Atome Framework - Documentation Étendue avec Exemples

Cette documentation exhaustive couvre l'ensemble des fonctionnalités du framework Atome, complétée par des exemples pratiques extraits de fichiers annexes. Elle est destinée à servir de référence complète pour les développeurs cherchant à maîtriser Atome.

Exemples et Guides d'Utilisation

Fichier: basic_understanding.rb

```
# frozen_string_literal: true


# The class Universe is used to retrieve some data needed for the atome framework
# per example you can retrieve the list of all available particles
puts Universe.particle_list
# this give at the date 14/11/2023 :
# or the list of all available atomes
puts Universe.atomes
# this give at the date 14/11/2023 :
# as well as the list of renderer available
puts Universe.renderer_list
# this give at the date 14/11/2023 :


# Universe hold all these instance variable :


# @counter is a integer  that store the total number of atome actually active for the current user on the current machine
# @atomes = is a hash that contains  a list all atomes actually active for the current user on the current machine,
# the key is the atome ID the value is the atome object itself
# atomes_specificities

# @atome_list is a hash that contains all atome's types available

# @particle_list is a hash that contains all particle's types available

# @renderer_list is an array that contai
```

Fichier: select_text.rb

```
#  frozen_string_literal: true



new({particle: :select})

t = text :hello

t.left(99)



t.edit(true)



b=box

b.touch(true) do

  puts t.data

  back_color = grab(:back_selection)

  text_color = grab(:text_selection)

  back_color.red(1)

  back_color.alpha(1)

  text_color.green(1)

  t.component({ selected: true })

end
```

Fichier: image.rb

```
#  frozen_string_literal: true
```

```
image(:red_planet)

image({path: 'medias/images/logos/atome.svg', width: 33})
```

Fichier: border.rb

```
# frozen_string_literal: true







b=box({id: :my_b_box, left: 150, top: 150})

b.shadow({

           id: :s1,

           # affect: [:the_circle],

           left: 9, top: 3, blur: 9,

           invert: false,

           red: 0, green: 0, blue: 0, alpha: 1

         })

border1= b.border({ thickness: 15, red: 1, green: 1, blue: 0, alpha: 1, pattern: :solid ,id: :border_1, inside: true})

wait 2 do

  b.remove(:border_1)

end

wait 1.5 do

 border({ thickness: 30, red: 1, green: 1, blue: 0, alpha: 1, pattern: :solid ,id: :poil, inside: true})

end



c = circle({ id: :the_circle, color: :green })

b = box({ left: 333, id: :the_box })
```

```
circle({ top: 190, width: 99, height: 99, id: :dont_break_too })

c2 = circle({ top: 190, width: 99, height: 99, id: :dont_break, color: :orange })

# let's add the border

wait 1 do

  c2.shadow({

              left: 9,

              top: 3,

              blur: 9,

              invert: false,

              option: :natural,

              red: 0, green: 0, blue: 0, alpha: 1
```

Fichier: over.rb

```
#  frozen_string_literal: true



b = box({ left: 666, color: :blue, smooth: 6, id: :the_box2 })

b.over(true) do

  b.color(:black)

  # puts "I'm inside"

end

b.over(:enter) do

  puts "in"

  puts "enter"

  b.width= b.width+30

  b.color(:yellow)

end

b.over(:leave) do
```

```
  b.height= b.height+10

  puts "out"

  puts "leave"

  # alert :out

  b.color(:orange)

end


#

t=b.text('touch me to stop over leave')

b.touch(true) do

  b.over({ remove: :enter })

  t.data('finished')

end
```

Fichier: int8.rb

```
# frozen_string_literal: true


# t = text({ int8: { english: :hello, french: :salut, deutch: :halo } })


# wait 1 do

#   t.language(:french)

#   wait 1 do

#     t.language(:english)

#     # data is updated to the latest choice

#     puts t.data

#     wait 1 do
#      t.data(:hi)
#    end
#  end
# end

Universe.translation[:hello] = { english: :hello, french: :salut, deutch: :halo }

b = box({ left: 155,
          drag: true,
          id: :boxy })

b.text({ data: :hello, id: :t1, position: :absolute, color: :black })
t2 = b.text({ data: :hello, id: :t2, left: 9, top: 33, position: :absolute })

Universe.language = :french
wait 2 do
  t2.refresh
  Universe.language = :deutch
  wait 2 do
  grab(:boxy).refresh
  end
end
```

Fichier: video.rb

```ruby
# frozen_string_literal: true



if Universe.internet

  v = video({ path: "http://commondatastorage.googleapis.com/gtv-videos-bucket/sample/ElephantsDream.mp4" })

else

  v = video(:video_missing)

end



v.touch(true) do

  v.play(true)

  wait 3 do

    v.play(66)

  end

end
```

Fichier: compute.rb

```ruby
# frozen_string_literal: true





c = circle({ height: 400, width: 200, top: 100, left:99, top: 79 })

b = c.box({ width: 200, height: 100, left: 280, top: 190, id: :my_box })

i= image(:red_planet)

c.touch(true) do

  c.fit({ value: 100, axis: :x })

end



puts '------'
puts "b.compute  left return the position on the screen of the item : #{b.compute({reference: c.id, particle: :left, metrics: :pixel})}"
puts "b.compute left : #{b.compute({ particle: :left })[:value]}, c left : #{b.left}"
puts "b.compute top :#{b.compute({ particle: :top })[:value]}, c top: #{b.top}"
puts  "i.compute width :#{i.compute({ particle: :width })[:value]}, i width: #{i.width}"
puts "i.compute height :#{i.compute({ particle: :height })[:value]}, i height: #{i.height}"
```

Fichier: read.rb

```
# frozen_string_literal: true



# works only in native for now
A.read('Cargo.toml') do |data|
  text "file content  :\n #{data}"
end

# if Atome.host == 'tauri'
#   JS.eval("readFile('atome','Cargo.toml')")
# else
#   puts 'nothing here'
# end
```

Fichier: account.rb

```
# # frozen_string_literal: true
b=box

b.touch(:down) do
    A.message({ action: :authentication, data: { table: :user, particles: { email: 'tre@tre.tre', password: 'poipoi' } } }) do
```

```
|response|
    alert "=> #{response}"
  end
end


#
#
# # # 1 login attempt


wait 1 do
   A.message({ action: :authentication, data: { table: :user, particles: { email: 'tre@tre.tre', password:  'poipoi' } } }) do
|response|
    alert "=> #{response}"
  end
  wait 1 do
     A.message({ action: :authentication, data: { table: :user, particles: { email: 'tre@tre.tre', password:  'poipoi' } } }) do
|response|
      alert "=> #{response}"
    end
  end
end
#
# 2 account creation attempt
# wait 1 do
#     A.message({ action: :account_creation, data: { email: 'tre@tre.tre', password: 'poipoi', user_id: 'Nico' }   }) do
|response|
#     puts response
#   end
#
# end


# string=hello
#
# puts JS.global.sha256(string.to_s)
```

Fichier: keyboard.rb

```
#  frozen_string_literal: true


t = text :hello
t.left(99)


t.edit(true)


t.keyboard(:press) do |native_event|
  event = Native(native_event)
  puts "press : #{event[:key]} :  #{event[:keyCode]}"
end


t.keyboard(:down) do |native_event|
  event = Native(native_event)
  if event[:keyCode].to_s == '13'
    event.preventDefault()
    t.color(:red)
  end
```

```
end

t.keyboard(:up) do |native_event|
  event = Native(native_event)
  puts "up!!"
end

t.keyboard(true) do |native_event|
  event = Native(native_event)
  puts " true => #{event[:keyCode]}"
  puts "true => #{event[:key]}"

end

# t.keyboard(:input) do |native_event|
#   event = Native(native_event)
#   puts event
# end

# t.keyboard(:keydown) do |native_event|
#   event = Native(native_event)
#   puts "down : #{event[:keyCode]}"
# end

c = circle({ top: 123, left: 0, width: 55, height: 55 })
# c2 = circle({ top: 123, left: 80, width: 55, height: 55 })
# c3 = circle({ top: 123, left: 150, width: 55, height: 55 })


c.touch(true) do
  text
```

Fichier: hypertext.rb

```
#  frozen_string_literal: true


b = box({ id: :the_html, color: :orange, overflow: :auto, width: :auto, height: :auto, left: 100, right: 100, top: 100, bottom:
100 })
# html_desc=<<STR
# <!DOCTYPE html>
# <html>
#     <head>
#         <title>Une petite page HTML</title>
#         <meta charset="utf-8" />
#     </head>
#     <body>
#         <h1 id='title' style='color: yellowgreen'>Un titre de niveau 1</h1>
#
#         <p>
#             Un premier petit paragraphe.
#         </p>
#
#         <h2>Un titre de niveau 2</h2>
#
#         <p>
#              Un autre paragraphe contenant un lien pour aller
#              sur le site <a href="http://koor.fr">KooR.fr</a>.
#          </p>
#     </body>
# </html>
# STR
html_desc = <<STR
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Com 1 Image</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 0; padding: 0; }
        he
```

Fichier: match.rb

```
#  frozen_string_literal: true


# def add_css_to_atomic_style(css)
#   style_element = JS.global[:document].getElementById('atomic_style')
#   text_node = JS.global[:document].createTextNode(css)
#   style_element.appendChild(text_node)
# end
#
# def convert_to_css(data)
#   conditions = data[:condition]

#   apply = data[:alterations]

#

#   # Convert the conditions

#   condition_strings = []

#

#   if conditions[:max]

#     condition_strings << "(max-width: #{conditions[:max][:width]}px)" if conditions[:max][:width]

#     condition_strings << "(max-height: #{conditions[:max][:height]}px)" if conditions[:max][:height]

#   end

#

#   if conditions[:min]

#     condition_strings << "(min-width: #{conditions[:min][:width]}px)" if conditions[:min][:width]

#     condition_strings << "(min-height: #{conditions[:min][:height]}px)" if conditions[:min][:height]

#   end

#

#   operator = conditions[:operator] == :and ? "and" : "or"

#

#   # Convert properties to apply

#   property_strings = []

#   app
```

Fichier: history.rb

```
#  frozen_string_literal: true



b = box({ id: :the_box })

b.data(:canyouwritethis)
```

```
b.rotate(33)

b.rotate(88)

b.rotate(99)

b.rotate(12)

b.rotate(6)

b.data

b.touch(true) do

  puts  b.history

  # b.data(:super)

  # b.data

  # box_data_write_history=b.history({ operation: :write, id: :the_box, particle: :data })

  # puts "get data write operation :  #{box_data_write_history}"

  # box_data_read_history=b.history({ operation: :read, id: :the_box, particle: :data })

  # puts "get data read operation :  #{box_data_read_history}"

end




# box_rotate_history=b.history({ operation: :write, id: :the_box, particle: :rotate })

# puts "get all all rotate write operation :  #{box_rotate_history}"

#

# # we check if an operation synced (that means saved on atome's server)

# puts "first rotate operation state  :  #{box_rotate_history[0]}"

#

# box_data_history=b.history({ operation: :write, id: :the_box, particle: :data })

# puts "get data write operation :  #{box_data_history}"

#
```

Fichier: category.rb

```ruby
# frozen_string_literal: true



# assign a class to atom object in the webview



t=text('touch the box')

b=box({ left: 12, id: :the_first_box })

b.category(:matrix)

b.touch(true) do

  b.remove({ category: :matrix})

  t.data= " category is : #{b.category}"

  wait 1 do

    b.category(:new_one)

    t.data= " category is : #{b.category}"

  end

end

t.data= " category is : #{b.category} "
```

Fichier: executor.rb

```ruby
# frozen_string_literal: true

def act_on(obj)

  obj.color(:red)

  obj.left(56)

end



def act_off(obj)
```

```
  obj.color(:blue)

  obj.left(33)

end




b = box({ left: 12, id: :the_first_box, top: 30 })


b.touch(true) do

   b.alternate({ width: 33, color: :red, height: 33 , smooth: 0 }, { width: 66, color: :orange, blur: 8}, { height: 66, color:
:green, smooth: 9, blur: 0})

end


c = circle({ left: 99 , top: 30})


c.touch(true) do

  alt = b.alternate(true, false)

  if alt

    c.color(:yellowgreen)

  else

    c.color(:orange)

  end

end




c2 = circle({ left: 333 , top: 30})
```

```
c2.touch(true) do
  b.alternate({  executor: {act_on: b}  }, { executor: {act_off: b}})
end
```

Fichier: display_bck.rb

```
#  frozen_string_literal: true




new({ particle: :display, render: false }) do |params|
  # alert type
  unless params[:items]
    params[:items] = { width: 200, height: 33 }
  end
  container_width = params[:width] ||= width
  container_height = params[:heigth] ||= height
  container_top = params[:top] ||= top
  container_left = params[:left] ||= left

  item_width = params[:items][:width] ||= 400
  item_height = params[:items][:height] ||= 50
  item_margin = params[:margin] ||= 3

  mode = params[:mode]

  case mode
  when :none
  when :custom
  when :list
    if params[:data].instance_of? Array
    elsif params[:data] == :particles
      list_id = "#{id}_list"
      unless grab(list_id)
        container = ''
        attach.each do |parent|
          container = grab(parent).box({ id: list_id, left: container_left, top: container_top, width: container_width, height: container_height, overflow: :auto, color: :black, depth: 0 })
          container.on(:resize) do |event|
            p
```

Fichier: drop.rb

```
#  frozen_string_literal: true




dragged = box({ left: 33,top: 333, width: 333,color: :orange, smooth: 6, id: :drop_zone })


dragged.drop(true) do |event|
  grab(event[:destination]).color(:white)
  grab(event[:source]).color(:black)
end


dragged.drop(:enter) do |event|
  grab(event[:destination]).color(:red)
end
```

```
dragged.drop(:leave) do |event|
  grab(event[:destination]).color(:gray)
end


dragged.drop(:activate) do |event|
  grab(event[:destination]).color(:yellow)
  grab(event[:source]).color(:cyan)
end



dragged.drop(:deactivate) do |event|
  grab(event[:destination]).color(:orange)
end
box({ left: 333, color: :blue,top: 222, smooth: 6, id: :the_box, drag: true })
box({ left: 333, color: :red,top: 180, smooth: 9, id: :the_box2, drag: true })


t=text({data: 'touch me to unbind drop enter'})
t.touch(true) do
  dragged.drop({ remove: :enter })
end
```

Fichier: apply.rb

```
# # frozen_string_literal: true


b=box({ left: 12, id: :the_first_box })
```

```
color({ id: :the_lemon, red: 1, green: 1 })

wait 1 do

  b.apply(:the_lemon)

end
```

