border of an element
- **ClientTop** As Double
  Returns the width of the top border of an element
- **ClientWidth** As Double
  Returns the width of an element, including padding
- **CloseEventListener**
  Closes an AddEventListenerOpen or AddEventListenerOpenAsync method
- **Delete** (property As String)
  The delete operator deletes a property from an object
- **Execute** (params As List) As BANanoObject
  If BANanoObject is a function, then you can execute it directly with params.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **GetFunction** (functionName As String) As BANanoObject
  Gets a function. You can then directly call the Execute method on it.
- **GetScrollLeft** As Double
  Sets or returns the number of pixels an element's content is scrolled horizontally
- **GetScrollTop** As Double
  Sets or returns the number of pixels an element's content is scrolled vertically
- **HasOwnProperty** (property As String) As Boolean
  Check whether a property is inherited
- **Initialize** (jsObject As Object)
  Can be used e.g. to connect a BANanoObject to a JavaScript object
- **Initialize2** (jsObject As String, params As Object) As BANanoObject
  To initialize for a 'New libObjectName' javascript library

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  see RunMethod for more info on the Array system.

  e.g. Javascript:

```
let datepicker = new When({
   input: document.getElementById('...'),
   singleDate: true
});
datepicker.showHeader = true;
```

  Translated to B4J:

```
Dim datepicker As BANanoObject
datepicker.Initialize2("When", CreateMap("input":
BANano.GetElement("#datepicker").ToObject, "singleDate": True))
datepicker.RunMethod("showHeader", True)
```

- **Initialize3** (params As Object) As BANanoObject
  Initialize using a constructor on the JavaScript object.

NOTE: the outer Array will be removed in the javascript.
So if you want to pass an array, you have to add an extra array.

e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
Initialize3(Array(Array(0,0), "Alain"))
```

If only one, non-Array param is passed, you can ignore this.

e.g. this is valid

```
Initialize3("Alain")
```

- **Initialize4** (jsObject As String, params As Object) As BANanoObject
  Can be used e.g. to connect a BANanoObject to a JavaScript object, with parameters

  This is basically the .Initialize, but with parameters. It does NOT do a New like the Initialize2 method!
- **Initialize5** As BANanoObject
  Can be used create a plain Javascript object

  This is basically set the object in Javascript to {}
- **Initialize6** (javaScriptObject As String)
  Initialize a BANanoObject from a JavaScript object, defined as a SmartString.

  Initialize5 could also be written as b.Initialize6("{}")

  Example:

```
Dim city As String = "Ieper"

Dim b As BANanoObject
b.Initialize6($"{
   body: "myBody",
   name: "myName",
   city: "${city}"
}"$)

Log(b)

Log(b.GetField("body"))
b.SetField("city", "Ieper Stad")
```

  Log(b)
- **Initialize7** (javaScriptObject As Object, constructor As String, params As Object)
  Initialize a new BANanoObject from a BANanoObjects constructor.

  Example:

```
' in JavaScript
' var innerConn = ...
' var query = new innerConn.$sql.Query(SQL);
Dim Query As BANanoObject
Query.Initialize2(innerConn, "$sql.Query", SQL)
```

- **IsInitialized** As Boolean

- **OffsetHeight** As Double
  Returns the height of an element, including padding, border and scrollbar
- **OffsetLeft** As Double
  Returns the horizontal offset position of an element
- **OffsetTop** As Double
  Returns the vertical offset position of an element
- **OffsetWidth** As Double
  Returns the width of an element, including padding, border and scrollbar
- **RemoveEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  Removes event handlers that have been attached with the addEventListener() method

  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies the event phase to remove the event handler from.

  true - Removes the event handler from the capturing phase
  false - Removes the event handler from the bubbling phase
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **ScrollHeight** As Double
  Returns the entire height of an element, including padding
- **ScrollWidth** As Double
  Returns the entire width of an element, including padding
- **Selector** (selector As Object) As BANanoObject
  Useful for e.g. jQuery selectors

```
Dim jQ as BANanoObject
jQ.Initialize("$")
JQ.Selector("#btn2").RunMethod("Click", Array(BANano.CallBack(Me,
"btn2_clicked", Null))))

Sub Btn2_Clicked()
   BANano.Msgbox("btn2 clicked through BANanoObject and jQuery!")
End Sub
```

- **SetField** (field As String, value As Object)
  Sets a field value
- **SetScrollLeft** (x As Double)
  Sets or returns the number of pixels an element's content is scrolled horizontally
- **SetScrollTop** (y As Double)
  Sets or returns the number of pixels an element's content is scrolled vertically
- **ToString** As String
  Converts the object to a String
- **ToString2** (base As Int) As String
  Converts the object to a String with a base

  e.g.
```
Dim b as BANanoObject
b = 15000
Log(b.ToString(10)) ' base 10
Log(b.ToString(16)) ' hex
```

18.24 BANanoPromise

Functions

- **CallSub** (module As Object, methodName As String, params As List)
  Calls a method. This method must use BANano.ReturnThen and/or BANano.ReturnElse to finish the method.

  e.g.

