



# PHsPeed Courseware

## Lesson 0

### Concepts

## Table of contents

---

Introduction .....	3
Getting started .....	3
Concepts .....	3
Components .....	5
Properties .....	6
Events .....	6
Server events (PHP) .....	7
Client events (JavaScript) .....	8
Basic PHP crash course .....	8
Variables .....	9
Operators .....	12
Application flow .....	14
Object oriented programming .....	18
PHsPeed objects .....	21
The magic of \$\$ .....	24
Naming conventions .....	24
PHsPeed crash course .....	25
Development .....	25
Set up database connection .....	26
Create project .....	27
Create module .....	28
Form .....	29
Grid .....	31
Grid+ Form .....	32
Grid + popup Form .....	33
Generate code .....	34
Locate the generated code. ....	36
Run the code .....	39
Debug code .....	39
Finalize development .....	40
Deploy application .....	41
Initial test / acceptance .....	43
Production .....	43
Evaluate .....	43

## Introduction

---

### Welcome to the world of PHsPeed 3.x !

This eBook will explain what PHsPeed is and how it can help you in building web applications faster, consistently, and with fun. This introduction is a must-read before you start building the tutorial projects.

After reading this document you will:

- Understand the concept of PHsPeed low-code development
- Use the most important functions of the IDE
- Use the application wizard to create your applications
- Use the form designer to modify the form to your specifications
- Use the tool wizards for applying Role-Based Access (RBA) and audit trail
- To generate and run the code
- To debug your PHP event code
- To promote your code for production
- To deploy your application on a server

## Getting started

---

To get started you need to have PHsPeed installed. If you haven't already, then download, and install it from our website <https://www.phspeed.com>. Don't forget to register on our website, so that you can access our forum where you can drop your questions and remarks. Use a valid email address, not some fake ten minute mail or so. The European privacy laws belong to the most severe in the world, we do NOT share your data to anyone, unless we are forced to do so by law.

## Concepts

---

PHsPeed is a low-code, component-based, Rad PHP development tool. It enables you to create database-driven applications in a fraction of the time that you would need using traditional tools. PHsPeed makes use of Bootstrap to enable full responsive behavior. This means that you can create (progressive) web applications with one single code base.

So what is Low Code development? As the name implies you can create applications with a minimum of coding. In many situations, you don't need code at all, which makes PHsPeed a perfect tool for prototyping. However, although there are 'no-coding' tools, we still prefer to call PHsPeed low-code tooling as it still allows you to use code, integrate external libraries with ease, etc.

Look at it this way. If you go to the supermarket then you can find pre-fabricated microwave food. There are many different options but: you have no control over the content, the amount, or the flavor. You buy a pre-cooked meal and only need to press a button. It saves a maximum of the time, but you can argue about the quality. We would call this a 'no-code' platform. You don't need to know how to cook.

Your food comes from the land. If you have a garden and grow your own food then you must have a lot of knowledge. You need to know about the products you grow, how to do pest control, how to harvest, how to prepare, etc., etc. You have to do it all. For us, that's a high-code solution. There is a way of support, like machinery, but you have to do a lot, have enough time, etc.

Let's go back to the supermarket. There you can find potatoes, pasta, rice, couscous, vegetables, etc. You have all the ingredients to create a meal your style, like the components of PHsPeed. It still requires a bit of

preparation, but all the hard work is already done. You need to know how to cook (which is not that hard). That is what we call low-code.

Then there is no-code. It means that you can buy a microwave package. You can still choose, but the contents of the package is fixed, the amount is fixed and the taste is fixed. Perhaps it is a good solution for you, but mostly I prefer to cook my own meal.

## ***So what do you need before you can start?***

Well, first of all, a good idea about what you want to create. As PHsPeed is a code-generator, it is tempting to 'just start' and let the inspiration come to you. No offense, but that is not a good approach. As PHsPeed is database-driven, it requires a database to work against. This means that you need to start with a suitable data model for your application.

PHsPeed comes with a built-in MariaDB database. Depending on your installation, it also has created a standard *database connection* to your database.

Project workflow

## ***What programming skills do you need?***

Honestly, you can build applications without a single line of code. But the truth is, that your requirements eventually will mean that you have to write PHP code. The good news is, you don't need much, and you don't need to be a full qualified PHP developer to be able to build applications. With basic PHP knowledge you should be good to go. JavaScript is used, but you don't need to write it.

## ***Database connection***

A database connection is a configuration in PHsPeed that is needed to connect and maintain the metadata (list of tables and fields with data types) and contains a configuration that your PHP environment needs to connect to that same database.

The IDE connects to the database to retrieve metadata that enables the generator to create forms. Your application will connect to the database to perform Crud operations.

In principle, you can use any database you like, as long as it supports a PDO driver. PDO is a database-independent layer that controls the connection and maintenance of your database. If you do not use (the internal) MySQL/MariaDB, then you need to install your database manually and apply the correct drivers so that the IDE and the web environment can connect. You will find some sample settings in our manual.

## **PDO**

PHP requires a valid PDO driver and some settings in the PHP configuration file (php.ini) to enable database access. The PDO configuration is a connection string, that PHsPeed will fill with the provided data, like user name, password, etc.

Configuration outside your PHsPeed projects.

## **Project independent**

The database connection and PDO configuration are maintained in PHsPeed in a **project-independent** way. The intention is, that connections and PDO configurations can be **reused** in your several projects.

Note: If you deploy your application, then the credentials of your database server most likely will be different than your development environment. PHsPeed behaves differently depending on its state.

- In development  
The connection data is copied into your database component(s). The data is readable and changeable.
- In production  
In production, the only way to connect is to create an encrypted configuration file that contains data about the connection. It is possible to bypass this so that you can make use of a single sign-on or a special configuration where you can maintain different connections depending on the user that logs in. These features will not be explained in these lessons.

## Application generators

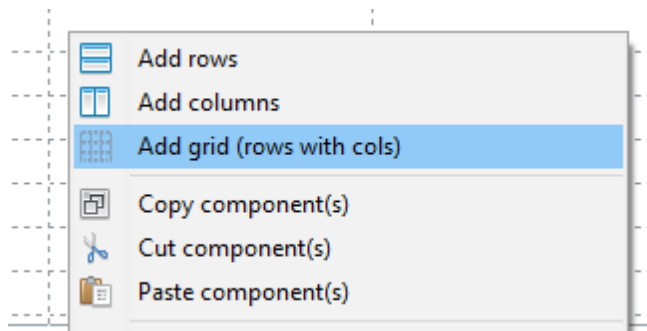
In PHsPeed you can create your forms using a forms designer. As a good starting point, you can make use of the application wizards that serve as your butler. Just select the table of your choice, apply a few preferences and PHsPeed will fully automatically create your form for crud-operations, grids, or combos. In the tutorial lessons, we will use the applications wizards extensively. But remember. Everything the wizards do can be done by hand. But why should you? Just create the application to get a good starting point, then modify them to your taste.

## Components

PHsPeed is component oriented. In the IDE there is a component palette where you can select the component you require and put them on your screen where you want it. PHsPeed uses the bootstrap grid to design your forms. Designing forms has three stages:

### 1. Layout

PHsPeed uses the Bootstrap grid system to layout forms. To understand this concept, you should read the documentation on [getbootstrap.org](http://getbootstrap.org). But in general you have rows and columns (like the traditional HTML table) and you can apply lay-out rules to the columns. To create grids you can design rows and columns individually, or use a function to create a grid in one go.