Fichier: login.rb

```
# #  frozen_string_literal: true



# puts "current user: #{Universe.current_user}"

# human({ id: :jeezs, login: true })

#

# puts "current user: #{Universe.current_user}"

# wait 2 do

#   human({ id: :toto, login: true })

#   puts "current user: #{Universe.current_user}"

# end


puts 'ok1'


  # Vérification que les champs email et password ne sont pas envoyés vides :

        # if (email_text.data.nil? || email_text.data.strip.empty?) && (password_text.data.nil? ||
password_text.data.strip.empty?)

  #   puts "Veuillez renseigner votre adresse email et votre mot de passe."

  # elsif email_text.data.nil? || email_text.data.strip.empty?

  #   puts "Veuillez renseigner votre adresse email."

  # elsif password_text.data.nil? || password_text.data.strip.empty?
```

```
 #   puts "Veuillez renseigner votre mot de passe."

 # else



   mail = 'tretre'

   pass = 'poipoi'

   pass = Black_matter.encode(pass)




   # A.message({ action: :authentication, data: { table: :user, particles: {email: mail, password: pass
```

Fichier: editor.rb

```
# frozen_string_literal: true

box

dragger = box({ width: 333, height: 16, top: 0 })

back = box({ width: 333, height: 222, top: dragger.height })

body = back.box({ top: 0, width: '100%', height: '100%', component: { size: 12 }, id: :poil })

code_runner = dragger.circle({ left: 3, top: 3, width: 12, height: 12, color: :red })

code_closer = dragger.circle({ left: :auto ,right: 3, top: 3, width: 12, height: 12, color: :black })



body.editor({ id: :the_ed, code: "def my_script\n

  return 100\n

end", width: 333, height: 192, color: :lightgray, top: 0 })



def create_editor(code_id)

  js_code = <<~JAVASCRIPT

    var editor = CodeMirror.fromTextArea(document.getElementById("#{code_id}"), {

        lineNumbers: true,
        mode: "ruby",

        theme: "monokai"

    });

    editor.getWrapperElement().id = "atome_editor_#{code_id}";

    document.getElementById("atome_editor_#{code_id}").CodeMirrorInstance = editor;


  JAVASCRIPT

  JS.eval(js_code)

end



def set_code(code_id, content)

  js_code =
```

Fichier: js&ruby.rb

```
# frozen_string_literal: true



# JS to ruby example & ruby to js example



def my_ruby_meth(val)

  puts "=> rb_meth call from js: #{val}"

end





if Atome::host.to_s == 'web-opal'

  JS.eval("my_opal_js_fct('js fct call with an eval')")

  JS.global.my_opal_js_fct('js fct call directly')

elsif Atome::host.to_sym  == :pure_wasm
```

```
  JS.eval("my_ruby_wasm_js_fct('js fct call with an eval')")
end



"js code is in  js/atome/atome.js"
```

Fichier: getter.rb

```
#  frozen_string_literal: true


the_text = text({ data: 'hello for al the people in front of their machine jhgj  jg jgh jhg  iuuy res ', center: true, top: 120,
width: 77, component: { size: 11 } })
the_box = box({ left: 12 })
the_circle = circle({ id: :cc, color: :orange })
the_circle.image('red_planet')
the_circle.color('red')
the_circle.box({ left: 333, id: :the_c })


element({ id: :the_element })
the_view = grab(:view)
puts "views_shape's shape are : #{the_view.shape}"
puts "the_circle color is : #{the_circle.color}"
puts "the_text data is : #{the_text.data}"
puts "the_box left is : #{the_box.left}"
puts "the_circle particles are : #{the_circle.particles}"
```

Fichier: localstorage.rb

```
# frozen_string_literal: true
t=text("touch the box to erase localstorage, long touch on the box to stop historicize")
```

```
b=box({top: 66})

c=circle({top: 99})

c.touch(true) do

  c.left(c.left+99)

  # c.left=c.left+33

  # box

end

b.touch(true) do

  JS.eval('localStorage.clear()')

end



b.touch(:long) do

  b.color(:red)

  Universe.allow_localstorage = false



end
```

Fichier: audio.rb

```
#  frozen_string_literal: true



# audio tag

a = audio({ path: 'medias/audios/clap.wav', id: :basic_audio })

b=box({id: :playButton})

b.text(:audio_tag)

a.left(333)

b.touch(:down) do
```

```
  a.play(true)
end
```

### Web Audio

```
 audio({ path: 'medias/audios/clap.wav', id: :audioElement })
@audio_context = JS.eval('return new AudioContext()')
@audio_element = JS.global[:document].getElementById('audioElement')
@track = @audio_context.createMediaElementSource(@audio_element)

@gain_node = @audio_context.createGain()
@gain_node[:gain][:value] = 0.6

@track.connect(@gain_node)
@gain_node.connect(@audio_context[:destination])

def play_audio
  @audio_context[:resume].to_s if @audio_context[:state].to_s == 'suspended'
  @audio_element.play
end
b2=box({left: 166})
b2.text(:web_audio)
b2.touch(:down) do
  play_audio
end
```

```
# ######### wadjs

bb=box({left: 333})
bb.text(:wadjs)


# Initialize window.snare

init_code = "window.snare = new Wad({source : 'medias/audios/clap.wav'});"
JS.eval(init_code)

# De
```

Fichier: holder.rb

```
# frozen_string_literal: true
# holder is a particle that contain an atome so we use my_objet.holder.left(33)
# and it will move the atome contain in the holder particle to be manipulated
# it facilitate the access of some atome without being worried about their id
# this is mainly used int context of input , slider , etc...


# simple example
b=box({color: :black})

c=b.circle({width: 10, height: 10, color: :red})
```

```
b.holder(c)

wait 1 do

  b.holder.center(true)

end









# second example ( holder is build in the input molecule)

text({ left: 33, top: 33, data: 'data collected', id: :infos })



inp = A.input({ width: 166,

                trigger: :up,

                back: :orange,

                shadow: {

                  id: :s2,

                  left: 3, top: 3, blur: 3,

                  invert: true,

                  red: 0, green: 0, blue: 0, alpha: 0.9

                },

                text: :black,

                smooth: 3,

                left: 66,

                top: 33,
```

Fichier: table.rb

```
# frozen_string_literal: true




c = circle({ id: :my_cirle, color: :red, drag: true })

c.box({ left: 0, width: 22, height: 22, top: 65 })

c.touch(true) do

  alert :okk

end

m = table({ renderers: [:html], attach: :view, id: :my_test_box, type: :table, apply: [:shape_color],

             left: 333, top: 0, width: 900, smooth: 15, height: 900, overflow: :scroll, option: { header: true },

             component: {

               border: { thickness: 5, color: :blue, pattern: :dotted },

               overflow: :auto,

               color: "white",

               shadow: {

                 id: :s4,

                 left: 20, top: 0, blur: 9,

                 option: :natural,

                 red: 0, green: 1, blue: 0, alpha: 1

               },

               height: 50,

               width: 50,

               component: { size: 12, color: :black }

             },

             data: [

               { titi: :toto },
        { dfgdf: 1, name: 'Alice', age: 30, no: 'oko', t: 123, r:
```

Fichier: schedule.rb

```
# frozen_string_literal: true




def format_time
  time = Time.now
  {
    year: time.year,
    month: time.month,
    day: time.day,
    hour: time.hour,
    minute: time.min,
    second: time.sec
  }
end


# Exemple d'utilisation


t=text({data: "message here", id: :messenger})


schedule_task('every_minute_task', format_time[:year], format_time[:month], format_time[:day], format_time[:hour],
format_time[:minute], format_time[:second]+5, recurrence: :minutely) do
  t.data("every minute i change from :#{format_time}, now : #{format_time[:minute]} , #{format_time[:second]}")
end
```

Fichier: tagged.rb

```
# frozen_string_literal: true


b=box

b.circle({left: 0, top: 0, tag: {group: :to_grid}})

b.box({left: 120, top: 120, tag: {group: :from_grid}})

b.circle({left: 240, top: 240,  tag: {group: :from_grid}})

b.box({left: 330, top: 330,tag: {group: :to_grid}})

b.box({left: 330, top: 600,tag: :no_tag})



wait 1 do

  tagged(:group).each do |atome_id|

    grab(atome_id).color(:green)

    wait 1 do

      tagged({group: :to_grid }).each do |atome_id|

        grab(atome_id).color(:blue)

      end

    end

  end

end
```

Fichier: scheduler.rb

```
# frozen_string_literal: true


######## check


# Relaunch all tasks

relaunch_all_tasks


# Example: Schedule a task to run at a specific date and time

schedule_task('specific_time_task', 2024, 11, 12, 15, 12, 30) do

  puts "Task running at the specific date and time"

end


# Example: Schedule a task to run every minute

schedule_task('every_minute_task', 2024, 05, 12, 15, 12, 3, recurrence: :minutely) do

  puts "Task running every minute"

end


# Example: Schedule a task to run every Tuesday at the same time

schedule_task('weekly_tuesday_task', 2024, 11, 12, 15, 12, 30, recurrence: { weekly: 2 }) do

  puts "Task running every Tuesday at the same time"

end


# Example: Schedule a task to run every second Wednesday of the month at the same time

schedule_task('second_wednesday_task', 2024, 11, 12, 15, 12, 30, recurrence: { monthly: { week: 2, wday: 3 } }) do

  puts "Task running every second Wednesday of the month at the same time"

end
```

```
# Stop a task

# wait 133 do

#   puts 'stop'

#   stop_task
```

Fichier: fonts.rb

```
# frozen_string_literal: true



# add new font face

A.add_text_visual({ path: 'Roboto', name: 'Roboto-Black' })

A.add_text_visual({ path: 'Roboto', name: 'Roboto-Thin' })

A.add_text_visual({ path: 'Roboto', name: 'Roboto-LightItalic' })



# now applying it

first_text=text({ data: :hello, component: { size: 55, visual: 'Roboto-Thin' } })

wait 1 do

  text({ data: :hello, component: { size: 55, visual: 'Roboto-Black' } })

  wait 1 do

    first_text.component({visual: 'Roboto-LightItalic'})

  end

end
```

Fichier: help.rb

```
# frozen_string_literal: true




b = box({ drag: true })

A.help(:left) do
```

```
  english = 'the left particle is,used to position the atome on the x axis, click me to get an example'

  french = "'la particle left est utilisée  pour positionner l'atome sur l'axe x, click moi pour obtenir un exemple"



  t = text({ int8: { english: english, french: french },  width: 666 })

  t.touch(true) do

    t.delete(true)

    example(:left)

  end

end




  b.help(:left)
```

Fichier: shortcut.rb

```
# frozen_string_literal: true




box({id: :my_box})

circle({id: :my_circle, left: 333})

box({id: :red_box, left: 666, color: :red})






shortcut(key: :b,  affect: :all) do |key, object_id|

  puts "Key #{key} press on #{object_id}"

end

text({data: "Key 'b'  on :all", top: 0})
```

```ruby
shortcut(key: :e, option: :meta,affect: [:my_circle, :red_box]) do |key, object_id|

  puts "Key #{key}  press on #{object_id}"

end

text({data: "Key 'e' with Meta  on [:my_circle, :red_box]", top: 30, left: 0, position: :absolute})




shortcut(key: :j, option: :ctrl, affect: :all, exclude: [:my_circle, :my_box]) do |key, object_id|

  puts "Key #{key} with Ctrl press on #{object_id}"

end

text({data: "Key 'j' with Ctrl  on :all but [:my_circle, :my_box]", top: 50,left: 0, position: :absolute})
```

Fichier: rotate.rb

```ruby
# frozen_string_literal: true




b=box

i=b.image({path: 'medias/images/icons/hamburger.svg'})

wait 2 do

  i.rotate(22)

end
```

Fichier: above_below_before_after.rb

```ruby
# frozen_string_literal: true



b=box

margin = 12
```

```
b2=box({top: below(b, margin)})

b3=box({top: below(b2, margin)})

b4=box({top: below(b3, margin)})

box({top: below(b4, margin)})

i=0




b = circle(left: 333, top: 333)

margin = "2%"

# margin = 120

i = 0



while i < 10 do

  #below first params is the object after which we place the objet, the second the margin

  # here in percent and the third is the reference object used for the percent

  # b = circle({top: below(b, margin, grab(:view)), left: b.left})

  # b = circle({top: :auto,bottom: above(b, margin, grab(:view)), left: b.left})

  b = circle({top: b.top,left: after(b, margin, grab(:view))})

  # b = circle({left: :auto,right: before(b, margin, grab(:view))})

  i += 1

end
```

Fichier: aid.rb

```
# frozen_string_literal: true




# aid is used to provide an unique and persistent id for any atome
```

```
b=box({ left: 12, id: :the_first_box })


puts " atome aid is : #{b.aid}"

wait 1 do

  hook(b.aid).color(:red)

end
```

Fichier: selected.rb

```
# frozen_string_literal: true


t = text({ data: 'touch me to select all', id: :the_text })

b = box({ left: 12, id: :the_box })

c = circle({ left: 230, id: :the_circle, color: { blue: 1, id: :c1 } })

c.color({ green: 1, id: :c2 })

# to change default selection style

Universe.default_selection_style = { border: { thickness: 3, red: 1, green: 0, blue: 1, alpha: 1, pattern: :dotted } }


c.touch(true) do

  if c.selected

    c.selected(false)

  else

    # c.selected(true)

    # example of custom selection style

    c.selected({ shadow: { id: :titi,

                           left: 9, top: 3, blur: 9,

                           invert: false,

                           red: 0, green: 0, blue: 0, alpha: 1
    }, border: { id: :toto, thickness: 5, red: 1, green: 1, blue: 1, alpha: 1,

                pattern: :dotted, inside: true }

              })

  end

end


image({ path: 'medias/images/red_planet.png', id: :the__red_planet, top: 233 })


t.touch(true) do

  puts "1 current_user - #{grab(Unive
```

Fichier: aXionJeezs.rb

```
# #  frozen_string_literal: true

#

#

#

# c=circle

#

# c.touch(true) do

#

#

#   # c.message({data: {prompt: "cherche un fichier qui se nomme  capture  et ouvre le avec  l'application par defaut" ,

user_key:

'sk-proj-30NyTRt_3DAjrK_W7LQl-0csVjmC2rABcNPiTihFo1Ag-JWHPKlhqdtkt5qLTXWcwmwKTrZtxmT3BlbkFJ525DX2

eMWY5E6MUiTUnJw_-FjZ4SNQXcypP-uj2sKoW6gEmTfU2TAYqhYwTSxZvJUpj2xUDr8A'},  action: :axion }) do |result|

#   #   puts  "my command return: #{result}"

#   # end
```

```
#

#        # c.message({data: { prompt: "liste moi tous les fichiers et dossiers que tu trouve", user_key:
'sk-proj-30NyTRt_3DAjrK_W7LQl-0csVjmC2rABcNPiTihFo1Ag-JWHPKlhqdtkt5qLTXWcwmwKTrZtxmT3BlbkFJ525DX2
eMWY5E6MUiTUnJw_-FjZ4SNQXcypP-uj2sKoW6gEmTfU2TAYqhYwTSxZvJUpj2xUDr8A' },  action: :axion }) do |result|
#   #   puts  "my command return: #{result}"
#   # end
#
#   A.message({data: {prompt: "il faudrait ecrire un texte de remerciement pour un service en rendu addressé a mr albert
et mettre ce texte dans un fchier, et ouvre le fichier" ,
```