```
Sub AFunction()
   Dim returnValue As Long 'ignore
   Dim promise As BANanoPromise

   promise.CallSub(Me, "myfuncwait", Array(25))
   promise.Then(returnValue)
      Log("The sum < 150 " & returnValue)
   promise.else(returnValue)
      Log("rejected: > 150 " & returnValue)
   promise.End
End Sub

public Sub myFuncWait(toAdd As Long)
   If 100 + toAdd < 150 Then
      BANano.ReturnThen(100 + toAdd)
   End If
   Sleep(5000)
   BANano.ReturnElse(100 + toAdd)
End Sub
```

- **Else** (returnValue As Object)
  Continues here after the CallSub with the returnValue from ReturnElse.

  See CallSub() for an example.
- **ElseWait** (returnValue As Object)
  Is the same as .Else, except the function will be async.

  This can be used if the code in the .ElseWait clause contains ...Wait functions or Sleep
- **End**
  Terminates the promise Then/Else/Finally
- **Finally**
  Will always run at the end
- **FinallyWait**
  Is the same as .Finally, except the function will be async.

  This can be used if the code in the .FinallyWait clause contains ...Wait functions or Sleep
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **IsInitialized** As Boolean
- **NewEnd**
  Close a NewStart or NewStartWait.
- **NewStart**
  Make a new Promise with this signature:

```
Dim prom as BANanoPromise
prom.NewStart
   ...
   BANano.ReturnThen(ret)
   ...
   BANano.ReturnElse(ret)
prom.NewEnd
prom.Then(response)

prom.Else(response)

prom.End


Transpiles to:

prom = new Promise(function(resolve, reject) {
   ...
   resolve(ret);
   ...
   reject(ret)
});
prom.then(function(response) {

}).else(function(response) {

});
```

Example:

```
Dim response As String
Dim prom As BANanoPromise 'ignore
prom.NewStart
   BANano.ReturnThen("Alain")
prom.NewEnd
   prom.Then(response)
   Return "Hello " & response & "!" 'ignore
prom.Then(response) 'ignore
   Log(response) ' prints: Hello Alain!
prom.end
```

- **NewStartWait**
  Same as NewStart, but needed if you use Wait methods

  Example:

```
Dim response As String
Dim prom As BANanoPromise 'ignore
prom.NewStartWait
   Sleep(3000) ' is a Wait method
   BANano.ReturnThen("Alain")
prom.NewEnd
prom.Then(response)
   Return "Hello " & response & "!" 'ignore
prom.Then(response) 'ignore
   Log(response) ' prints: Hello Alain!
prom.end
```

- **Result** As Object
  Gets the result

- **Return** (data As Object)
  Returns something in a then part. Can be passed on the next then.
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **Then** (returnValue As Object) As BANanoPromise
  Continues here after the promise.CallSub with the returnValue from ReturnThen.

  See CallSub() for an example.
- **ThenWait** (returnValue As Object)
  Is the same as .Then, except the function will be async.

  This can be used if the code in the .ThenWait clause contains ...Wait functions or Sleep

18.25 BANanoRegEx

Functions

- **Exec** (s As String) As String
  Tests for a match in a string. Returns the first match
- **ExecAll** (s As String) As String()
  Tests for a match in a string. Returns all matches
- **GetIndex** (execResult As String()) As Int
  The 0-based index of the match in the string.
- **Global** As Boolean
  Checks whether the "g" modifier is set
- **IgnoreCase** As Boolean
  Checks whether the "i" modifier is set
- **InitializePattern** (pattern As String)
  Creates a new BANanoRegEx object using the given pattern
  the outer quotes will be removed

  see Javascript RegEx reference for the pattern.
- **InitializeString** (patternS As String)
  Creates a new BANanoRegEx object using the given string

  see Javascript RegEx reference for the pattern.
- **LastIndex** As Int
  The lastIndex property specifies the index at which to start the next match.
- **MultiLine** As Boolean
  Check whether or not the "m" modifier is set
- **ReplacePattern** (s As String, pattern As String, byStr As String) As String
  The replace() method replaces a specified value with another value in a string

  you don't need to use BANano.RegEx to initiate it.
- **ReplaceString** (s As String, patternS As String, byStr As String) As String
  The replace() method replaces a specified value with another value in a string
  the outer quotes will be removed

  you don't need to use BANano.RegEx to initiate it.
- **SearchPattern** (s As String, pattern As String) As Int
  The search() method searches a string for a specified value and returns the position of the match
  the outer quotes will be removed

  you don't need to use BANano.RegEx to initiate it.
- **SearchString** (s As String, patternS As String) As Int
  The search() method searches a string for a specified value and returns the position of the match

  you don't need to use BANano.RegEx to initiate it.
- **Source** As String
  Return the text of the RegExp pattern

- **Test** (s As String) As Boolean
  Tests for a match in a string. Returns true or false

18.26 BANanoRouter

Functions

- **AddRoute** (path As String, handlerClass As String, initializeParams As List, loadHTML As Boolean)
  Adds a Route to the router.

  initializeParams: an array containing all the extra parameters needed to run the handlerClass.Initialize method.
  loadHTML: if true, then ./path/index.html will be loaded + a binding to all BANanoAutoelements in the class.
  HandlerClass: Must have the following Method(s):

