


GUYI DOCUMENTATION

TABLE OF CONTENTS

Chapter 1 :The Interface.....	2
Menus.....	2
File Menu.....	2
Full menu.....	3
Logs menu.....	4
Components(Comps).....	4
Basic.....	5
Html.....	5
Icons.....	6
Outline.....	9
Selected/Small Preview.....	10
Inspector.....	11
Comps(component) Props.....	12
The html props.....	13
The inspector Events tab.....	14
Chapter 2 : Simple Layouts.....	15
Column.....	15
Row.....	20
View.....	20
Some important procedures.....	20
Adding comps to your app tree.....	20
Changing comp's/component's id (name shown in the outline).....	20
Rearranging children of a comp/change parent.....	21
Layout 1.....	21
Try creating the layout below.....	23
Adding photos to an app.....	23
Where to Next?.....	25
Chapter 3 : Coding.....	26
Our first function.....	27
Preset variable and functions in guyi.....	27
App 1,Counter.....	28
How does it work?.....	31
The load function.....	31
The update function.....	32
App2 : BMI Calculator.....	33
Load and update functions.....	35
The heightChange and weightChange functions.....	36
Calculate function.....	37
How to download this module/app.....	37
Chapter 4: Modules.....	39
Our first module.....	39
4. Understanding the table.....	40
Using the module.....	41
Using the props and events provided from the inspector in the module.....	42
Uploading Modules.....	44
Downloading modules.....	44
Chapter 5: Advanced comps.....	45
	45
Above are MapList and CondList respectively.....	45
MapList.....	45
Example.....	46
CondList(Condition list).....	47
Finishing notes.....	48
Chapter 6 : Using Runners.....	49
Currently(guyi 0.8.2) guyi has 3 runner.....	49
Installing runners.....	49

Runner interface.....	49
Using guyi web(the guyi.html file you downloaded).....	50
Setting the default app for guyi windows runner.....	50
New methods that work on specific platforms.....	50
Android functions.....	51
Windows(Win32 functions).....	52
Change Log.....	55
Guyi 0.8.2.....	55
New layout suitable for smart phones.....	55

Chapter 1 :The Interface

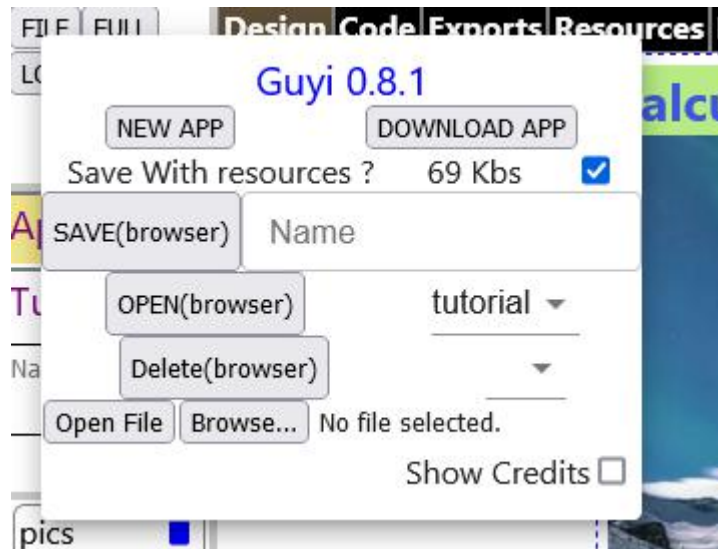


The Red markings are names of the different parts of the interface.
Below we go through each part in detail.

Menus

File Menu

- Gives access to some of the most used features in Guyi.
- When clicked opens a popover shown below.



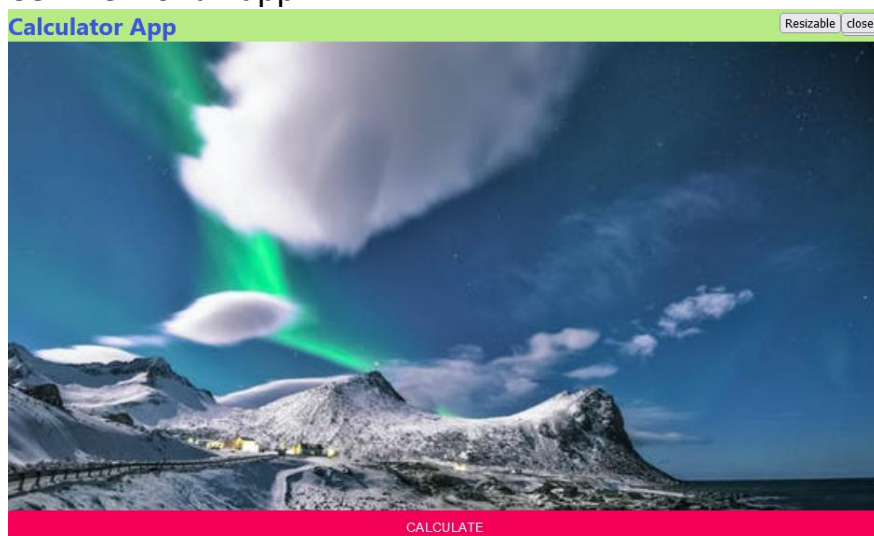
Functions of the various buttons;

- NEW APP- starts a new empty app, old app not saved.
- Download app-saves the app to a file with .guyi extension(ending)
- SAVE(browser)-saves the app to current browser
- OPEN(browser)-opens a save app in this browser
- Delete(browser)-deletes an app stored in current browser.
- Open file-loads an app from a downloaded file, great for inter-browser transfers of apps.
- Check the “save with resource” if you need photos saved with the project, usefull if app doesn’t fit browser storage.
- Check “Show Credits” if you want to see info about guyi and its developer(T.S.O)

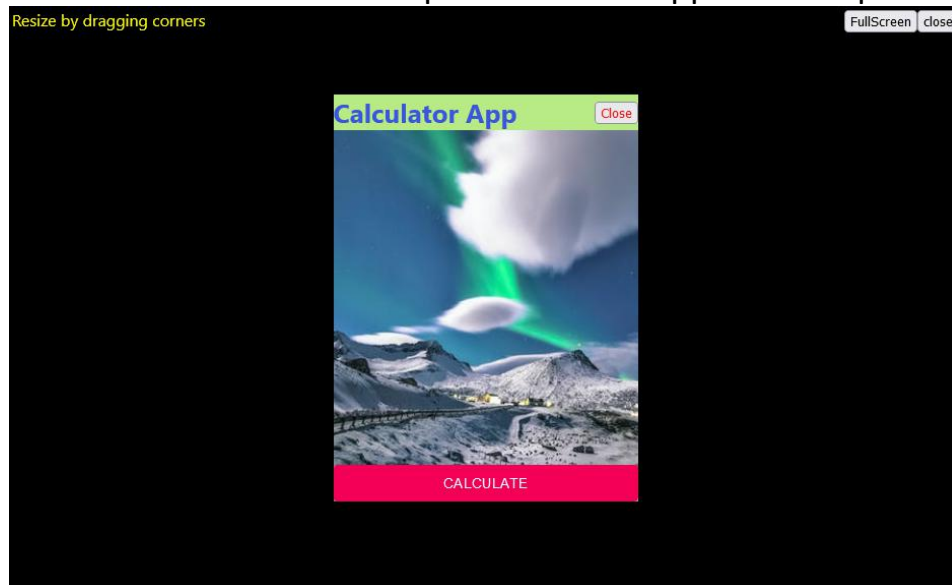
Full menu

Click if you need a full view of the app click this.

Below is a full screen view of an app.



- To have a re-sizable preview - so as to test the app in different screen sizes click the Re sizeable button on the top right corner.
- The close button return us to the edit preview of the app we had opened initially.



- In the re-sizable preview we can go back to the full screen preview by clicking the FullScreen button on the top right
- Click the close button on any of the previews to go back to the Edit screen.

Logs menu

The log menu allows us when we start coding to know the current values of some important variables like

- ◆ State
- ◆ Local
- ◆ Glob

Components(Comps)

This are the basic building blocks of interfaces in Guyi.
We combine the in different way to create great interfaces.



Lets know their names first, in Order from left to right;

Basic



1. Text- a read only display of text/words/strings
2. Row- a horizontal display of child comps.
3. Column- a vertical display of child comps.
4. Button-a clickable display with text.
5. TextInput- used to get data from a page user.
6. View- a layout component like row or column but takes one child and has many props to customize.
7. Image- displays an image resource from net or local file.
8. Ripples-creates waves/ripples when clicked.
9. Html-displays plain html content on the page.
10. MapList-creates views from give data - powerful and advance comp.
11. ConList - shows only its children that pass a condition,used to toggle comp's visibility on and off.

Html



1. MButton - better looking button.
2. MTextField - better looking text field
3. MAvatar -used to show circular images of profiles mostly.
4. MBadge - used to create a badge
5. MChip - used to show list of things
6. MTooltip - shows a messages when child is hovered or long pressed
7. MText - better looking text
8. MCProgress - circular progress dialog
9. MLProgress - linear progress dialog
10. MDialog - creates a popup on a page when open condition is true.
11. MSnackBar - a message display that shows up from a screen edge
12. MCard - a box with great box shadow used to create cards.
13. MPaper - just like Card but with less elevation
14. MSwitch - a switch component
15. MSlider - a slider component
16. MFab - floating action button
17. MCheckBox - a checkbox comp
18. MRadio - a radio comp
19. MFormCL- a form control label takes a comp and gives it a label.
20. MLink - creates hyper links to pages on the net
21. MSelect- creates a select dialog for users.

Html comps start with M since they are actually just wrappers of material-ui components of react.

NOTE: The Html comps will only be supported in the build system of the web and so may not work when Windows and Android renders of guyi apps are released.

Icons

Are found below the html comps

Enter the name or a query to find icons that march.

To get results click outside the input box

The icons can then be used as any basic comp.

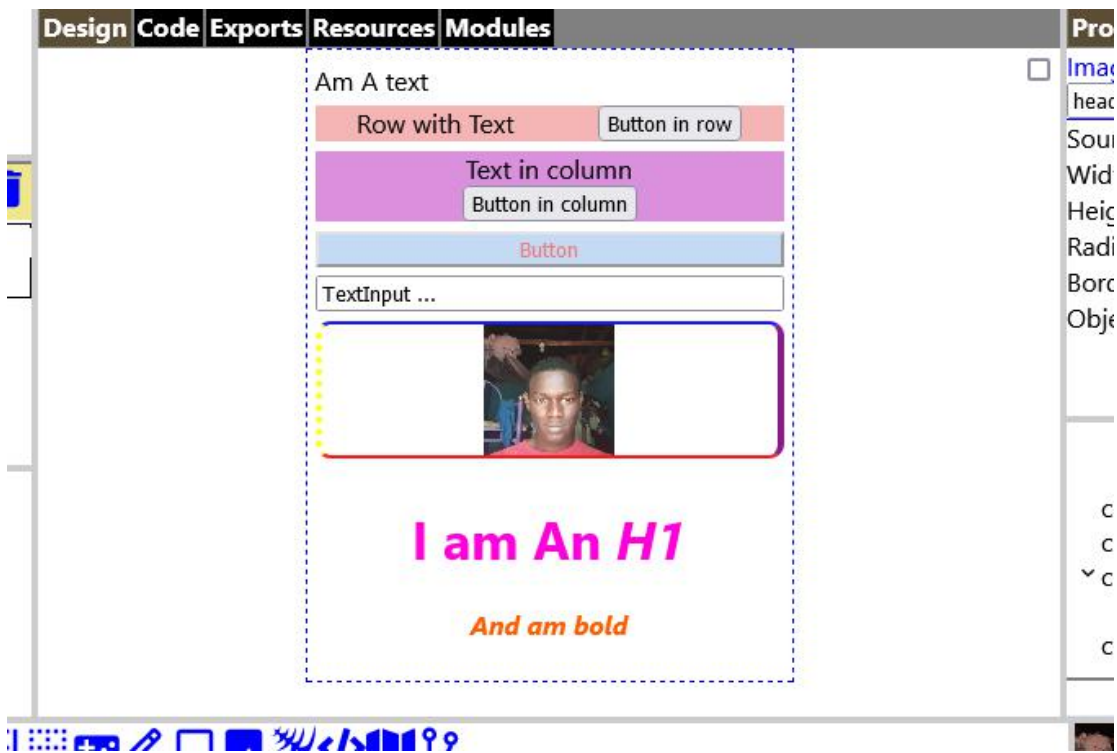
Icons are provided from material-ui , fontawsome and 'bi-react-icons',so the names start as Md,Fa,Bi respectively

Display of basic comps

Below is a pic of the basic comps not including the MapList, CondList and Ripples which will be look at latter.