Fichier: shadow.rb

```
# frozen_string_literal: true


c = circle({ id: :the_circle, left: 122, color: :orange, drag: { move: true, inertia: true, lock: :start } })
c.color({ id: :col1, red: 1, blue: 1 })


 c.shadow({
             id: :s1,
             # affect: [:the_circle],
             left: 9, top: 3, blur: 9,
             invert: false,
             red: 0, green: 0, blue: 0, alpha: 1
           })


shadow({
          id: :s2,
          affect: [:the_circle],
          left: 3, top: 9, blur: 9,

          invert: true,

          red: 0, green: 0, blue: 0, alpha: 1

        })


c.shadow({

          id: :s4,

          left: 20, top: 0, blur: 9,

          option: :natural,

          red: 0, green: 1, blue: 0, alpha: 1

        })


wait 2 do

  c.remove(:s4)

  wait 2 do

    c.remove({ all: :shadow })

  end

end




the_text = text({ data: 'text with shadow!', center: true, top: 222, width: 777, component: { size: 66 }, id: :my_text })




the_text.shadow({

          id: :my_shadow,

          left:
```

Fichier: blocks.rb

```ruby
# frozen_string_literal: true





a = application({
                  id: :arp,
                  margin: 3,
                })

page1_code = lambda do
  b = box({ id: :ty, left: 90, top: 90, color: :black })
  b.touch(true) do
    b.color(:red)
  end
end

page1 = {
  run: page1_code,
  menu: false,
  id: :page1,
  color: { red: 0.5, green: 0.5, blue: 0.5 },
  name: :accueil,
  # footer: { color: :green, height: 22 },
  header: { color: { red: 0.3, green: 0.3, blue: 0.3 }, height: 90, shadow: { blur: 12, left: 0, top: 0 } },

}
```

```
a.page(page1)

c = a.show(:page1)

c.color(:orange)

header = grab(:arp_content_header)

header.color(:orange)

# header.height(66)

# header.subs({ "contact" => { "width" => "33%" }, "project" => { "width" => "33%" }, "calendar" => { "width" => "33%" } })



bloc_to_add = { height: 33, color: :cyan }

bloc_to_add2 = { height: 99, color: :blue }

bloc_to_add3 = { height: 133, color: :red }

bloc_to_add4 = { height: 33, color: :gray }

###########@

grab(:page1).blocks({ dire
```

Fichier: atome_particle_validation.rb

```
# frozen_string_literal: true



# we check if the atome or the particle we want to create has already been defined in atome



new ({ atome: :image })



new ({ particle: :left })
```

Fichier: text.rb

```
# frozen_string_literal
```

```
t2 = text({ data: ['this is ', :super, { data: 'cool', color: :red, id: :new_one }], component: { size: 33 }, left: 120 })

the_text = text({  data: 'hello for al the people in front of their machine', center: true, top: 120, width: 77, component: {
size: 11 } })
```

Fichier: refresh.rb

```
# frozen_string_literal: true


b = box({ top: 166, data: :hello })

c=color({ id: :col1, red: 1, blue: 1})


b.instance_variable_set("@top", 30)

b.instance_variable_set("@apply", [c.id])

b.instance_variable_set("@path", './medias/images/red_planet.png' )


b.instance_variable_set("@smooth", 30)

wait 1 do

  b.refresh

  b.instance_variable_set("@left", 300)

  wait 1 do

    b.refresh

    b.instance_variable_set("@type", :text)

    wait 1 do

      b.refresh

      b.instance_variable_set("@type", :image)

      wait 1 do

        b.refresh
      end

    end

  end

end

i=image(:green_planet)

# alert i.path

i.instance_variable_set("@path", './medias/images/red_planet.png')

wait 2 do

  i.refresh

  # i.path'./medias/images/red_planet.png'

end




#

# b.instance_variable_set("@left", 300)

# b.instance_variable_set("@top", 400)

# # b.instance_variable_set("@width", 150)

#

# # b.instance_variable_set("@smooth", 9)

# # new({particle: :tototo})

#

# wait 1 do

#   b.refresh

#   # b.instance_variable_set("@typ
```

Fichier: media_video_thumbnail.rb

```
# frozen_string_literal: true
```

```
video({id: :video, path: 'medias/videos/avengers.mp4', width: 300, height: 222 })

waveform_container=box({id: 'thumbnails-container', top: 190,width: 666, height: 39, color: :gray})

waveform_container.draw({width: 666, height: 33,  id: :thumbnails})

waveform_container.box({id: 'progress', width: 3, height: '100%', color: :red})



box({id: :file, top: 666, left: 12, width: 300, height: 40, smooth: 9, color: { red: 0.3, green: 0.3, blue: 0.3 } })

box({id: :load_file, top: 777, left: 12, width: 300, height: 40, smooth: 9, color: { red: 0.3, green: 0.3, blue: 0.3 } })



  JS.eval <<~JS


const video = document.getElementById('video');


  const thumbnailsCanvas = document.getElementById('thumbnails');

  const thumbnailsCtx = thumbnailsCanvas.getContext('2d');

  const progress = document.getElementById('progress');

  // const loadFileButton = document.getElementById('load-file');

  // const fileInput = document.getElementById('file-input');

  let isDragging = f
```

Fichier: opacity.rb

```
# frozen_string_literal: true


image({id: :planet,path: 'medias/images/red_planet.png', width: 66,height: 66,  left: 33, top: 33})
```

```
b=box({width: 66, height: 66, color: :yellowgreen})

  wait 1 do

    b.opacity(0.3)
  end
```

Fichier: input.rb

```
# frozen_string_literal: true
t = text({ left: 33, top: 33, data: 'data collected', id: :infos })
b=box({drag: true, id: :the_b})
# Important to trigger on 'return' add the parameter :  {trigger: :return}
inp=b.input({ width: 166,
              trigger: :up,
              back: :orange,
              shadow: {
                id: :s2,
                left: 3, top: 3, blur: 3,
                invert: true,
                red: 0, green: 0, blue: 0, alpha: 0.9
              },
              component: {size: 8},
              text: { color: :black , top: 5, left: 6},
              smooth: 3,
              left: 66,
              top: 33,
              # height: 8,
            default: 'type here'
          }) do |val|

  grab(:infos).data(val)
end

inp.top(12)

  wait 1 do
    inp.width(666)
    wait 1 do
      inp.holder.data('new data')
    end
end



c=circle({top: 99})
c.touch(true) do
  alert b.fasten
end
```

Fichier: unit.rb

```
# frozen_string_literal: true
```

```
box({ left: 50, id: :the_first_box,  color: :blue })

b1=box({ left: 12, id: :the_second_box ,top: 3, unit: {left: '%', width: '%'}, color: :red})

box({ left: 550, id: :the_third_box , unit: {left: :px}, color: :green})

wait 2 do

  b1.unit({left: 'cm'})

  b1.unit({top: 'cm'})

  # b1.unit[:top]='cm'

  puts b1.unit

end
```

Fichier: debug.rb

```
# # frozen_string_literal: true



class Atome

  class << self

    def monitoring(atomes_to_monitor, particles_to_monitor, &bloc)

      atomes_to_monitor.each do |atome_to_monitor|

        particles_to_monitor.each do |monitored_particle|

          # storing original method

          original_method = atome_to_monitor.method(monitored_particle)

          # redefine the method

          atome_to_monitor.define_singleton_method(monitored_particle) do |*args, &proc|

            # monitoring bloc before calling original method

            value_before = atome_to_monitor.instance_variable_get("@#{monitored_particle}")

            if args.empty?
      # args = nil
    else
      if monitored_particle == :touch
        # instance_variable_set("@#{monitored_particle}", { tap: args[0] })
        # instance_variable_set("@#{monitored_particle}_code", { touch: proc })
        # args = { tap: args[0] }
      elsif monitored_particle == :apply
```

Fichier: allow_copy.rb

```
# frozen_string_literal: true



t=text(:hello)
t.edit(true)
b=box({left: 99})

b.touch(true) do
  allow_copy(true)
  allow_right_touch(true)
end
```

Fichier: buttons.rb

```
# frozen_string_literal: true

box({color: :gray, width: 666, height: 666})
box({ id: :the_box, drag: true, color: { alpha: 2 } })



but =buttons({
     id: "my_menu",
     depth: 9999,
     attach: :the_box,
     inactive: { text: { color: :gray }, width: 66, height: 12, spacing: 3, disposition: :horizontal,
           color: :orange, margin: { left: 33, top: 12 } },
     active: { text: { color: :white, shadow: {} }, color: :blue, shadow: {} },
    })

c = text({ top: 99, left: 99, data: 'add buttons' })

c.touch(:down) do
 but.add_button(new_button: {
  text: :button1,
  code: lambda { puts :button1_touched }
 })
  but.add_button(new_button2: {
   text: :button2,
   code: lambda { puts :button1_touched }
  })
 but.add_button(new_button3: {
  text: :button3,
  code: lambda { puts :button1_touched }
 })
```

```
  wait 0.2 do

    grab(:my_menu).remove_menu_item(:new_button2)

  end


  end
```

```
# TODO: remove menu_item ,reset_menu,
```

Fichier: to_percent.rb

```
# frozen_string_literal: true


b=box

t=text({width: 66, left: 99,top: 66, data: "touch the bow and resize the window"})


b.touch(true) do

  b.width(t.to_percent(:width))

  b.left(t.to_percent(:left))

end
```

Fichier: actor&role.rb

```
# frozen_string_literal: true
```

```
bbb = box({left: 66})

ccc = bbb.circle(id: :the_circle)



bbb.role(:first)

bbb.role(:second)

bbb.delete(:left)

bbb.delete(:role)



bbb.role(:fourth)

bbb.role(:five)

bbb.role({ remove: :last })



bbb.actor({ the_circle: :buttons })

bbb.actor({ the_circle: :dummy })

bbb.actor({ the_circle: :menu })



bbb.actor({ remove: { the_circle: :dummy } })



puts "1 ===> #{bbb.role}"

puts "2 ===> #{bbb.actor}"

puts "3 ===> #{ccc.role}"
```

Fichier: to_px.rb

```
# frozen_string_literal: true
```

```ruby
view_width = parent_found.to_px(:width)

view_height = parent_found.to_px(:height)




text({data: "view width in px : #{view_width}, height: #{view_height}" })
```

Fichier: vr.rb

```ruby
# frozen_string_literal: true



vr({width: 700,height: 390,path: 'medias/images/puydesancy.jpg', id: :tutu})
```

Fichier: find.rb

```ruby
# frozen_string_literal: true



new({ particle: :find }) do |params|

  puts params



end



b = box

# alert 'use category top assign class then port hybrid.html to atom'

16.times do |index|

  width_found = b.width

  b.duplicate({ left: b.left + index * (width_found + 45), top: 0, category: :matrix })

end



def calculate_dynamic_value(particle)

  500
```

```
end

b.find(
  condition: [{
              operator: :and,
              rules: [
                {
                  property: :left,
                  comparison: :gt,
                  value: { type: :dynamic, content:[22] }
                },
                {
                  operator: :or,
                  rules: [
                    {
                      property: :width,
                      comparison: :eq,
                      value: { type: :static, content: 50 }
                    },
                    {
                      property: :width,
                      comparison: :eq,
```

Fichier: target.rb

```
#  frozen_string_literal: true
```

```
b = box({ left: 333, color: :blue, smooth: 6, id: :the_box2 })


t = text({ id: :the_text, data: 'touch the box and wait!' })


exec_code=lambda do


  wait 2 do
    t.data('it works!! ')
  end


end
b.code(:hello) do
  circle({ left: rand(333), color: :green })
end
b.run(:hello)
b.touch(:tap) do
  {
    color: :cyan,
    target: { the_text: { data: :super! } },
    run: exec_code
  }
end
```

Fichier: flash.rb

```
# frozen_string_literal: true
```

```
wait 1 do
  flash(:msg)
end
```

Fichier: smooth.rb

```
# frozen_string_literal: true


b = box({ width: 333, left: 333 })
b.smooth(9)


wait 2 do
  b.smooth([33, 2, 90])
end
```

Fichier: sub_atome_manipulation.rb

```
# frozen_string_literal: true



b=box({id: :the_box})
b.text({id: :the_text, left: 90, top: 30, data: :ok})
b.text({id: :the_text2, left: 190, top: 30, data: :hello})


wait 1 do
  b.text.each_with_index do |el, _index|
    grab(el).left(30)
  end
  # b.text.left(30)
  wait 1 do
    b.text.color(:white)
    b.text.each_with_index do |el, index|
      grab(el).left(30+30*index)
    end
    b.color(:black)
  end
end
```

Fichier: tick.rb

```
# frozen_string_literal: true


# tick allow you to automatise any action counting
# it can be added into any new created particle ex: here a dummy


new({ particle: :dummy }) do |_p|
  tick(:dummy )
end


new({ particle: :dummy2 }) do |_p|
  tick(:dummy2 )
end


a=box
a.dummy(:hi)


puts a.tick[:dummy]
```

```
a.dummy(:ho)
puts a.tick[:dummy]


a.dummy2(:ho)
puts a.tick[:dummy2]


c=circle({left: 99})


c.touch(true) do
  c.tick(:my_counter)
  puts  c.tick[:my_counter]
end


bb=box({left: 333})


bb.touch(true) do
  if   bb.tick(:my_counter)%2 == 0
    bb.color(:red)
  else
    bb.color(:blue)
  end
end
```

Fichier: touch.rb

```
#  frozen_string_literal: true


b = box({ left: 333, color: :blue, smooth: 6, id: :the_box2 })
```

```ruby
t = text({ id: :the_text, data: 'type of touch : ?' })


t.touch(:down) do |event|

  puts :down

  puts event[:pageX]

  puts event[:pageY]

  b.touch({ remove: :down })

  t.data('down removed !! ')

end


touch_code = lambda do

  b.color(:red)

  puts 'box tapped'

end

b.touch(tap: true, code: touch_code)


b.touch(:long) do

  { color: :cyan }

  t.data('type of touch is : long ')

end


b.touch(:up) do

  t.data('type of touch is : up ')

  b.color(:orange)

end
```

```
b.touch(:down) do

  t.data('type of touch is : down ')

  b.color(:white)

end


b.touch(:double) do

  t.color(:red)

  t.data('type of touch is : double ')

  b.color(:yellowgreen)

end
```

