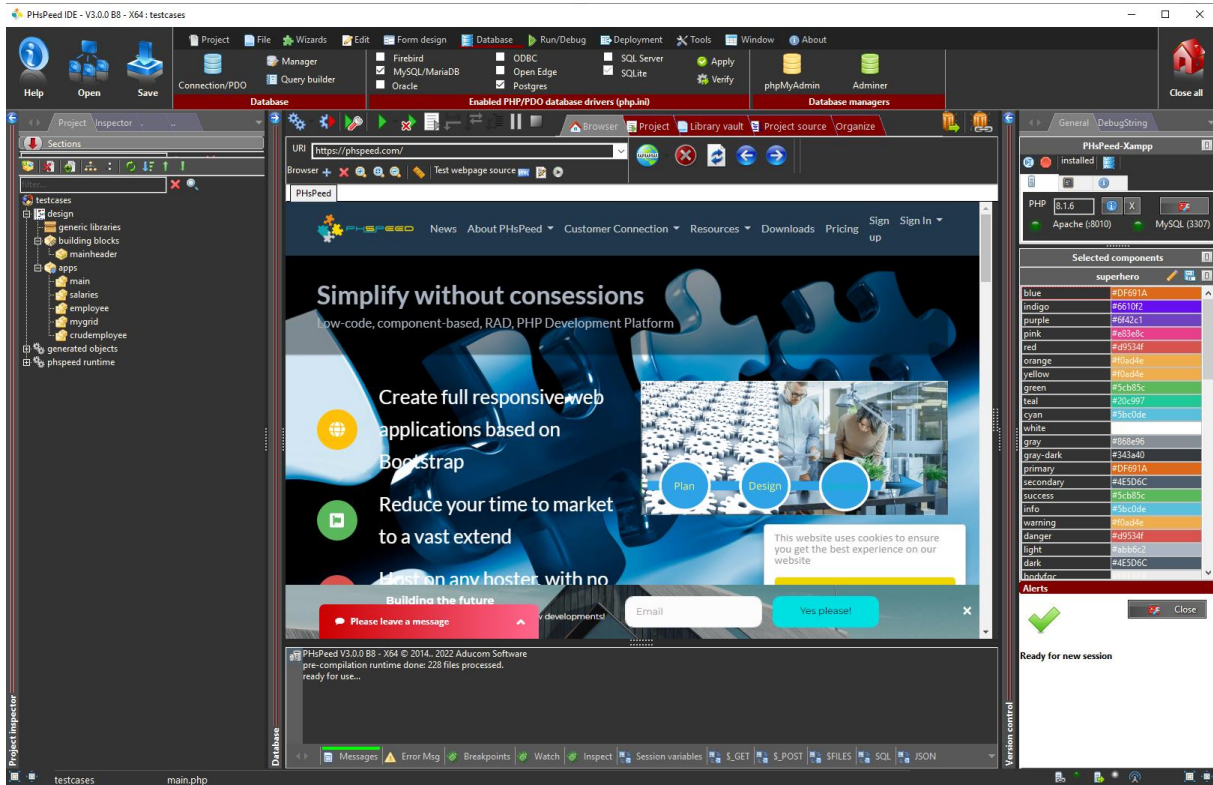


# PHsPeed Introduction

Programming (Yes, it's low-code, not no-code..)



## PHsPEED: Getting Started



Welcome to the world of PHsPEED! This manual is designed to introduce you to our low-code PHP development tool and help you work faster and smarter, aligning with the modern approach to web application development.

In the previous documents, we discussed the fundamental features of the Integrated Development Environment (IDE), created the first basic PHsPEED project demonstrating the 'hello world' example, and showed you how to leverage the application wizards to expedite your development process.

While it is possible to create applications without writing a single line of code using PHsPEED, there may be cases where you want to achieve something beyond the provided tooling and require writing custom code. This document will explore the concepts of events and moments where you can intervene with the standard behavior of components and your application.

In PHsPEED, events are actions or occurrences that take place within your application. User interactions, system events, or other components can trigger these events. By utilizing events, you can control the flow of your application and customize its behavior according to your requirements.

### Interfering with Component Behavior:

PHsPEED allows you to interfere with the standard behavior of components. This means that you can modify the behavior of existing components, such as buttons, forms, or data grids, to suit your specific needs. By leveraging this feature, you can create dynamic and interactive applications that respond to user actions in a desired manner.

