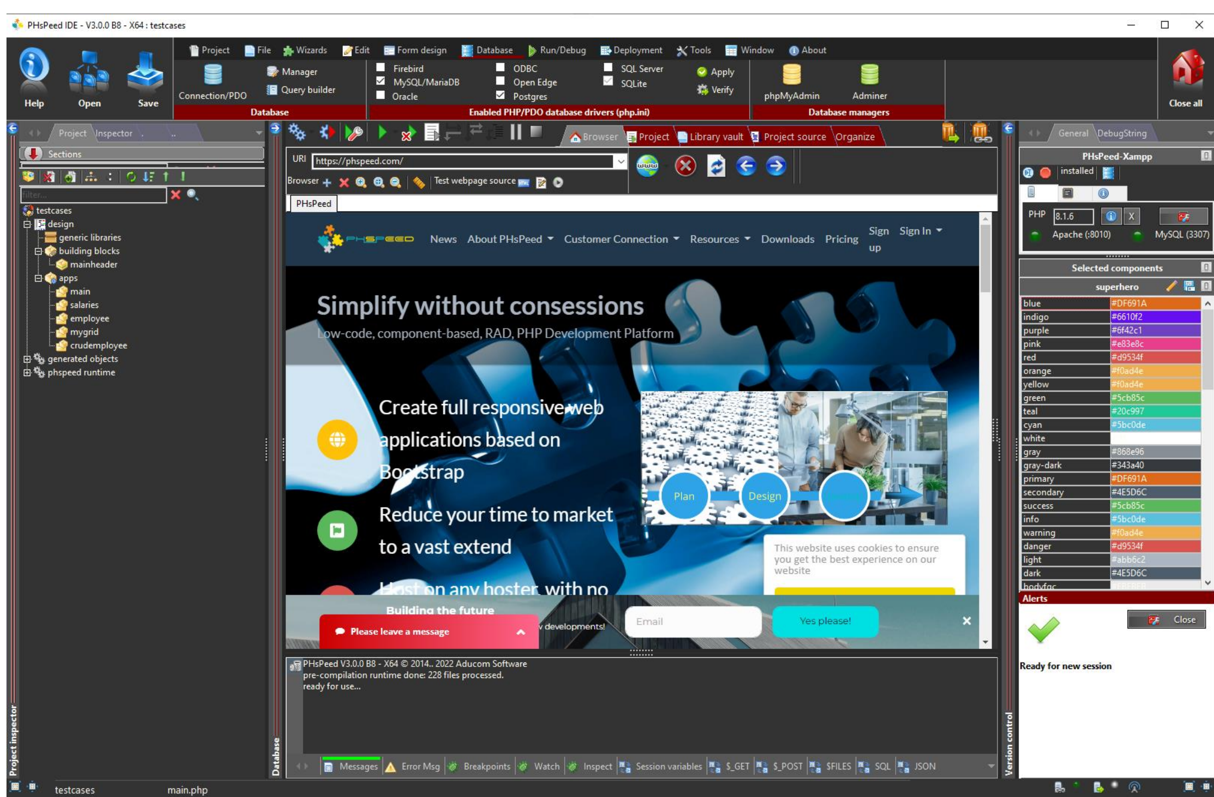
PHsPeed Introduction

More on events and coding

© 2023 Aducom Software Netherlands



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. You also know by now how to create a form manually and wrote your first PHP code in an Ajax event. We also explained the bootstrap grid system to lay out your forms. Now we will dive into customizing your application by adding code. We will explain the application flow, events, and how PHsPeed handles code generation and classes.

Basic application structure.

PHsPeed is a powerful web development framework that allows you to create applications manually or through the assistance of wizards. When you create a new application, PHsPeed automatically generates a basic controller application, which serves as the backbone of your module. This manual will guide you through the process of creating and using controller applications effectively.

- **Generating a Basic Controller Application**

  Upon creating a new application, either manually or by utilizing one of the available wizards, PHsPeed generates a basic controller application. The name of this application corresponds to the module you've created.

- **Component Generation**

© 2023 Aducom Software Netherlands

For each component you add to the form, PHsPeed generates a PHP class. This separation of code ensures clean organization and modularity.

- **Separation of Code Files**

  PHsPeed efficiently divides the application's code into separate files, including PHP, JavaScript, CSS, and HTML. This approach enhances code readability and simplifies maintenance.

- **Starting the Controller PHP Application**

  When you initiate the controller PHP application, it generates a basic HTML template that is sent to the client. The form is initially displayed on the client side.

- **onDocumentReady Event**

  Upon loading the form in the browser, an onDocumentReady event is triggered. This event calls the controller application to execute further actions. This event is executed as an Ajax event.