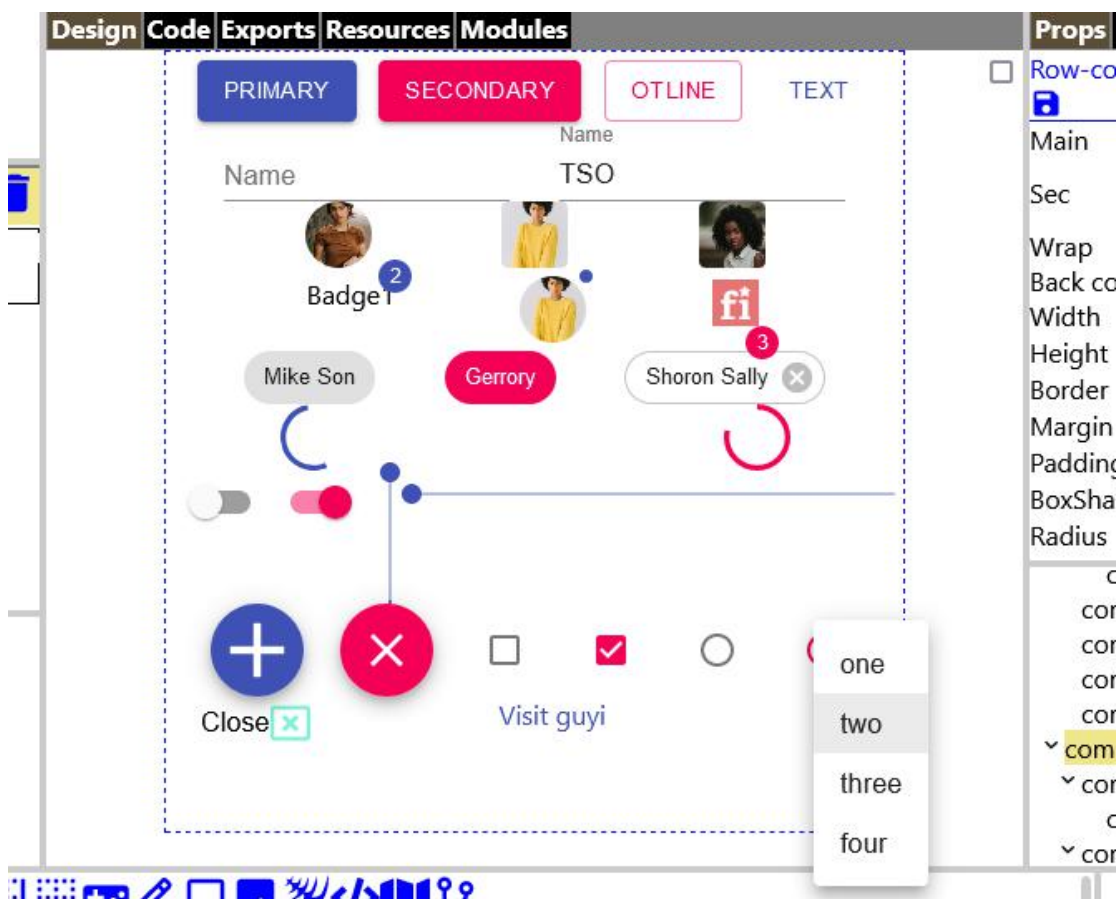
Note: The row , column and view comps are not displayed but used for layout purposes.

Note:Most of the Html comps also take other comps as children to display.



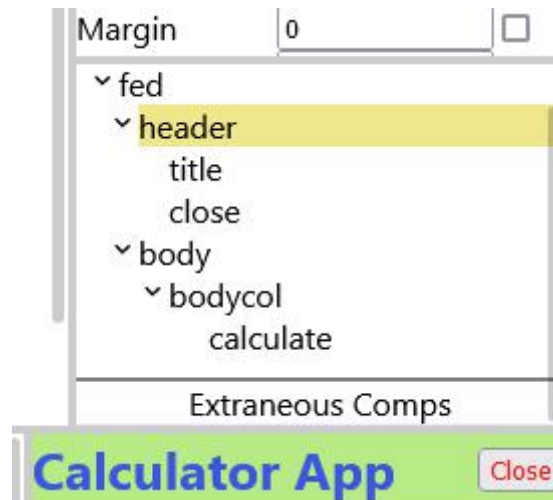
- The rows 2 children are arranged horizontally.
- The columns 2 children are arranged vertically.
- The image above is in a view component that gives the borders and border radius shown
- The last two texts are actually 1 html comp (component).

Html Comps preview



- This app page shows some of the html comps
- These comps may only work only in web build of guyi not Android and Windows version of the apps.
- We will learn how to create Modules (components made up of basic comps that we can reuse as if they where basic comps)

Outline



Shows the app tree.

The app tree - the layout of the app in terms of parent and its children

The ids of the comps are the one shown in the outline

Eg From above

- ✓ feb is the parent of header and body - both have same indentation and are below feb in tree
- ✓ Header as 2 children title and close
- ✓ Body has 1 child bodycol which also has 1 child calculate.

That's how we analyze the app tree

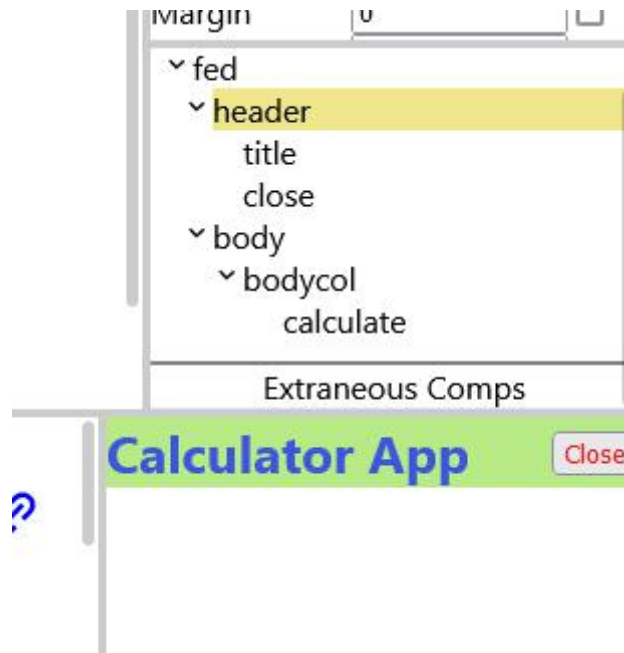
The type of comp represented by the id is shown in the props Inspector tab discussed latter

The currently selected component for editing is shown in the color 'khaki' e.g header show above

To select a new comp click on its id

NOTE:The collapse mechanism has issues to be fixed soon.

Selected/Small Preview



Show the currently selected comp and its children if any

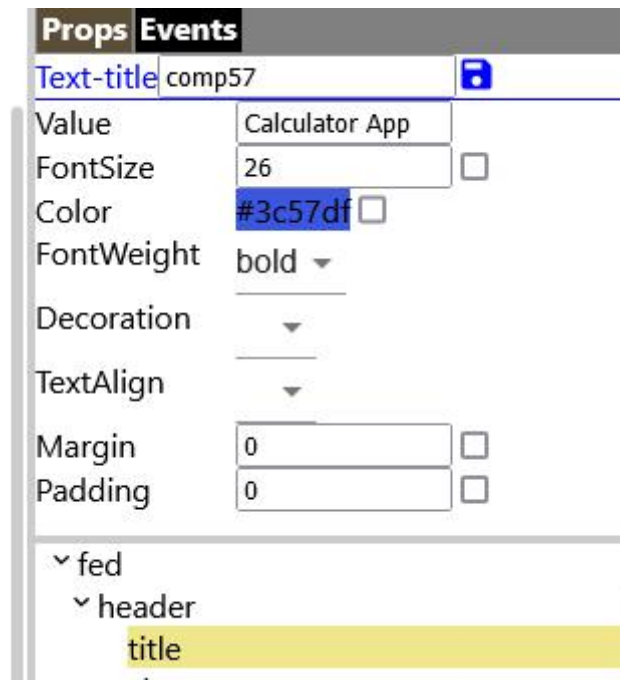
e.g from above the selected comp from the outline is the one whose id is header.

We see the preview as show ;

- ✓ Header as 2 children title and close , from outline.
- ✓ Title is a text comp with the value 'calculator app' while close is a button comp with value close
- ✓ Header is a row as its children are displayed horizontally(left to right)

When we change the selected component we see its preview here immediately.

Inspector



The inspector is the tool we use most when designing interfaces. Once we have a component selected in the outline we customize it in the inspector. Changing the selection updates the inspector too.



The first row gives us some info on the selected component eg from above :-

- ✓ The type is Row and the id is header.
- ✓ You can enter a new id and click save to apply, useful after adding new comps to app tree.

E.g. from above we have title which is a text comp selected in the outline, so its props are shown in the inspector.

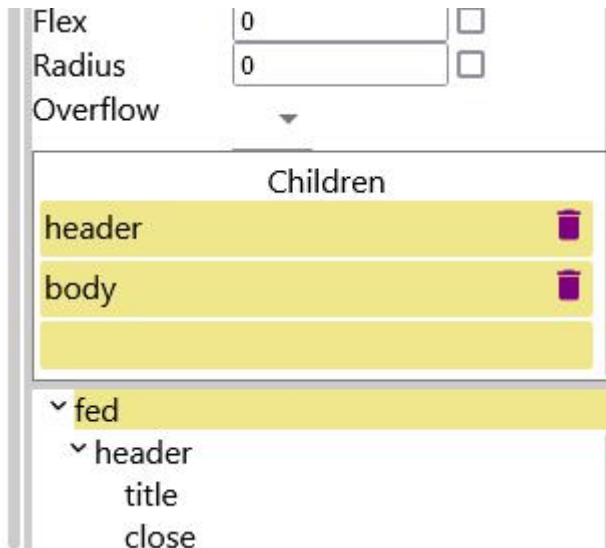
From the props tab of the inspector we see that we can change the following values/props of the Text comp with id title.

- ✓ Value -text prop we enter value displayed by the text.
- ✓ FontSize- number prop change how long/big the text is.
- ✓ Color - color prop change the color of the text value.
- ✓ FontWeight - select prop, choose the fontWeight/thickness to use with text value.
- ✓ Etc -etc

We also have other types of props like;

1. component props that allow children to be added to a comp present in Row , column and View comps
2. Json props allow users to enter json, used by map list basic comp.
3. Html props allows you to enter html by using inbuilt html editor, used by html comp.
4. Cond Props, allows use to set conditions used by CondList to display/hide items, used by condlist.

Comps(component) Props



It is a prop used to add child components to a layout comp like row column and view, also most of the Html comps take children.

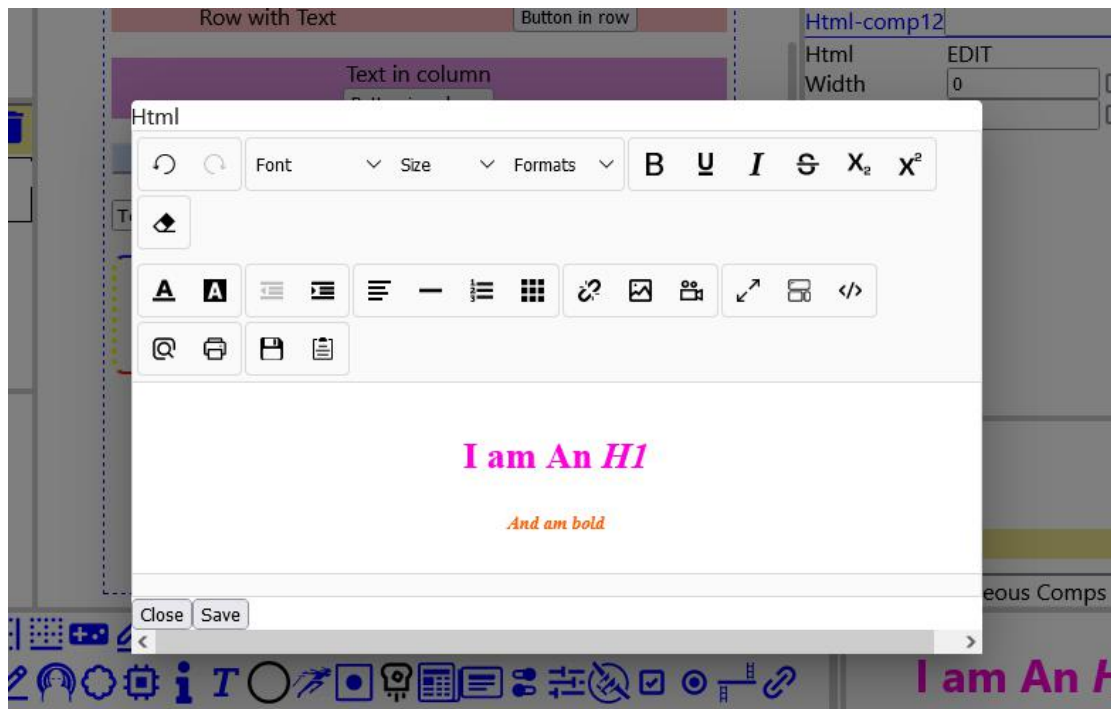
Above comps prop shows that the comp has 2 children
One with id header and the other id body.