```
' handling the call (data and/or params can be null!)
Sub BANano_RouterHandle(url As String, data As Map, params As Map)

End Sub

' (Optional) if you want to do some checks before one leaves the page.
' Can cancel the page change.
Sub BANano_RouterLeaving() As Boolean

   Return True (or False if you do not want to leave)
End Sub

Examples:
Router.AddRoute("/foo", "FooClass", Null, False)
- matches specifically "/foo"

Router.AddRoute("/foo/:name", "FooClass", Null, False)
- matches "/foo/my-name-here"

Router.AddRoute(":page", "FooClass", Null, False)
- matches "/about-page"

Router.AddRoute("/foo/*", "FooClass", Null, False)
- matches "/foo/a/b/c"

Router.AddRoute("*", "FooClass", Null, False)
- matches "/foo/bar/moo"

Router.AddRoute("/foo/:id/?", "FooClass", Null, False)
- matches "/foo/20/save" and also "/foo/20"

Router.Addroute("/testPage1", "testPage1", Array("Something extra"),
True)
- will load ./testPage1/index.html and bind all BANanoAutoElements
with the variables declared in the testPage1 class.
```

- **Initialize** (rootPath As String, matchAll As Boolean)
  rootPath: the root path of your application. For example, if you are hosting the application at https://site.com/my/project you have to specify the following:
  matchAll: default false, meaning that when a match is found the router stops resolving other routes. If set true, it will continue searching for other matches

e.g. Router.AddRoute("/foo/:id/?", "FooClass") matches "/foo/20/save" and also "/foo/20"

NOTE: use the hash string as path for routing. For example /my/app/#/about/team is treated as /about/team when entered in the browser's navigation bar.

```
Dim Router as BANanoRouter
Router.Initialize("/my/project", False)
```

- **Navigate** (path As String)
  Navigate does the following:
  1. Checks if there is a match. And if the answer is "yes" then ...
  2. It calls the old page route leaving (if exists). If this returns True (or does not exist) then ...
  3. It calls the new page route handler.
  4. Updates the internal state of the router.
- **NotFound** (handlerClass As String, initializeParams As List)
  A special handler for the cases where a no match is found.

  initializeParams: an array containing all the extra parameters needed to run the handlerClass.Initialize method.
- **RemoveRoute** (path As String)
  To remove a route call the RemoveRoute method by passing the path of the route.
- **Start** (initialPath As String)
  Starts the router, going to your initial Path

18.27 BANanoSQL

Events

- **SQLExecuteError** (Tag As String, Reason As String)
- **SQLExecuteResult** (Tag As String, Result As List)
- **SQLOpened()**

Functions

- **Execute** (Query As String, Args As List, tag As String)
  Will return the result in the SQLResult(Tag as String, Result as List) event. The tag can be used to see where it is coming from.

  Result returns a list of maps containing the requested data in case of a SELECT

  if an error occurs, SQLExecuteError() will be raised with Tag you passed
- **ExecuteCallback** (Query As String, Args As List, module As Object, methodName As String)
  Will return the result in the callback. MUST BE defined as:

  ```
  functionName(success As Boolean, Result as List, Reason as String)
  ```

  Result returns a list of maps containing the requested data in case of a SELECT
- **ExecuteWait** (Query As String, Args As List) As List
  Returns the result as a List of maps containing the requested data in case of a SELECT

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.
- **LastInserted** (tableName As String, fieldName As String) As Object
  Returns the last auto incremented value after an insert
- **NextInserted** (tableName As String, fieldName As String) As Object
  Returns the next auto incremented value that will be used
- **Open** (eventName As String, databaseName As String)
  Opens the database and creates it if it does not exist. Uses IndexedDB so everything works with promises.

  When the database is created and open the event SQLOpened() will be raised

  databaseName cannot be a variable and must be a literal String

  if an error occurs, SQLExecuteError() will be raised with Tag="CREATEDB"
- **OpenWait** (eventName As String, databaseName As String) As Object
  Opens the database and creates it if it does not exist. Uses IndexedDB so everything works with promises.

  databaseName cannot be a variable and must be a literal String

  Note: Do not use a BANano.AWait around this method as it already does it internally and needs some other settings before being able to run.

18.28 BANanoScreen

Functions

- **AvailHeight** As Double
  Returns the height of the screen (excluding the Windows Taskbar)
- **AvailWidth** As Double
  Returns the width of the screen (excluding the Windows Taskbar)
- **ColorDepth** As Double
  Returns the bit depth of the color palette for displaying images
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Height** As Double
  Returns the total height of the screen
- **PixelDepth** As Double
  Returns the color resolution (in bits per pixel) of the screen
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **Width** As Double
  Returns the total width of the screen

18.29 BANanoTranspilerOptions

Functions

- **ActivateFirebaseMessaging** (messageSenderId As String, apiKey As String, projectId As String, appId As String)
  Activates the Firebase Messaging.

  Setup and the value of the parameters can be found here|https://www.itwonders-web.com/blog/push-notification-using-firebase-demo-tutorial

  A file called firebase-messaging-sw.js will be created. It MUST be copied to the root of your domain!

  Add the following:

```
BANano.TranspilerOptions.ActivateFirebaseMessaging("317703310xxx",
"AIzaSyDC16GI7KqNo0vLbHF-xxxxxxxxxxxx-EDE8",
"onetwoxxxx","1:317703310964:web:xxxxxxxxxxxxxxx")
```

  Sending a message can be done by sending a POST message to the url:

```
https://fcm.googleapis.com/fcm/send
```

  With Headers:

```
"Content-Type": "application/json","Authorization": "key=Server_key"
```

  ***Server_Key**: get from the Firebase Console (Project Settings - Cloud Messaging)*

  Body:

```
{
   "to": "cXpswYrkmcnrwk3rqolMuR:APA91bE8piwYq-
NTezNUhpj3pIfgX0_vgddqjw8dKkLLOhCYCq1t3-B_9r_1lD5LNo0OA1u...",
   "notification": {
      "title": "Alain",
      "body": "Alain and some more",
      "icon": "./assets/onetwo.png",
      "content_available": true,
      "priority": "high",
      "requireInteraction": true,
      "image": "./assets/scanner.jpg",
      "vibration": [300, 100, 400],
   }
}
```

  ***to:** can be retrieved with the `BANano.GetFirebaseToken` method after the user gave permission.*
- **DoNotDeleteFileOnCompilation** (fullPath As String)
  Prevents the Transpiler from deleting this file. Useful e.g. for assets that are not in the /Files folder.

- **DoNotDeleteFolderOnCompilation** (fullPath As String)
  Prevents the Transpiler from deleting this folder. Useful e.g. for assets that are not in the /Files folder.
- **ExcludePWACachingUrlContaining** (str As String)
  Url containing the given string will not be cached by the PWA Service Worker. Case sensitive.
- **GZipGeneratedWebsite** (minSizeKB As Double)
  Will GZip your html/css/js/json files on compilation. Set a minimum filesize so small files are not compressed

  This is ONLY useful if you use NGinx with gzip_static set to 'on'
- **IgnoreB4JLibrary** (libName As String)
  A B4J library the BANano Transpiler should ignore.

  By default, the following are ignored:

  ```
  BANano
  BANanoServer
  jCore
  jFx
  json
  jMQTT
  jServer
  JavaObject
  ABJJWT
  ```
- **RedirectOutput** (dir As String, fileName As String)
  Redirects the logs to a file. Must be set in AppStart
- **SetFireReadyWhenReadyStateComplete** (bool As Boolean)
  Raises the Ready state only when the loading state is 'Complete'. Default = True
- **SetIgnoreAutoID** (ignore As Boolean)
  Ignore Transpiling the AutoID property of a CustomView to a random ID/Name.
- **SetPWAStartUrl** (StartURL As String)
  Sets the Start Url in the manifest.json file for a PWA. Default is HTML_NAME

  e.g. `PWA/index.html`
- **SetSessionMaxInactiveInterval** (sessionMaxInactiveInterval As Int)
  Sends a heartbeat. Only applicable if a BANanoServer is used.

  Should be the same as the BANanoServer.SessionMaxInactiveInterval
- **SetStaticFolder** (staticFolderName As String)
  Sets the static folder name. is by default the appShortVersion. Can only be set in AppStart, after the initialize.
- **UseServiceWorkerWithUpdateMessage** (bool As Boolean, UpdateColor As String, UpdateTitle As String, UpdateMessage As String)
  Use a service worker where an update toast is showed if an update is available. The user can then click the toast to do the update.

Properties

- **Author** As String [write only]
  Must be set before Build(). Only applicable for .b4xlibs.

- **DisableShortenVariableNames** As Boolean [write only]
  Disables the shorting of the variable names in release mode
- **EnableLiveCodeSwapping** As Boolean [write only]
  Enable Live Code Swapping and watch live changes made in the B4J source code.
  On Save, the changed B4J code is Transpiled again and reloaded by the browser.

  Default = true
- **ExternalTestConnectionServer** As String [write only]
  By default the connection to the internet is tested by checking if donotdelete.gif can be retrieved
  from the assets folder where the app is hosted.

  However, if you do not put it on a host (e.g. just by opening the .html file from disk),
  You can upload the donotdelete.gif to some host on the internet to test for an internet connection.
- **IDEComment** As String [write only]
  Must be set before Build(). Only applicable for .b4xlibs.
  Adds a comment in the libs manifest
- **IgnoreWarningsOldBrowsers** As Boolean [write only]
  If True the transpiler will not show warnings for older browsers
- **MergeAllCSSFiles** As Boolean [write only]
  Must be set before Build(). Only used when in Release mode.
- **MergeAllJavascriptFiles** As Boolean [write only]
  Must be set before Build(). Only used when in Release mode.
- **MinifyOnline** As Boolean [write only] DEPRECIATED (Does not work anymore)
  Using the API of:

  https://javascript-minifier.com, the generated Javascript file will be minified
  https://cssminifier.com, the CSS files will be minified
- **RemoveDeadCode** As Boolean [write only]
  Only works in Build

  The transpiler does not GENERATE dead code (never used). It does NOT remove the B4J code!

  Use ShowWarningDeadCode beforehand to check if the transpiler is correct.
  Methods with a _ in their name are always considered to be needed.
- **ShowLogPosition** As Boolean [write only]
  Shows the position in the B4J code of the log in DebugMode
- **ShowWarningDeadCode** As Boolean [write only]
  Only works in Build

  Shows a warning in the log if the transpiler suspects some code is dead (never used).
  This is handy, especially in the final stage of development to remove code (or comment out) that is never used.

  Methods with a _ in their name are always considered to be needed.

You can then use the RemoveDeadCode property to prevent GENERATING dead javascript code.

- **UseServiceWorker** As Boolean [write only]
  USE UseServiceWorkerWithUpdateMessage INSTEAD *Can ONLY be used in AppStart(). Set this param to true if you do want to use a ServiceWorker*
  Default false

18.30 BANanoURL

Functions

- **CreateObjectUrl** (object As Object) As BANanoURL
  The URL.createObjectURL() static method creates a DOMString containing a URL representing the object given in the parameter.
  The URL lifetime is tied to the document in the window on which it was created. The new object URL represents the specified File object or Blob object.
  To release an object URL, call revokeObjectURL().
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **GetHash** As String
  returns a String containing a '#' followed by the fragment identifier of the URL.
  The fragment is not percent-decoded. If the URL does not have a fragment identifier, this property contains an empty string - "".
- **GetHost** As String
  The host property of the URL interface is a String containing the host, that is the hostname, and then,
  if the port of the URL is nonempty, a ':', and the port of the URL.
- **GetHostname** As String
  Gets the hostname property of the URL interface is a String containing the domain of the URL.
- **GetHref** As String
  Gets the href property of the URL interface is a String containing the whole URL.
- **GetPassword** As String
  Gets the password property of the URL interface is a String containing the password specified before the domain name.
  If it is set without first setting the username property, it silently fails.
- **GetPathname** As String
  Gets the pathname property of the URL interface is a String containing an initial '/' followed by the path of the URL (or the empty string if there is no path).
- **GetPort** As String
  Gets the port property of the URL interface is a String containing the port number of the URL. If the URL does not contain an explicit port number, it will be set to ''.
- **GetProtocol** As String
  Gets the protocol property of the URL interface is a String representing the protocol scheme of the URL, including the final ':'.
- **GetSearch** As String
  Get the search property of the URL interface is a search string, also called a query string, that is a String containing a '?' followed by the parameters of the URL.
  Modern browsers provide the URL.searchParams property to make it easy to parse out the parameters from the querystring.
- **GetUsername** As String
  Gets the username property of the URL interface is a String containing the username specified before the domain name.
- **Initialize** (url As String, base As String)
  *returns a newly created URL object representing the URL defined by the parameters. **

- **Result** As Object
  Gets the result
- **RevokeObjectURL** (url As BANanoURL)
  The URL.revokeObjectURL() static method releases an existing object URL which was previously created by calling URL.createObjectURL().
  Call this method when you'll've finished using an object URL to let the browser know not to keep the reference to the file any longer.
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **SetField** (field As String, value As Object)
  Sets a field value
- **SetHash** (hash As String)
  set a '#' followed by the fragment identifier of the URL
- **SetHost** (host As String)
  sets the host property, a String containing the host, that is the hostname, and then, if the port of the URL is nonempty, a ':', and the port of the URL.
- **SetHostname** (hostname As String)
  Sets the hostname property of the URL interface is a String containing the domain of the URL.
- **SetHref** (href As String)
  Sets the href property of the URL interface is a String containing the whole URL.
- **SetPassword** (password As String)
  Sets the password property of the URL interface is a String containing the password specified before the domain name.
  If it is set without first setting the username property, it silently fails.
- **SetPathname** (pathname As String)
  Sets the pathname property of the URL interface is a String containing an initial '/' followed by the path of the URL (or the empty string if there is no path).
- **SetPort** (port As String)
  Sets the port property of the URL interface is a String containing the port number of the URL. If the URL does not contain an explicit port number, it will be set to ''.
- **SetProtocol** (protocol As String)
  Sets the protocol property of the URL interface is a String representing the protocol scheme of the URL, including the final ':'.
- **SetSearch** (search As String)
  Set the search property of the URL interface is a search string, also called a query string,

that is a String containing a '?' followed by the parameters of the URL.
Modern browsers provide the URL.searchParams property to make it easy to parse out the parameters from the querystring.

- **SetUsername** (username As String)
  Sets the username property of the URL interface is a String containing the username specified before the domain name.
- **ToJSON** As String
  The toJSON() method of the URL interface returns a String containing a serialized version of the URL, although in practice it seems to have the same effect as URL.toString().
- **ToString** As String
  The URL.toString() stringifier method returns a String containing the whole URL. It is effectively a read-only version of URL.href.

Properties

- **Origin** As String [read only]
  The origin read-only property of the URL interface returns a String containing the Unicode serialization
  of the origin of the represented URL. The exact structure varies depending on the type of URL
- **SearchParams** As Map [read only]
  The searchParams read-only property of the URL interface returns a URLSearchParams object allowing access to the GET decoded query arguments contained in the URL.

18.31 BANanoWebSocket

Events

- **OnClose** (event As BANanoEvent)
- **OnConnecting** (event As BANanoEvent)
- **OnError** (event As BANanoEvent)
- **OnMessage** (event As BANanoEvent)
- **OnOpen** (event As BANanoEvent)
- **OnServerReady()**
- **WebSocket_Connected()**
- **WebSocket_Disconnected** (event As BANanoEvent)

Fields

- **CLOSED** As Int
- **CLOSING** As Int
- **CONNECTING** As Int
- **OPEN** As Int

Functions

- **B4JSend** (methodName As String, data As Map)
  Same as send, but for a B4J Webserver. Is send as an event

  The methodName MUST end with _BAN and have only one parameter: Params as Map!

  e.g. `myServerFunc_BAN(Params as Map)`