### Interfering with Application Behavior:

In addition to modifying component behavior, PHsPEED allows you to interfere with the overall behavior of your application. You can define custom actions and behaviors for your application, such

as executing specific code when certain events occur or manipulating data based on user inputs. This flexibility empowers you to create tailored applications that go beyond the standard functionality provided by the low-code environment.

By understanding the concepts of events and interfering with component and application behavior, you can unlock the full potential of PHsPeed and build powerful web applications that meet your unique requirements.

### **Concept: PHsPeed events**

Events are occurrences within your application where you can intervene in the standard behavior. PHsPeed has two types of events: PHP events and JavaScript events.

#### **PHP Events:**

PHP events in PHsPeed enable you to handle server-side interactions and logic. With PHP events, you can execute code and manipulate data on the server, providing dynamic and personalized functionality to your application.

#### **JavaScript Events:**

JavaScript events, on the other hand, allow you to handle client-side interactions and behavior. While it is unlikely to require writing JavaScript for your PHsPeed applications, you must assign the events. For example, if you want to create an AJAX event, you will need to incorporate JavaScript to enable the execution of PHP code within the Event.

#### **Ajax Events:**

Ajax events involve asynchronous communication between the client and server, allowing you to update parts of your application without refreshing the entire page. PHsPeed provides a consistent naming convention for Ajax events in PHP and JavaScript to simplify the process. This ensures seamless integration between the two languages, allowing you to handle Ajax events effortlessly.

In the subsequent sections of this manual, we will explore PHP and JavaScript events in detail, providing examples and instructions to help you leverage these event types effectively.

### **Concept: properties and events.**

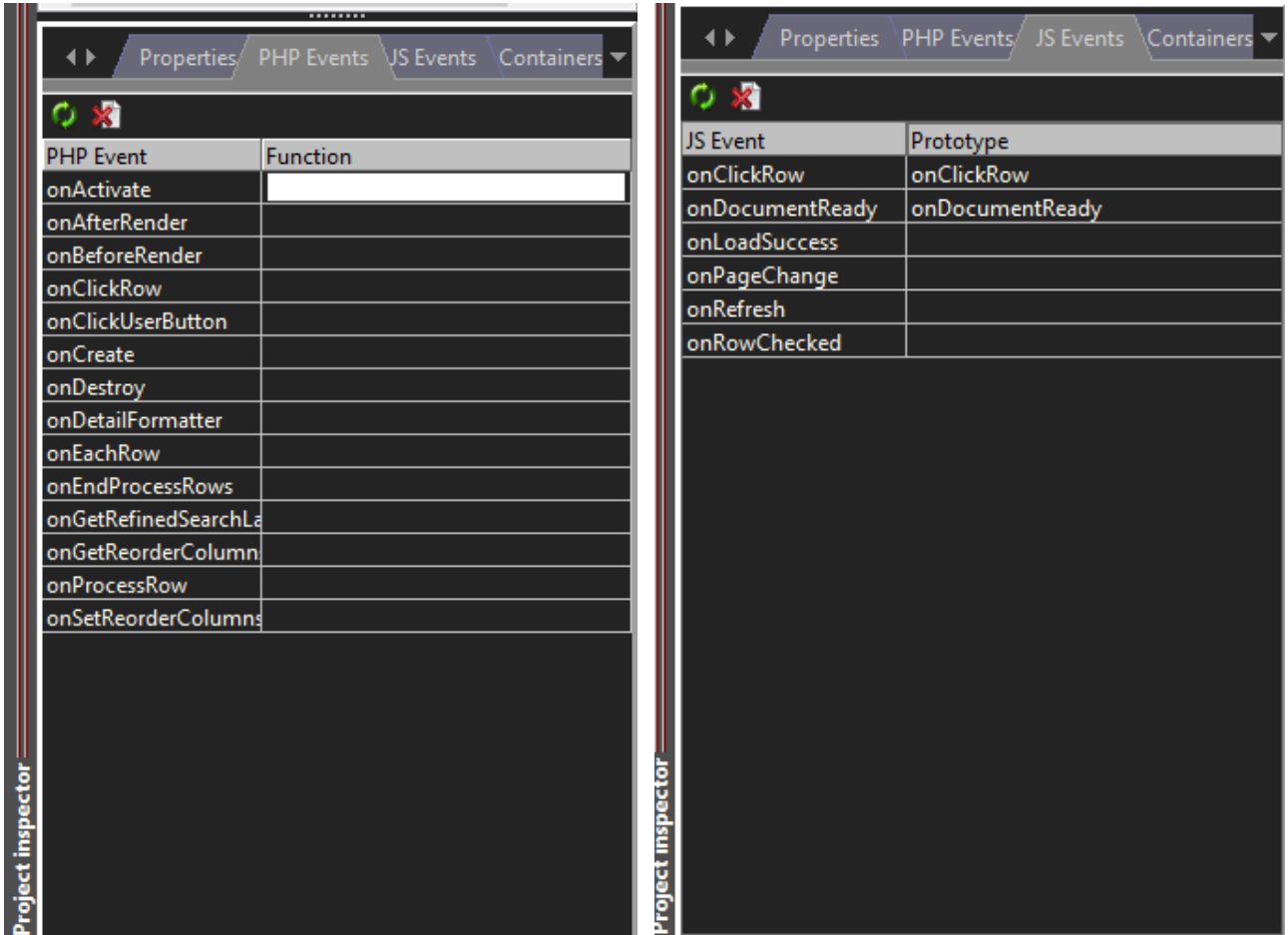
In the previous documents, we discussed how components in PHsPeed have properties that define their behavior and appearance. However, components also have events that allow you to respond to specific occurrences within the application flow. The available events for each component can be found in the component inspector.