How to add children to a component

- ❖ Select the comp you want to add children to, should be one that accepts children e.g row, column, view, most html comps too.
- ❖ Scroll to the comp props field/area
- ❖ Drag the comp from the Components area to the row you want it placed
- ❖ You can place the comp at any position in the list of children the others will be moved accordingly.
- ❖ The new comp will get a random id starting with comp+a number
- ❖ It's best to change the id as explained above after adding a component.

NOTE: using the randomly generated id is not safe when you save and reopen the editor so change it before closing.

The html props



The html props field allows you to create html from a WYSIWYG editor like shown above. You can switch to manual content creation at any time. And also paste html to the editor at will. Hover over the buttons to know their functions.

Json props



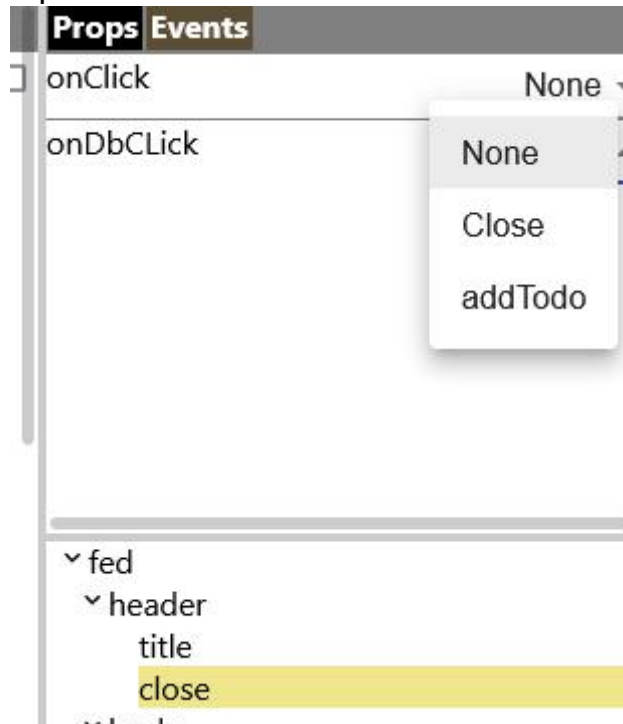
Json editor allows us to create json string and ensures they are well formatted before saving .

NOTE: Above types of props (json, comps, html) will be explained further in their respective areas.

The inspector Events tab

The inspector has two tabs the default we looked about above is the props tab, and the second is the events tab.

Click on the events tab to open it.



The events tab is used to assign functions to events emitted/produced by components. Events are functions called when a user performs an action in a component or the component is in a give state

E.g

- When a user clicks a button the an onClick event is fired.
- When a user enter some text in a textInput comp then a onChange event is fired/emitted.
- If we need to do some thing at such times then we register function that run when the events occur using the events tab.

From above we have the button with id close selected so we may register a click or double click event listeners/function.

NOTE: We will take more on events and the events tab when we reach the coding chapter.

Chapter 2 : Simple Layouts

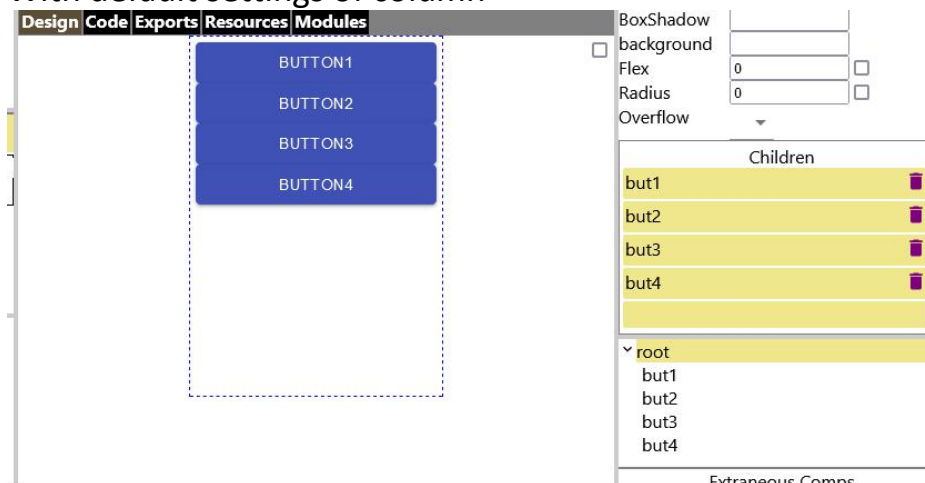
Before we can start creating simple layout lets understand how Guyi layout system the flex box works.

Column

Column is a flex box that arranges it children in a vertical direction, I.e top to bottom in the order they are provided. Eg

I will use MButton comps for these example but any comp including other columns,rows and view etc can be used.

With default settings of column

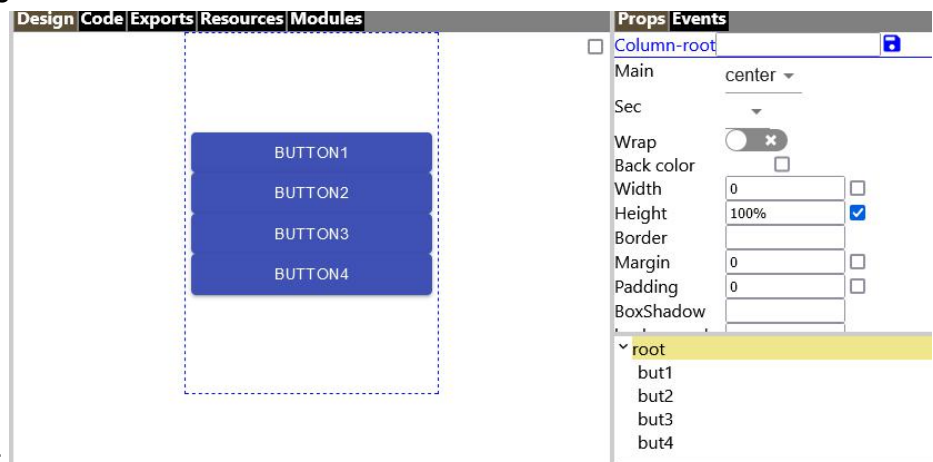


Main prop/Justify-content for web developers

The main prop of column determines the spacing between comps horizontally.

The default is center

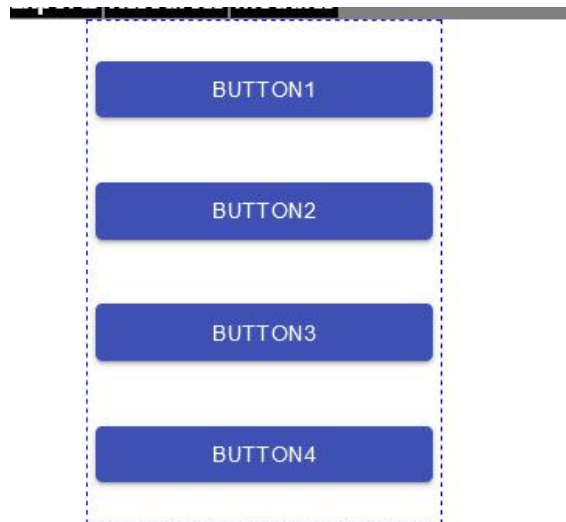
The following are the options



When Main is set to center

When main is set to space around

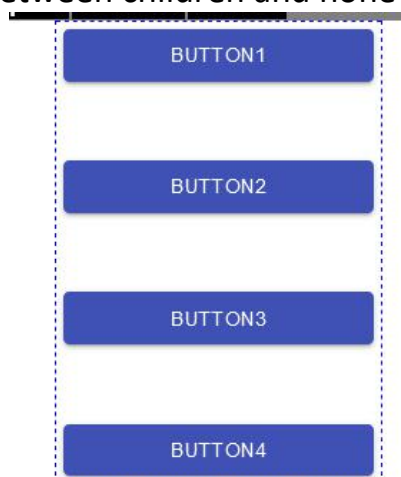
Spaces at end halve the space between children



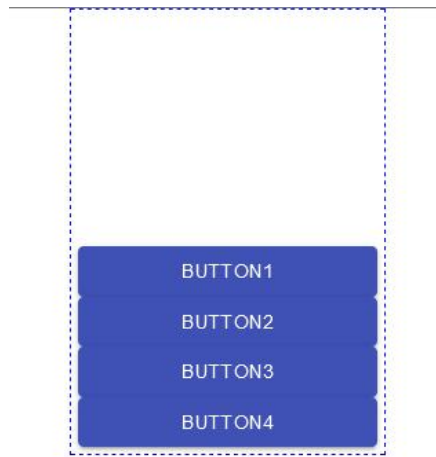
When main is set to space evenly
Equal spaces between children including top and bottom



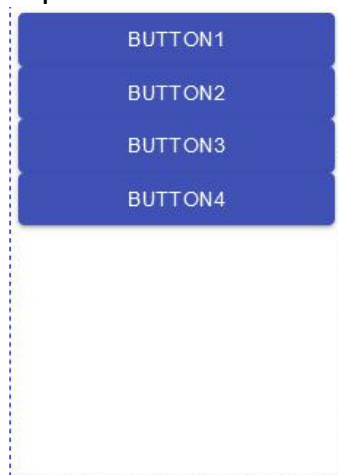
When main is set to space between
Equal spaces between children and none at start and end



When main is set to flex end
Children placed at end of container



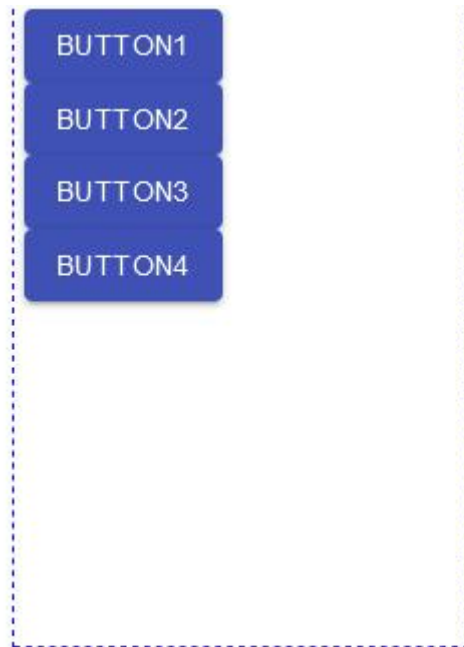
When main is set to flex-start(the default)
Children placed at start of container



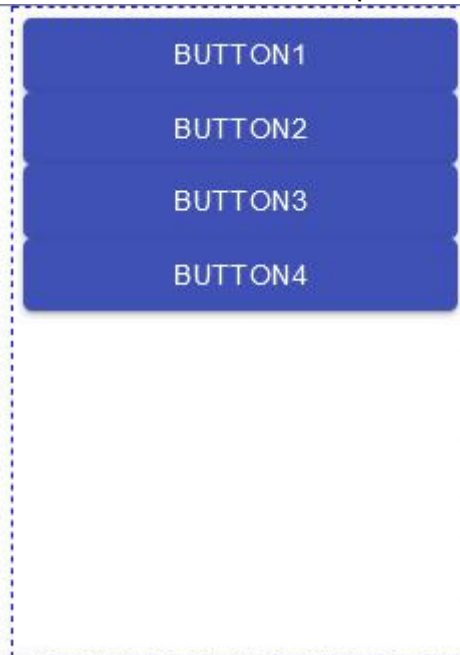
Sec prop/Align items prop

Determines arrangement of children horizontally,
Default is stretch

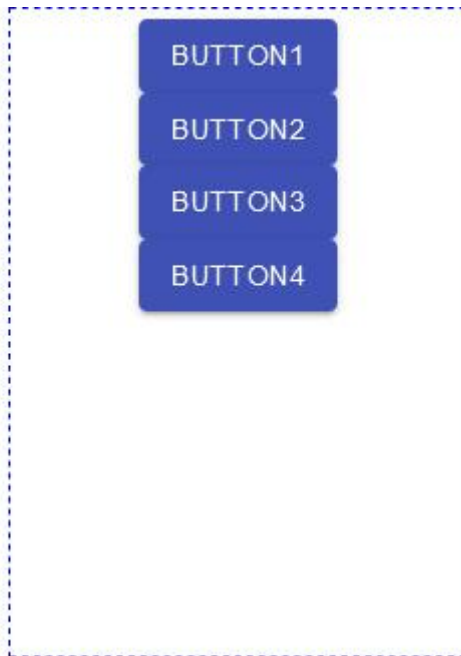
When sec is set to flex start



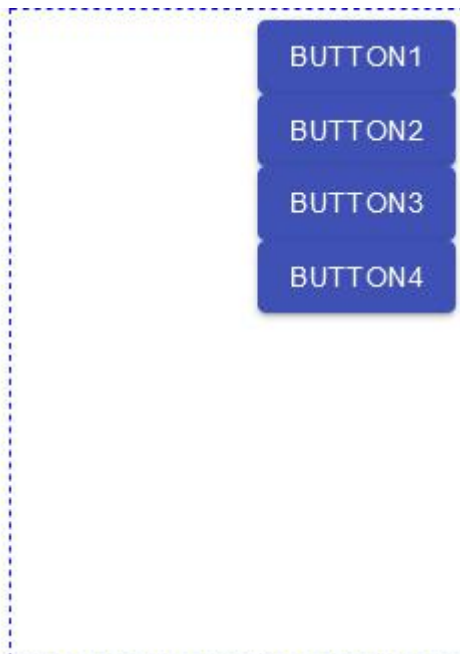
When sec is set to stretch (the default)



When sec is set to center



When sec is set to flex end



Row

- Is a layout component like column discussed above.
- Arranges its children horizontally from left to right
- It also has the main and sec props.
- The main prop arranges content horizontally while the sec prop arranges contents vertically , I.e the opposite direction as these props in the column comp.
- I won't fill these space by examples so test them in the editor.