  ```
  ws.B4JSend("myServerFunc_BAN", CreateMap("Message": "My message"))
  ```
- **Close**
  Closes the WebSocket connection or connection attempt, if any. If the connection is already CLOSED, this method does nothing.
- **CloseReason** (code As Int, Reason As String)
  Closes the WebSocket connection or connection attempt, if any. If the connection is already CLOSED, this method does nothing.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **Initialize** (Url As String)
  If using BANanoServer. Raises classic B4J WebSocket_Connected and WebSocket_Disconnected events.

  Do NOT use if you do not use a BANanoServer! Use InitializeExtended instead.

  Creates a new WebSocket using the url (ws:// or wss://).
- **InitializeExtended** (eventName As String, Url As String, protocols As String, isReconnectingWebSocket As Boolean)
  Extended version that raises events `_OnOpen, _OnClose, _OnMessage, _OnError, _OnConnecting and _OnServerReady` *(if using BANanoServer)*

Creates a new WebSocket using the url (ws:// or wss://).

protocols: leave empty ("") if not used.
isReconnectingWebSocket: if true, then a special WebSocket will be used which tries to reconnect if the connection is lost

- **IsReconnected** As Boolean
  In case of a Reconnecting Websocket, you can check if this was a reconnected session
- **Result** As Object
  Gets the result
- **RunFunction** (Function As String, Args As List)
  Runs a B4J function. Pass null to Args if is not needed.
- **RunFunctionWithResult** (Function As String, Args As List) As BANanoPromise
  Similar to RunFunction. Returns a BANanoPromise with the result.
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **Send** (data As Object)
  Enqueues the specified data to be transmitted to the server over the WebSocket connection, increasing the value of bufferedAmount
  by the number of bytes needed to contain the data. If the data can't be sent (for example, because it needs to be buffered
  but the buffer is full), the socket is closed automatically.
- **SetField** (field As String, value As Object)
  Sets a field value

Properties

- **BinaryType** As String
  CAN NOT BE USED WITH A RECONNECTING WEBSOCKET!

  Get/Set the type of binary data being transmitted by the connection.

  Possible values "blob" or "arraybuffer"
- **BufferedAmount** As Long [read only]
  CAN NOT BE USED WITH A RECONNECTING WEBSOCKET!

  Read-only property returns the number of bytes of data that have been queued using calls to send() but not yet transmitted to the network.

This value resets to zero once all queued data has been sent. This value does not reset to zero when the connection is closed:
if you keep calling send(), this will continue to climb.

- **Extensions** As String [read only]
  CAN NOT BE USED WITH A RECONNECTING WEBSOCKET!

  Read-only property returns the extensions selected by the server.
  This is currently only the empty string or a list of extensions as negotiated by the connection.
- **IsSupported** As Boolean [read only]
  Check if the browser does support WebSockets
- **Protocol** As String [read only]
  Read-only property returns the name of the sub-protocol the server selected:
  this will be one of the strings specified in the protocols parameter when creating the WebSocket object.
- **ReadyState** As Int [read only]
  Read-only property returns the current state of the WebSocket connection.
  See the constants on this object.
- **ReconnectAttempts** As Int [read only]
  The number of attempted reconnects since starting, or the last successful connection.
  Read only.
  Only for a Reconnecting WebSocket
- **Url** As String [read only]
  Read-only property returns the absolute URL of the WebSocket as resolved by the constructor.

18.32 BANanoWindow

Functions

- **AddEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)
  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.

  true - The event handler is executed in the capturing phase
  false - The event handler is executed in the bubbling phase
- **AddEventListenerOpen** (eventName As String, params As Object)
  All the code between AddEventListenerOpen and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpen MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpen("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = function(aEvt) {
...
};
```

- **AddEventListenerOpenAsync** (eventName As String, params As Object)
  All the code between AddEventListenerOpenAsync and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpenAsync MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

```
req.AddEventListenerOpenAsync("onreadystatechange", aEvt)
...
req.CloseEventListener
```

  transpiles to:

```
req.onreadystatechange = async function(aEvt) {
...
};
```

- **Alert** (message As String)
  Displays an alert box with a message and an OK button

  Same as BANano.Msgbox() or BANano.Alert()
- **Atob** (base64String As String) As String
  DEPRECIATED: Use BANano.Atob() instead