#### **Component Inspector: |**

The component inspector in PHsPeed provides a comprehensive view of the components used in your application. It displays the components in a tree order, allowing you to navigate and select individual components. When a component is selected, the inspector displays its properties, including the available events that you can utilize. You can also choose a component from the design panel to achieve the same.

**PHP Events and JavaScript Events:**

Within the component inspector, you will notice separate tabs for PHP events and JavaScript events. These tabs organize the events according to the programming language you wish to use for handling them.



**Application flow**

Understanding the sequence of events and how the application flow works in PHsPeed is crucial for effectively utilizing events and developing robust web applications. This section will guide you through the main application process and explain the event sequence in separate stages.

**Main Application Process (first stage):**

In PHsPeed, the main application is the controller, handling various actions and orchestrating the flow. It generates HTML forms, manages to submit inputs, handles AJAX requests, and more. Let's explore the main application process step by step:

**Application Initialization:**

When you start the PHsPeed application, it begins by initializing the process. The application verifies role-based access and internal security measures like the CSRF token during this stage.

**HTML Template Generation:**

After the initialization, the main application generates an HTML template based on your form design. This template will serve as the foundation for your application's user interface.

### **Component Creation and onCreate / onActivate Event:**

Once the HTML template is generated, PHsPeed creates all the components defined within the template. As each component is created, it triggers the onCreate Event. This Event signifies the initialization of the component and allows you to execute custom logic or initialization routines.

After the creating all the components, PHsPeed activates each component, triggering the onActivate Event for each one. This Event indicates that the component is now active and ready to interact with the user. It is an opportunity to set initial values, perform data retrieval, or any other necessary setup tasks. As all the components are already created, you can access the properties of other components.

### **HTML Template Output:**

With all the components activated, PHsPeed generates the final HTML template and returns that to the browser.

### **Component Destruction and onDestroy Event:**

Once the HTML template is output, PHsPeed destroys all the components, firing the onDestroy Event for each component. This Event allows you to perform cleanup tasks, release resources, or execute any necessary finalization actions.

### **JavaScript Events in the First Pass:**

It's important to note that no interceptable JavaScript events are involved in this initial pass. The event sequence described above focuses on the server-side processing and generation of the HTML template. JavaScript events come into play during subsequent interactions and user actions, which we will cover below. But after the form template has been output, the browser will fire an ajax event to retrieve the data. This will inhibit the second pass of the application controller.

### **Follow up: Application Process (second stage):**

With the screen loaded, the web browser initiates an AJAX (Asynchronous JavaScript and XML) event, a method of exchanging data with a server without interfering with the current web page. This Event acts as a bridge, connecting the browser to a vast repository of data. Its purpose is to retrieve the necessary information to fuel the upcoming stages of the application's output.

### **Component Creation and onCreate / onActivate Event:**

The same process as the initial stage will occur: components are created and the OnCreate and OnActivate events occur.

### **Component rendering onBeforeRender / on AfterRender**

Instead of creating an HTML template, PHsPeed will now render all the components and send them back to the web browser to replace the placeholders in the template. These events can be used to intervene in the output. Some components have specialized events for

certain tasks. It is possible that more than one of these events will be triggered during the rendering phase.