- **Rendering Components**

  The controller application proceeds to render all components into HTML, CSS, and JavaScript. These rendered elements are then sent back to the client. Depending on the complexity of the components, this process may repeat several times until the form is fully rendered and visualized.

PHsPeed Manual: Controller Application Events and Component Lifecycle

In PHsPeed, the controller application plays a vital role and is executed multiple times during the lifecycle of your application. This manual will guide you through the event-driven nature of the controller application and the lifecycle of components used in your application.

Component Creation and Event Triggering

During each execution pass, the PHP code creates objects for all the components used in the application. As the components are created, events are triggered, allowing you to perform specific tasks in response to these events.

onCreate Event

When a component is being created, the onCreate event is triggered. You can use this event to initialize properties of the respective component. However, keep in mind that you cannot set properties of other components at this stage, as their creation might not have occurred yet.

onActivate Event

After all the components have been created, the onActivate event is triggered for each created component. At this point, all components are available, and you can access their properties. However, exercise caution when changing properties of other components during this event, as you cannot be certain that they have already processed the onActivate event.

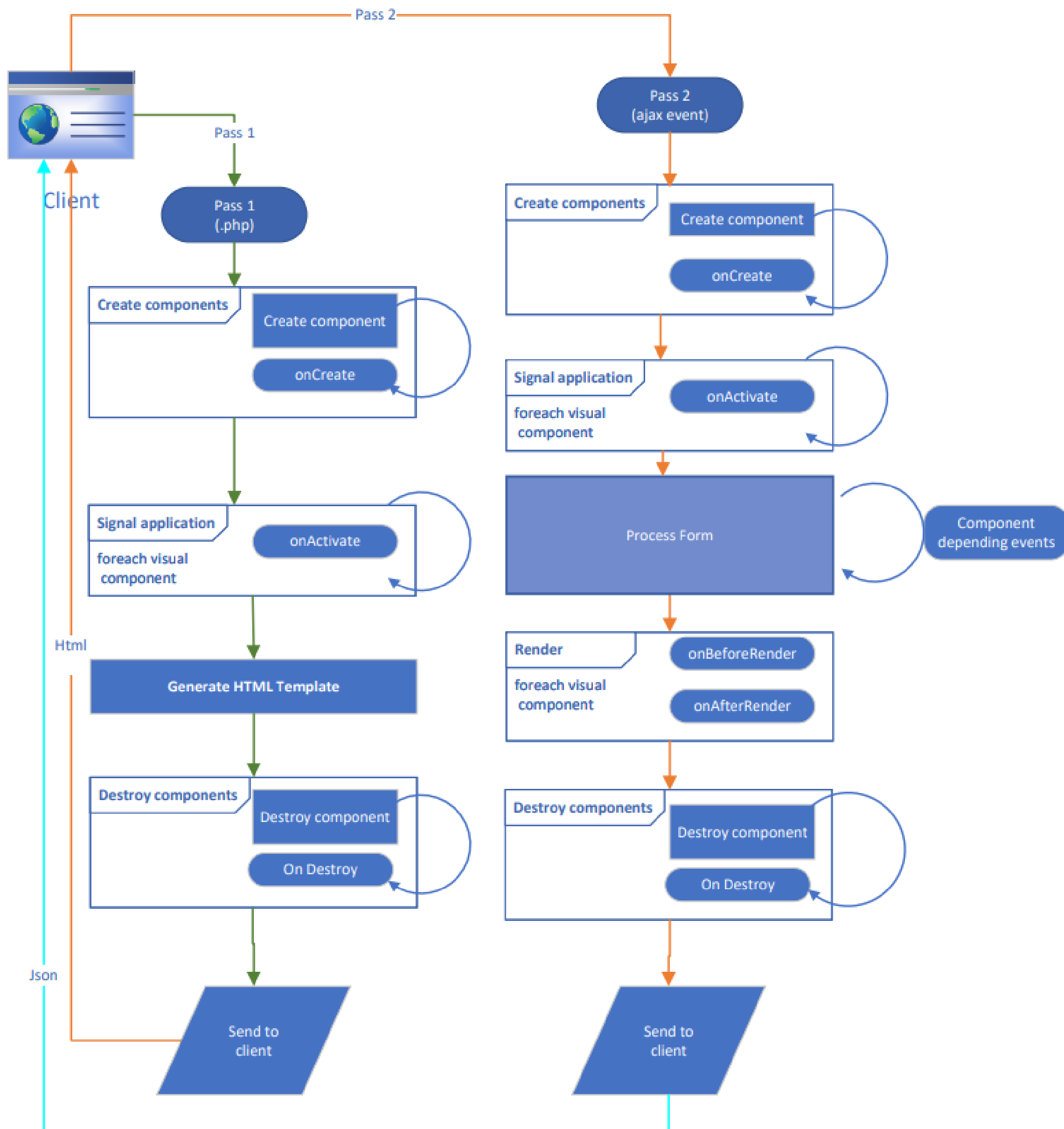
onBeforeRender Event

During the rendering phase, the onBeforeRender event is triggered for each component. This event allows you to intervene with the output and perform additional actions based on component values.