  Decodes a base-64 encoded string
- **Blur**
  Removes focus from the current window
- **Btoa** (string As String) As String
  DEPRECIATED: Use BANano.Btoa() instead

  Encodes a string in base-64
- **CancelAnimationFrame** (requestID As Long)
  cancels an animation frame request previously scheduled through a call to window.requestAnimationFrame().
- **ClearInterval** (timerVar As Object)
  Clears a timer set with setInterval()
- **ClearTimeout** (timerVar As Object)
  Clears a timer set with setTimeout()
- **Close**
  Closes the current window
- **Closed** As Boolean
  Returns a Boolean value indicating whether a window has been closed or not
- **CloseEventListener**
  Closes an AddEventListenerOpen or AddEventListenerOpenAsync method
- **Confirm** (message As String) As Boolean
  Displays a dialog box with a message and an OK and a Cancel button
- **Focus**
  Sets focus to the current window
- **FrameElement** As BANanoObject
  Returns the iframe element in which the current window is inserted
- **Frames** As BANanoObject()
  Returns all ifram> elements in the current window
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **GetName** As String
  Sets or returns the name of a window
- **InnerHeight** As Double
  Returns the height of the window's content area (viewport) including scrollbars
- **InnerWidth** As Double
  Returns the width of a window's content area (viewport) including scrollbars
- **Length** As Int
  Returns the number of iframe elements in the current window

- **MoveBy** (x As Double, y As Double)
  Moves a window relative to its current position
- **MoveTo** (x As Double, y As Double)
  Moves a window to the specified position
- **Open** (URL As String, name As String, specs As String, replace As Boolean)
  Opens a new browser window

  For more info: https://www.w3schools.com/jsref/met_win_open.asp
- **Opener** As BANanoWindow
  Returns a reference to the window that created the window
- **OuterHeight** As Double
  Returns the height of the browser window, including toolbars/scrollbars
- **OuterWidth** As Double
  Returns the width of the browser window, including toolbars/scrollbars
- **PageXOffset** As Double
  Returns the pixels the current document has been scrolled (horizontally) from the upper left corner of the window
- **PageYOffset** As Double
  Returns the pixels the current document has been scrolled (vertically) from the upper left corner of the window
- **Parent** As BANanoWindow
  Returns the parent window of the current window
- **Print**
  Prints the content of the current window
- **Prompt** (message As String, defaultValue As String) As String
  Displays a dialog box that prompts the visitor for input
- **RemoveEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  Removes event handlers that have been attached with the addEventListener() method

  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies the event phase to remove the event handler from.

  true - Removes the event handler from the capturing phase
  false - Removes the event handler from the bubbling phase
- **RequestAnimationFrame** (module As Object, methodName As String) As Long
  The window.requestAnimationFrame() method tells the browser that you wish to perform an animation and requests
  that the browser call a specified function to update an animation before the next repaint.

  The method to call when it's time to update your animation for the next repaint. MUST be defined as:

  funcName(timeStamp As double)

- **ResizeBy** (width As Double, height As Double)
  Resizes the window by the specified pixels
- **ResizeTo** (width As Double, height As Double)
  Resizes the window to the specified width and height
- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

  ```
  RunMethod("myMethod", Array(Array(0,0), "Alain"))
  ```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

  ```
  RunMethod("myMethod", "Alain")
  ```
- **ScreenLeft** As Double
  Returns the horizontal coordinate of the window relative to the screen
- **ScreenTop** As Double
  Returns the vertical coordinate of the window relative to the screen
- **ScreenX** As Double
  Returns the horizontal coordinate of the window relative to the screen
- **ScreenY** As Double
  Returns the vertical coordinate of the window relative to the screen
- **ScrollBy** (x As Double, y As Double)
  Scrolls the document by the specified number of pixels
- **ScrollTo** (x As Double, y As Double)
  Scrolls the document to the specified coordinates
- **Self** As BANanoWindow
  Returns the current window
- **SetField** (field As String, value As Object)
  Sets a field value
- **SetInterval** (callback As Object, ms As Int) As Object
  Calls a function or evaluates an expression at specified intervals (in milliseconds)
- **SetName** (name As String)
  Sets or returns the name of a window
- **SetTimeout** (callback As Object, ms As Int) As Object
  Calls a function or evaluates an expression after a specified number of milliseconds
- **Stop**
  Stops the window from loading
- **Top** As BANanoWindow
  Returns the topmost browser window

18.33 BANanoXMLHttpRequest

Note: Use the more modern BANanoFetch instead.