We will come to that later.

Each layout-able object is divided in 12 units. Bootstrap 4 has 5 device contexts that can be defined to hold a certain number of units to a maximum of 12. Each device context describes an output device of a certain horizontal resolution.

DesktopXL	Desktop	Tablet	Phone	Mobile
≥ 1200px	≥ 992px	≥ 768px	≥ 768px	< 576px
xl	lg	md	sm	xs

Suppose you have a row with two columns. To define this, you need to add a row to the form, with two columns and apply 6 units to both columns of the desktop xl class:

Property	Value
backgroundcolor	
bordercolor	none
borderradius	none
class	
columnclass	container
deviceclass	
deviceclassdesktop	
deviceclassdesktopxl	6
deviceclassmobile	
deviceclassphone	
deviceclasstablet	

With this definition, you will get a row with two columns of half the size ( $2 \times 6 = 12$ ). If you require a column of  $2/3$  you can use a setting of 9 and 3 ( $9 + 3 = 12$ )

Depending on the presented data, it is likely that you will not be able to see two columns on a mobile phone. In older systems, you would get a horizontal scroll bar and scroll over your form to read its data. But what you really (should) want is that the columns are not horizontally on a mobile phone, but below each other so that you only need to scroll vertically. By applying 12 units to each column on the `deviceclassphone` property you achieve this. Bootstrap will read the resolution of your browser, determine its device class and then apply the given setting. This will cause your form to look differently on different platforms with different screen resolutions. This is called 'responsive design'.

## 2. Non visual components

When you are building a database application then you need components to connect to your database, your tables, your data sources; when you want to send an email you need an email component. These components help you to make those functions accessible without complexity, but they have no visual representation like an edit box for instance. These components are called non-visual. Although you can add any type of component any time, you need a table component on your form before you define an edit field to a member of that table.

## 3. Visual components

Visual components are components that have a visual presentation, like grids, fields, menus etc.

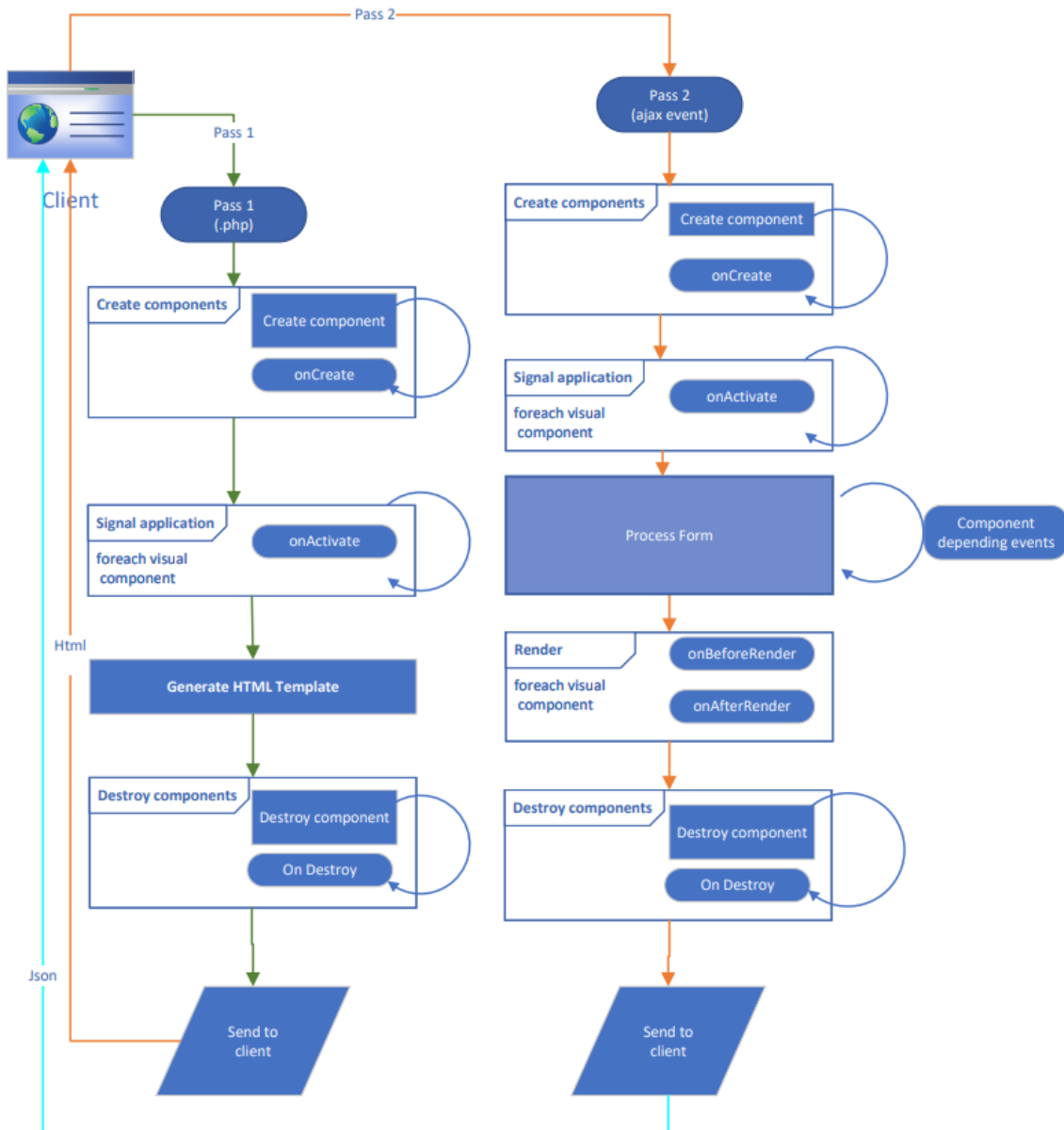
### Properties

Components have *properties*. Properties define a certain behavior of the component that it applies to. Some properties (like left, top, width, and height) are only used to reconstruct the designed form and have no meaning for the code generator (as the form is build based upon the Bootstrap rules). Some others can change the appearance of the component, like color, font type, or it's behavior like linking to other forms components, management of master-detail etc.

Properties can be set directly in the property editor, many can also be set in code.

### Events

PHsPeed applications follow a standard application flow, that is depending on the application type. A grid application has a different flow than a crud application, but the concepts are the same. In many situations it can be necessary to change the standard behavior of the application, and to apply custom code. In PHsPeed this is done by using **events**. Events are moments where PHsPeed looks if there is some custom code to integrate. In the below flow chart you can see which standard events apply.



Standard application flow.

Each component that is created will generate an 'onCreate' event at runtime, that can be used to set some basic properties. Some are done by the developer in the IDE, but if a property has some dependencies, it is possible to change them by code. Do NOT refer to properties of other components, because you cannot be sure that this component is already created. Components are created in a certain sequence, and if you have properties to set, then wait until all components are available. This is the onActivate event. Each component will receive this message. Again, be aware that if you look into a property of a certain component that is triggered later in the sequence that you cannot rely on the value. But there are some standards:

Non visual components like database, queries, tables etc, are generated before the visual components. So if you have to setup all kinds of queries the onCreate/onActivate components are key. But if you have visual components, then you have to be careful. Most of the time, the onBeforeRender is suitable as every component will have it's value, .... unless you change another component in this event that already has been rendered. Looks complex. But once you start working with events, you will see that it is not really of an issue.