View

- Is also a layout comp that only accepts 1 child.
- It has a center prop that determines if the element is center inside the view comp.
- View has the most props and so is the most customizable component in guyi.
- We will cover most of them in the simple examples in future chapters.

Some important procedures

Adding comps to your app tree

- Select a comp in the outline of a type that takes a child/ children eg row,column,view and most html comps.
- Scroll to the comp prop children/ child prop that takes a child/children
- Drag the comp type you want as the child from the components area and drop it on the comps field in the position you want
- Drop the comp in the position you want not necessarily the last position

Changing comp's/component's id (name shown in the outline)

- Select the comp in the outline
- The inspectors props tab should show the selected comps props/properties.
- Enter the new id at the input field in the first row of props tab and click the save icon to apply it

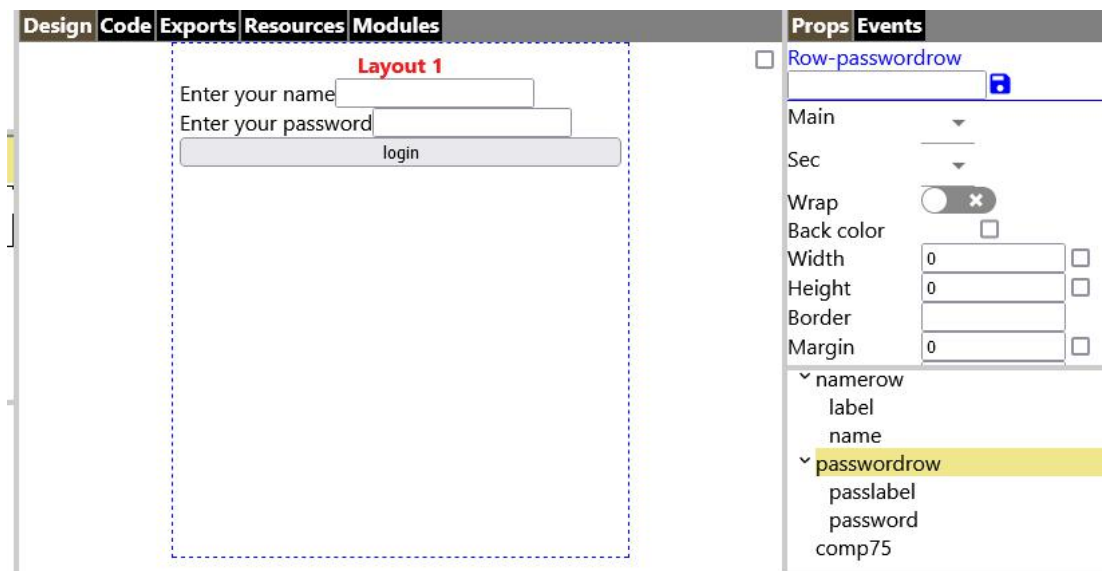


- The 1st row also tells the type and current id of a component eg from above type is row and id is passwordrow.

Rearranging children of a comp/change parent

- In the props tab of the inspector go to the children/child field and delete the comp from there.
- The outline extraneous components area now shows the component you deleted.
- You can now drag this comp to a new child position or even drop it on a different component ie change the parent of the component.
- The comp will be removed from the extraneous comps list

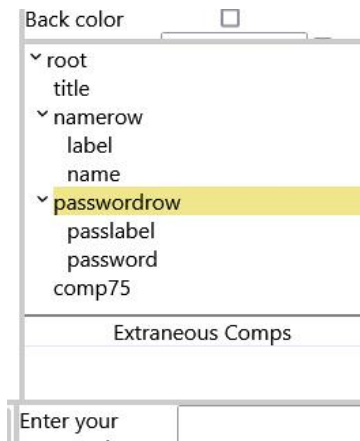
Layout 1



Steps ;

Note: The root comp is a column and starting point for all components, see in the props tab.

- ◆ Add a Text comp to the root comp (the usual drag and drop discussed earlier)
- ◆ Select the text in the outline (it will have a random id)
- ◆ Change the id to title.
- ◆ Re-select the text in the outline.
- ◆ Set the following props value to "layout 1", select color "red", fontweight "bold", textAlign "center".
- ◆ Add a row to root below the title comp
- ◆ Change the new row's id to namerow .
- ◆ Select the row we have added and scroll to its children props and add a text and TextInput field and give them the ids label and name
- ◆ Add the new row for password and give it two children a text and text input.
- ◆ Finish by adding a button and set its value to "login".



- ◆ You should have the tree below
- ◆ Set all the ids and props to have the finished app



- ✓ Try changing the various props of the components to see what they do.
- ✓ Try the following
 1. Changing the main of root comp to 'space-around', and all the other to see the effect
 2. Change the sec of the root to the various options to see the effect
 3. Change the main of namerow to space-between
 4. Change the main of passwordrow to space-between

To download the above layout/module :

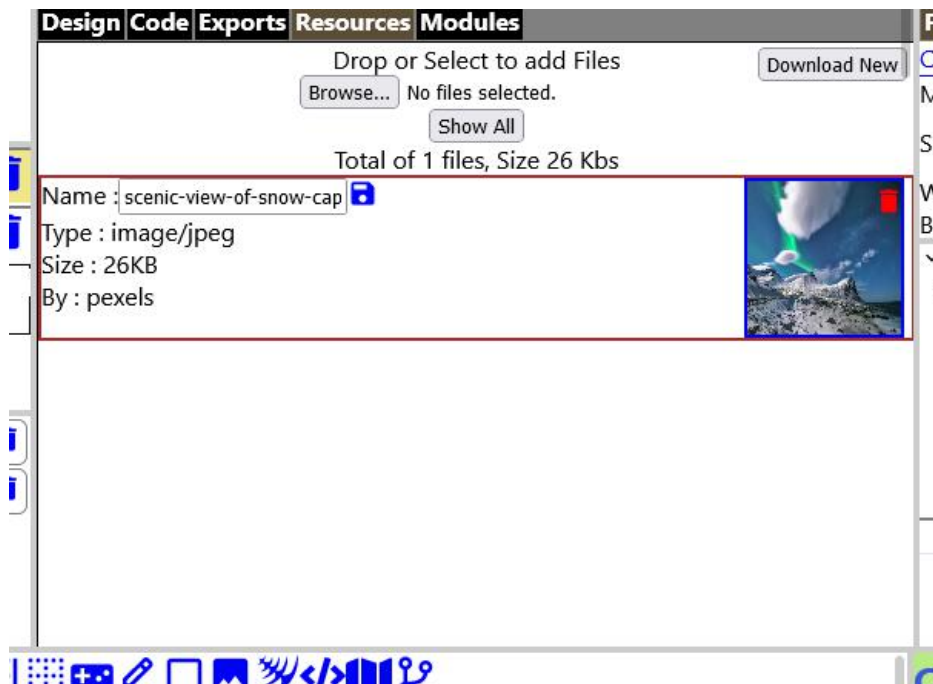
- ✓ In the work area (center of editor) open the modules tab.
- ✓ In the modules search for "layout 1" or "t.s.o"
- ✓ In the results find the layout and press the comp download icon to download it.
- ✓ In the modules area click the layout 1 to open it.

Try creating the layout below



Adding photos to an app

- In the work space select the resources tab
- In resources tab has two pages/sub-tabs
- The view resources tab and download resources tab
- You change between the two sub-tabs using the top right button
- Below is the view resources tab-it shows current resources/files/photos in the app



- Below is the download resources tab - use it to download images from pexels - select the images you want and press save button to save them.



- The resource will appear in the components area below the icons after you do some modifications to the app .ie set a prop or change tabs or modules.
- Photos can be used with Image comp, MAAvatar comp and view comp as background image. Set them as follows ;
 - For images set the source prop to “res:-name-of-image” or drag the image from components area to the source prop to set.
 - For MAAvatar set the source prop manually as “res:-name-of-image” or by dragging the image from the components area to the source prop.
 - For view set the background image prop manually or by drag and drop as done above for the source prop.

Where to Next?

- ❖ Create new apps and add other components, icons and images
- ❖ Use the inspector to experiment with what the various props do.
- ❖ When setting resource names manually it is easier to rename the photos first, you can do so in the resource view tab(not the resource download tab,with pexel images)

Chapter 3 : Coding

- We programme guyi using JavaScript.
- Js is easy to learn and use.
- If you are not familiar with js take a course on YouTube or even better read some tutorials online as you practice.
- Coding is guyi uses normal js, interpreted by the browser, nothing special.



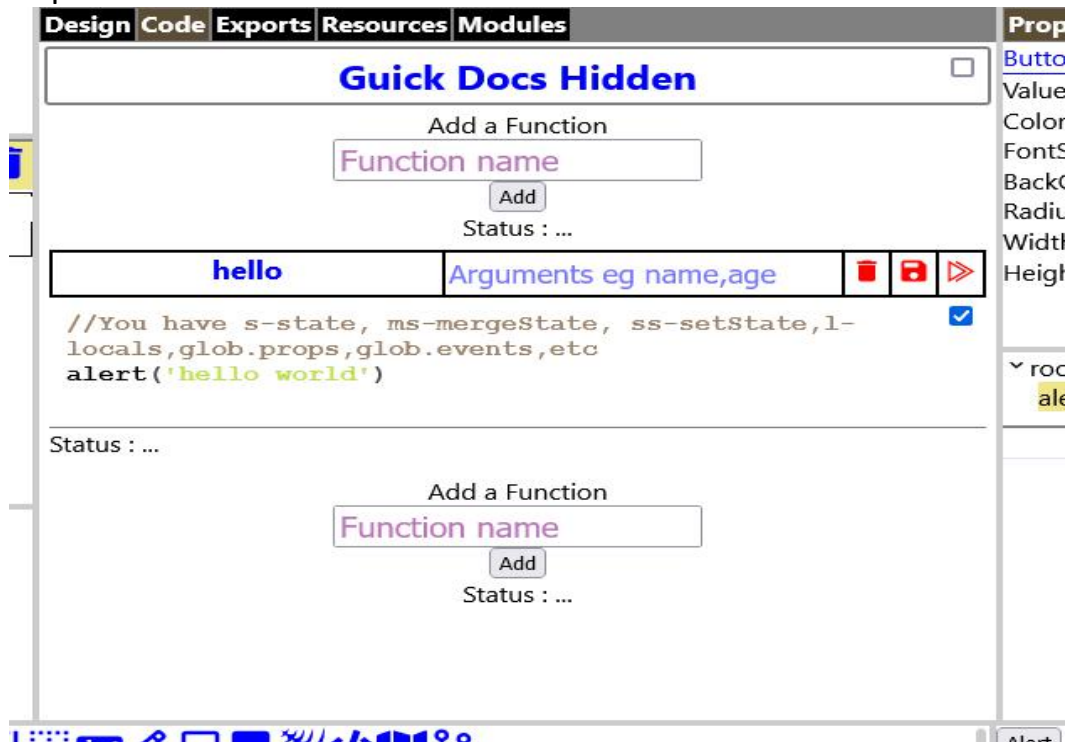
The code tab is where we do all the coding in guyi.

Note the following about the code tab

- You can hide/collapse the body of a function by un-checking the checkbox at the top right of the function.
 - To add a function give it a name and click add.
 - You can move functions around in the interface by dragging their headers and dropping on another function header to place it there.
 - Currently we can not rename a function so just copy its body to a new function and Delete the old one.
-
- Functions are run when events occur eg button clicked, text changed, app started, app updated.
 - In guyi we have 2 special functions the update and load functions.
 - update-function is called when the app is about to be updated, more on that later.
 - load-function called when app is started.