Fichier: group.rb

```
# frozen_string_literal: true


text({ id: :the_text,data: 'Touch me to group and colorize', center: true, top: 120, width: 77, component: { size: 11 } })

box({ left: 12, id: :the_first_box })

the_circle = circle({ id: :cc, color: :yellowgreen, top: 222 })

the_circle.image({path:  'medias/images/red_planet.png', id: :the__red_planet })

the_circle.color('red')

the_circle.box({ left: 333, id: :the_c })


element({ id: :the_element })


the_view = grab(:view)
```

```
color({ id: :the_orange, red: 1, green: 0.4 })

color({ id: :the_lemon, red: 1, green: 1 })

the_group = group({ collect: the_view.shape })



wait 0.5 do

  the_group.left(633)

  wait 0.5 do

    the_group.rotate(23)

    wait 0.5 do

      the_group.apply([:the_orange])

      the_group.blur(6)

    end

  end

end

puts the_group.collect

grab(:the_first_box).smooth(9)

grab(:the_text).touch(true) do

bibi=box({left: 555})

the_group2= group({ collect: [:the_c,:the_first_box, :the_text, :cc , bibi.id] })

the_group2.top(55)

# puts we remove the circl
```

Fichier: css.rb

```
# frozen_string_literal: true



b=box({right: 45, left: :auto})
```

```
b.css[:style][:border] = '2px solid yellow'

puts  b.css[:style][:border]

puts b.css
```

Fichier: blur.rb

```
# frozen_string_literal: true



b=circle({left: 333})

b.blur(6)



image(:red_planet)

b2=box({color: {alpha: 0.1, red: 1, green: 0, blue: 0.2}, left: 99, top: 99, width: 99, height: 99})

b2.drag(true)

b2.border({ thickness: 0.3, color: :gray, pattern: :solid })

b2.smooth(12)

b2.shadow({

      invert: true,

      id: :s4,

      left: 2, top: 2, blur: 9,

      # option: :natural,

      red: 0, green: 0, blue: 0, alpha: 0.3

     })



b2.shadow({

      # invert: true,

      id: :s5,

      left: 2, top: 2, blur: 9,
          # option: :natural,
          red: 0, green: 0, blue: 0, alpha: 0.6
        })
b2.blur({affect: :back, value: 15})
```

Fichier: calendar.rb

```
# frozen_string_literal: true


new(molecule: :calendar) do |params, &bloc|


  cal = box(params)
  cal.resize(true)
  cal_id = cal.id


  #########################  create calendar #########################
  cal_name = cal_id
  calendar = <<~JAVASCRIPT
      window.#{cal_name} = new tui.Calendar('##{cal_id}', {
      defaultView: 'month',
      usageStatistics: false,
      month: {
       startDayOfWeek: 0,
      },
      week: {
       showTimezoneCollapseButton: true,
       timezones: [{ timezoneOffset: 0, displayLabel: 'UTC', tooltip: 'UTC' }],
      },
  });



  JAVASCRIPT

  JS.eval(calendar)

  ######################## Update view methode ###########################



  cal.define_singleton_method(:view) do |view_mode|
    update_calendar = <<~JAVASCRIPT
        function changeCalendarView(view) {
       const validViews = ['day', 'week', 'month'];
       if (!validViews.includes(view)) {
        console.error(`Vue non valide: ${view}. Les vues valides sont: $
```

Fichier: chronology.rb

```
# frozen_string_literal: true


new({molecule: :chronology}) do |params|
  chr=box({width: '100%', height: 333, color: :white, smooth: 9})
  chr_id=chr.id
```

```
 JS.eval <<~JS
// Create a dataset with items
        var items = new vis.DataSet({
            type: { start: 'ISODate', end: 'ISODate' }
        });

        // Add items to the DataSet
        items.add([
            {id: 1, content: 'item 1<br>start', start: '2014-01-23'},
            {id: 2, content: 'item 2', start: '2014-01-18'},
            {id: 3, content: 'item 3', start: '2014-01-21', end: '2014-01-24'},
            {id: 4, content: 'item 4', start: '2014-01-19', end: '2014-01-24'},
            {id: 5, content: 'item 5', start: '2014-01-28', type: 'point'},
            {id: 'kjhdkfjghdkjfgh', content: 'item 6', start: '2014-01-26'}
        ]);

        // Log changes to the console
        items.on('*', function (event, properties) {
            console.log(event, properties.items);
        });

    var container = docume
```

Fichier: animation.rb

```
# # frozen_string_literal: true
#
# bb = text({ id: :the_ref, width: 369, data: "touch me!" })
# bb.color(:orange)

# box({ id: :my_box, drag: true })

# c = circle({ id: :the_circle, left: 222, drag: { move: true, inertia: true, lock: :start } })

# c.shadow({ renderers: [:html], id: :shadow2, type: :shadow,

#            attach: [:the_circle],

#            left: 3, top: 9, blur: 19,

#            red: 0, green: 0, blue: 0, alpha: 1

#          })

#

# Atome.new(animation: { renderers: [:browser], id: :the_animation1, type: :animation, attach: [],fasten: []})

# aa = animation({

#                  targets: %i[my_box the_circle],

#                  begin: {

#                    left_add: 0,

#                    top: :self,

#                    smooth: 0,

#                    width: 3

#                  },

#                  end: {

#                    left_add: 333,

#                    top: 299,

#                    smooth: 33,

#                    width: :the_ref

#                  },

#
```

Fichier: svg_img_to_vector.rb

```
# frozen_string_literal: true



grab(:black_matter).image({ path: 'medias/images/icons/color.svg', id: :atomic_logo, width: 33, left: 333 })

img=vector({ width: 333, height: 333, id: :my_placeholder })

A.fetch_svg({ source: :atomic_logo, target: :my_placeholder })

wait 2 do

  img.color(:cyan)

end

# grab(:atomic_logo).delete(true)
```

Fichier: particles.rb

```
# frozen_string_literal: true



b = box({ left: 777 })

puts "b contain the following particles : #{b.particles}"
```

Fichier: on_the_fly_ruby_code_loading.rb

```
#  frozen_string_literal: true







b=box({color: :red})

b.touch(true) do

  JS.eval('loadFeature()') # found in atome.js file

end
```

Fichier: map.rb

```ruby
#  frozen_string_literal: true



# new({ atome: :map, type: :hash })


# new({particle: :longitude}) do |params, _user_proc|
#   render(:map, {longitude: params })
#   params
# end
#
# new({particle: :latitude}) do |params, _user_proc|
#   render(:map, {latitude: params })
#   params
# end


# new({ method: :map, renderer: :html, type: :int }) do |params, _user_proc|
#   latitude_found=@latitude
#   longitude_found=@longitude
#   location_hash={longitude: longitude_found, latitude: latitude_found}.merge(params)
#   html.location(location_hash)
# end




m=map({id: :hgfh, longitude: 55.9876876, latitude: 33.987687, width: 333, height: 222,})
# wait 3 do
p=map({id: :poilo, location: :auto, width: 333, height: 333, top: 333 , left: 333, zoom: 3})
```

```
# end

b=box

b.touch(true) do

  m.zoom(33)

  # p.zoom(3)

  # wait 2 do

    p.pan({ left: 370, top: 190 })

  # end

end



# m=map({id: :locator, location: :auto})



# alert m.longitude
```

Fichier: example.rb

```
# frozen_string_literal: true




b = box({ drag: true })



A.example(:left) do

  english = 'here is an example, touch me to get some help, or click the code to exec'

  french = "voici un example, click moi pour de l'aide, ou  clicker le code pour l'executer"

  code = <<STR

b=box
```

```
puts b.left

b.left(155)

puts b.left

STR

  example = text({ int8: { english: english, french: french }, language: :english, width: 666 })

  code_text = text({ int8: { english: code }, language: :english, width: 666, top: 33 })

  example.touch(true) do

    example.delete(true)

    help(:left)

  end

  code_text.touch(true) do

    eval(code)

  end

end



  b.example(:left)
```

Fichier: media_audio_thumbnail.rb

```
# frozen_string_literal: true




audio({id: :audio})

waveform_container=box({id: 'waveform-container', width: 666, height: 270, color: :gray})

waveform_container.draw({width: 666, height: 270,  id: :waveform})

waveform_container.box({id: 'progress', width: 3, height: '100%', color: :red})
```

```
draw({width: 666, height: 270, top: 280,color: :orange, id: :realtime})



box({id: :load_file, top: 666, left: 12, width: 300, height: 40, smooth: 9, color: { red: 0.3, green: 0.3, blue: 0.3 } })

box({id: :file_input, top: 777, left: 12, width: 300, height: 40, smooth: 9, color: { red: 0.3, green: 0.3, blue: 0.3 } })




JS.eval <<~JS

const audio = document.getElementById('audio');

  const waveformCanvas = document.getElementById('waveform');

  const waveformCtx = waveformCanvas.getContext('2d');

  const realtimeCanvas = document.getElementById('realtime');

  const realtimeCtx = realtimeCanvas.getContext('2d');

  const progress = document.getElementById('progress');

  const loadFileButton = document.g
```

Fichier: grip.rb

```
# frozen_string_literal: true



b=box

b.circle({role: :header, left: 55, id: :first_one})

b.text({role: [:action], data: "hello", top: 90})

b.box({role: :header, left: 155, id: :second_one})
```

```
puts"header grip : #{ b.grip(:header)}"

puts "last header grip #{b.grip(:header).last}"
```

Fichier: on_resize.rb

```
# frozen_string_literal: true



# please note that whatever the atome resize will return the size of the view!

view = grab(:view)

view.on(:resize) do |event|

  puts "view size is #{event}"

end



b=box

b.touch(true) do

  view.on(:remove)

end




c=circle({ left: 333 })



c.touch(true) do

  view.on(:resize) do |event|

    puts "Now size is : #{event}"

  end

end
```

Fichier: text_align.rb

```
# frozen_string_literal
```

```
text({data: :centering,align: :center, width: 180, top: 33, left: 0, position: :absolute, color: :red})
```

Fichier: atome.rb

```
# frozen_string_literal: true



Atome.new( { renderers: [:html], attach: :view,id: :my_test_box, type: :shape, apply: [:shape_color],
             left: 120, top: 0, width: 100, smooth: 15, height: 100, overflow: :visible, fasten: [], center: true
           })
```

Fichier: site.rb

```
# frozen_string_literal: true



# new(application: {name: :compose })
# new(application:  :compose ) do |params|
#   alert params
# end
s=application({ name: :home })



# alert s.class
# alert "a.class : #{a.class}"
s.page(:hello)
# grab(:toto).color(:cyan)
#
# def layout
#   compose_back=box
#
#   compose_back.color({ alpha: 0 })
#   media_reader=compose_back.box({left: 99, width: 250, height: 250, top: 99})

#   viewer_1=compose_back.box({left: 360, width: 250, height: 250, top: 99})

#   viewer_2=compose_back.box({left: 690, width: 250, height: 250, top: 99})

#   timeline=compose_back.box({left: 99, width: 250, height: 250, top: 399})

#   login=compose_back.text(:log)

#   login.touch(true) do

#     compose_back.delete(true)

#     # grab(:view).clear(true)

#     form

#   end

#

# end

#

# def form

#   form1=box

#   form1.text(:login)

#

#   form1.touch(true) do

#     form1.delete(true)

#     layout

#   end

#

# end

# form
```

Fichier: repeat.rb

```
#  frozen_string_literal: true
```

```
c=circle({width: 66, height: 66})

t1=c.text({id: :first, data: 0, left: 28})


first_repeater=repeat(1, repeat = 99) do |counter|

  t1.data(counter)

end



c.touch(true) do

  stop({ repeat: first_repeater })

  t1.data(:stopped)

end



cc=circle({width: 66, height: 66, left: 90 })

t2=cc.text({id: :second, data: 0, left: 28})


# # alert first_repeater

my_repeater=repeat(1, repeat = 9) do |counter|

  t2.data(counter)

end

#


#

cc.touch(true) do
```

```
  stop({ repeat: my_repeater })

  t2.data(:stopped)

end



# use Float::INFINITY to infinite repeat
```

Fichier: resize.rb

```
# frozen_string_literal: true



m = shape({ id: :the_shape, width: 333, left: 130, top: 30, right: 100, height: 399, smooth: 8, color: :yellowgreen, })

m.drag(true)

m.on(:resize) do |event|

  puts event[:dx]

end



m.resize({ size: { min: { width: 90, height: 190 }, max: { width: 300, height: 600 } } }) do |event|

  puts "width is  is #{event[:rect][:width]}"

end



t=text({data: ' click me to unbind resize'})

t.touch(true) do

  t.data('resize unbinded')

  m.resize(:remove)

end



c=circle({left: 99, top: 99, right: 100, height: 99})
```

```
c.touch(true) do
  m.resize({ size: { min: { width: 90, height: 190 }, max: { width: 300, height: 600 } } }) do |event|
    puts "ooooooo"
  end
  m.on(:resize) do |event|
    puts 'yes'
  end
end
```

Fichier: drop_down_list.rb

```
# frozen_string_literal: true



data_f = %w[initiate suspect prospect abandoned finished archived]


d_d_l = box({ id: :the_ddl, width: 160 })
d_d_l.touch(:down) do
  grab(:view).drop_down({ data: data_f, }) do |params|
    d_d_l.clear(true)
    d_d_l.text(params)
  end
end
```

Fichier: file.rb

```
#  frozen_string_literal: true


# see import for drag and drop import
```

```
b = box({ drag: true })

b.import(true) do  |content|

  puts "add code here, content:  #{content}"

end
```

Fichier: dig.rb

```
# frozen_string_literal: true




c = circle({ height: 400, width: 200, top: 100, left: 0, top: 100 , id: :the_circle})

b = c.box({ width: 200, height: 100, left: 600, top: 200, id: :my_box })

c.circle({ width: 200, height: 100, left: 120, top: -80, id: :my_text, data: :hi })

b.circle({ color: :yellow, width: 55, height: 88, left: 100 })

b.box




atome_founds = c.dig

puts "dig allow to retrieve all fasten atomes recursively,
```

it return a table of ID including the ID of the parent (here : :the_circle) :\n#{atome_founds}"

Fichier: duplicate.rb

```
# frozen_string_literal: true


# new({ particle: :duplicate, store: false }) do |params|
#   if @duplicate
#     copy_number = @duplicate.length
#   else
#     copy_number = 0
#   end
#
#   new_atome_id = "#{@id}_copy_#{copy_number}"
#   new_atome = Atome.new({ type: @type, renderers: @renderers, id: new_atome_id })
#
#   fasten_atomes = []
#   fasten_found = fasten.dup
#   particles_found = instance_variables.dup
#
#   particles_found.delete(:@history)
#   particles_found.delete(:@callback)
#   particles_found.delete(:@duplicate)
#   particles_found.delete(:@touch_code)
#   # touch_code=instance_variable_get('@touch_code')
#   particles_found.delete(:@html)
#   particles_found.delete(:@fasten)
#   particles_found.delete(:@id)
#   params[:id] = new_atome_id

#   fasten_found.each do |child_id_found|

#     child_found = grab(child_id_found)

#     if child_found

#       new_child = child_found.duplicate({})

#       fasten_atomes << new_child.id

#     end

#   end

#   particles_found.each d
```