© 2023 Aducom Software Netherlands

As all components are created and have their values set, the onBeforeRender event proves to be extremely useful. Nevertheless, keep in mind that the sequence of components firing the onBeforeRender event is not guaranteed.

Understanding the event-driven nature of the controller application and the lifecycle of components is crucial to developing efficient and reliable applications using PHsPeed. By utilizing the appropriate events, you can initialize, modify, and intervene with components at various stages of their lifecycle, resulting in powerful and dynamic web applications.



Component depending events.

In addition to the standard events outlined in the documentation, PHsPeed components offer a range of unique events that empower you to intercept and modify data within components. These specialized events also enable the integration of alternative authentication methods such as LDAP (Lightweight Directory Access Protocol) or SAML (Security Assertion Markup Language). Please note

© 2023 Aducom Software Netherlands

that, as of the current version, PHsPeed does not provide pre-built standard components specifically dedicated to LDAP and SAML.

LDAP Integration:

LDAP integration is seamlessly supported by PHP through a concise code implementation. Crafting a custom component for LDAP is not the most efficient approach due to the simplicity of integrating LDAP with minimal lines of code.

SAML Integration:

PHsPeed has refrained from developing a standard SAML connector due to the presence of an excellent external library called "simpleSAML." This library can be effortlessly imported and used within your projects to streamline SAML integration.

SAML implementations often vary widely in terms of required parameters and setup specifics. Designing a comprehensive SAML component would entail an overwhelming number of properties, and comprehensive testing of all possible features would be virtually impractical.

Furthermore, many organizations adopt their own mechanisms to wrap SAML functionality. These mechanisms might involve redirecting to a specialized login process or inspecting headers to determine the next course of action. The intricacies of these diverse approaches make it challenging to create a universally usable, straightforward SAML component.

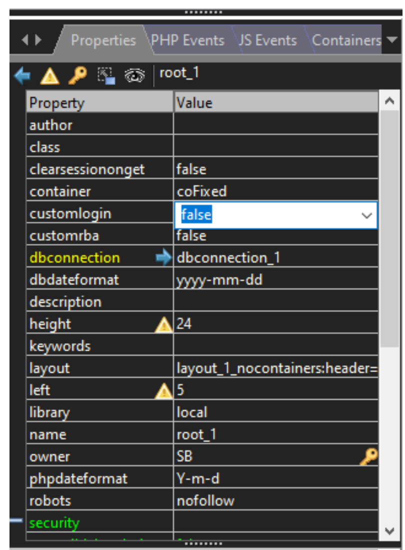
Sample: intercept login procedure

In this section, we will illustrate the process of implementing a custom login procedure using PHsPeed's custom events. The core principle of PHsPeed revolves around maintaining role-based access to internal modules within the database, rather than relying on an external access farm. Once the custom login procedure successfully completes, the standard role-based module of PHsPeed seamlessly takes over.

Step-by-Step Guide:

Identify Custom Login Requirements:

Begin by identifying the specific requirements for your custom login procedure. This might involve integrating with an LDAP server, validating user credentials against an external system, or implementing a unique authentication mechanism, i.e. a select to an external user table.



Creating a Custom Event:

In your PHsPeed project, select the root component. In this component you need to enable the custom login event by setting it's property to true. As a security measure this feature is disabled by default.

Implementing the Custom Login Logic:

The next step is to create the login event procedure. Select the PHP Events tab and find the onLogin event. This event will get fired when PHsPeed requires to login. If you have used the

© 2023 Aducom Software Netherlands

templates, then this event will get fired when the user commits his credentials. If you write your own login form, then the event will get fired when you call the PHsPeed login procedure.

Within the custom event handler, write the necessary logic to execute the custom login procedure. This may involve interacting with external systems, validating user input, and ensuring the security of the authentication process.

In the onLogin event you have to write your own login code. If the credentials are ok then call:

setLoggedOn(); // set logon status

or

SetLoggedOn($usernum); // set logon status and the user identifier is stored as UserNum in the session variables. To retrieve this value at any time, use ***GetSessionVar('UserNum','');***

If you want to make use of the onLoginFail / onLoginSuccess events, then do not use the SetLoggedOn procedure, but simply return true (successful) or false (not successful).

```
<< my login code >>
If ok  {
    return true;
} else {
   return false;
}
```

Above sample also shows you how you can save variables in your session that will live during your session.