### Server events (PHP)

Server events are occurring on the server and are written in PHP. There are some basic PHP events that

apply to all components:

onCreate	When the component is created
onDestroy	When the component is destroyed
onActivate	When all components are created

Visual components have additional events that can be used

onBeforeRender	To influence the rendering of the component
onAfterRender	To influence the rendered code

But other components might have other events. I.e. a database grid has events to process selected rows, to modify a column under certain circumstances etc.

## Client events (JavaScript)

Client events occurs on the client and is JavaScript. In general, developers do NOT need to write JavaScript. All required code is generated. But it is necessary to invoke some of the events as they might be used to open certain functionality.

A special case if you want to implement Ajax events. Suppose you want to perform some action when a user clicks on a button, but you do not want to submit the data, but use an Ajax event.

The button has a server AND a client event of the same name (onClick). As the user clicks on the button an JavaScript event is required to send a message to the server. To invoke, you can click on the JavaScript event, that will add the event to the property editor and open the editor that contains the required code (no need to change). The PHP event can be created and the editor opens the PHP editor with a function prototype. That is where you can put your custom code.

Don't worry, during this course you will learn all about this.

## Basic PHP crash course

---

This section provides you with a minimum knowledge of PHP, it is NOT a full in-depth course. If you already know PHP, then I still advise you to scan through these pages as we will also will describe how you can modify the behavior of the many PHsPeed components.

### ***Web applications***

Before going into PHP it is important that you know a bit of how web applications work, and how PHsPeed web applications work. Simplified:

If you have been working with desktop applications like Word or Excel, then you actually are running a program locally on your workstation. Before you close your program, you must save your data or it gets lost. Web applications work differently. To start your web application you use a web browser that seeks contact to a server by the url you have given. This starts the PHP application that will create output that is shown on your browser, and **then closes the program and forget all about you**. The output that is sent can contain several parts, but mostly it contains references to HTML which describes your page, a style which describes the looks and JavaScript that performs client-side interactive behavior. But the most important message here is, that after sending the output of the application to the browser the application is closed!

So how does it work when you login on a page, then goes to another page containing a menu and start all kinds of forms? Surely the system must know something about you? The answer is 'sessions'. Whenever you start a PHP application a session must be created and the reference must be kept within the web page. When the application closes, it will store the session data and puts the reference to that data hidden in your web page. When you click on a button and the PHP application starts up, it will see that there is a session and retrieve it's data. It is up the application to handle this sequence in a valid way.



## PHsPeed

When you use PHsPeed, you don't have to worry about sessions, as it is handled for you automatically. But it is important to know the basic principle. When we explain variables, you will know that there is a session pool where your overall data is managed. PHsPeed **does not send** individual data to the client to be sent this data back to the application when a user clicks a button. This method was used often in the early days, but allows potential attackers to interfere and therefore can be considered as a vulnerability. PHsPeed stores all required information in the session along with some token information to avoid this problem.

## Variables

This section provides you with a minimum knowledge of PHP. If you already know PHP, then I still advise you to scan through these pages as we will also describe how you can modify the behavior of the many PHsPeed components.

### Variables

Variables are placeholders for values. They have a unique name that starts with a dollar sign. If you come from other languages, PHP is weakly typed, which means that a variable can store virtually any data structure, which also can change over time. (In PHP 8.x it is also possible to do some strict typing, but that is out of scope here). A variable name always starts with a '\$' and is followed by a sequence of letters and numbers, but the first character of the name must be a letter. Use a significant identifier. \$name, \$address are valid names that also is self-explanatory. \$a, \$b are not...

You can declare variables anywhere in your code, but they only exist in the context that they are used. So if you have created a procedure with a variable, then this variable will be forgotten as soon as you leave the procedure. We'll get back to that later. You declare a variable by assigning a value to it. See the next sample:

```
$name='janssen';           // name will be a string
$housenumber=12;         // house number is an integer.
```

Because variables can be declared anytime, it is possible to generate issues if you reference a variable without a value. For instance

```
echo $person;
```

will generate an error if this is the first time that the variable is used and does not have an (initial) value. When you receive data from the web, it is not always clear that the data you require will be there. To overcome this issue, you can test if there is a value:

```
if(isset($person)) {
    echo $person;
}
```

This sample shows two new concepts: a conditional statement that will be executed when the value is true (isset(\$person)). A function (isset) that performs a certain task (here to test if the variable has a value). A block between {} that shows the full code that is going to be executed when the condition is true.

Variable types

In the above sample we've seen a string and an integer. In PHP you can have different types of data:

### Integer.

This An integer is a whole number and has the same range as the "long" data type of C. In most machines (e.g. Intel Pentium processors), an integer is a 32-bit signed number that can be as low as -2,147,483,648 or as high as 2,147,483,647 Some examples:

- 10
- 100
- -1023

## Floating point numbers

A floating-point number is a number that can contain decimals and an exponential notation. When accessed using a typical platform, a floating-point number is 8 bytes in size and can reach as high as 1.8E+308 down to 2.2E-308. A floating-point value can have a "+" or "-" sign. Some examples;

- 12.40
- 34.3E2
- -23.44

## String

Strings are a sequence of characters and numbers between single or double quotes. Some examples:

```
"this is a string"
'this is a string'
```

The difference is that a string between double quotes will be interpreted to find and replace variables.

Suppose the following:

```
$name= 'john';
$text = "hello $name";
```

After execution \$text will contain **hello john** where

```
$name= 'john';
$text = 'hello $name';
```

\$text will contain **hello \$name**

When to use the single and double quotes? If you have no replacement to be executed use single quotes, as it is more efficient. But if you require a string containing a single quote, then use double quotes:

```
$text="my name is O'connor";
```

## Boolean

A boolean variable can only contain two logical values: true and false. The result of a condition is always true or false.

```
$result=8 > 3
```

\$result will contain **true**

## Null

When a variable has the value null, then it exists, but has **no value**. You can test if the value is null, but you cannot use it in expressions, without generating a run time error.

## Arrays

An array is a list of values of any kind. In PHP you can have two separate array types, a regular array, than contains a number of different elements and an associative array that handles key=>value pairs.

An example of a regular array is:

```
$a[0]=12
$a[1]=14
$a[2]=15
```

A regular array contains a variable name and its **index** in that array to represent a value. So the value of `$a[1]` is 14

An associative array is similar but with a 'bite':

```
$a['mynumber']=12
$a['yournumber']=1
$a['hisnumber']=12
```

Here the value of `$a[1]` is undefined. But the value of `$a['yournumber']` is 1

## ***Scope of variables***

The scope of a variable describes its validity at any point. In general, when you have written functions, variables are only valid within the procedure that they are created. So what if you need that value anywhere in your program. In that case you can declare the variable as **global**. In this case the variable will be created once and live until the application terminates.

## ***Session variables***

Session variables are variables that persists over your complete session. So if your PHP application terminates then these variables

## ***Superglobal variables***

When you have created a web application then the content of a form is sent to your application. So how do you read those variables. First of all there are two way to send data to your form. The first on is when you type an url with parameters.

```
https://www.mysite.com/myapp.php?name=janssen&housenumber=12
```

These variables are part of the url and are send as GET variables.