**Component Destruction and onDestroy Event:**

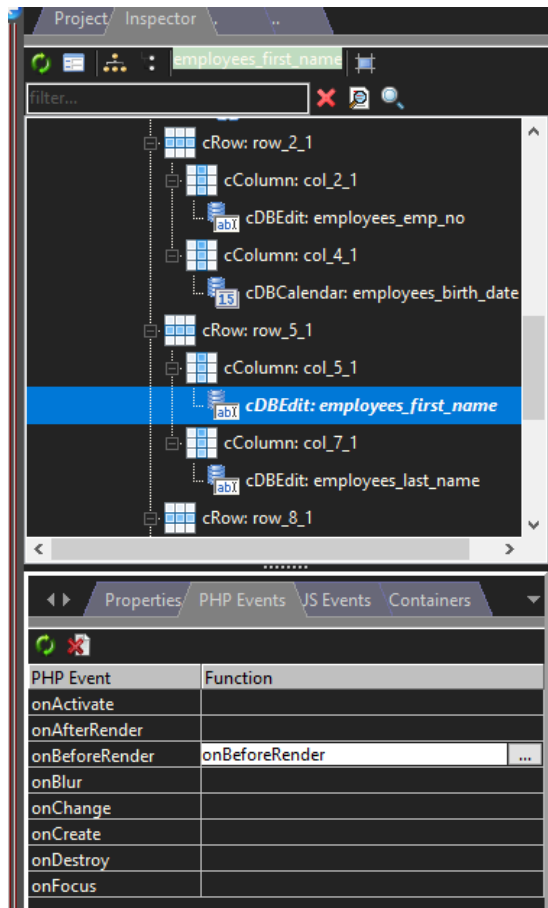
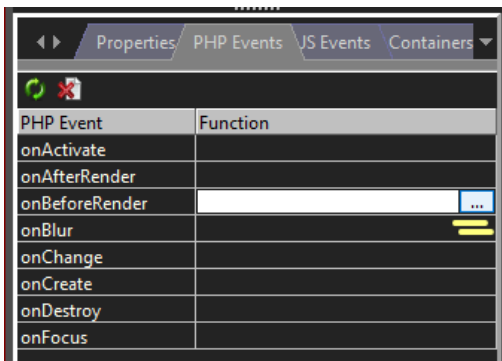
Once the data has been sent, PHSpeed destroys all the components, firing the onDestroy Event for each component. This Event allows you to perform cleanup tasks, release resources, or execute any necessary finalization actions.

**Follow up: Application Process (other passes)**

Depending on the used components or your defined (ajax) events, the rendered form may inhibit new events to the controller application. In this case, specialized events may be triggered. But in any case, the same create and destroy events will be executed.

**Writing event code**

To add code to an event, click in the Event, next to the small button behind it. This will open the PHP editor and generates an empty function:



```

gridpus_employees
1 function employees_first_name_onBeforeRender($app)
  {
  .
  .
  4 }
  
```

## The function of \$app and \$\$

Each event function within PHsPeed carries a set of parameters that provide developers with the means to respond and react accordingly. One constant presence among these parameters is \$app, a class structure encompassing all components, properties, and additional features, including standard class functions. However, accessing event properties directly through the \$app variable can be inconvenient, requiring precise knowledge of the field's name.

To illustrate this point, let's consider the example of a component named "label\_1" in our previous "hello world" demo. Upon inspecting the generated code, you will notice that PHsPeed has altered the variable name to "hw\_label\_1". This renaming is aimed at maintaining clarity and avoiding potential conflicts or naming collisions.

```
class hw extends spapplication {  
    public $action;  
    protected $hw_root_1;  
    protected $hw_form_1;  
    protected $hw_panel_1;  
    protected $hw_label_1;  
    protected $hw_csrfToken;  
    ....  
}
```

Initially, PHsPeed incorporates the module's name as a prefix to component names. This approach serves a logical purpose, especially when employing headers and footers, where components may share the same name on the form. Adding the application name as a prefix ensures uniqueness and helps avoid conflicts arising from non-unique component names.

However, when accessing a specific field, you need to reference the variable from the \$app variable. For instance, to access the value property of the hw\_label\_1 component, you would use

```
$app->hw_label_1->value.
```

Now, let's consider a scenario where you have utilized this reference and subsequently modify the name of the module. In such a case, it is important to acknowledge that the reference to the field will still contain the original module name prefix. Therefore, changing the module name without updating the corresponding references may lead to inconsistencies and errors in your code.