SetSessionVar('MyVar', 'my value');

To retrieve the value back use:

$MyVal = GetSessionVar('MyVar', '');

The second parameter will be returned when the session variable is not found.

Transition to Standard Role-Based Module:

Upon successful completion of the custom login procedure, PHsPeed's built-in role-based access module takes over. This ensures that the authenticated user gains access to the appropriate internal modules according to their assigned roles.

© 2023 Aducom Software Netherlands

PHSPEED

The magic of $app and $$ variable

To learn more about the internals of PHsPeed, it is important to know that there is a 'super variable', that contains references to all your components. Actually it is not a variable but a class of type spapplication.

Suppose you have a module called 'shops' then PHsPeed will generate the following line of code:

$app = new shops($config, $connectionstrings);

The application class is declared in the top of the module:

```
class shops extends spapplication {
   component declarations
   public function __construct($config, $connectionstrings) {
       creation of all components
   }
}
```

Simplified example

```
class shops extends spapplication {
  protected $shops_root_1;
  protected $shops_dbconnection_1;
  protected $shops_dbtable_1;
  protected $shops_datasource_1;
  protected $shops_form_1;
  protected $shops_gridpanel_1;
  protected $shops_dbgrid_1;

  public function __construct($config, $connectionstrings) {
   $this->shops_form_1 = new shops_form_1($this, $currentform);
   $this->registerComponent($this->shops_form_1);
   $currentform=$this->shops_form_1;
   $this->registerForm($this->shops_form_1);
   …  etc for each component on the form.

  }
}
```

Upon creating the `$app` object, you gain access to all available components within your form. However, a closer look at your form design might reveal that the components lack a prefix. For instance, `root_1` on your form will appear as `$shops_root_1` in your code. This prompts the question: why does PHsPeed add a prefix?

Consider a scenario where your application features a header and footer, each containing labels and edit fields. By default, if you adhere to these naming conventions, PHsPeed will assign names like `edit_1`, `edit_2`, and so forth to your edit fields. However, this naming scheme applies not just within modules but across all modules. Consequently, when these modules are eventually merged, conflicts can arise due to duplicate names.

To avert such conflicts, PHsPeed implements prefixes derived from the module names. This ensures that the module name remains unique, both for Building Blocks and Application names. The prefix helps differentiate components and circumvents naming clashes during the merging process.

© 2023 Aducom Software Netherlands

Navigating this naming convention and referencing form fields may raise queries. This is where the `$$` variable comes into play. When you need to access a form field, you can employ `$$`, and PHsPeed intelligently appends the appropriate module name.

For instance, if you were to reference an edit field within a module named `edit_1`, you can utilize the `$$` variable in the following manner: `$$edit_1`. This instructs PHsPeed to locate the specific edit field within the designated module, resolving any potential naming conflicts.

In the event that you decide to rename a module, rest assured that PHsPeed will seamlessly accommodate this modification. By utilizing the `$$` variable, the referencing of form fields remains consistent, even if the module name evolves.

To summarize, the prefixing strategy in PHsPeed ensures clear identification and differentiation of components within modules, thereby preventing name clashes during merging. The `$$` variable simplifies the referencing of form fields and maintains accuracy when module names undergo alterations.

Actually, you have seen this in the Ajax sample a few lessons ago.

Naming of HTML fields

In the realm of PHsPeed application development, consistency in field naming is of paramount importance. The PHsPeed template mechanism employs a robust approach to seamlessly transition placeholders into functional fields. These placeholders, often identified by the nomenclature "phid_modulename" – for instance, "phid_shops_dbgrid_1" – undergo a transformation process for seamless integration within the rendered output.

Consider the placeholder transformation process exemplified below:

1. Placeholder: phid_shops_dbgrid_1

2. Identifier Post-Transformation: id_shops_dbgrid_1

3. Field Name Post-Transformation: shops_dbgrid_1

Notably, this systematic approach extends to intricate components like dbgrids, buttons, and checkboxes. These components follow a discerning naming scheme, influenced by their owning context. To elucidate, observe the following button element:

<button type="button" id="id_shops_dbgrid_1_cbtn_0" ...>

In this illustration, the button component embedded within the "shops_dbgrid_1" entity boasts a distinctive identifier ("id_shops_dbgrid_1_cbtn_0"), intrinsically linked to its parent component. This strategic naming methodology fosters unambiguous identification and streamlined intercommunication among diverse elements entrenched within the component hierarchy.

© 2023 Aducom Software Netherlands