If you have a form and hit the submit button then the content of the field are send as POST variables.

PHP has 'superglobal' variables that contains this data:

```
$_ENV[] - This is an array that contains environment variables.
$_GET[] - This array holds all of the "GET" variables gathered from the user's web browser
```

Because these variables are 'always' there they are mentioned here for clarity. PHsPeed retrieves this data and makes them available as session variables under their own name while protecting the data against XSS. Therefore using the variables directly is not recommended. But if you need certain information, then it is there.

## ***Casting variables***

If you perform calculations, then it can be necessary to 'change' the datatype. Suppose you have a string "10" then you cannot calculate with that value. Before you do, you need to *cast* the value to a number:

```
$a="10";
```

```
$b=(int) $a + 10;
```

## **Object variables ('properties')**

Just to be complete, PHsPeed is fully object oriented. That means that there are rules to access variables that are bound to that object (usually called 'properties'). In the paragraph 'object oriented programming' we will go deeper into this but know that if you have to access a variable of a certain object (or 'class') that you need to 'point' to them like:

```
$myclass->property. (Class variables are called properties)
```

Your html form contains fields that you want to process in your code. These fields are also objects. To access those, you can use the short \$\$ notation. So if you have a field on the form then the object in your code is called 'address', then you can access this object with \$\$address->some property.

## **PHsPeed considerations**

### **Get variables.**

The 'get' variables are available as session variables. To set and retrieve a session variable use the function getSessionVar and setSessionVar. We'll get back to this later. Avoid to access the \$\_Get and \$\_Post arrays directly, to avoid security issues.

### **Put variables.**

The put variables are fully managed by PHsPeed. All data becomes available in the value property of the component that manages the field on the form.

## **Operators**

An operator is a symbol that represents a certain action. In

```
$result=5 * 3;
```

The \* (Multiplier) is an operator.

In PHP (and many other programming languages) you can have different operators.

## **Assignment operators**

=, -=, /=, \*=, +=, .=, |=, >>=, <<=, &=, ^=, and &

Assignment operators are used to assign a value to a variable. The most elementary is the = sign that is a simple operator:

```
$result = 5;
```

But there are more assignment operators. The mostly used are

.= which concatenates strings like

```
$result= 'hello t '.$value;
$result.= 'welcome!';
```

+= which adds a number to the variable.

-= which subtracts a number from the variable.

```
$result=12;
$result+=1; // adds 1 to $result and is equivalent with $result=$result + 1;
```

## Calculations

We have already seen some operators that perform calculation. As an overview:

```
+      Add operator
-      Subtract operator
*      Multiplication operator
/      Division operator (returns the quotient)
%      Modulo function (returns the remainder of a division)
```

The variable type of the result depends on the result. If you have two integers like 6 and 5, and you divide them, then the result will become a float.

There are two operators that can be used for fast increment or decrement.

```
++     Increase by one
--     decrease by one
```

Examples;

```
$a++; $a--;
```

It is also possible to put the ++ and -- before the variable.

```
$++a, $--a;
```

This is rarely used, but can be helpful if the variable is used as a loop variable. It means that the increment of the variable is performed initially before the expression is used.

## Comparisons

You will have the need to compare values often. Depending on the comparison you have to execute or skip a block of code. In PHP we have the following compare operators

```
==     Is equal
!=     Is not equal
<=    Is less equal or equal
<     Is less equal
>     Is greater than
>=    Is greater or equal
```

The result of a compare is a boolean, always 'true' or 'false'.

The ! operator can be used to turn the compare

```
if (!$a < $b) { ... }
```

Needs to be read as **if not \$a < \$b then...** That is the same as **if(\$a >= \$b)**.

In this example the use of ! can be easily avoided, but the use can be quite handy when you have to respond to the result of a function. Example:

```
$var = 10;
$message = ($var>10) ? 'greater': 'smaller or equal';
echo $message;
```

will output smaller or equal.

For larger expressions, it is possible that the code will become fuzzy. For that reason you can always fall back to the traditional structure:

```
if($a < $b) {
    some code when true
} else {
    code when not true
}
```

## ***Special comparisson (ternary operator)***

A very handy construction to avoid if-then-else (see application flow) constructions is the use of ternary operators. It is mostly used in relative simple expressions:

```
if(condition) ? { expression 1 } : {expression 2};
```

if the condition is true then expression 1 is executed otherwise expression 2 will be executed.

## ***Logical operators***

Sometimes you have more complex decision rules. In that case you need to be able to combine arguments with 'and' or 'or'. In PHP there are separate operators for these kinds of comparisons

```
&&    Logical and
||    Logical or
xor   (not very commonly used) .
```

It is mandatory to use () to prioritize the decision:

```
if( (condition) or (condition) ) {...}
```

```
if( ( (condition) or (condition)) and (condition) ) { ...}
```

To avoid misunderstanding, use brackets for prioritizing. It makes reading a lot easier to others, although there are rules what condition (and/or) comes first. But using brackets is clear to anybody.

Another thing to consider is the sequence of comparison. The execution of a condition is terminated as soon as the outcome is clear. So if you have an or condition and the first condition is true, then the second one will not be determined. If you consider this rule, then you can create your ifs more efficient. Although the computers are very powerful nowadays, it still can become a point of concern when processing much data.

## ***Other operators***

There are more operators to work with bits and bytes, but they fall out of scope of this crash course. If you are not an experience developer you will not miss these as they are not widely used.

## **Application flow**

Under normal circumstances your application runs the programming lines in sequence. But in many case you need to interfere. Some parts are depending on the status of a condition, or needs to be repeated until a certain condition is met. Here we introduce the term 'compound block'. A compound block is a sequence of statements (program lines) that is executed as a whole withing { and }.

## Conditional application flow

### if-then-else

In the previous chapters you already have met the if-then-else construction. In general:

```
if (condition) {
    compound block
} else {
    compound block
}
```

There are cases where you can have an if condition within an if condition:

```
if (condition) {
    if (condition) {
        compound block
    } else {
        compound block
    }
} else {
    compound block
}
```

The indentation that is used to make the code more readable is not mandatory, like in Python. Spaces have no meaning in PHP. But it is good practice to indent blocks so that the readability improves. Sometimes you have more conditions in a row.

```
if (condition) {
    compound block
} else {
    if (condition) {
        compound block
    } else {
        if (condition) {
            compound block
        } else {
            compound block
        }
    }
}
```

Although there is indentation, the readability of nested ifs is not optimal. To make the code better readable you can consider using elseif. The code then becomes

```
if (condition) {
    compound block
} elseif (condition) {
    compound block
} elseif (condition) {
    compound block
} elseif (condition) {
    compound block
} else {
    // finalize with a regular else for the situation that none of the conditions
    are met.
}
```

## switch

another way to implement more if-elseif constructions is by using a switch. This is especially helpful when you only have to validate a value from a single variable / expression:

```
switch($var) {
    case 1: some code
           some code
           break;

    case 2: some code
           some code
           break;

    case 3: some code
           some code
           break;

    default: some code
            some code
            break;
}
```

The variable does not need to be numeric:

```
switch($word) {
    case 'yes': some code
               some code
               break;

    case 'no'  some code
               some code
               break;

    default:  some code
               some code
               break;
}
```

## Loop structures

PHP has a number of different loop structures. Loop structures repeats a compound block until a certain condition is met. Although obvious, it is important that you verify that a certain loop will end, otherwise you will have a situation where your application becomes unresponsive because the application never ends. Eventually the webserver will interfere because you consume too much time and the application will be aborted. But it is a situation you must avoid.