To effectively address any challenges that may arise from changing the module name, PHsPeed offers a convenient approach using the \$\$ notation. By employing this technique, you can access the properties of components placed on the form without relying on the explicit \$app variable reference.

Let me explain straightforwardly: when accessing component properties on the form, you can utilize the \$\$ notation. For example, instead of using

```
$app->hw_label_1->value = 'hello world';
```

you can simplify the code to

```
$$label_1->value = 'hello world';
```

This change eliminates the need to reference the \$app variable directly and enables smoother handling of module name modifications.

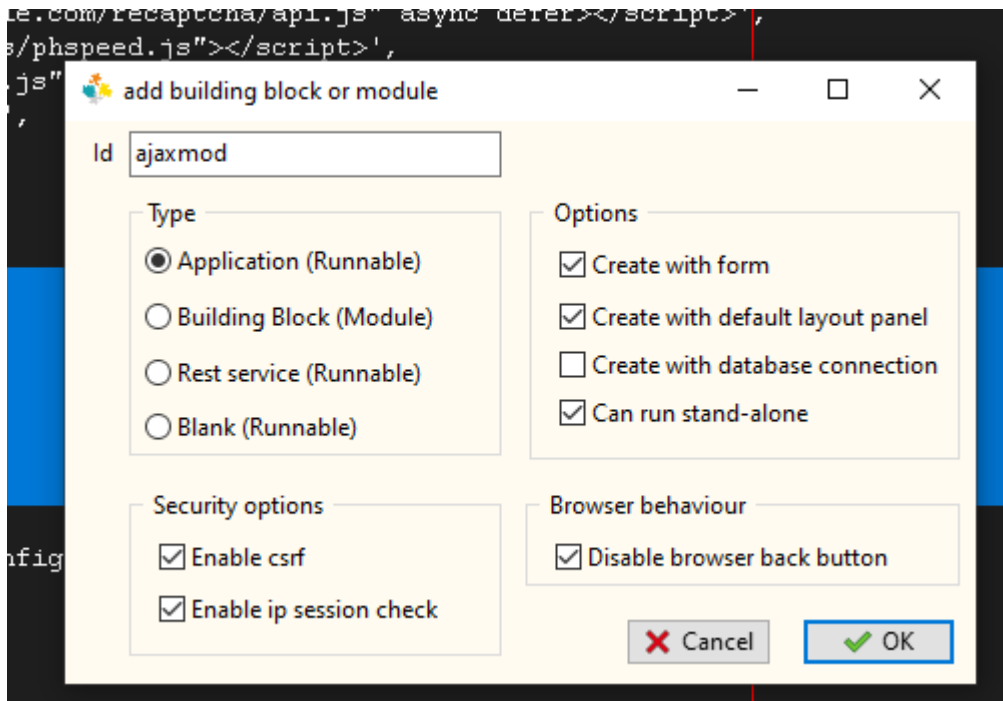
PHsPeed takes care of the rest, seamlessly managing the connection between the modified module name and the associated component properties. Therefore, even if you decide to change the name of the module, the code utilizing the \$\$ notation will continue to function as expected.

This approach enhances code readability and maintainability, reducing reliance on explicit references and allowing for more streamlined development. However, it is important to note that the \$\$ notation should only be used when accessing properties of components placed on the form.

**Coding in practice, create an ajax event.**

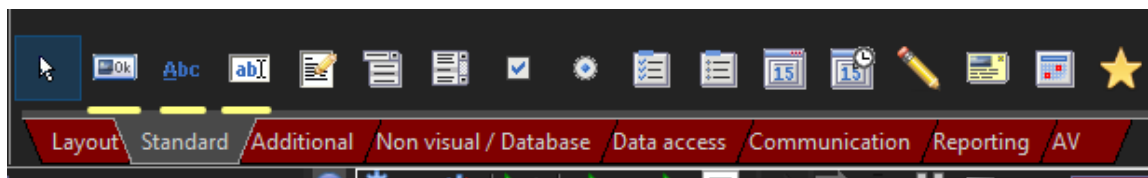
In this section, we will delve into writing a concise piece of code that demonstrates setting the content of a label field based on user input from an edit field. While this sample may appear simple, it encompasses key concepts such as using the editor, handling user input, and accessing component properties within the code.

- Open the hello world project and add a new application. Disable the database connection.