Our first function

1. Add the function “hello” to your app, start a new one if you can ie file=>new App commands to do so.
2. Give it the code ‘ alert(“hello world”) ’ as shown below and save it, if no error it will show compile successful.



3. To run the function we must register it as an event.
4. So create an app with a button as show below



5. To run the function when the button is pressed, lets register it.
6. Select the button in the outline.
7. Open the events tab of the inspector.
8. Select the hello function as the event handler of the onClick of the button.
9. Open the code tab and check the function if correct then back to the design tab click the button.
10. You should see an alert (pop up) showing the message hello world.
11. There we have it, our first function.

Preset variable and functions in guyi

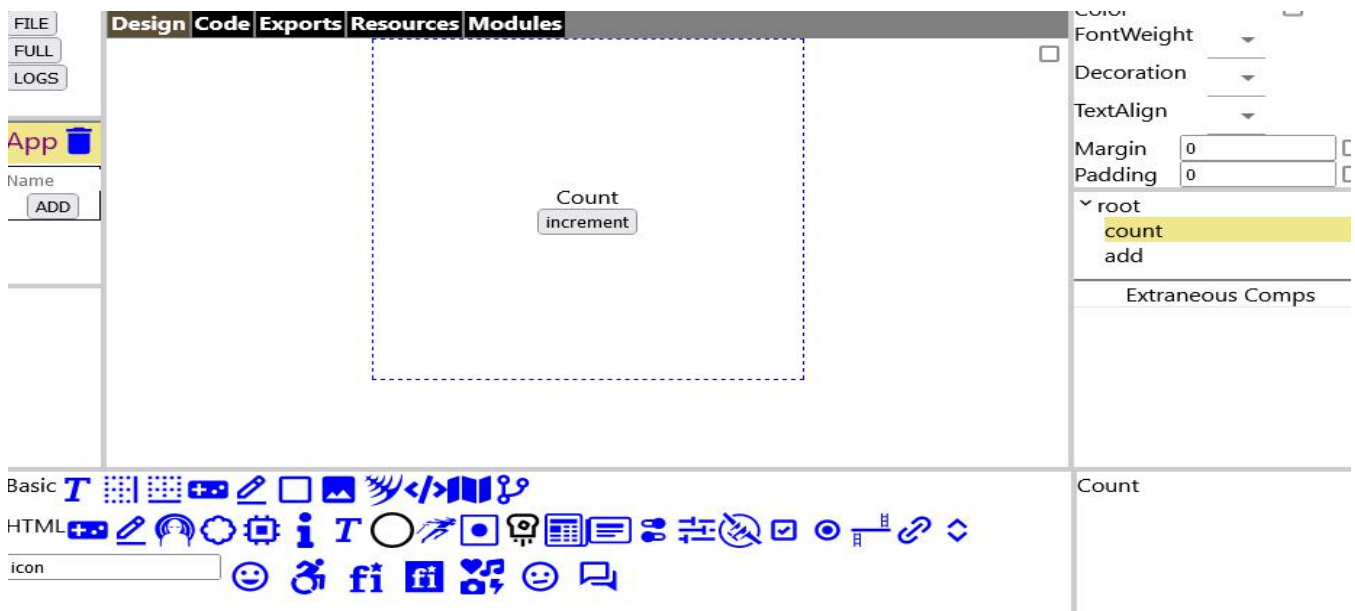
name	Info about	Usage
s	State variable, constant and should not	s.no,s.text,s['no'],s['text']

	be assigned to. When changed the app updates.	
l	Local variable like state but not constant can be assigned directly.	l.no, l.text , l['no'], l['text']
ss	A function available to set the state variable	ss({no;1,text:'Guyi page'}) ss({...s,text:'Guyi page'})
ms	Merge state Makes setting state easier	Eg ss({...s,text:'Guyi page'}) Same as ms('text','Guyi page')
glob	Global object carries important info and functions	glob.props.name glob.events.name
glob.props	Carries props given to the module	glob.props.Title glob.props.PageColor glob.props['Title page']
glob.events	Carries events passed to the module	glob.events['onClick'] glob.events.submitted
glob.tiePS	Very useful function that ties a prop to a variable's/state's value	glob.tiePS('id of comp','the props to tie', 'variable to tie to','default value')
glob.getFunc	Return one of the function you have defined, so that you can use it from another function.	const add=glob.getFunc('add') let c=add(1,2) C is 3

- ❖ Above are preset in all function and you can use them directly.
- ❖ These are passed as function arguments in position 6 onwards till 10 so don't have more than 5 parameters for your functions.
- ❖ We can use arrow functions inside these functions but they are only accessible from the scope of that main function.

App 1,Counter

When the add button is clicked the number on the text is increased by one.



This is the layout of the app.
See the root.

Creating the layout

1. Start a new app by, go to file menu and press “start new app”.
2. The outline has one comp root, which is always a column ie arranges its children vertically.
3. Select the root
4. Scroll to the children props of the root in the props tab.
5. Drag a text to the children props from components area ‘area with basic, html, icons and images if present at the bottom left of the editor’.
6. The text comp is the one with the ‘T’ icon.
7. Next drag a button not MButton to the children props of root.
8. Change the id of the text to count and that of button to add.
9. The above step is important, explanation on changing ids was discussed before.
10. Set these props for count value - ‘count’
11. Set these props for add value- ‘increment’
12. Props for root, sec - ‘center’, main - ‘center’,height-‘100%’
13. Chance are you could not set height of root to 100% since it allows only numbers.
14. To do so check the box at the end of the field, this works for all number fields.
15. You should have the layout and tree shown above by now.

Adding the code

Add and write the body of the function shown below.

We can reposition function in the editor by dragging its header (top part with name,arguments ,save and delete icons) to the header of another function to place it there.










Quick Docs Hidden

Add a Function

Function name

Add

Status : ...

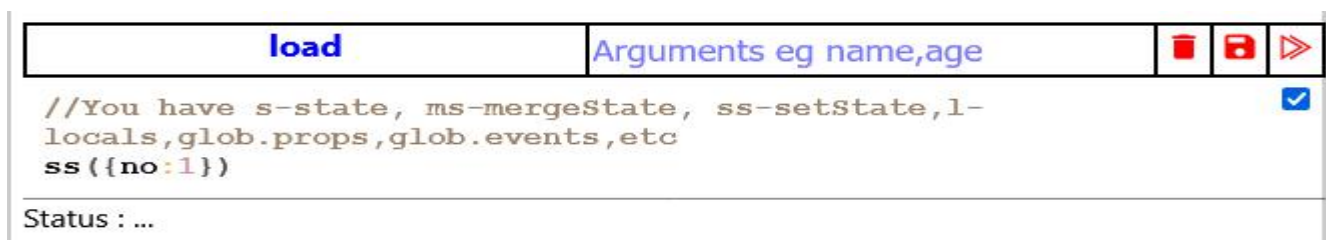
update	Arguments eg name,age	  
<pre>//You have s-state, ms-mergeState, ss-setState, l- locals,glob.props,glob.events,etc glob.tiePS('count','value',s.no)</pre> <div>Status : ...</div>		
load	Arguments eg name,age	  
<pre>//You have s-state, ms-mergeState, ss-setState, l- locals,glob.props,glob.events,etc ss({no:1})</pre> <div>Status : ...</div>		
increment	Arguments eg name,age	  
<pre>//You have s-state, ms-mergeState, ss-setState, l- locals,glob.props,glob.events,etc ms('no',s.no+1)</pre> <div>Status : ...</div>		

- The update and load functions are special functions that should not be assigned to an event.
- More on the two funcs later.
- Select the add comp, remember the selected preview will show you the select comp and its childrens if any in the inspector
- Open the events tab of the inspector to assign the functions
- Assign increment func to the onClick event of add(button with id add)
- To test if the function works, open the code tab, then open the design tab.
- Now click the button and the number will increase from 1 upwards.
- NOTE
 - ✓ If a function is not working properly, double check if you assigned it.
 - ✓ The assigned funcs do not show up immediately in the events tab, only after selection of another comp and back to the initial comp again(an issues to be fixed)
 - ✓ The design view will loose focus when you use the props tab, events tab or outline, and so functions will not run, to refocus the design view open another tab of the work area like code tab and then reopen then design tab (also an issue to be fixed)
 - ✓ Use the full menu to have a full screen view of the app.
 - ✓ In the full screen view we can change to re-sizable view and back using the top Right button, we can go back to edit mode by click the close button at the top right.

How does it work?

- Guyi manages the state for us, so we don't need to set values manually in code.
- Changes to the state variable will cause a re-render/update of the app with the new value.




The load function



- The first function to be called is the load function.
- In the load function is called when the app is started or restarted when we close the preview.
- The load function is great for initializing the state ie putting the first value of the state,
- In the above we set the state to the object {no:1} meaning “s.no will be 1 at the start of the app”
- “ss()” is a preset function that sets the state variable eg ss({no:1}) makes the state to be the object with the field no equal to 1.




- If we had another state variable to set eg text we would have `ss({no:1 , text: 'MyApp' })` , etc
- Any other code that will run once when the app is started, can be added here.

The update function

update	Arguments eg name,age			
<pre>//You have s-state, ms-mergeState, ss-setState,l- locals,glob.props,glob.events,etc glob.tiePS('count', 'value', s.no)</pre>				
Status : ...				

- The update func is called at the start after the load function and also before every update of the app.
- Ie it is called when we call the set state function (ss) or merge state function (ms)
- Do not call ms() or ss() functions here, unless you really know what you are doing, because doing so will cause an infinite recursion as ms and ss calls the update function
- The function `glob.tiePS`- 'global tie prop to state/value' is mostly used here.
- It takes 4 arguments ;
 - Id - the id of the comp to tie its prop to a state/value,
 - Prop- the prop we want to tie a value to
 - Value-the value to assign to the prop of the comp.
 - Default value- the value to use if the given value is null or undefined
- Above we set id to count , prop to 'value' and value to s.no (which initially is one from the load function),
- So the app's count comp will have the value of s.no, if that value changes (by use of ms or ss) then the counts value also changes with it.
- Instead of s.no we could have used any variable like l.no, l.message ,s.name etc
- Later we will see that we can tie to `glob.props.value` for modules when we reach there.

The increment Function

increment	Arguments eg name,age			
<pre>//You have s-state, ms-mergeState, ss-setState,l- locals,glob.props,glob.events,etc ms('no', s.no+1)</pre>				
Status : ...				

- This is not a special function like load and update.
- We have registered to the onClick event of the add button.

- So it is called every time we click the button.
- It has one line `ms('no',s.no+1)`
- The line that starts with `-` is a comment so not executed, it just gives us information.
- The `ms()` is a func that allow us to set the state in an easier manner
- `ms()` takes the following props:
 - a) Field - the field of the state to set a new value for.
 - b) Value - the value to set to the state field specified.
- `ms('no',s.no+1)` set the no field of the state object (s) and sets it to the old value (gotten by `s.no`) but after adding 1 to it eg `s.no+1` will read old value of state and adds 1 to it so 1 changes to 2 etc, `ms` will then set the value to `s.no`. Eg `s.no` will become 2.
- Since in the update func we tied the prop value of count to `s.no` when the app updates the value of count also update automatically
- The app updates only when we call `ms()` or `ss()`
- The `ms` is a shorthand/easier way of setting state the above would be same as `"ss({...s,no:s.no+1 })"`, albeit long but sometimes necessary for big projects.

Getting this app as a module/comp

- Open the modules tab and search for "counter" or "t.s.o"- the author name.
- In the results you see various modules that march the search, in the preview look for the one that resembles our app.
- In the preview try pressing the add button, it should work right in the preview area.
- To see the comp code and layout download it as a comp.
- We will learn more on modules and download as module later.
- Note : The preview is live and can be tested right before download.
- Downloading as a module(mod) will allow as only to use it but not see code or layout.