### *while loop*

A while loop will initially verify the condition and as long as the expression is true, it will repeat a compound block. So the block is repeated 0 to x times.

```
$c=read_record();
while( $c != " " ) {
    {
        process record
    }
}
```



```

}
$c=read_record;
}

```

### **do-while loop**

A do-while loop will execute the block and will verify at the end of a condition is met to leave the loop. So this block is repeated 1 to x times.

```

do {
    $c=read_record();
    process record
} while( $c != " )

```

### **for loop**

The for loop is used when you know the amount of times that a loop must be executed. Although there are constructs to process arrays (for-each) you sometimes needs to keep track of the element number

```

for ($i=0; $i< 10; $i++) {
    compound block
}

```

$\$i$  gets it's initial value of 0, then at the end of the loop the value will increase until the value is 10. You do not need to start with 0, and you can use an expression like  $\$i+=2$  if you want to increase  $\$i$  by 2. But always be aware of the danger

What is wrong with the following sample?

```

for ($i=0; $i = 9; $i+=2) {
    compound block
}

```

(if you haven't figured it out,  $\$i$  gets the values 0, 2, 4, 6, 8, 10... and will **never** get the value of 9. This loop never ends!

Another dangerous (but valid!) way of a for loop is:

```

for(;;) {
    compound block
}

```

This loop also never ends. So there needs to be a construct to leave a loop in these situations.

### **aborting loop structures**

There are situations where you want to abort a loop because of some unexpected error condition.

#### **break;**

The break statement aborts a loop and jumps to the next line after the compound block

```

if (condition) {
    statements
    break;
    statements;
}

```

The application flow continues after } when reaching break;

```
for ($i=0; $i = 9; $i+=2) {
  compound block
  if($i>15) { break; }
}
```

Loop will abort.

### ***continue;***

The continue statement will restart the loop at the place of the condition. Especially useful in for loops:

```
for ($i=0; $i = 9; $i+=2) {
  statements 1;
  if(condition) { continue; } // the $i will increase and the applciation continues at statements 1
  statements 2;
}
```

### ***foreach loop***

If you have arrays, then sometimes you need to process each element. In the PHsPeed runtime code this is a very common situation. foreach will perform the compound block for each element of the array:

```
$a=[];
$a[]=1;
$a[]=2;
$a[]=3;
foreach($a as $element) {
  $element will contain 1 the first time the loop is executed, 2 on the next iteration, etc.
}
```

In the variable section we also have discussed associative arrays where elements have a key:

```
$a=[];
$a['pet']='dog';
$a['owner']='john';
$a['breed']='boxer';

foreach($a as $key => $value) {
  $key will contain the key of the array, $value the content of the element so
  pet and dog in the first iteration
  owner and john in the second etc.
}
```

## **Object oriented programming**

This paragraph is about the object oriented structure of PHsPeed. It can be confusing to novice developers to see what objects are and why they are important. If you want to become an experience developer then it is important to understand the concepts as OOP (object oriented programming) is mainstream in any programming language nowadays.

In OOP (i.e. object-oriented programming), you will combine codes and data to create an object. A computer application created using this style consists of different objects that can communicate with each other. Often, these objects are self-contained and possess different methods and properties. Properties serve as an object's data. Thus, these are variables owned by the object they point to. The methods, on the other hand, are functions an object supports. Classes serve as templates for a programming object. They describe the properties and methods that an object will possess. You can see each object as an Island. The

Island is self-contained and communicates with elements on other Islands. To move data between the Islands you use ferries, a formal way of transporting data.

## ***SPObject***

In PHsPeed it all begins with a standard object (simplified):

```
class spobject {
    public $name;
    protected $app;

    public function __construct($name) {
        code
    }

    public function __get($property) {
        code
    }

    public function __set($property, $value) {
        code
    }

    public function __destruct() {
        code
    }
}
```

This class is the 'mother' of all objects. If you write code in PHsPeed, you most likely will work with the available objects and don't have to create new, but for the concept it is important to know that spobject is the root of the PHsPeed object model.

The declaration starts with 'class spobject'. Every class start with a class statement. In PHsPeed all classes are contained in a PHP file of the same name. This is very common in PHP environment that use *autoloaders* for PHP modules (for experience devs: you will not see many \$includes in PHsPeed).

The we see two class variables, called 'properties'. These are variables that have an *access type*. There are three of them:

**public:** property can be access by any other object

**protected:** property can only be accessed by the class itself and objects that are descendants of this class (we get to that)\

**private:** property can only be accessed by the class itself and not outside.

Next we see a number of functions that belong to that class. They too can have an accessibility declaration. These functions are called 'methods'. So `__construct` is a *method* of the *class* spobject.

### **Important!**

- To be able to use classes you need to *create* them. You do that with the statement ***new***.

```
$myobject = new spobject('myname');
```

The function new requires that you define a initialization function called `__construct`(some parameters). `__construct` is **always** called when you create a new object. (If you define a class without a constructor then you have to define some initialization method to initialize the object. So use `__construct`).

- If the object gets destroyed (i.e. at the end of your php program, php will clean up the trash) then the ***destructor*** function will be called.

- getters and setters

You see a `__get` and `__set` method. These are formal ways to move data around from objects to objects. PHP invokes these functions automatically and as all components of PHsPeed are descendants of `sproject` you don't have to worry about this.

- references

To access a property, you must point to that variable:

```
public function __construct($name) {
    $this->name=$name;
}
```

`$this` is required if you access a property of the current class.

## Inheritance

The `sproject` is a very simple object, that has limited functionality, yet is very important because it is the mother of all other PHsPeed objects. These other objects inherits the properties of the underlying objects. To understand lets look at the next sample:

Suppose we have a class 'graphic object'. There are several graphical objects, like a square, a triangle, etc. They all share some properties, like a position, color... But there are also differences. A square has four lines, a triangle three. So the structure could look like this (simplified):

```
class graphicalobject {
    public $x;
    public $y;
    public $color;

    public __construct($x, $y);
    public __destruct();
}

class triangle extends graphicalobject {
    $line1
    $line2
    $line3
}

class squaire extends graphicalobject {
    $line1
    $line2
    $line3
    $line4
}
```

In above sample the triangle and square *inherits* the properties and methods of the graphical object. You can see this as a 'kind-of' normalization of the object structure, similar to the normalization of a data structure of your database.

To improve we can define an object 'line' that has properties, like, x, y coordinates for the start position and end position of the line. And a method to draw that line:

```
class line {
    $x1; $y1;
    $x2; $y2;
```

```

public __construct($x1, $y1, $x2, $y2) {
    code
}

public function draw() {
    code
}
}

```

So now we have a class that can draw a line, and two classes that needs to draw lines:

```

class triangle extends graphicalobject {
    $line1
    $line2
    $line3
    public __construct (($point1, $point2, $point3) {
        $line1=new line($x1, $y1, ...
        $line2=new line($x1, $y1, ...
        $line3=new line($x1, $y1, ...
        parent::__construct(...;
    }

    public function draw() {
        $line1->draw();
        $line2->draw();
        $line3->draw();
    }
}

```

\$point would need to be an object too, containing properties to hold the start and end position. As creating an object will call the constructor of that object, you need to take care that the constructor of the extended (=inherited from) objects are called. This is done by `parent::__construct(..)`; The `::` sign means that you are working in the same object (as it is the same object you cannot use `->` as it points to a different location).