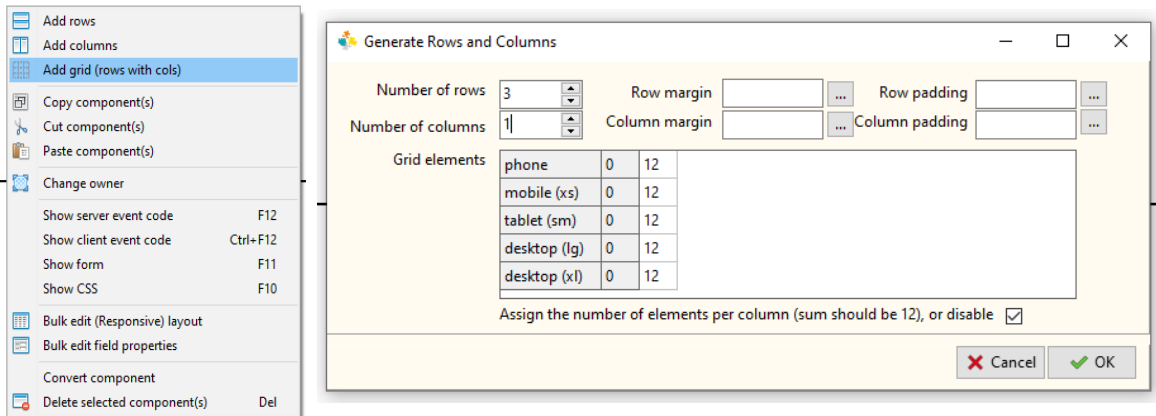


- In the panel we need three rows with one column.

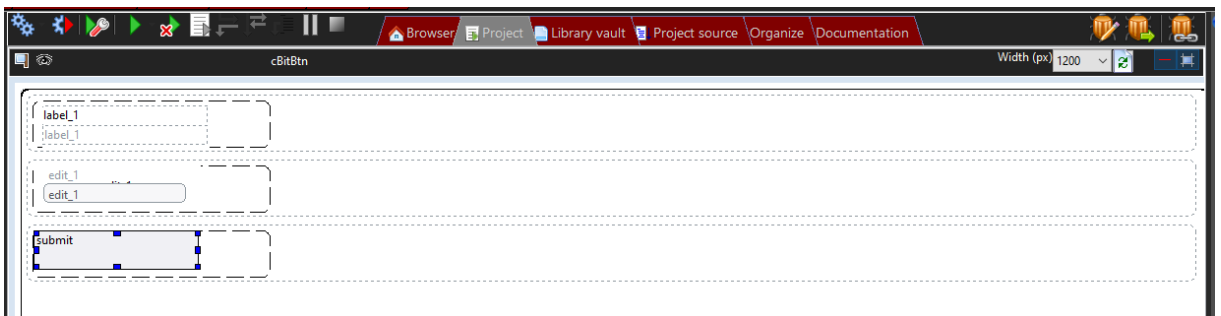
You can add each component one-by-one, from the component panel, but you can also use the right mouseclick in the form panel to add the rows and columns in one go. In the three rows add an edit component, a label component and a button component.



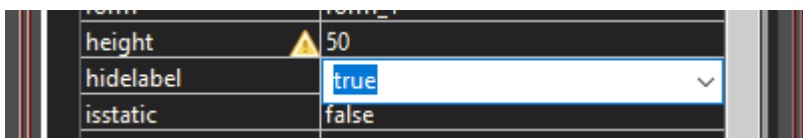




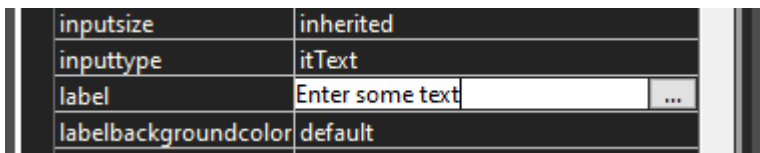
- The design should be something like:



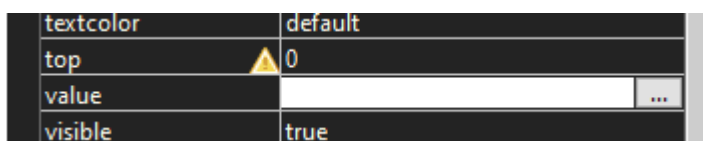
- As the label will show the data that is entered in the edit field, set the hide label property to true. Set the value to empty (by removing the content).