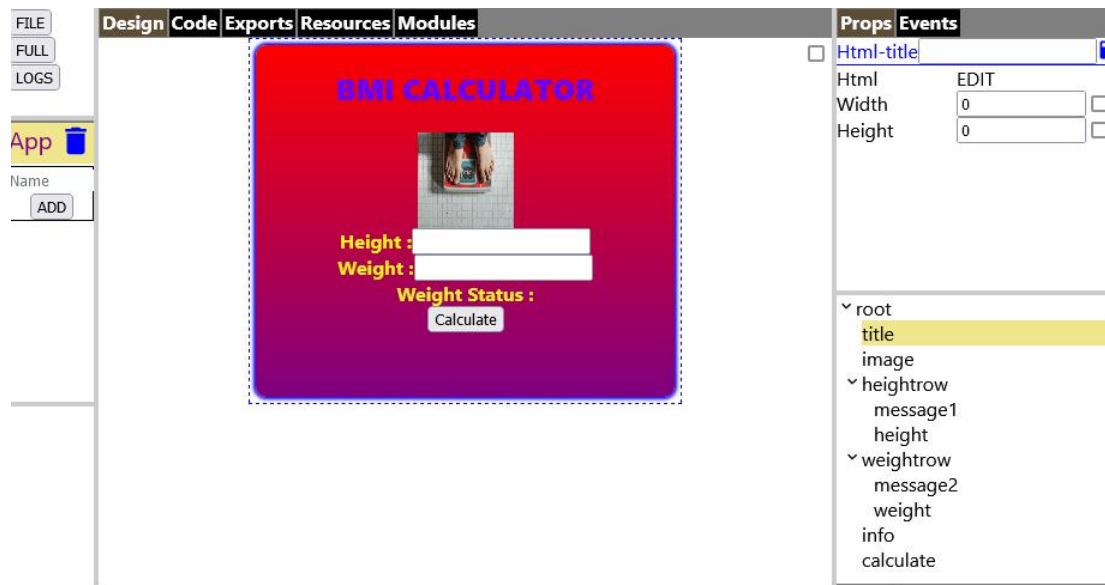
All Life cycle Methods

1. `init`- method called on the fist when the app is loading,use to initialize local variables.(I)
2. `update`-method called before app dates are drawn to screen.(before render). use to tiePS.
3. `load`-method called after 1st render. If returns function will be called on unmount. Use to initialize state(s),use fo ref or fetch data.
4. `drawn`-method called after every render.Returned method called before next render.Use if you need ref, of to fetch data.
- 5.

App2 : BMI Calculator

BIM- Body mass index - ratio of height to weight of a person.

Used to determine if one is overweight, underweight, fit , acceptable weights.






- Create the layout shown below by yourself
- Set all props , note on some props:
 - ✓ Roots background prop set to “linear-gradient(to bottom, red ,purple)”, you can use circular-gradient with background props too, info about this on the web.
 - ✓ Height of the root also set to 100% and main to center, and sec to ‘center’
- You can download it, instruction at end of example.
- heighrow and weighrow are row comps with 2 children each a text and textInput to enter the data.

Load and update functions




```
//You have s-state, ms-mergeState, ss-setState,l-  
locals,glob.props,glob.events,etc  
ms('height',e.target.value)
```

Status : ...

update	Arguments eg name,age			
---------------	-----------------------	---	---	---

```
//You have s-state, ms-mergeState, ss-setState,l-  
locals,glob.props,glob.events,etc  
glob.tiePS('weight','value',s.weight,'')  
glob.tiePS('height','value',s.height,'')  
glob.tiePS('info','value',s.info,'BMI info')
```

Status : ...

load	Arguments eg name,age			
-------------	-----------------------	---	---	---

```
//You have s-state, ms-mergeState, ss-setState,l-  
locals,glob.props,glob.events,etc  
ss({bmi:0,info:'',height:2.5,weight:60})
```

Status : ...

Add a Function

Function name

Add

Status : ...

- As usual we initialize the state in the load function.
- And tie props to values in the update function.

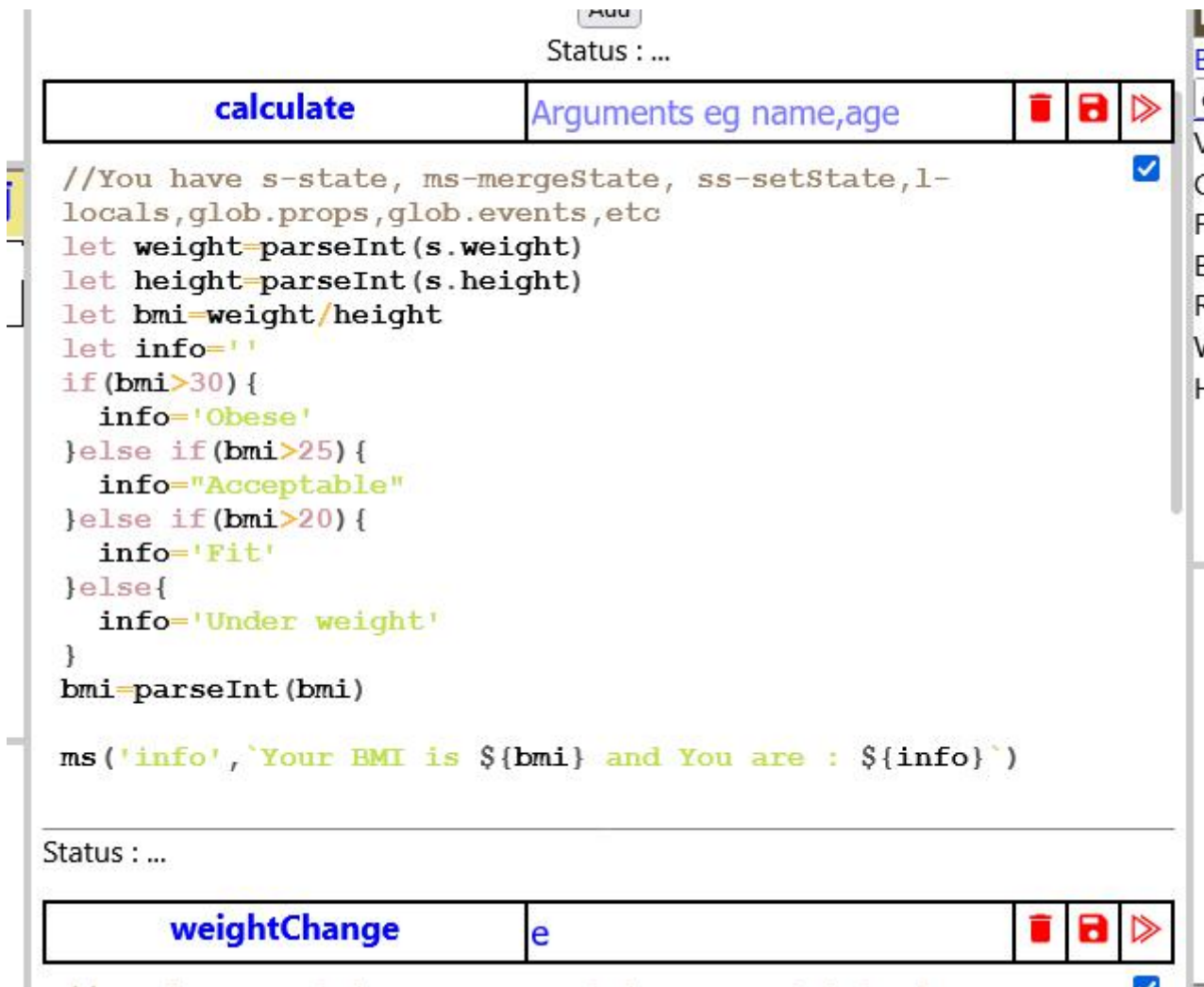
The heightChange and weightChange functions

The screenshot shows a code editor with three function definitions:

- calculate**: Arguments eg name,age. Status: ...
- weightChange**: Argument e. Status: ...
Code: `//You have s-state, ms-mergeState, ss-setState, l-locals, glob.props, glob.events, etc
ms('weight', e.target.value)`
- heightChange**: Argument e. Status: ...
Code: `//You have s-state, ms-mergeState, ss-setState, l-locals, glob.props, glob.events, etc
ms('height', e.target.value)`
- update**: Arguments eg name,age. Status: ...
Code: `//You have s-state, ms-mergeState, ss-setState, l-`

- When the value of a TextInput or MTextField changes then onChange event is fired.
- The two function above are registered for the weight and height comps onChange events respectively.
- The onChange event gives the handler/func an event object eg the one captured as e in the arguments header section of the functions.
- The event object carries the new value in the “e.target.value”
- Since the value of height and weight are tied to state height and state weight, check the update func.
- NOTE how to capture input:
 - Tie the value of the TextInput or MTextField to a state objects field eg height and width above
 - Assign a listener for the onChange and capture the event object eg using ‘e’ as above.
 - `ms(‘field tied to value’,e.target.value)`, ie update the state field tied to this.
- Explanation for heightChange function:
 - ✓ `ms(‘height’,e.target.value)`
 - ✓ `ms` -is used to merge state and so it updates the state field ‘height’ to a new value gotten from user.
 - ✓ Event object capture above as “e” carries other information. `console.log(e)` to see all the data in it.

Calculate function



- This function is assigned to the click of the calculate button.
- So every time the button is clicked it runs.
- Since weight and height are inside the state object we access them using s.height and s.weight
- First we use parseInt() to convert them to whole numbers(integers) since they are strings by default.
- We then divide the weight with the height and store it in bmi
- We do if check to get the correct information to show in the info object.
- We then use parseInt to convert the bmi to a whole number again, remove decimals from division by height.
- We then update the state field 'info' using the template string containing bmi and info.
- And that's it we have our BMI calculator
- It does not use the same table as the standard one.

How to download this module/app

1. Open the modules tab of the work area
2. Search "bmi calculator" or 't.s.o' - author name.

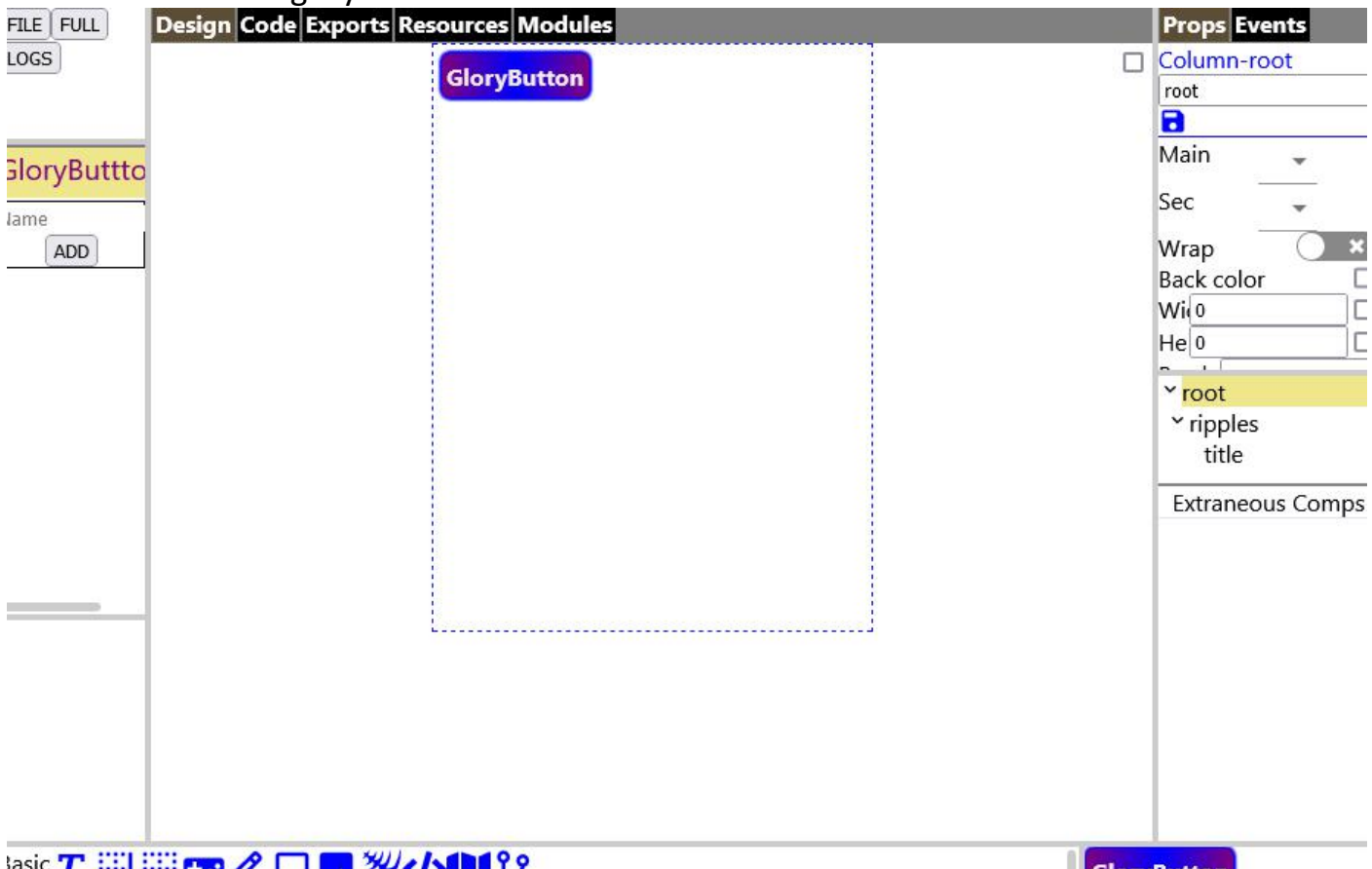
3. Look at the preview of the result to find the right one.
4. Test the bmi calculator on the site to see the result.
5. Download as a comp or mod(module)
6. Download as comp makes it editable unlike download as a module.
7. Note the image used by the module will be added to your resources
8. To view it go to resources tab, under the view resources sub-tab click the “view all” button
- 9.