Fichier: universe.rb

```
# frozen_string_literal: true



puts "atomes : #{Universe.atomes}"

puts "user_atomes : #{Universe.user_atomes}"

puts "particle_list : #{Universe.particle_list}"

puts "users : #{Universe.users}"

puts "current_machine : #{Universe.current_machine}"

puts "internet connected : #{Universe.internet}"
```

Fichier: www.rb

```
#  frozen_string_literal: true



b = box



b.www({ path: "https://www.youtube.com/embed/usQDazZKWAk", left: 333 })



Atome.new(
            renderers: [:html], id: :youtube1, type: :www, attach: :view, path:
"https://www.youtube.com/embed/fjJOyfQCMvc?si=lPTz18xXqIfd_3Ql", left: 33, top: 33, width: 199, height: 199,


)
```

Fichier: edit.rb

```
#  frozen_string_literal: true


new({particle: :select})

t = text :hello

t.left(99)


t.edit(true)


b=box

b.touch(true) do

  puts t.data

  t.component({ selected: true })

end


# # frozen_string_literal: true

#

# c = circle({ id: :the_circle, left: 122, color: :orange, drag: { move: true, inertia: true, lock: :start } })

# col = c.color({ id: :col1, red: 1, blue: 1 })

# wait 2 do

#   col.red(0.6)

#   wait 2 do
#     col.red(0) # Appel en écriture
#   end
# end
```

Fichier: copy.rb

```
# frozen_string_literal: true


b = box
c = circle
t = text('touch me')


b.copy([c.id, b.id, t.id])
b.copy(b.id)


wait 1 do
  c.paste([0, 2])
  wait 1 do
    t.paste(0)
  end


end


t.touch(true) do
  copies = t.paste(0)
  copies.each do |atome_paste|
    wait 1 do
      grab(atome_paste).color(:red)
    end

  end

end
```

Fichier: code.rb

```
# frozen_string_literal: true



a = box

a.code(:hello) do

  circle({ left: 333, color: :orange })

end

wait 1 do

  a.run(:hello)

end
```

Fichier: attach.rb

```
# frozen_string_literal: true



# Here is the attach explanation and example

# the attach method in atome is both a getter and a setter

# attach and fasten particles serve the same purpose but just in the opposite direction

# please note that atome.attach([:atome_id]) means that atome will be the parent of the atome with the id :atome_id

# to sum up :  attach and fasten are both setter and getter :

# a.attach(b.ib) will attach the current object to the IDs passed in the params. The current atome will be the child of the

the atomes width IDS passed in the the params,

# a.attach(b.ib) means (insert 'b' into 'a') or a is parent b is child
```

# while a.fasten(b.id) (insert 'a' into 'b')is the opposite to fasten it will attach IDs passed in the params to the current atome. The current atome will be the parent of of the the atomes width IDS passed in the the params

# a.fasten(b.ib) means (insert 'a' into 'b') or a is child b is parent

# atome.attach([:atome_id]) means that atome will be the ch

Fichier: exchange.rb

```
# frozen_string_literal: true


b = box({ width: 200, height: 200, color: :white })


a = b.box({ color: :green, left: 33, id: :box, shadow: {
  id: :menu_active_shade,
  left: 9,
  top: -3,
  blur: 10,
  invert: false,
  red: 0,
  green: 0,
  blue: 0,
  alpha: 1 } })
wait 2 do
  a.exchange({ color: :red, top: 33})
end
```

Fichier: inspector.rb

```
#  frozen_string_literal: true
```

```
b = text({ id: :toto, left: 0, data: :inspect, depth: 12 })


c = text({ id: :the_c, left: 190, data: 'stop inspect', depth: 12 })

box({ left: 120, top: 120, width: 333, height: 333, id: :helper })


class Atome


  def follow_cursor(div_id, item_to_be_monitored, &proc)

    @inspector_active = true

    last_collided_element = nil


    JS.global[:document].addEventListener('mousemove', @mousemove_listener = proc do |native_event|

      next unless @inspector_active

      event = Native(native_event)

      element = JS.global[:document].getElementById(div_id)

      width = element[:offsetWidth].to_i

      height = element[:offsetHeight].to_i


      left = event[:clientX].to_i - (width / 2)

      top = event[:clientY].to_i - (height / 2)


      element[:style][:left] = "#{left}px"

      element[:style][:top] = "#{top}px"


      last_collided_element = check_collision(element, item_to_be_monitored, last_collided_element,&proc)

    end)
    JS.global[:document]
```

Fichier: detach.rb

```ruby
#  frozen_string_literal: true


b = box({ drag: true, id: :the_b })
c = b.circle({ left: 99, id: :the_c })
d = b.text({ data: :hello, left: 44, top: 44, id: :the_t })
c.touch(:down) do
  c.detach(b.id)
end
```

Fichier: paint.rb

```ruby
# frozen_string_literal: true


c=circle({drag: true, id: :the_circle})


c1=c.color(:white).id
c2=c.color(:red).id
c3=c.color(:yellow).id
color({id: :my_col1, red: 1 , alpha: 0.5})
wait 0.5 do
  c.paint({ gradient: [c1,c2], direction: :left })
  wait 0.5 do
    wait 0.5 do
      c.paint({ gradient: [c1,c2], diffusion: :radial })
      wait 0.5 do
        cc= c.paint({ gradient: [c1,c2, c3], diffusion: :conic })
        wait 0.5 do
          # cc.delete(true)
          #   alert c.paint
          c.remove({all: :paint})
          # alert c.paint
          wait 0.6 do
            c.color(:red)
          end
          # c.paint({ gradient: [c3, c3], diffusion: :conic })
        end
      end
    end
  end
end
```

Fichier: hierarchy.rb

```
# frozen_string_literal: true


#  here is how to setup a hierarchy within atome using a more simple way than fasten and attach .simply adding atome
inside another atome. here is a example to do to so : b = box({ id: :the_box })
b=box
# the line below will create a circle inside the box b
c = b.circle({ id: :the_circle })
# we can add any atome inside another atome, below we add a text inside de th box b
t = b.text({ data: :hello, left: 200, id: :the_cirle })
# theres no limit in the depht of atome, we can create an image inside the text, exemple:
t.image({ path: 'medias/images/logos/atome.svg', width: 33 })
```

```
# note that creating a hierarchy this way automatically


# Note that when you create a hierarchy in this way, it automatically creates a relationship by populating the 'attach' and 'fasten' properties. So, if you enter:



puts "b attach : #{b.attach}" # prints [:view] in the console as it is fasten to the view atom
puts "b fasten :#{b.fasten}" # prints [:the_circle, :the_cirle] in
```

Fichier: type_mutation.rb

```
# frozen_string_literal: true


b = box({ top: 166, data: :hello,path: './medias/images/red_planet.png'  })
b.color({ id: :col1, red: 1, blue: 1})


# b.instance_variable_set("@top", 30)
# b.instance_variable_set("@apply", [c.id])
# b.instance_variable_set("@path",  )


# b.instance_variable_set("@smooth", 30)
wait 1 do
  b.type=:text
  b.refresh
  wait 1 do
    b.type=:image
    b.refresh
  end

end
```

Fichier: markup.rb

```
# frozen_string_literal: true



# For now markup can only be specified at creation time, it will be possible later

the_one = text({ data: :hello, markup: :h1 })
```

Fichier: play.rb

```
# frozen_string_literal: true



if Universe.internet

  # v = video({ path: "medias/videos/avengers.mp4", id: :my_video })

  v = video({ path: "http://commondatastorage.googleapis.com/gtv-videos-bucket/sample/ElephantsDream.mp4" })

else

  v = video(:video_missing)

end

v.left(200)

v.touch(true) do

  alert v.play

end



t=text({id: :my_text, data: "play video"})



t.touch(true) do

  v.data=0

  v.play(26) do |event|
    t.data("event is : #{event}")

    if event[:frame] ==  900 && v.data <3

      puts v.data

      v.data(v.data+1)

      v.play(26)

    end

  end

end



c=circle({left: 123})



c.touch(true) do

  v.play(:pause)

end



cc=circle({left: 0, width: 55, height: 55})



left=0

cc.drag(:locked) do |event|

  dx = event[:dx]

  left += dx.to_f

  min_left = 0

  max_left = 600

  left = [min_left, left].max

  left = [left, max_left].min

  v.html.currentTime(left/10)

  cc.left(left)
```

```
end



puts "add lock x and y when drag"

puts "restrict ro :view doesnt work"
```

Fichier: display.rb

```
# #  frozen_string_literal: true

#

# new({ particle: :display, render: false }) do |params|

#   # alert type

#   unless params[:items]

#     params[:items] = { width: 200, height: 33 }

#   end

#   container_width = params[:width] ||= width

#   container_height = params[:heigth] ||= height

#   container_top = params[:top] ||= top

#   container_left = params[:left] ||= left

#

#   item_width = params[:items][:width] ||= 400

#   item_height = params[:items][:height] ||= 50

#   item_margin = params[:margin] ||= 3

#

#   mode = params[:mode]

#

#   case mode

#   when :none
#   when :custom

#   when :list

#     if params[:data].instance_of? Array

#     elsif params[:data] == :particles

#       list_id = "#{id}_list"

#       unless grab(list_id)

#         container = ''

#         attach.each do |parent|

#             container = grab(parent).box({ id: list_id, left: container_left, top: container_top, width: container_width, height:
container_height, overflow: :auto, color: :black, depth: 0 })

#
```

Fichier: random.rb

```
# frozen_string_literal: true



b = box



16.times do |index|

  width_found = b.width

  b.duplicate({ left: b.left + index * (width_found + 45), top: 0, category: :matrix })

end




Universe.user_atomes.each do |atome_id|

  atome_found = hook(atome_id)

  if atome_found.type == :shape

    atome_found.color(:orange)
    atome_found.smooth(200)

    atome_found.top(200)

  end

end



random_found =Universe.user_atomes.sample(7)

random_found.each do |atome_id|

  atome_found = hook(atome_id)

  if atome_found.type == :shape

    atome_found.top(rand(600))

    atome_found.width(rand(120))

    atome_found.height(rand(120))

    atome_found.smooth(rand(120))

    atome_found.color(:red)

  end

end



random_found =Universe.user_atomes.sample(9)

random_found.each do |atome_id|

  atome_found = hook(atome_id)

  if atome_found.type == :shape

    atome_found.left(rand(700))

    atome_found.width(rand(120))

    atome_found.height(rand(120))

    atome_found.smooth(rand(120))

    atome_found.color(:blue)

  end
```

```
end
```

Fichier: size.rb

```
# frozen_string_literal: true



c = circle({ height: 400, width: 200, top: 100, left: 0, top: 100 })

b = c.box({ width: 200, height: 100, left: 600, top: 200, id: :my_box })

c.circle({ width: 200, height: 100, left: 120, top: -80, id: :my_text, data: :hi })

b.circle({ color: :yellow, width: 55, height: 88, left: 500 })

b.box



wait 1 do

  # recursive apply the new size to all fasten atomes recursively

  # reference : change the size according the to wanted axis

  c.size({value:  50, recursive: true, reference: :y })

end
```

Fichier: clones&monitoring.rb

```
# # frozen_string_literal: true



# TODO : clones alteration must be bidirectional, to do so :



c = circle({ id: :the_circle, left: 12, top: 0, color: :orange, drag: { move: true, inertia: true, lock: :start } })

b = box({ top: 123 })
```

```
t = text({ data: :hello, left: 300 })


t.touch(true) do
  puts "#{b.touch} , #{b.touch_code}"
  b.touch_code[:touch].call
end
color({ id: :col1, red: 1, blue: 1 })
# ######################
atomes_monitored = [c, b]
# particles_monitored=[:left, :width, :touch, :apply]
particles_monitored = [:left, :width, :apply]
# particles_monitored = [:touch]
Atome.monitoring(atomes_monitored, particles_monitored) do |monitor_infos|
  puts "1 ==> #{@id} : #{monitor_infos[:particle]},#{monitor_infos[:altered]}"


  atomes_monitored.each do |atome_to_update|
    # we exclude the current  changing atome to avoid infinite loop
    unless atome_to_update == self || (monitor_infos[:original] == monitor_infos[:altered]) || !monitor_infos[:altered]
      puts "2 ==> #{atome_to
```

Fichier: drag.rb

```
#  frozen_string_literal: true


a=box({width: 666, height: 777, color: :orange})
b = box({ left: 666, color: :blue, smooth: 6, id: :the_box2, depth: 1 , top: 66})
cc=circle({color: :red, left: 0, top: 0})
```

```
clone = ""

b.drag(:start) do

  b.color(:black)

  b.height(123)

  # beware you must use grab(:view) else it'll be fasten to the context, that means to 'b' in this case

  clone = grab(:view).circle({ color: :white, left: b.left, top: b.top, depth: 3 })

end



b.drag(:stop) do

  b.color(:purple)

  b.height=b.height+100

  clone.delete(true)

end



b.drag(:locked) do |event|

  dx = event[:dx]

  dy = event[:dy]

  x = (clone.left || 0) + dx.to_f

  y = (clone.top || 0) + dy.to_f

  clone.left(x)

  clone.top(y)

  puts "x: #{x}"

  puts "y: #{y}"

end

cc.drag({ restrict: {max:{ left: 240, top: 190}} }) do |event|

end
```

```
c=circle


c.drag({ restrict: a.id }) do |event|


end


t=text({data: 'touch me to unbind drag stop for b (clone will not deleted anymore)', left: 250 })
t.touch(true) do
  b.drag({remove
```

Fichier: preset.rb

```
# frozen_string_literal: true




# here how ti use preset in the atome framework
# presets available are :
render_engines,image,video,animation,element,box,vector,circle,shape,text,drm,shadow,color,www,raw,code,audio,gro
up,human,machine,paint


my_box=box
# using the code line above a lot of particles will be implicitly created, if we inspect my_box
puts my_box.inspect # this will print :
#[Log] #<Atome: @broadcast={}, @callback={}, @tag={}, @fasten=[], @unit={}, @collected={}, @id=:box_14,
@type=:shape, @html=#<HTML:0x0662a164 @element=[object HTMLDivElement], @id="box_14",
@original_atome=#<Atome: @broadcast={}, @callback={}, @tag={}, @fasten=[], @unit={}, @collected={},
@id=:box_14, @type=:shape, @html=#<HTML:0x0662a164 ...>, @attach=[:view], @renderers=[:html], @width=99,
@height=99, @apply=[:box_color], @left=100, @top=100, @clones=[], @preset={:box=>{:width=>99, :height=>99,
:apply=>[:box_color], :left=>100, :top=>100, :clones=>[]}}>, @element_type="div">, @attach=[:view], @rendere
```