Fields

- **DONE** As Int
- **HEADERS_RECEIVED** As Int
- **LOADING** As Int
- **OPENED** As Int
- **UNSENT** As Int

Functions

- **Abort**
  Abort the call
- **AddEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)
  callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
  useCapture: A Boolean value that specifies whether the event should be executed in the capturing or in the bubbling phase.

  true - The event handler is executed in the capturing phase
  false - The event handler is executed in the bubbling phase
- **AddEventListenerOpen** (eventName As String, params As Object)
  All the code between AddEventListenerOpen and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

  An AddEventListenerOpen MUST always be closed by an CloseEventListener!

  params: params it has to pass in the function() method

  ```
  req.AddEventListenerOpen("onreadystatechange", aEvt)
  ...
  req.CloseEventListener
  ```

  transpiles to:

  ```
  req.onreadystatechange = function(aEvt) {
  ...
  };
  ```
- **AddEventListenerOpenAsync** (eventName As String, params As Object)
  All the code between AddEventListenerOpenAsync and CloseEventListener is transpiled between those lines

  eventName: A String that specifies the name of the event. (Do not use the 'on' prefix!)

An AddEventListenerOpenAsync MUST always be closed by an CloseEventListener!

params: params it has to pass in the function() method

```
req.AddEventListenerOpenAsync("onreadystatechange", aEvt)
...
req.CloseEventListener
```

transpiles to:

```
req.onreadystatechange = async function(aEvt) {
...
};
```

- **CloseEventListener**
  Closes an AddEventListenerOpen or AddEventListenerOpenAsync method
- **GetAllResponseHeaders** As String
  Returns a string containing all header value pairs from the response.
- **GetField** (field As String) As BANanoObject
  Gets a field value
- **GetResponseHeader** (header As String) As String
  Returns the value for the specified header. Returns null if the headers do not contain a value for header.
- **GetResponseType** As String
  Determines the type returned by response:

```
'' (default) Same as 'text'
'text' String
'arraybuffer' ArrayBuffer
'blob' Blob
'document' Document
'json' Object
```

- **GetTimeout** As Double
  Gets the timeout value
- **GetWithCredentials** As Boolean
  Gets if credentials are used
- **Initialize**
  Creates a new XMLHttpRequest object.
- **Open** (method As String, url As String)
  Specifies the url to read from and the http method ('GET', 'POST', 'PUT', 'DELETE', etc) to use when reading the url.
- **Open2** (method As String, url As String, async As Boolean, user As String, password As String)
  Specifies the url to read from and the http method ('GET', 'POST', 'PUT', 'DELETE', etc) to use when reading the url.
  If async is true, the request will be asynchronous and you should provide an onload callback to be called when the read completes.
  In general, it is best to use the asynchronous request so the browser remains responsive while the request is in progress.
- **RemoveEventListener** (eventName As String, callbackMethod As Object, useCapture As Boolean)
  Removes event handlers that have been attached with the addEventListener() method

callbackMethod: Specifies the function to run when the event occurs. Use BANano.CallBackMethod()
useCapture: A Boolean value that specifies the event phase to remove the event handler from.

true - Removes the event handler from the capturing phase
false - Removes the event handler from the bubbling phase

- **Result** As Object
  Gets the result
- **RunMethod** (methodName As String, params As Object) As BANanoObject
  Runs a method on the JavaScript object.

  NOTE: the outer Array will be removed in the javascript.
  So if you want to pass an array, you have to add an extra array.

  e.g. if you want to pass "[0,0], "Alain", you actually have to pass [[0,0], "Alain"]

```
RunMethod("myMethod", Array(Array(0,0), "Alain"))
```

  If only one, non-Array param is passed, you can ignore this.

  e.g. this is valid

```
RunMethod("myMethod", "Alain")
```

- **Send**
  Sends the request
- **Send2** (obj As Object)
  Sends the request, The param obj can be a Document, Formdata, Blob, ArrayBuffer or a String
- **SetField** (field As String, value As Object)
  Sets a field value
- **SetRequestHeader** (header As String, value As String)
  sets the value of an HTTP request header. When using setRequestHeader(), you must call it after calling open(), but before calling send().
  If this method is called several times with the same header, the values are merged into one single request header.
- **SetResponseType** (rspType As String)
  Determines the type returned by response. Must be set to one of the following:

```
'' (default) Same as 'text'
'text' String
'arraybuffer' ArrayBuffer
'blob' Blob
'document' Document
'json' Object
```

  Must be set before readyState reaches LOADING.
- **SetTimeout** (timeoutValue As Double)
  Sets the timeout value

- **SetWithCredentials** (withCred As Boolean)
  Sets if credentials have to be used

Properties

- **OverrideMimeType** As String [write only]
  specifies a MIME type other than the one provided by the server to be used instead when interpreting the data being transferred in a request.
  This may be used, for example, to force a stream to be treated and parsed as "text/xml", even if the server does not report it as such.
- **ReadyState** As Int [read only]
  The current state of this. Will be one of UNSENT, OPENED, HEADERS_RECEIVED, LOADING, or DONE (see consts).
- **Response** As Object [read only]
  Returns the response from the server in the type specified by responseType. Only valid after the load event fires.
- **ResponseText** As String [read only]
  Returns the response from the server as a string. Only valid after the load event fires and if responseType is set to '' (the default) or 'text'.
- **ResponseXML** As Object [read only]
  returns a Document object
- **Status** As Int [read only]
  The http status code for the request. See statusText for a description of the code.
- **StatusText** As String [read only]
  A description of the status return code.
- **Upload** As BANanoObject [read only]
  Returns a BANanoObject associated with this XMLHttpRequest that can be used to track the upload status of the send() call.