Chapter 4: Modules

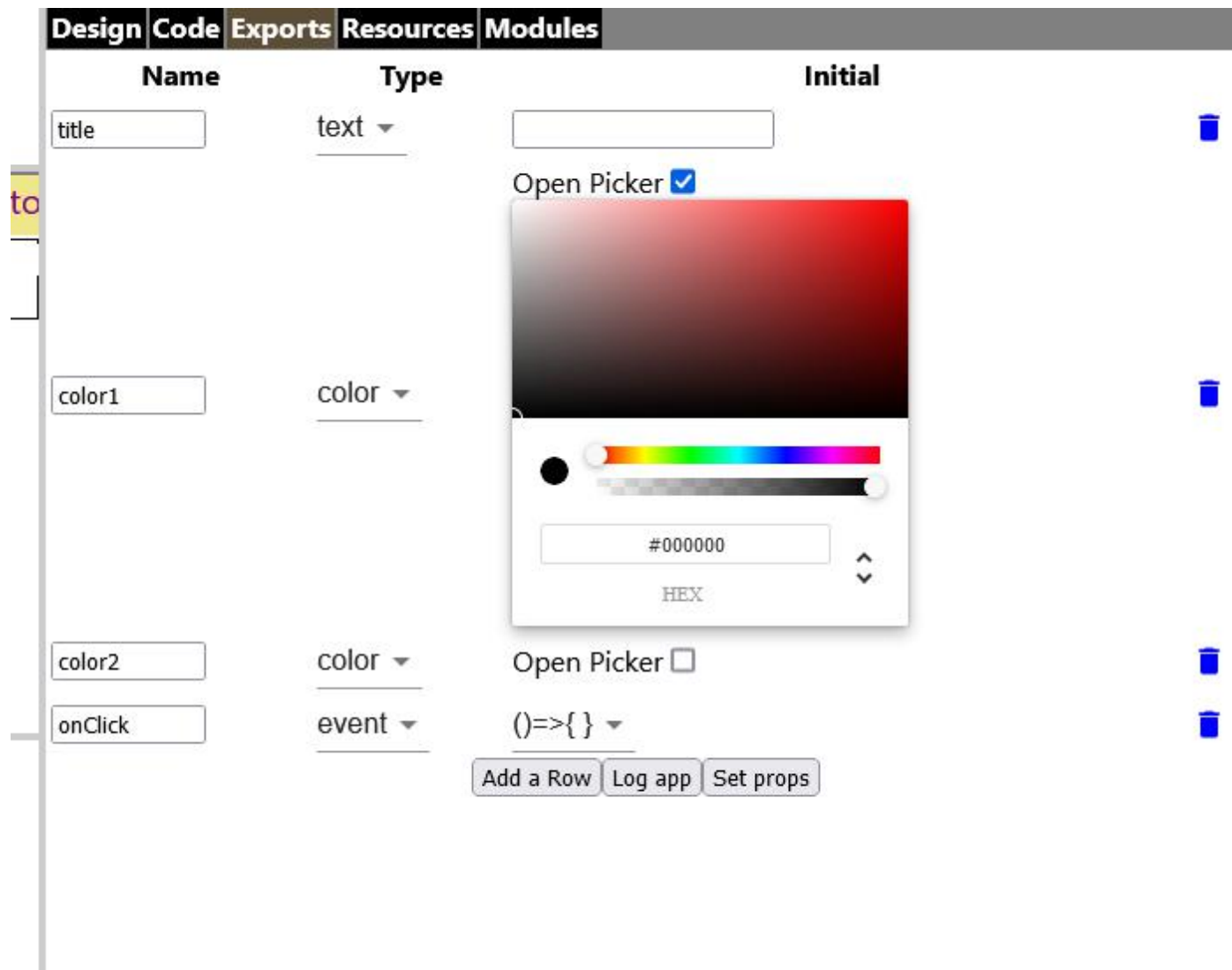
- It is a tree of basic components that act as a component is called a module.
- What we have been calling apps from the other chapters are indeed modules.
- Modules enable us to create our own components.
- Modules like basic components take 0 or more props and can also take 0 or more events.
- Modules are just apps which take props and can be added to the tree of other apps.

Our first module

- Create the following layout.



- To allow user to customize it we add prop types to it as follows:
 1. Open the exports tab and add the following props types show below.



2.

3. we have added 3 props and 1 event.

4. Understanding the table

- ✓ The table has 3 columns
- ✓ Name is the name show in props tab when comp of this modules is selected.
- ✓ Type determines the values that can be entered and how in the props.
- ✓ Initial is the value used when none is specified ie the default value, its also used when designing the module.

Understanding the prop types

Type	Description
text	Accepts words/string entries
number	Accepts integers, can be turned to text type
bool	Displays a switch gives true and false values only.
color	Shows a color picker and allows text entries
json	Allows user to enter js object using json string
Select	Shows a popup with options to select one Set options separated by commas eg 'Monday, Tuesday, Friday'

html	Opens an editor for creating html, allows raw html entry too.
comps	Allows entry of child components, Has issues don't use for now.
events	Used to create events for a module. Has two defaults: 1. ()=>{} an empty arrow function 2. Undefined -no function at all.

After adding or modifying click then setProps button to apply the change.

Using the module.

We can use the module just has any basic component.

Drag and drop it as a child of another module.

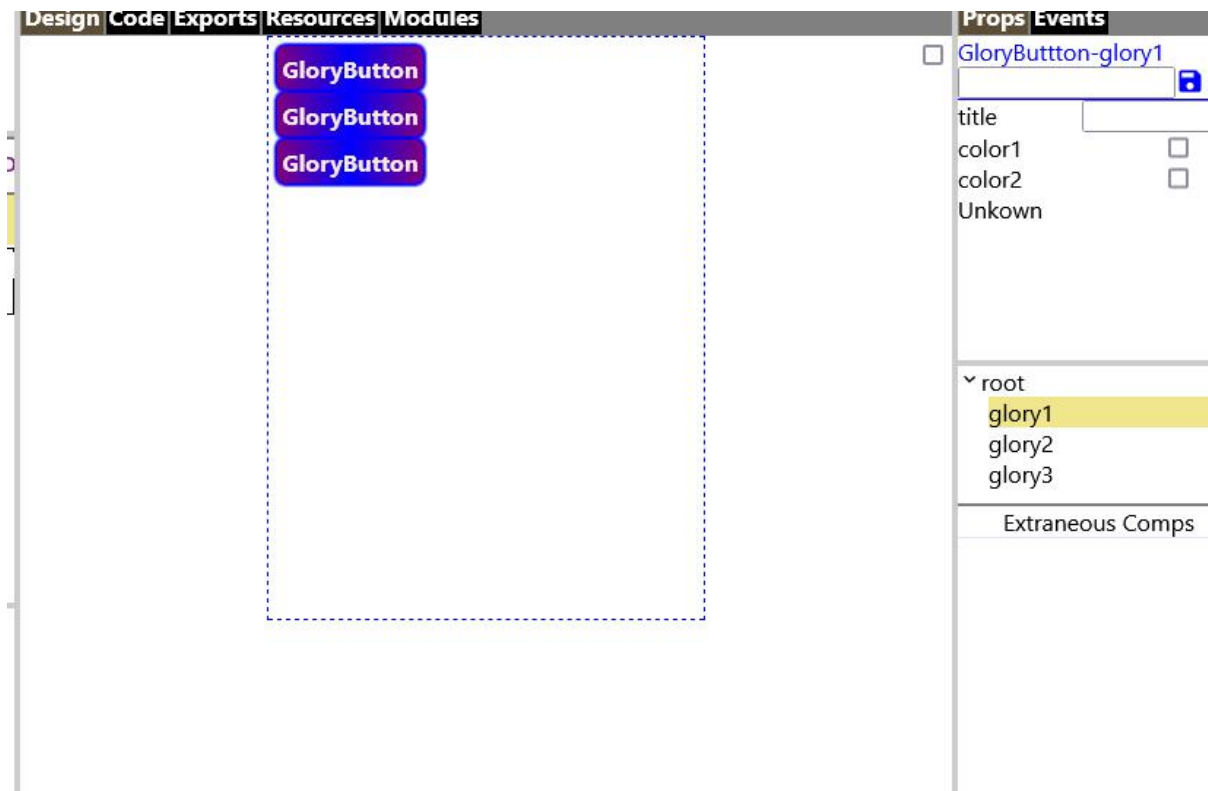
Module can have as many child modules.

We can have a module with child modules which can be used as a child module for another module.

When we modify a module it is updated in every place it is used.

Eg

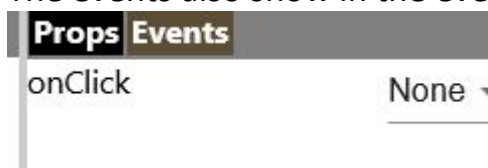
- ✓ Add a new module using the modules area.
- ✓ Rename it to testing by double clicking its name in the modules area and entering a new name and save it.
- ✓ add 3 GloryButton modules to testing module by drag(from modules area) and drop just as we do basic comps to create the layout below.



- ✓
- ✓ You can change the ids of the glory buttons.
- ✓ When we select the glory items, the props tab tells us the type is a glory button, check it out.
- ✓ The props tab also allows us to set the props we specified for the module.



- ✓
- ✓ The events also show in the events tab



- ✓
- ✓ The module can now be used as a basic component.

Using the props and events provided from the inspector in the module

- ❖ For now changing the props does not change our GloryButton, lets fix that.
- ❖ We will use the title prop's value to set the text of glory button.
- ❖ The color1 and color2 will be used for the linear gradient of button background.
- ❖ The onClick event will be called when the button is clicked.

Status : ...

buttonClicked	Arguments eg name,age			
----------------------	-----------------------	--	--	--

```
//You have s-state, ms-mergeState, ss-setState,l-
locals,glob.props,glob.events,etc
glob.events.onClick()
```

Status : ...

update	Arguments eg name,age			
---------------	-----------------------	--	--	--

```
//You have s-state, ms-mergeState, ss-setState,l-
locals,glob.props,glob.events,etc
glob.tiePS('title','value',glob.props.title)
glob.tiePS('root','background',
`linear-gradient(to left,
${glob.props.color1},${glob.props.color2},${glob.props.color1})
`)
```

Status : ...

When we add the above two functions our module now changes when the props change eg



- We don't need the load function,since we don't need to initialize any variable.
 - The props given to the module are stored in glob.props and accessed as glob.props.name ,where names is the name of the prop
 - The events are likewise in the glob.events object to access we use glob.events.name, where name is the name of the event we set in the exports.
 - The update func is used to tie the values in glob.props to the the props of the various comps of the tree.
 - The buttonClick is assigned to the click of the view of the module.
 - Clicking the view will call the buttonClick which calls the function passed in the glob.events as glob.events.onClick()
 - And that's it our glory button is done.
- ❖ To download this module use the name 'GloryButtton' or 't.s.o'

Uploading Modules

To upload a module so other can use it :

1. Open the modules tab
 2. Click the top right “upload” button to switch to the upload button the same button returns you to the download, view.
 3. Select the module.
 4. You will see its info.
 5. Write the documentation/short tutorial on how to use it.
 6. Enter your name.
 7. Note: the name of user and modules is used to select the module.
- ✧ Please download spam guyis’ modules repo only upload modules when they are done.
 - ✧ Don’t you can save a module to a file to use later.
 - ✧ Modules can be imported from project files too.

Downloading modules

8. Enter the search query and press the search icon
9. There are two download options.
10. A live preview of modules if provided, for testing and identification of a module.
11. The doc is also show next to the module preview.
12. Note; both doc, preview and entire row of module is re-sizable, also for testing with diff sizes.
13. With Comp download- it will be added to the modules area and you can edit it.
14. With the mod download- it will be added to downloaded modules area and you can only use it.
15. Modules with resources/photos will add them to your resources after download.
16. You can import modules from a file but support for resources not still available.

Chapter 5: Advanced comps

Here we will cover the last 2 components ie MapList and CondList.



Above are MapList and CondList respectively

MapList

Takes data and create comps out of it and adds them to the app tree.

MapList props

Data - js array from which objects are created.

Type - string of the type of comps to create.

Mapping - and object explaining which data is used for which prop.

Extras - data and functions that are passed to the created components.

- all mapped comps get an index in the extras, so you don't need to pass it explicitly.
- extras only accessible when we map to a module not basic comp.