Fichier: overflow.rb

```ruby
# frozen_string_literal: true


b = box({ id: :the_container, width: 300, height: 300 })
b.box({ top: 500, color: :red })
cc = b.circle({ top: 160, id: :the_circle })


initial_height = cc.height
initial_width = cc.width
b.overflow(:scroll) do |event|
  new_height = initial_height + event[:top]
  cc.height(new_height)
  { left: event[:top] }
end
c = circle({ top: 370, color: :red })
c.touch(:up) do
  b.overflow(:remove)
  c.delete(true)
  c = circle({ top: 370, left: 90, color: :green })
  c.touch(true) do
    b.overflow(:scroll) do |event|
      puts 'removed!!'
      new_width = initial_width + event[:top]
      cc.width(new_width)
    end
  end
end
```

Fichier: test.rb

```
# frozen_string_literal: true


def contact_template
{ id: :humans, role: nil, date: { companies: [], project: {}, events: {}, last_name: nil, first_name: nil ,
                                  emails: { home: nil }, phones: {}, address: {}, groups: [] } }
end


element({id: :testing, data: contact_template})
# grab(:testing).data(contact_template)


wait 2 do
  grab(:testing).data
end
```

Fichier: gradient.rb

```
# frozen_string_literal: true

circ = circle({ drag: true })
circ.remove({ all: :color })
```

```
col_1 = circ.color(:white)

col_2 = circ.color({ red: 1, id: :red_col })

col_4 = circ.color({ blue: 1, id: :red_col2, alpha: 0.3 })

col_5 = circ.color({ red: 0, green: 1, id: :red_col3, alpha: 0.7 })

col_3 = circ.color(:yellow)

wait 0.5 do

  circ.paint({ gradient: [col_1.id, col_2.id], direction: :left })

  wait 0.5 do

    circ.paint({ id: :the_painter, rotate: 69, gradient: [col_1.id, col_2.id] })

    wait 0.5 do

      circ.color(:cyan)

      circ.paint({ gradient: [col_1.id, col_2.id, col_3.id], rotate: 33, diffusion: :conic })

      wait 0.5 do

        painter = circ.paint({ id: :the_painter2, gradient: [col_1.id, col_2.id, col_3.id], direction: :left })

        wait 0.5 do

          # circ.color(:blue)

          circ.paint({ gradient: [col_4.id, col_5.id], diffusion: :conic })

          wait 1 do

            circ.color(:blue)

            # circ.paint({ gradient: [col_5.id, col_5.id], diffusion:
```

Fichier: layout.rb

```
# frozen_string_literal: true



b = box({ color: :red, id: :the_box, left: 3 })

5.times do |index|

  width_found = b.width
  b.duplicate({ left: b.left + index * (width_found + 45), top: 0, category: :custom_category })

end



grab(:view).fasten.each do |atome_found|

  grab(atome_found).selected(true)

end

grab(:the_box_copy_1).text(:hello)



selected_items = grab(Universe.current_user).selection # we create a group

# we collect all atomes in the view

atomes_found = []

selected_items.each do |atome_found|

  atomes_found << atome_found

end





selected_items.layout({ mode: :default, width: 500, height: 22 })



wait 1 do

  selected_items.layout({ mode: :grid, width: 900, height: 500, color: :green, element: { rotate: 22, height: 100, width: 150

} })

  wait 1 do

    selected_items.layout({ mode: :grid, width: 1200, height: 500, overflow: :scroll })

    wait 1 do

      selected_items.layout({ mode: :default, width: 500, height: 22 })

      wait 1 do

        selected_items.layout({ id: :my_layout,
```

Fichier: categories.rb

```
# frozen_string_literal: true



# Universe.categories is used to get the existing category to sort particles , ex:



puts Universe.categories
```

Fichier: percent_to_px.rb

```
# frozen_string_literal: true



bb=box({width: '90%'})

puts bb.to_px(:width)
```

Fichier: svg_vectorizer.rb

```
# frozen_string_literal: true




svg_content = <<-SVG

<svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="1024"
height="1024"  xml:space="preserve" id="colorCanvas">

<!-- Generated by jeezs - http://www.atome.one

<path d="M150 0 L75 200 L225 200 Z" stroke="red" stroke-width="37" fill="white" />

<circle id="colorCanvas-oval" stroke="none" fill="rgb(255, 0, 0)" cx="274" cy="306" r="198" />

<circle id="colorCanvas-oval2" stroke="none" fill="rgb(0, 142, 255)" cx="767" cy="306" r="198" />

<circle id="colorCanvas-oval3" stroke="none" fill="rgb(50, 255, 0)" cx="499" cy="702" r="198" />

<ellipse id="colorCanvas-ellipse" stroke="black" stroke-width="5" fill="yellow" cx="512.5" cy="256" rx="150" ry="100" />

<rect id="colorCanvas-rect" stroke="green" stroke-width="5" fill="blue" x="100" y="500.7" width="300" height="150" />

<line id="colorCanvas-line" stroke="purple" stroke-width="110" x1="50" y1="800" x2="300.6" y2="950" />
```

<polygon id="colorCan

Fichier: attached.rb

```
# frozen_string_literal: true


# Here is the fasten explanation and example :


# the fasten method in atome is both a getter and a setter

# attach and fasten particles serve the same purpose but just in the opposite direction

# please note that atome.attach([:atome_id]) means that atome will be the parent of the atome with the id :atome_id

# to sum up :  attach and fasten are both setter and getter :

# attach will attach the current object to the IDs passed in the params. The current atome will be the child of the the atomes width IDS passed in the the params,

# while fasten is the opposite to fasten it will attach IDs passed in the params to the current atome. The current atome will be the parent of of the the atomes width IDS passed in the the params


# Here is how to use it as a setter :

grab(:black_matter).color({ red: 1, green: 0.6, blue: 0.6, id: :active_color })

grab(:black_matter).color({ red: 0.3, green: 1, blue: 0.3, id: :inactive_color })


b = box({ left: 99, drag: true, id:
```

Fichier: retreive.rb

```
# frozen_string_literal: true


b = box({ left: 155, drag: true, id: :boxy })


t=b.text({ data: :hello, id: :t1, position: :absolute, color: :black })
```

```
t2 = b.text({ data: :hello, id: :t2, left: 9, top: 33, position: :absolute })



wait 1 do
  grab(:view).retrieve do |child|
    child.left(33)
  end
  wait 1 do
    grab(:boxy).retrieve do |child|
      child.color(:green)
    end
    wait 1 do
      grab(:view).retrieve({ ascending: false, self: false }) do |child|
        child.delete(true)
      end
    end
  end
end
```

Fichier: trigger_abstraction.rb

```
# frozen_string_literal: true


new ({particle: :trigger})


a=circle
a.trigger({record: true})
```

});

### wad JS

```
bb=box({left: 333})

bb.text(:wadjs)
```

## Midi test

```
js_midi_code = <<~JAVASCRIPT

async function startMidi() {

    try {

        await window.__TAURI__.invoke('start_midi');

        console.log('MIDI listener started');

    } catch (error) {

        console.error('Failed to start MIDI listener', error);

    }

}

function listenForMidiEvents() {

    window.__TAURI__.event.listen('midi-event', event => {

        console.log('MIDI Event found:', event.payload);

    });

}

startMidi();
```

```
listenForMidiEvents();


JAVASCRIPT

if Atome::host == 'tauri'

  JS.eval(js_midi_code)

end




# Initialize window.snare


init_code = "window.snare = new Wad({source : 'medias/audios/clap.wav'});"

JS.eval(init_code)


# Define the JavaScript playSnare function

js_code = <<~JAVASCRIPT

  window.playSnare = function() {

    window.snare.play();

    // setTimeout(function() {

    //  window.snare.stop();

    //}
```

Fichier: increment.rb

```
# frozen_string_literal: true


cc=color({red: 1, blue: 0.1,id: :the_col})

b=box({ left: 12, id: :the_first_box, apply: cc.id  })

c=circle({ left: 99, top: 99 })
```

```
wait 1 do
  c.increment({left: 33, top: 99})
  b.increment({left: 33, top: 99})
  wait 1 do
    c.increment({width: 33, top: -22})
    b.increment({width: 33, top: -9})
    cc.increment({red: -0.5})
    wait 1 do
      cc.increment({blue: 1})
    end
    # Atome.sync(:ok)
  end
end


# wait 3 do
#   color(:red)
# end
```

Fichier: raw_html.rb

```
#  frozen_string_literal: true


raw_data = <<STR
<iframe width="560" height="315" src="https://www.youtube.com/embed/8BT4Q3UtO6Q?si=WI8RlryV8HW9Y0nz" title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
STR
```

```
raw_data = <<STR

<svg width="600" height="350" xmlns="http://www.w3.org/2000/svg">

<!-- Style for the boxes -->

                        <style>

.box { fill: white; stroke: black; stroke-width: 2; }

   .original { fill: lightblue; }

   .clone { fill: lightgreen; }

   .arrow { stroke: black; stroke-width: 2; marker-end: url(#arrowhead); }

                                                        .text { font-family: Arial, sans-serif; font-size: 14px; }

   </style>



  <!-- Arrowhead definition -->

  <defs>

    <marker id="arrowhead" markerWidth="10" markerHeight="7"

    refX="0" refY="3.5" orient="auto">

      <polygon points="0 0, 10 3.5, 0 7" fill="black"
```

Fichier: security.rb

```
#  frozen_string_literal: true




c=circle({left: 220})

t=text({left: 550,data: :hello,password: { read: { atome: :my_secret} }})

b = box({ id: :the_box, left: 66,smooth: 1.789,

          password: {
```

```
      read: {

        atome: :the_pass,

        smooth: :read_pass

      },

      write: {

        atome: :the_write_pass,

        smooth: :write_pass

      }

    }

  })




b.authorise({ read: { atome: :the_pass, smooth: :read_pass }, write: { smooth: :write_pass}, destroy: true}  )

puts b.smooth

# next will be rejected because destroy: true

puts b.smooth

#

b.authorise({ read: { atome: :wrong_pass, smooth: :no_read_pass }, write: { smooth: :wrong_write_pass}, destroy: false}

)

puts 'will send the wrong password'

puts b.smooth


b.authorise({ read: { atome: :wrong_pass, smooth: :read_pass }, write: { smooth: :wrong_write_pass}, destroy: false}  )

puts "'with send the right password it'll works"

puts b.smooth

# authorise has two para
```

Fichier: infos.rb

```ruby
# frozen_string_literal: true



c = circle({ height: 400, width: 200, top: 100, left: 0, top: 100 })


puts "infos : #{c.infos}"

puts "width : #{c.infos[:width]}"
```

Fichier: generator_and_build.rb

```ruby
# frozen_string_literal: true


gen = generator({ id: :genesis, build: {top: 66, copies: 1} })

gen.build({ id: :bundler, copies: 32, color: :red, width: 33, height: 44,  left: 123, smooth: 9, blur: 3, attach: :view })

grab(:bundler_1).color(:blue)




#   Atome.new(

#   { renderers:  [:html], id: :atomix, type: :element, tag: { system: true }, attach: [], fasten: [] }

# )

#

#

# {:id=>:eDen, :type=>:element, :renderers=>[], :tag=>{:system=>true}, :attach=>[], :fasten=>[]}
# {:renderers=>[], :id=>:eDen, :type=>:element, :tag=>{:system=>true}, :attach=>[], :fasten=>[]}
```

Fichier: sliders.rb

```ruby
# frozen_string_literal: true


label = text({ data: 0, top: 400, left: 69, component: { size: 12 }, color: :gray })


aaa = grab(:intuition).slider({ id: :toto, range: { color: :yellow }, min: -12, max: 33, width: 333, value: 12, height: 25, left: 99, top: 350, color: :orange, cursor: { color: :orange, width: 25, height: 25 } }) do |value|
  label.data("(#{value})")
end


aa = grab(:intuition).slider({ orientation: :vertical, range: { color: :white }, value: 55, width: 55, height: 555, attach: :intuition, left: 555, top: 33, color: :red, cursor: { color: {alpha: 1, red: 0.12, green: 0.12, blue: 0.12}, width: 33, height: 66, smooth: 3 } }) do |value|
  label.data("(#{value})")
end


b=box
b.touch(true) do
  aa.value(12)
  aaa.value(-6)
end
```

Fichier: wait.rb

```ruby
# #  frozen_string_literal: true
b = box
```

```
first_wait=wait 2 do
  b.color(:red)
end
wait 1 do
  puts 'now'
  stop({ wait: first_wait })
  # or
  # wait(:kill, first_wait)
end


wait 3 do
  b.color(:green)
end
```

Fichier: remove.rb

```
# frozen_string_literal: true

b = box({ top: 166, id: :the_box, left: 333 })
b.color({ id: :new_col, red: 1 })

b.touch(true) do
  # alert b.color
  b.remove(:box_color)

  # alert b.color
  wait 1 do
```

```
    grab('box_color').red(1)

  end

end

# b.color({id: :other_col,  green: 1})

# # b.paint({gradient: [:other_col, :new_col]})

# color({id: :last_col,  green: 0.3, blue: 0.5})

# color({id: :last_col2,  red: 1, blue: 0.5})

#

# b.shadow({

#            id: :s1,

#            # affect: [:the_circle],

#            left: 9, top: 3, blur: 9,

#            invert: false,

#            red: 0, green: 0, blue: 0, alpha: 1

#          })

#

#

# wait 1 do

#   b.remove(:other_col)

#   wait 1 do

#     b.remove(:new_col)

#     wait 1 do

#       b.remove(:box_color)

#

#       wait 1 do

#         b.apply(:last_col)

#         wait 1 do
#           b.apply(:last_col2)
#           b.remove(:s1)
#         end
#       end
#     end
#   end
# end
# b.touch(true) do
#   b.shadow({
#              id: :s1,
#
```

Fichier: recorder.rb

```
# frozen_string_literal: true


# native recording (server mode only) :
text({ data: 'native operation only work in server mode ', top: 60 })
a = circle({ color: :red, left: 30, top: 90 })
a.text('native Audio')
record_callback = 'unset'
a.touch(true) do
   A.record({ media: :audio, duration: 5, mode: :native, name: :my_audio_rec, type: :wav, path: '../src', data: { note: :c,
velocity: 12, robin: 3, author: :vie, tags: [:voice, :noise, :attack] } }) do |result|
    puts "result: #{result}"
    record_callback = result
  end
  nil# we must return nil else the event methods take the bloc for a Hash and crash the code
```

```
end



aa = circle({ color: :red, left: 120, top: 90 })

aa.text('native video')



aa.touch(true) do

  A.record({ media: :video, duration: 5, mode: :native, name: :my_video_rec, type: :mp4, path: '../src/', data: { type: :thriller, } }) do |result|

    puts result

    record_callback = result

  end

  nil # we must return nil else the event methods take the bloc for a Hash and cra
```