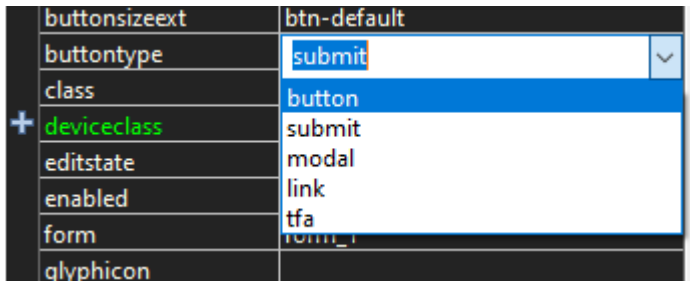
- Set the label of the edit field to 'enter some text'.



- Clear the value property of the edit text to have a blank input

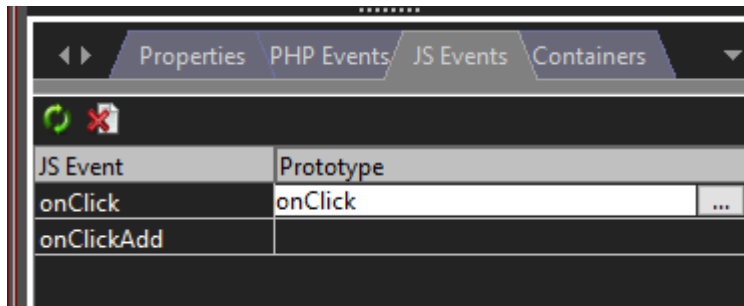


- Set the button type of the button to type 'button'



This is required as the standard type is 'submit', which will send the input to your application and then refreshes your form. Setting it to 'button', will allow you to apply a javascript to enable the ajax event.

- Open the JS Events tab of the component inspector, and click in the onClick Event, next on the small icon to open the JavaScript editor and to generate it's code.

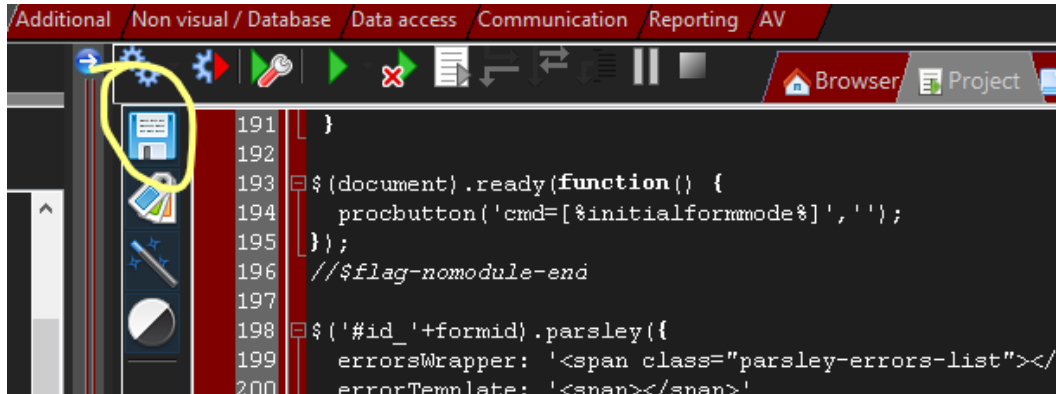


```

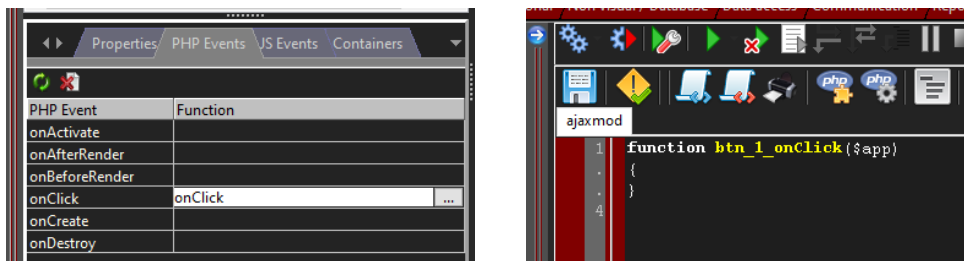
205     $('#bs-callout-info').toggleClass('hidden', !ok);
206     $('#bs-callout-warning').toggleClass('hidden', ok);
207     });
208     .on('form:submit', function() {
209         return true; // Do or Don't submit form
210     });
211 </script>
212
213 <script id="btn_1_onClick">
214
215     $("#ph[%name%]").click(function() {
216         var str='';
217         !IF [%bsubmit%]
218         bsubmit=true;
219         !ENDIF
220         !IF [%form%]
221         str = $( "#[%form%]" ).serialize();
222         !ENDIF
223         procbutton(str+'&cmd=onclick&member=[%_name%]&sender=[%name%]', '');
224     });
225 </script>
226
    
```

**Concept:** All JavaScript code is maintained in one file. The code generated will extract the required code from this file when applicable.