Example

Data

```
[
  { "name": "Mary ", "color": "primary" },
  { "name": "Sheldon", "color": "secondary" },
  { "name": "Nancy", "color": "default" }
]
```

Close Save

Mapping

```
{
  "name": "Value",
  "color": "Color"
}
```

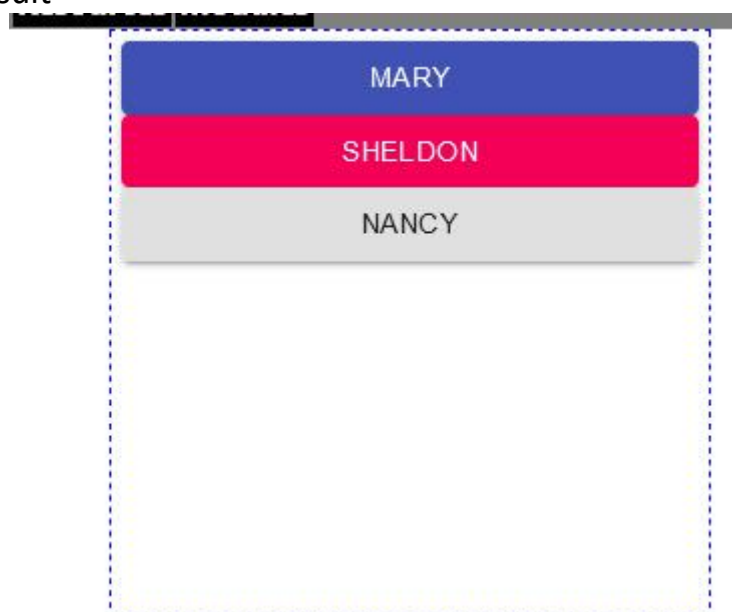
Close Save

From mapping above :

- The name from each data array element will be supplied to the Value prop of MButton.
- The color of each data element will likewise be used for the Color prop of the MButton.

The type is MButton , the spelling and case must march the name. Ie names are case sensitive.

We get the below result

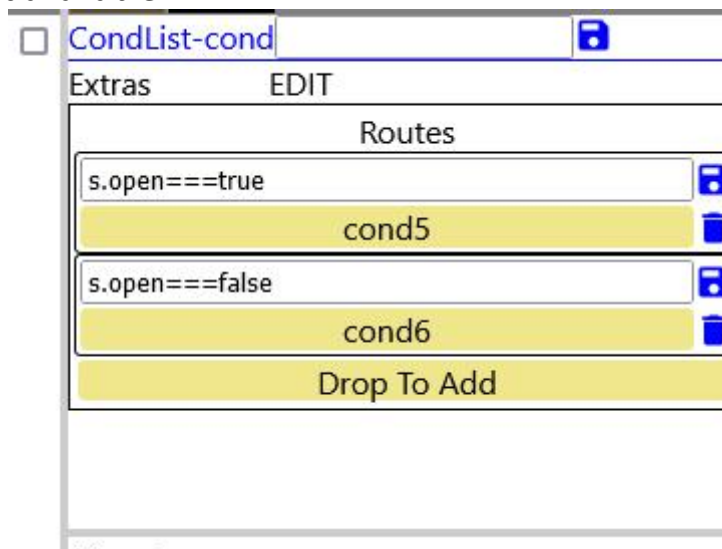


➤ All the props can be set from code.

- When you code them just supply js objects and array no need for json strings.
- You can change the type of element, data and mapping dynamically in code.
- MapList is very useful in real world apps.
- If we were making a Todo app then it would render the todos items.
- The MapList also renders modules.
- When you render modules pass in the extras props, and you can use it from the module from “glob.props.extras.name” where name is the name of the extra
- The extras contain the index of the component in the data, zero based indexing of js applies.

CondList(Condition list)

It is a comp that accepts 1 or more children and shows them only when a condition is true. The condition of the condlist get access to state as (s) and locals as (l), the other variables from functions are not available.



- From the above we see that the comp ‘cond5’ will render only when s.open is equal to true.
- While the cond6 comp will render only when s.open ===false
- We can change s.open to true or false on given condition in code and the relevant component will be displayed.
- This is called conditional rendering, we show comps on give conditions.
- This is great for opening one page at a time, and another when user clicks a button.
- We can now hide and show components conditionally

Finishing notes

- ❖ Understanding MapList and Condlist will allow you create powerful modules.
- ❖ Most of the props have same name to their web equivalents so do web search on props you find difficult.
- ❖ The view component is the most customizable and also accepts a onclick event, make good use of it.
- ❖ The overflow hidden of the view comp can allow you create amazing borders for images.
- ❖ ~~The basic Props when set from code use the html names of the props not the ones shown on the inspector (will be fixed) -fixed in guyi 0.8.2~~

Chapter 6 : Using Runners

- Runners are interpreters of guyi apps on specific platforms
- Runners allow you/users of your app to run them on android device, windows desktop, web server and many other platform to come.
- When an app is open on a runner it come perform platform specific functions like read and write files, show toast and dialogs on android etc.

Currently(guyi 0.8.2) guyi has 3 runner

- 1.Windows-opens guyi project on windows operating system
- 2.Android-opens guyi project on android operating system(the common smart phones)
- 3.Web-opens guyi app as a web app without the editor, can be placed on a server.

Installing runners

- On guyi online editor open the file menu and check to show credits
- In the credits section at the top there are links to download the various runners.
- You can also download the latest version of this documentation from there
- install the runner and follow next steps below to run apps on them.

Runner interface



- Click the load button to show the file picker.
- Use the file picker to select a guyi app.
- On android you may have to change the project extension to “.txt” from “.guyi” to be able to open it.
- If project is valid it will open.
- The last opened project becomes the default app run on next start of the runner.

Using guyi web(the guyi.html file you downloaded)

- To place the file on the server and rename it to index.html
- Place your project file next to it and rename it to app.guyi
- Visiting the url of the file, you will see your app open.
- The guyi web runner is not yet optimized and may open slower for apps with many resources(images), so use smaller images.

Setting the default app for guyi windows runner

- Go to the install directory of guyi windows, default is *c:/programs file x86/Guyi-Windows*
- Then navigate to *resources/app/ folder*
- Place your project folder there and rename it app.guyi
- Now every time the “Guyi Windows” is started it will open this default project
- If you want to go back to using the last project as the default when app is open then just remove the app.guyi in the above mentioned folder.

New methods that work on specific platforms

- ✧ The `window.process.platform` object can tell you if your app is running on windows eg

```
if( window.process.platform==='win32'){  
    -Write code here  
}
```

✧ Works for also `process.platform('android' | 'web' | 'Windows')`

Android functions

1. `Android.showToast('message','length')` - length ('long' or 'short')
2. `Android.writeFile('example.txt','This is the string to write inside')`
3. `Let fileData=Android.readFile('example.txt')`
4. `Let files=Android.GetFiles()`-returns string of file names separated by commas,use `file.split(',')` to get array of files
5. `Android.showDialog('message','title','Yes,No')`-will write the result when user click yes or no button to the file 'result.txt', also a message is send to app of the clicked button , when the dialog is dismissed a message is also sent,dismissed message is sent even if user click on of the button.
6. `Android.flashOn()`-opens torch on phone, has issues to be fixed
7. `Android.flashOff()`=closes torch if open, has issues to be fixed
8. `Android.notifyBasic(id, title, content:, visibility)`-shows a simple notification on the phone:
 - i. Id-no to use to update the notification use a defrent id to create a new notification.
 - ii. Title-off the notification(text)
 - iii. Contents-body text to show,if longer than one line,body is collapse till user expands it(text)
 - iv. Visibility-use public to show notification even on lock screen,only public is used now,guyi 0.9.1
- When user clicks the notification it is closed and app open if not running and message sent to app eg 'notification clicked 120'
9. `Android.stt()`-speech to text, starts google speech to text, and sends message of the got speech to the app or error, messages start with "speech,error or message follows ",eg "speech, turn left".

Return maximum of 5 possible alterations of the speech if its not clear.

10. `Android.tts(word,now)`-say the word given to it, uses the current voice.now-true/false value to show if speech is spoken immediately or waits in line for others to end.
11. `Android.getVoices()`-returns a comma separated list of voices in the system.Use `split(',')` to get Array.
12. `Android.setVoice(voiceName,speed)`-set voice for `tts()`, from on of the gotten voices plus the speech rate of the voice 1=normal 0.5 =half rate 1.2 faster etc,any value applies.(above 1 wont work till next version of guyi)
13. `Android.checkPermission(name)`-names like CAMERA,RECORD_AUDIO ,READ_SMS,SEND_SMS,ACCESS_FINE_LOCATION, etc ,returns true if accepted else false.
14. `Android.sendSms(no,body)` no-tel number ,body =string message.
15. `Android.call(no,sim)`,no-tel no, sim=int 0|1 for sim slot.
16. `Android.getSms(type)`-type can be inbox|sent|draft|contacts, if contacts return all contacts on phone.
17. `Android.playSound(type)`-alarm|ringtone|none|notification.
- 18.

Windows(Win32 functions)

- 1) For below Methods to work use as `Win32.method(arg1,arg2)` eg `Win32.writeFile('example.txt','Your data here')`
- 2) `writeFile(path,stringData)`
- 3) `readFile(path,asText)`-return base64 if `asText=false` else string
- 4) `getFiles(path)`-return names of files and directories as list
- 5) `deleteFile(path)`-deletes a file pointed to by path
- 6) `deleteDir(path,recursive)`-delete the folder,if recursive it deletes also items in it,dangerous-can format disks!!
- 7) `exists(path)` -if path refers to item on disk folder/file,true for shortcuts.

- 8) `isFile(path)`-if item is a file
- 9) `isDirectory(path)`-if item is directory , false for shortcuts.
- 10) `stats(path)`-info about item pointed to by path eg
size,modification date, birthdate,etc
- 11) `mkdir(path,mode,recursive)`=creates a directory at the path,
mode is permissions and recursive -if missing parent folders to be
created too.
- 12) `copyFile(oldpath,newpath)`-copies file to new location, to move
delete old after copy.
- 13) `maximize()`-make window take full desktop size `minimize()`-send
window to taskbar
- 14) `setSize(w, h)` -resize window
- 15) `setBrightness(val)` -set screen brightness 0-1
- 16) `getBrightness()` -obtain screen brightness 0-1
- 17) `setVolume(val)` -change volume 0-100
- 18) `getVolume()` -get current volume 0-100
- 19) `setMuted(val)` -val =true to mute,val=false to unmute
- 20) `getMuted()` -get if muted
- 21) `setMinSize(w, h)` -set minimum size for user resize
- 22) `setMovable(val)` -set max size
- 23) `setPosition(x, y)` -set position of window on screen
- 24) `setFullScreen(val)`-val=true for full screen,val=false to cancel
- 25) `isFullScreen()` -return is in full screen
- 26) `setProgressBar(val)` -set task bar icon progress val -1 removes 0-1
is range
- 27) `setIcon(val)`-val is base64 , use Guyi's `glob.getRes('filename')` to
get base64 of a file in the project
- 28) `setOverlayIcon(val, desc)`-sets smaller icon ontop of icon val is
base64
- 29) `screenshot()`-return base64 image of the screen use
 `let img= "data:image/jpg;base64,"+Win32.screenshot()`
- 30) `screenshotAll()`-return array of screenshots for each screen
- 31) `center()` -puts the window on screen center
- 32) `flashFrame()` -makes icon blink on taskbar,will work on next
version of guyi

- 33) `getInfo()`-return a lot of info about user and computer
- 34) `setAlwaysOnTop(val, level)` -makes app be on top of other and cancels it too
- 35) `openConsole(val)`-`val=true` open ,`val=false` closes
- 36) `evalNode(body)`-the code is evaluated by node.You have access to the variables(constants)
 - a) `robot`- `robotjs` import. Use to control mouse and keyboard
 - b) `say`-use to convert speech to text.
 - c) `electron`-use to create windows/dialogs etc
 - d) `express`-use to create servers.
 - e) `ss`=use to create ws servers and clients
 - f) `fileUpload`-`express-fileupload` import useful for multipart form data.
 - g) `Loudness`-control volume
 - h) `brightness`-control screen brightness
 - i) `screenshot`-capture screen shots
 - j) `config` -instance of `Conf` store config info.
 - k) `WinRef`- reference to app window. Use carefully!
 - l) `app`-the electron app itself.
- ❖ ProTip- one file node-js modules can be run using `evalNode`.

- ❖ Check the platform before using the ,above methods they will crash the app if app is running on wrong platform.

Support for Libraries

Since guyi 0.9.1when you add **javascript** files or **css** as resources you can mark them as scripts to be executed after window load.

With allows use of the enormous amount of libraries for web JavaScript, like;

1. `Tensorflow.js` if you want artificial intelligence in your app.

2. Three.js if you want 3d graphics on your apps-COMES PREINSTALLED.
3. Jimp for image processing.
4. Google maps scripts- to embed maps
5. Etc:anything that works with script tags should work .

1. Css like bootstrap.
2. Css for varius js libraries,
3. Etc any css that works with link <rel='stylesheet' > should work

Change Log

Guyi 0.8.2

New layout suitable for smart phones

To use FILE-MENU/LAY2 and change back by FILE-MENU/LAY1