Fichier: online.rb

```
# frozen_string_literal: true



text online?
```

Fichier: center.rb

```
# frozen_string_literal: true



b= box({ center:  { x: 0, y: 0, dynamic: true }})



# b.center({ x: '10%', y: '20%' })

# b.center({ x: true, y: true })

# box({center: true})
```

Fichier: midi.rb

```
# frozen_string_literal: true


new({molecule: :system}) do |params|
  alert Atome::host
  alert Universe.engine
end


puts "connect a midi device and run atome in native mode then look in the console"


system({message: "open 'test autoload.bwproject'"})
```

Fichier: aXion.rb

```
#  frozen_string_literal: true




c=circle


c.touch(true) do



   # c.message({data: {prompt: "cherche un fichier qui se nomme  capture  et ouvre le avec  l'application par defaut" ,
user_key: 'user_key'},  action: :axion }) do |result|
  #   puts  "my command return: #{result}"
  # end
```

```
   # c.message({data: { prompt: "liste moi tous les fichiers et dossiers que tu trouve", user_key: 'user_key' },  action:
:axion }) do |result|
  #   puts  "my command return: #{result}"
  # end

  A.message({data: {prompt: "il faudrait ecrire un texte de remerciement pour un service en rendu addressé a mr albert
et mettre ce texte dans un fchier, et ouvre le " , user_key:
'sk-proj-30NyTRt_3DAjrK_W7LQl-0csVjmC2rABcNPiTihFo1Ag-JWHPKlhqdtkt5qLTXWcwmwKTrZtxmT3BlbkFJ525DX2
eMWY5E6MUiTUnJw_-FjZ4SNQXcypP-uj2sKoW6gEmTfU2TAYqhYwTSxZvJUpj2xUDr8A'},  action: :axion }) do |result|
    puts  "my command return: #{result}"
  end

  {} #must add an empty hash else events events method will interpret keys of the hash an
```

Fichier: applications.rb

```
# frozen_string_literal: true

a = application({
                  id: :arp,
                  margin: 3,
                  spacing: 6
                })

page1_code = lambda do |back|
  alert :kooly
end
```

```
verif = lambda do
  b = box({ id: :ty, left: 90, top: 90 })
  b.touch(true) do
    alert grab(:mod_1).touch
  end
end


page1 = {
  id: :page1,
  color: :cyan,
  name: :accueil,
  footer: { color: :green, height: 22 },
  header: { color: :yellow },
  left_side_bar: { color: :yellowgreen },
  right_side_bar: { color: :blue },
}


color({ id: :titi, red: 1 })
page2 = { id: :page2,
          color: :white,
          menu: false,
          run: verif,
          box: { id: :mod_1, left: 333, top: 123, touch: { down: true, code: page1_code } }
}


page0 = { id: :page0,
          color: :purple,
}


a.page(page0)

a.page(page1)

a.page({ id: :page3,

         color: :red,

         footer: { color: :green, height: 22 }

       })




menu_f=a.menu

menus_found= menu_f.fasten # replace fasten for entrie
```

Fichier: convert.rb

```
#  frozen_string_literal: true


b = box({ id: :the_html, color: :orange, overflow: :auto, width: :auto, height: :auto, left: 100, right: 100, top: 100, bottom:
100 })

html_desc = <<STR

<!DOCTYPE html>

<html>

    <head>

        <title>Une petite page HTML</title>

        <meta charset="utf-8" />

    </head>

    <body>
```

```
        <h1 id='title' style='color: yellowgreen'>Un titre de niveau 1</h1>


        <p>

            Un premier petit paragraphe.

        </p>


        <h2>Un titre de niveau 2</h2>


        <p>

            Un autre paragraphe contenant un lien pour aller

            sur le site <a href="http://koor.fr">KooR.fr</a>.

        </p>

    </body>

</html>

STR


b.hypertext(html_desc)


def markup_analysis(markup) end


def convert(params)

  case

  when params.keys.include?(:atome)


    # Atome.new({type})

    puts params[:atome]

  else
```

```
    # ...

  end

end



b.hyperedit(:title) do |tag_desc|

  convert({ atome: tag_desc })

end



# Bien sûr ! Voici une liste des principaux types de balis
```

Fichier: tools.rb

```
# frozen_string_literal: true





# new({ tool: :color2 }) do

#   active_code = lambda {

#     puts 'color activated1'

#   }

#   color_code2=lambda {

#     puts  "object id is : #{id}"

#     # color(:green)

#   }

#   inactive_code = lambda { |data|

#     data[:treated].each do |atome_f|

#       # atome_f.drag(false)

#       # atome_f.color(:green)

#     end
#   }
#
#   { activation: active_code,
#     alteration: { event: color_code2 },
#     inactivation: inactive_code,
#     target: :color,
#     particles: { red: 0, green: 0.5, blue: 1, alpha: 1 }
#   }
# end


new({ tool: :toolbox1 }) do
  active_code = lambda {
    toolbox({ tools: [:combined], toolbox: { orientation: :ew, left: 90, bottom: 9, spacing: 9 } })
  }
  { activation: active_code }
end



new({ tool: :combined }) do |params|

  active_code = lambda {
    # puts :alteration_tool_code_activated
  }

  inactive_code = lambda { |param|
    # puts :alteration_tool_code_inactivated1
  }
  pre_code = lambda { |params|

    # puts ''
```

Fichier: atomizer.rb

```
# frozen_string_literal: true



# dummies html objects :



#object 1

div_rouge = JS.global[:document].createElement( "div")



div_rouge[:style][:backgroundColor] = "red"

div_rouge[:style][:width] = "100px"

div_rouge[:style][:height] = "100px"

div_rouge.setAttribute('id', "my_div")

div_view = JS.global[:document].getElementById('view')

div_view.appendChild(div_rouge)




#object 2

span_bleu =  JS.global[:document].createElement( "span")

span_bleu[:style][:backgroundColor] = "blue"

span_bleu[:innerHTML] = "blue"

span_bleu[:style][:width] = "10px"

span_bleu[:style][:height] = "8px"

div_rouge.appendChild(span_bleu)
```

```
#object 2

span_white =  JS.global[:document].createElement( "h1")

span_white[:style][:color] = "white"

span_white[:innerHTML] = "Hello"

span_white[:style][:width] = "10px"

span_white[:style][:height] = "80px"

span_white[:style][:top] = "80px"

span_bleu.appendChild(span_white)




#   usage example

#   div_result =  HTML.locate(id: 'my_div') # Recherche par ID

#   alert "id found : #{
```

Fichier: encode.rb

```
#  frozen_string_literal: true



my_pass = Black_matter.encode('hello')

puts my_pass

checker = Black_matter.check_password('hello,', my_pass)

puts checker
```

Fichier: monitor.rb

```
# frozen_string_literal: true

puts 'deprecated use clone monitoring'

# b = box({ id: :the_box })

# c = circle({ top: 3, id: :the_cirle })
# A.monitor({ atomes: [:the_box, :the_cirle], particles: [:left] }) do |atome, particle, value|
#   puts "changes : #{atome.id}, #{particle}, #{value}"
# end
#
# wait 2 do
#   b.left(3)
#   wait 2 do
#     c.left(444)
#   end
# end
```

Fichier: list.rb

```
# frozen_string_literal: true


styles = {
  width: 199,
  height: 33,
  margin: 6,
  shadow: { blur: 9, left: 3, top: 3, id: :cell_shadow, red: 0, green: 0, blue: 0, alpha: 0.6 },
  left: 0,
  color: :yellowgreen
}


element = { width: 33,
            height: 33,
            component: { size: 11 },
            left: :center,
            top: :center,

            color: :black,

            type: :text }



listing = [

  { data: :'hello' },

  { data: :'salut', color: :red },

  { data: :hi },

  { data: :ho }

]

b = box({ drag: true })

list_1 = grab(:intuition).list({

                                styles: styles,

                                element: element,

                                listing: listing,

                                left: 33,

                                attach: b.id,

                                action: {touch: :down, method: :my_method }

                              })



# test2



styles = {

  width: 199,

  height: 33,

  margin: 6,

  shadow: { blur:
```

Fichier: database_handling.rb

```
# frozen_string_literal: true



A.message({ action: :insert, data: { table: :security, particle: :password, data: 'my_pass'} }) do |datas|

  puts "0 data received:  #{datas}"

end



A.message({ action: :insert, data: { table: :identity, particle: :name, data: 'jeezs' } }) do |data_received_from_server|

  puts "1 my first insert #{data_received_from_server}"

end



A.message({ action: :insert, data: { table: :identity, particle: :name, data: 'jeezs2' } })



A.message({ action: :query, data: { table: :identity } }) do |data_received_from_server|

  puts "2 another insert  : #{data_received_from_server}"

end



A.message({ action: :query, data: { table: :identity } }) do |data_received|

  puts "3 received : #{data_received}"

end



A.message({ action: :insert, data: { table: :identity, particle: :name, data: 'jeezs3' } }) do |result|

  puts "4 insert done : #{result}"

end



A.message({ action: :insert, data: { table: :identity, particle: :name, data: 'jeezs4' } }) do |result|

  puts "5 last message r
```

Fichier: timeline.rb

```ruby
# frozen_string_literal: true



new(molecule: :roller) do |params = {}|

  roller_id = params[:id] ||= identity_generator

  roller = box({ id: roller_id, width: 900, height: 333, color: :orange })

  JS.eval("aRoll('#{roller_id}_roller','#{roller_id}', #{roller.width}, #{roller.height})")

  roller

end

new({ molecule: :button }) do |params, bloc|

  but = box({ smooth: 6, shadow: { alpha: 0.3 }, width: 25, height: 25, color: :red })

  but.shadow({ alpha: 0.6, left: -3, top: -3, blur: 3, invert: true })

  label = params.delete(:label) || 'button'

  idf_f = params.delete(:id) || identity_generator

  but.text({id: idf_f, data: label, component: { size: 9 }, center: true, position: :absolute })



  but.instance_variable_set('@on', true)

  but.set(params)



  def code_logic(but, bloc)

    but.instance_exec(&bloc) if bloc.is_a?(Proc)

    if but.instance_variable_get('@on') == true

      but.instance_variable_set('@on', false)

    else

      but.instance_variable_set('@on', true)

    end

  end
  bu
```

Fichier: affect.rb

```
# frozen_string_literal: true

box({ left: 12, id: :the_first_box })
c=color({ id: :the_col, blue: 0.21, green: 1 })

wait 1 do
  c.affect(:the_first_box)
end
```

Fichier: vector.rb

```
 # frozen_string_literal: true

edition = "M257.7 752c2 0 4-0.2 6-0.5L431.9 722c2-0.4 3.9-1.3 5.3-2.8l423.9-423.9c3.9-3.9 3.9-10.2 0-14.1L694.9 114.9c-1.9-1.9-4.4-2.9-7.1-2.9s-5.2 1-7.1 2.9L256.8 538.8c-1.5 1.5-2.4 3.3-2.8 5.3l-29.5 168.2c-1.9 11.1 1.5 21.9 9.4 29.8 6.6 6.4 14.9 9.9 23.8 9.9z m67.4-174.4L687.8 215l73.3 73.3-362.7 362.6-88.9 15.7 15.6-89zM880 836H144c-17.7 0-32 14.3-32 32v36c0 4.4 3.6 8 8 8h784c4.4 0 8-3.6 8-8v-36c0-17.7-14.3-32-32-32z"

v = vector({ data: { path: { d: edition, id: :p1, stroke: :black, 'stroke-width' => 37, fill: :red } } })

wait 1 do
  v.data([{ circle: { cx: 300, cy: 300, r: 340, id: :p2, stroke: :blue, 'stroke-width' => 35, fill: :yellow } }, { circle: { cx: 1000, cy: 1000, r: 340, id: :p2, stroke: :green, 'stroke-width' => 35, fill: :yellow } }])
  wait 1 do
    v.color(:cyan) # colorise everything with the color method
    wait 1 do
      v.shadow({
                 id: :s4,
                 left: 20, top: 0, blur: 9,
                 option: :nat
```

Fichier: color.rb

```
# frozen_string_literal: true


# frozen_string_literal: true


# puts 'type you problematic code here!'
col=color({green: 1, id: :the_col})


b=box({top: 3})
t=text(data: :red, left: 0, top: 123)
t1=text(data: :green, left: 100, top: 123)
t2=text(data: :blue, left: 200, top: 123)
t3=text(data: :yellow, left: 300, top: 123)
t4=text(data: :orange, left: 400, top: 123)
t5=text(data: :cyan, left: 500, top: 123)


item_to_batch=[t.id,t1.id,t2.id, t3.id, t4.id, t5.id]
the_group= group({ collect: item_to_batch })
the_group.apply([:the_col])
t.touch(true) do
  b.color({id: :red, red: 1 })
  # puts "number of atomes : #{Universe.atomes.length}"

end

t1.touch(true) do

  b.color({id: :green, green: 1 })

  # puts "number of atomes : #{Universe.atomes.length}"

end

t2.touch(true) do

  b.color({id: :blue, blue: 1 })

  # puts "number of atomes : #{Universe.atomes.length}"

end

t3.touch(true) do

  b.color({id: :yellow,  red: 1, green: 1 })

  # puts "number of atomes : #{Universe.atomes.length}"

end

t4.touch
```

Fichier: fit.rb

```
# frozen_string_literal: true



c = circle({ height: 400, width: 200, top: 100, left: 0, top: 100 })

b = c.box({ width: 200, height: 100, left: 600, top: 200, id: :my_box })

c.circle({ width: 200, height: 100, left: 120, top: -80, id: :my_text, data: :hi })

b.circle({ color: :yellow, width: 55, height: 88, left: 100 })

b.box

i=c.image({path: 'medias/images/red_planet.png', id: :the_pix })

# b.text(:red_planet)
```

```
wait 1 do


  c.fit({ value: 100, axis: :x })


  wait 1 do
    c.fit({ value: 66, axis: :y })
    wait 1 do
      c.fit({ value: 600, axis: :x })
    end
  end
end
# alert i.width
# alert i.height
# i.fit({ value: 66, axis: :x })
#  i.width(66)
#  i.height(66)
```

Fichier: shapes.rb

```
# frozen_string_literal: true


shape(
  { renderers: [:html], id: :my_test_box, type: :shape, apply: [:shape_color],
    left: 120, top: 0, width: 100, smooth: 15, height: 100, overflow: :visible, fasten: [], center: true
  })
```