This example is not complete, but you now have an idea about how object orientation functions. If you do not understand fully then don't worry. Most of the time you will not need to use this. Important to know is that objects have properties that you access through object `->` property.

## PHsPeed objects

So, if you design a form, then all form components are actually objects in PHsPeed. If you generate code you can easily find this in the main module. PHsPeed will **always** create a variable `$app` and that variable is passed in almost every event that you can use. This `$app` variable instantiates a class inherited from `spapplication`, and the constructor will create all the form objects (simplified):

```

class comment extends spapplication {
    protected $comment_root_1;
    protected $comment_dbconnection_1;
    protected $comment_dbtable_1;
    protected $comment_datasource_1;
    protected $comment_form_1;
    protected $comment_gridpanel_1;
    protected $comment_dbgrid_1;

    public function __construct($config, $connectionstrings) {
        global $connections;
        global $apikey;
        $this->sessionmethod='P';
        $this->comment_form_1 = new comment_form_1($this, $currentform);
        $this->registerComponent($this->comment_form_1);
        $currentform=$this->comment_form_1;
    }
}

```

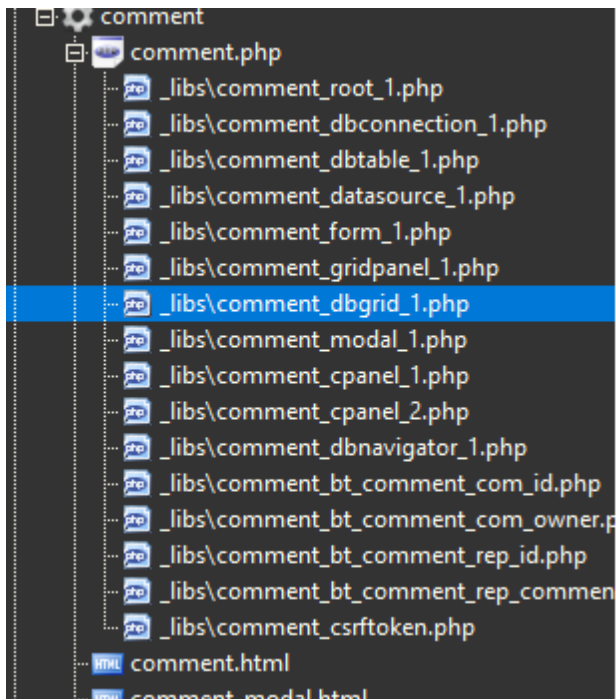
```

$this->registerForm($this->comment_form_1);
$this->comment_csrf_token = new comment_csrf_token($this, $currentform);
$this->registerComponent($this->comment_csrf_token);
$this->comment_root_1 = new comment_root_1($this, $currentform);
$this->registerComponent($this->comment_root_1);
$this->registerRoot($this->comment_root_1);
$this->bodyclass=$this->comment_root_1->bodyclass;
$this->comment_dbconnection_1 = new comment_dbconnection_1($this, $currentform);
$this->registerComponent($this->comment_dbconnection_1);
$this->comment_dbtable_1 = new comment_dbtable_1($this, $currentform);
$this->registerComponent($this->comment_dbtable_1);
$this->comment_datasource_1 = new comment_datasource_1($this, $currentform);
$this->registerComponent($this->comment_datasource_1);
$this->comment_gridpanel_1 = new comment_gridpanel_1($this, $currentform);
$this->registerComponent($this->comment_gridpanel_1);
$this->comment_dbgrid_1 = new comment_dbgrid_1($this, $currentform);
$this->registerComponent($this->comment_dbgrid_1);
}
}

```

```
$app = new comment($config, $connectionstrings);
```

The \$app allows you to access all the properties of all the (other) components. Now let us zoom in into one of the objects: `$this->comment_dbgrid_1 = new comment_dbgrid_1($this, $currentform);`  
 In the IDE this is very easy as you can 'zoom in' into any object. First we need to locate the object. In the project tree you need to locate the object in the generated code:

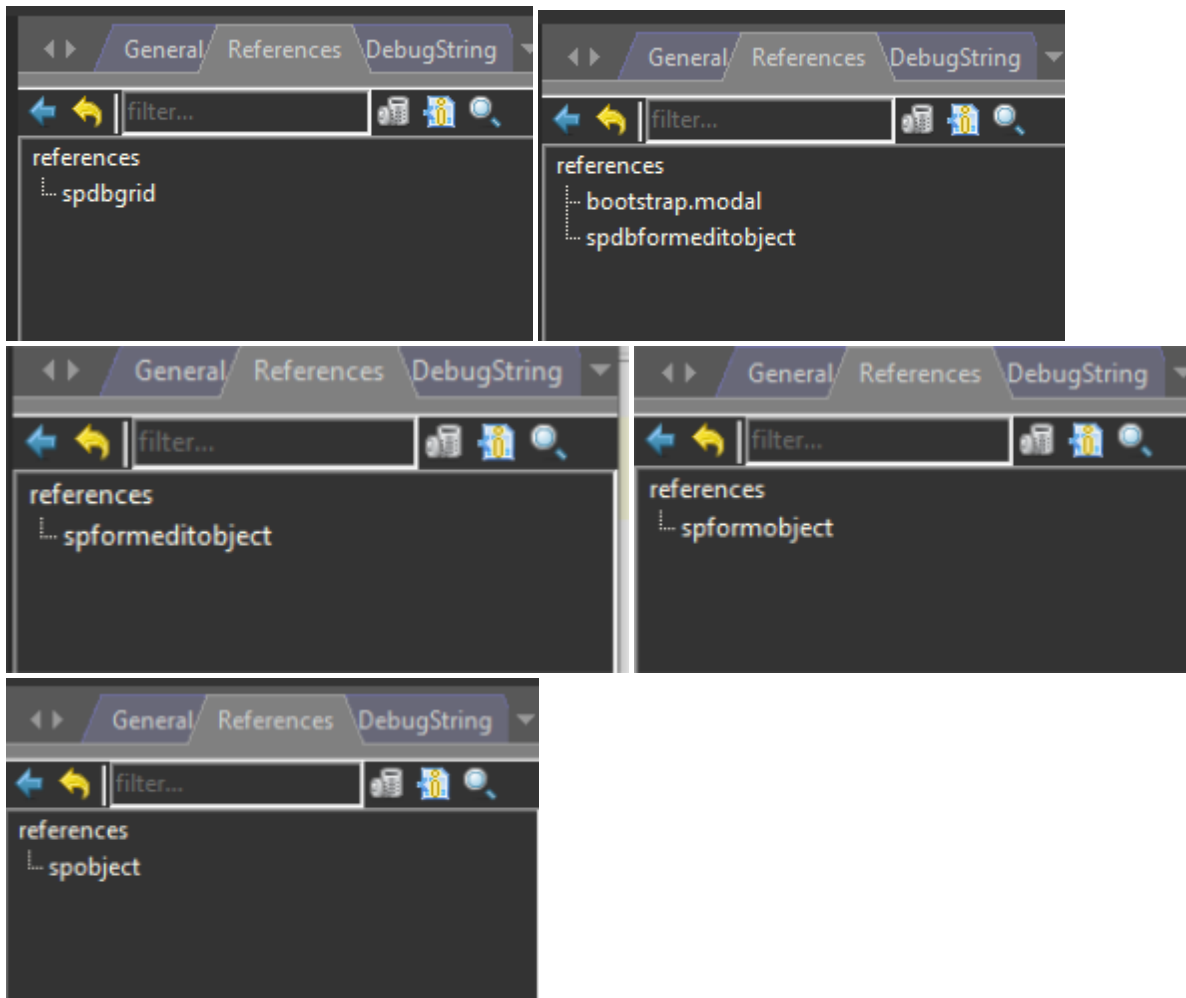


```

rt\sparea comment_dbgrid_1
1 <<?php
2 //=====
3 // PHsPeed V2.3.1 B1 - X64
4 // © 2014.. 2022 Aducom Software
5 // generation date: 2022-12-08 14:20:27
6 //=====
7
8 class comment_dbgrid_1 extends spdbgrid {
9
10 public function __construct($app, $form) {
11     parent::__construct('comment_dbgrid_1',$form);
12     $this->app=$app;
13     $this->visible='true';
14     $this->formclass='dbgrid';
15     $this->idprefix='ph';
16     $this->form='id_comment_form_1';
17     $this->formid='id_comment_dbgrid_1';
18     $this->formname='comment_dbgrid_1';
19     $this->formlabelid='lbl_comment_dbgrid_1';
20     $this->modaltarget='modal_1';
21     $this->modalbuttontitle='new';
22     $this->modalbuttonclass='btn-default';