- Click on Save to save the generated JavaScript.



- Open the component inspector's PHP Events tab and click on the onClick Event, next on the small icon to open the PHP editor and generate the function prototype.



**Concept:** All PHP code is maintained in one file. The code generated will extract the required code from this file when applicable.

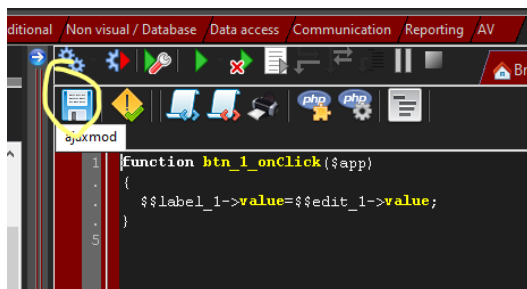
- Write the code

**Concept:** you do not have to perform special operation to read from the input. PHSpeed will do that for you. By default PHSpeed will perform functions to sanitize your input to avoid hazardous input like XSS attacks.

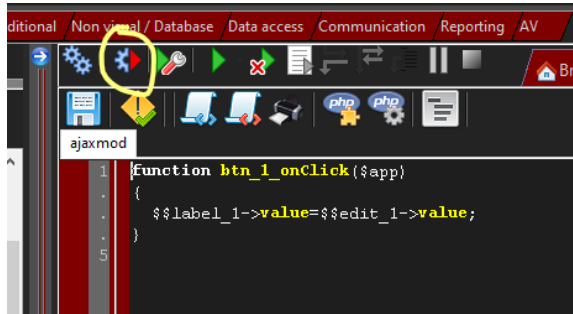
So the code will be

***\$\$label\_1->value=\$\$edit\_1->value;***

- Click save.

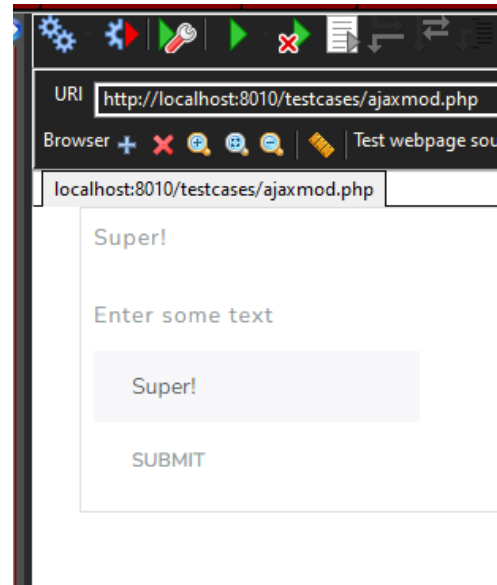
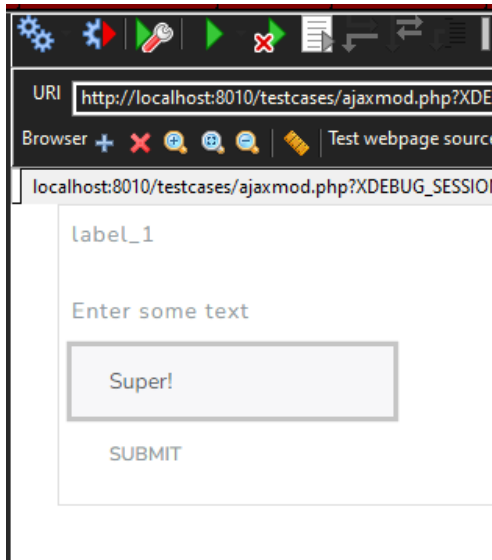


- Now all is set to generate and run your program. Click on the generate and run button.



```

function btn_1_onClick($app)
{
    $$label_1->value=$$edit_1->value;
}
    
```



You have written your first program. Only one line of code. That is what low-code is all about! And as you have seen, writing Ajax events are a breeze!