Fichier: interop_ruby_js.rb

```
#  frozen_string_literal: true
# caling a js methode

js_func(:js_test, :super)



# using class

my_class_instance=js_class(:my_test_class)

my_class_instance.myTestFunction("Bonjour depuis Ruby!")

# to call a ruby methode from js use :

#       atomeJsToRuby('box'); or  atomeJsToRuby("my_meth('my_params')")

#
```

Fichier: terminal.rb

```
# frozen_string_literal: true

A.terminal('pwd') do |data|

  text "terminal response  :\n #{data}"

end



# alert A.inspect
```

Fichier: style.rb

```
# frozen_string_literal: true

b = box



b.style({ left: 33, width: 44, rotate: 23, color: :yellowgreen, blur: 44 })
```

Fichier: alternate.rb

```
# frozen_string_literal: true

def act_on(obj)
```

```
  obj.color(:red)
  obj.left(56)
end


def act_off(obj)
  obj.color(:blue)
  obj.left(33)
end



b = box({ left: 12, id: :the_first_box, top: 30 })


b.touch(true) do
   b.alternate({ width: 33, color: :red, height: 33 , smooth: 0 }, { width: 66, color: :orange, blur: 8}, { height: 66, color:
:green, smooth: 9, blur: 0})
end


c = circle({ left: 99 , top: 30})


c.touch(true) do
  alt = b.alternate(true, false)
  if alt
    c.color(:yellowgreen)
  else
    c.color(:orange)
  end
end
```

```
c2 = circle({ left: 333 , top: 30})



c2.touch(true) do
  b.alternate({  executor: {act_on: b}  }, { executor: {act_off: b}})
end
```

Fichier: clear.rb

```
# frozen_string_literal: true


# here is how to clear the content of an atome
b=box
c=circle
b.left(0)
c.left(222)
wait 2 do
  # Important : please note that the view is also an atome, this this a system atome that can't be deleted,
  # There are a few system atomes created at init time
  # Here are the list of the system atomes created at system startup:
  #  we can clear it's content using .clear(true) its the same action as if I have done : b.delete(true) and c.delete(true)
  grab(:view).clear(true)
end



# here are the list of system atomes created at system startup :
```

```
#Atome.new(

#   { renderers: [], id: :eDen, type: :element, tag: { system: true }, attach: [], fasten: [] }

# )

# Atome.new(

#   { renderers: [], id: :user_view, type: :element, tag: { system: true },

#  attach: [:eDen], fasten: [] }

# )

#

# # color creation

# Atome.new(

#   { renderers: default_render, id: :view_color, type: :color, tag: ({ system: true, persistent: true }),

#  red: 0.15, green: 0.15, blue: 0.15, a
```

Fichier: fill.rb

```
# frozen_string_literal: true



b=box({width: 300, height: 333, color: {alpha: 0}})

image({id: :logo,path: 'medias/images/logos/atome.svg', width: 66, left: 555})

grab(:black_matter).image({id: :planet,path: 'medias/images/red_planet.png', width: 66,height: 66,  left: 555, top: 180})




b.fill([atome:  :logo, width: 33, height: 33 ])

b.overflow(:hidden)

wait 1 do

  b.fill([atome:  :planet, width: 33, height: 33 ])
  wait 1 do

    b.fill([{atome:  :planet,repeat: {x: 5, y: 3}}])

    wait 1 do

      b.fill([{atome:  :planet,width: 33, height: 33 ,rotate: 33, size: { x: 800,y: 600 }, position: { x:-200,y: -200 } }])

      wait 3 do

        b.fill([{atome:  :planet,repeat: {x: 5, y: 3}}, { atome: :logo, width: 33, height: 33 ,  opacity: 0.3} ])

      end

    end

  end

end



b.drag(true)
```

Fichier: behavior.rb

```
# frozen_string_literal: true



# Behaviors allow you to add specific code to any particle, enabling the particle to behave differently.

# Here, when the first box receives a value, it behaves differently from the second box even if they received

# the same params .



text({ data: :hello, id: :the_txt, left: 120 })



b=box



my_lambda= lambda do |new_value|

  grab(:the_txt).color(:red)

end
```

```
b.behavior({value: my_lambda})


my_second_lambda= lambda do |new_value|
  grab(:the_txt).data('from cirle')
end
c=box({top: 69})
c.behavior({value: my_second_lambda})


wait 1 do
  c.value(:ok)
end
wait 2 do
  b.value(:ok)
end
```

Fichier: browse.rb

```
# frozen_string_literal: true


# browse only works with  application version of atome or using server mode , it allow the browse local file on your computer or remote file on server, if operating in server mode
# here is an example :

A.browse('/') do |data|

  text "folder content  :\n #{data}"

end



# if Atome.host == 'tauri'

#   # JS.eval("readFile('atome','Cargo.toml')")

#   JS.eval("browseFile('atome','/')")

# else

#   puts 'nothing here'

#   # JS.eval("terminal('A.terminal_callback','pwd')")

# end
```

Fichier: import.rb

```
# frozen_string_literal: true



support = box({ top: 250, left: 12, width: 300, height: 40, smooth: 9, color: { red: 0.3, green: 0.3, blue: 0.3 }, id: :support })



support.shadow({

                 id: :s3,

                 left: 3, top: 3, blur: 9,

                 invert: true,

                 red: 0, green: 0, blue: 0, alpha: 0.7

               })
```

```
box({ id: :the_boxy })




support.import(true) do |content|

  puts "add code here, content:  #{content}"

end




importer do |val|

  puts "case 21 #{val}"

end


# importer(:all) do |val|

#   alert "case 21 #{val}"

# end


importer('the_boxy') do |val|

  puts "yes !!! exception found : #{val}"

end
```

Fichier: atome_sparkle_use.rb

```
# frozen_string_literal: true




text("a whole new way to use atome :\n

 create a ruby file, ex : index.rb then type atome sparkle index\n
```

it will create an app and run it immediately")

Fichier: unfasten.rb

```
#  frozen_string_literal: true

b = box({ drag: true, id: :the_b, top: 63, left: 63 })

c = b.circle({ left: 99, id: :the_c })

b.box({left: 99, top: 99, width: 33, height: 33, id: :second_one})

t = b.text({ data: 'touch the circle', left: 44, top: 44, id: :the_t })

c.touch(:down) do

  b.unfasten([c.id])

  b.color(:green)

  t.data('circle unfasten')

  grab(:infos).data("number of item(s) fasten to the box : #{b.fasten}")

  wait 2 do

    grab(:second_one).delete((true))

    grab(:infos).data("number of item(s) fasten to the box : #{b.fasten}")

    wait 2 do

      b.color(:red)

      t.data('unfasten all attached atomes')

      b.unfasten(:all)

      grab(:infos).data("number of  item fasten to the box : #{b.fasten}")

    end

  end

end


text({id: :infos,left: 155, data: "number of  item fasten to the box : #{b.fasten}"})
```

Fichier: matrix.rb

```
# #  frozen_string_literal: true

#

matrix_zone = box({ width: 333, height: 333, drag: true, id: :the_box, color: {alpha: 0.4} })

#

# # matrix creation

main_matrix = matrix_zone.matrix({ id: :vie_0, rows: 8, columns: 8, spacing: 6, size: '100%' })

main_matrix.smooth(10)

main_matrix.color(:red)




# ##########################################################@

matrix_to_treat = main_matrix.cells

matrix_to_treat.color(:blue)

matrix_to_treat.smooth(6)

matrix_to_treat.shadow({

                         id: :s1,

                         left: 3, top: 3, blur: 6,

                         invert: false,

                         red: 0, green: 0, blue: 0, alpha: 0.6

                       })

# ###################

col_1 = color(:yellow)

col_2 = color({ red: 1, id: :red_col })


wait 3 do

  matrix_to_treat.paint({ gradient: [col_1.id, col_2.id], direction: :top })
```

```
end

#

# ###################



test_cell = grab(:vie_0_2_3)

wait 1 do

  test_cell.color(:red)

  test_cell.text('touch')

  grab(:vie_0_backgroun
```

Fichier: sync.rb

```
#  frozen_string_literal: true

b = box({ id: :the_box })

b.data(:canyouwritethis)

b.rotate(33)

b.rotate(88)

b.rotate(99)

b.rotate(12)

b.rotate(6)

b.data

b.touch(true) do

  b.data(:super)

  puts b.data

  # operation has two option write or read, it filter the history on those two options,  write retrieve all alteration

  # of the particle , read list everytime a particle was get

  # id retrieve all operation on a given ID

  # particle retrieve all operation on a given particle
```

```
end

# alert b.instance_variable_get('@history')

# box_rotate_history=b.history({ operation: :write, id: :the_box, particle: :rotate })

# puts "get all all rotate write operation :  #{box_rotate_history}"

# first_rotate_operation_state=b.history({ operation: :write, id: :the_box, particle: :rotate })[0][:sync]

#

# # we check if an operation synced (that means saved on atome's server)

# puts "first rotate operation state  :  #{box_rotate_history[0]}"

#

# # we check if an operation synced (that means saved on atome
```

Fichier: encrypt.rb

```
# frozen_string_literal: true

encoded=A.encrypt('hello')

text("encrypted string : #{encoded}")
```

Fichier: b64_to_image.rb

```
# frozen_string_literal: true

image({ id: :logo })

def_2 = "M 536.75,-0.25 C 536.75,-0.25 536.75,-0.08 536.75,0.25 536.75,25.82 536.75,1023.75 536.75,1023.75
536.75,1024.08 536.75,1024.25 536.75,1024.25 L 486.75,1024.25 C 486.75,1024.25 486.75,1024.08 486.75,1023.75
486.75,998.18 486.75,0.25 486.75,0.25 486.75,0.24 486.75,-0.2 486.75,-0.2 L 536.75,-0.25 536.75,-0.25 Z M
```

```
536.75,-0.25"

vector({ id: :my_svg, top: 33, left: 99, data: { path: { d: def_2, id: :p2, stroke: :red, 'stroke-width' => 3, fill: :green } } })



wait 1 do

  grab(:view).b64_to_tag({ id: 'my_svg', target: :logo })

end
```

Fichier: copybck.rb

```
# frozen_string_literal: true



new({ particle: :copy }) do |items_id|

  # alert items_id

  unless items_id.instance_of? Array

    items_id = [items_id]

  end

  grab(:copy).collect << items_id

  # new_copy_group=group({ collect: items_id })

  # @copy << items_id

  # @copy

  # items_id

  grab(:copy).collect

end

Atome.new({ renderers: [:html], id: :copy, collect: [], type: :group, tag: { system: true } })



new({ read: :copy })



new({ particle: :paste }) do |params|
  all_copies = grab(:copy).collect

  if params == true

    copies_found = all_copies.last

  elsif params.instance_of? Integer

    copies_found = all_copies[params.to_i]

  elsif params.instance_of? Array

    copies_found = [all_copies[params[0]][params[1]]]

  end



  copies_found.each do |copy_found|

    if grab(copy_found)

      pasted_atome = grab(copy_found).duplicate({ left: 333 })

      pasted_atome.attach(@id)

    end

  end

  copies_found

end



b = box

c = circle

t = text(:hello)



# b.copy([c.id, b.id])

# b.copy(b.id)



# tes
```

Fichier: delete.rb

```ruby
# frozen_string_literal: true


b = box({left: 99, top: 99})

b.text({ data: 'click me' })


# wait 5 do

#   b.delete(:left)

#   puts 'o'

# end

orange=''

b.touch(true) do


  c = grab(:view).circle({id: :circling, left: 222, color: :orange, blur: 1.9 })

  orange=c.box({id: :boxing,color: {id: :orange_col, red: 1, blue: 0.2 }, width: 33, height: 33, left: 123})

  orange.shadow({

            id: :s1,

            # affect: [:the_circle],

            left: 9, top: 3, blur: 9,

            invert: false,

            red: 0, green: 0, blue: 0, alpha: 1

          })

  c.box({id: :boxy,color: {id: :red_col, red: 1 }, width: 33, height: 33, left: 333})

  c.text('tap here')

  wait 0.5 do

    c.delete(:left)

    wait 0.5 do
```

```
      # orange.color(:pink)

       c.delete(:blur)

    end

  end



  c.touch(:down) do

    grab(:circling).delete({ recursive: true }) if grab(:circling)

  end

  # alert orange.apply

  # wait 4 do

  #   grab(:circling).delete({ recursive: true })if grab(:circling)

  # end

end
```

Fichier: grab.rb

```
#  frozen_string_literal: true



# the grab method is used to retrieve atome using their ID

a = box({ id: :my_box })



# to alter or add a particle you can use the variable, here we set the left value

a.left(33)



# to alter or add a particle you can use the variable

# it's also possible to alter or add a particle without a variable using grab and the ID of the atome , here we set the top value
```

```
wait 1 do
  grab(:my_box).top(5)
end
```

Fichier: allow_system_right_click.rb

```
# frozen_string_literal: true


b=box({ left: 12, id: :the_first_box })
b.touch(true) do


  alt=b.alternate(true, false)
  if alt
    b.color(:green)
  else
    b.color(:red)
  end
  allow_right_touch(alt)


end
```

Fichier: server.rb

```
# frozen_string_literal: true


user_password = {global: :all_star, read: { atome: :all_star }, write: { atome: :all_star } }


human({ id: :jeezs, login: true, password: user_password, data: { birthday: '10/05/2016' },selection: [], tag: { system: true
} , attach: :user_view })
```

```
c = box({ color: :yellow, left: 333 })

c.touch(true) do

  c.message({data: 'cd ..;cd server;ls; pwd', action: :terminal }) do |result|

    puts "shell command return: #{result}"

  end

  c.message({data: {source: 'capture.rb',operation: :read  }, action: :file}) do |result|



    puts "file read encoded_content: #{result[:data].gsub('\x23', '#')}"

  end

  c.message({ action: :file,data: {source: 'user_created_file.rb', operation: :write, value: :hello }})do |result|

    puts "file creation result : #{result}"

  end



  A.message({ action: :terminal , data: 'cd ..;cd server;ls; pwd'}) do |result|

    puts "result : #{result}"

  end

  {} #must add an empty hash else events events method will interpret keys of the has
```

Fichier: console.rb

```
# frozen_string_literal: true



box({id: :my_box})

console(true)
```

Fichier: run.rb

```
#  frozen_string_literal: true
```

```
b = box({ left: 333, color: :blue, smooth: 6, id: :the_box2 })




exec_code=lambda do


  wait 1 do

    b.color(:violet)

  end


end


b.run(exec_code)
```

Fichier: meteo.rb

```
#  frozen_string_literal: true


b = box

b.meteo('chamalieres') do |params|

  text({ data: params[:main][:temp] })

  puts params

end
```