```

If you click on the component then the editor will show the generated code. As you can see here, the `comment_dbgrid_1` is extended from `spdbgrid`. In the sidebar you will find a reference to all parents (here only one).



Now you click on this line to zoom into the underlying object, until you come to the end, which is ....

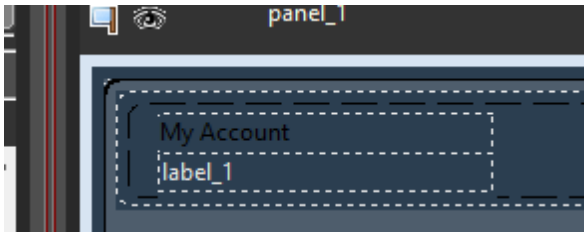
spobject.

Following the code is very handy while debugging. If you step through your code, PHsPeed will automatically enter the object code so you can always debug into depth.

More about `$this->comment_dbgrid_1`. On the designed form you have put a dbgrid. It's name was `dbgrid_1`. The code generated appended the module name to it. Suppose you rename the module, or you rename the component. After all, `dbgrid_1` is not a useful name, you might want to give it a more logical name. If you have written event code, then you have to search and replace the names of all the changed properties. To avoid this and to make coding more easier, you can use the `$$` notation. We call it 'the magic of `$$`'.

## The magic of `$$`

If you put a label component on a form, then by default the name will become `label_1`:



In this case PHsPeed creates a class name `"cust_detail_label_1"`. The main reason for this is that you can add other classes to this application. Suppose that you have selected a template with a header. This header can also contain a `"label_1"` which would cause a conflict when running code like:

```
$label_1->value='something'.
```

PHP will not be able to decide which variable to use and generates an error. So if PHsPeed generates a unique name adding the application name into: `$cust_detail_label_1`, then the valid syntax to assign a value to this label would be:

```
$cust_detail_label_1->value='myvalue';
```

But what if you decide for whatever reason, to rename your application? Then you must redo all your code as `$cust_detail_label_1` will have become something different. Besides, if you have to prepend all your variables with the module name, then that becomes a lot of hard and boring work.

To overcome this, PHsPeed uses the `$$` notation, which you can use for all the components that you have put on the form. So

```
$cust_detail_label_1->value='myvalue' will become $$label_1->value='myvalue';
```

The code generator will then append the module name while generating code, and if you rename the module, the generator will change the variable names accordingly.

## Naming conventions

PHsPeed will generate unique names for components based upon the application name and the component



name. But what about your HTML? PHsPeed does not work with IFrames, but generates DIVs and as PHsPeed uses Bootstrap, these DIVS have names and classes

PHsPeed has a standard for this too. All HTML objects will have an id, rendered HTML objects have a name and an id. Predefined id's are based upon phid\_ and the name of the component, similar to the php variables. phid\_ is used to replace these sections by the rendered code of your application. The rendered HTML objects have the same name as their PHP variable counterparts, and an id\_ as prefix in case of an id.

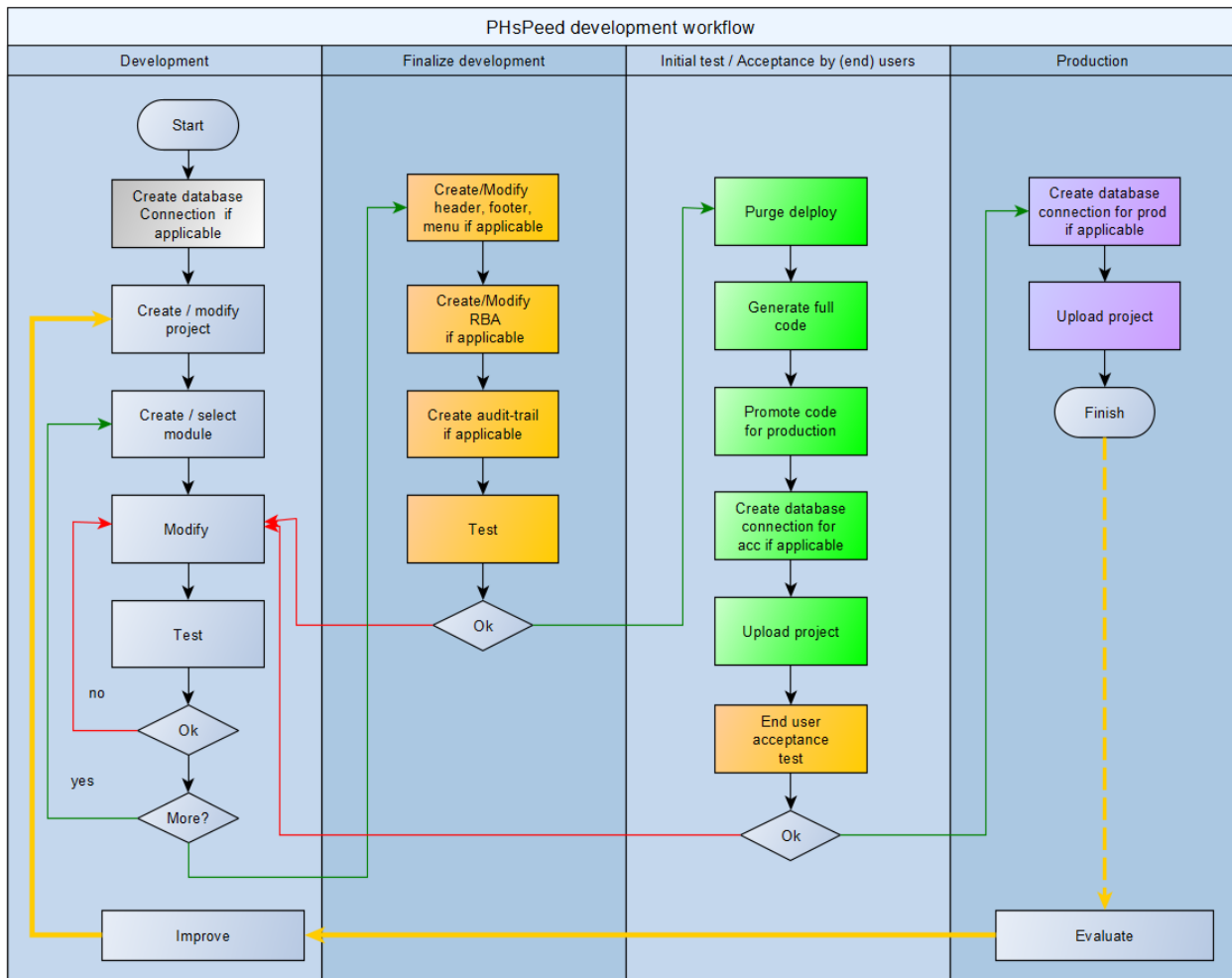
So if you have a field edit\_1 in application cust\_detail then the following names will be generated:

*php: \$cust\_id\_edit\_1*

*html: name="cust\_id\_edit\_1" id="id\_cust\_id\_edit\_1".*

## PHsPeed crash course

This section describes in short the features of the IDE. During the several tutorials you will learn these features more in detail.



## Development

The following paragraphs will follow the development workflow. Before you can work with any project, you need to specify the database connection. If you are new, then you can work with the provided MySQL database that already contains a sample database. The provided sample project works with this database, and all settings are in place. You can add new tables to the schema, or create new schemas. The current settings can be a good example how to setup the connections.

## Set up database connection

Depending on your situation you probably need to start with creating a database connection. If your application is using a database, then initially the IDE must be able to connect in order to retrieve its metadata (tables, views, fields, etc.). This connection is defined once for each database and can be reused over your projects that require this connection. If you define your project, you can apply the default connection to use. It is possible to have more connections, even to different databases in one project or module by applying the database connection component.

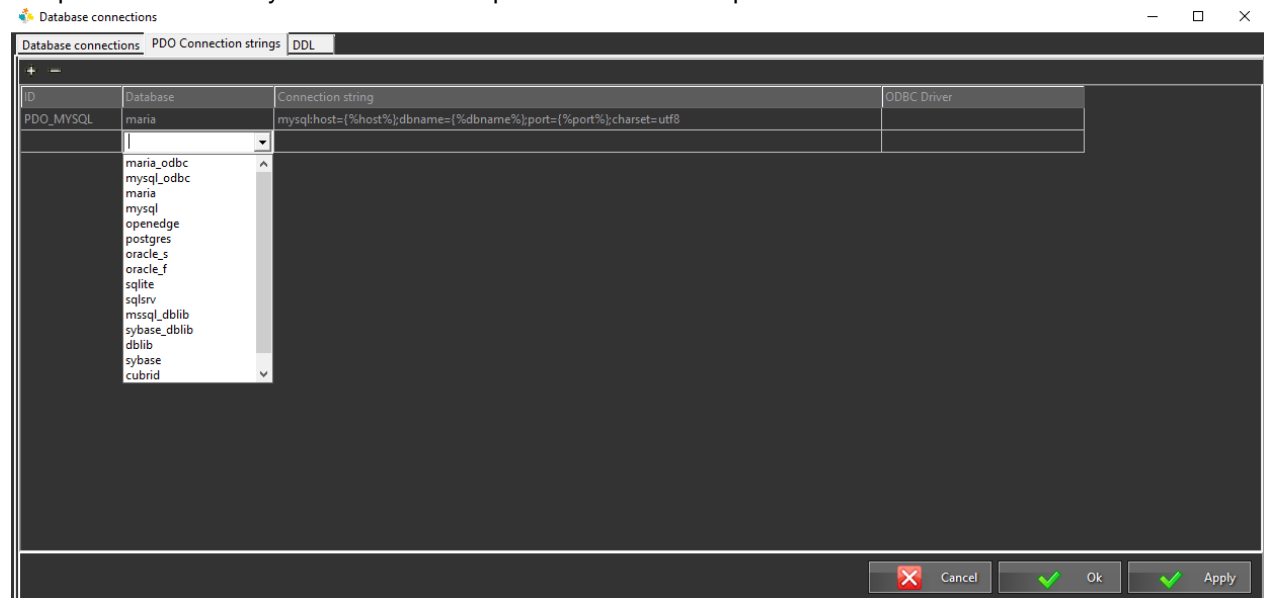
The database connection has the following parts.

- Connection data for the IDE.

The IDE needs to be able to retrieve metadata from your database.

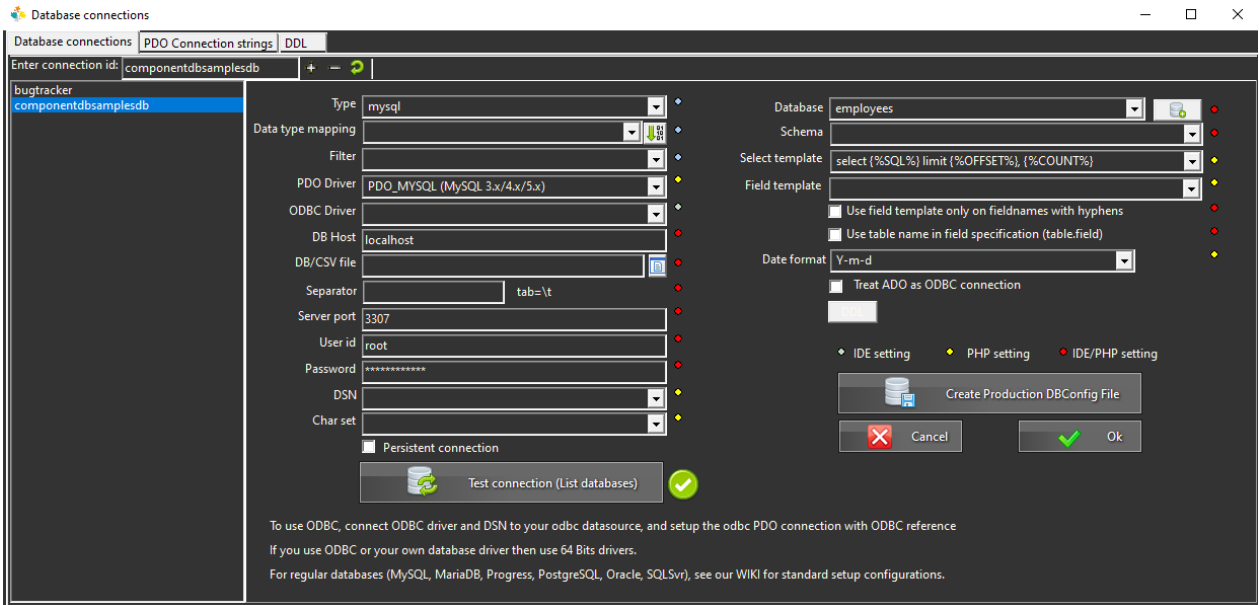
- Connection data for PHP

This consists of two parts. First you need to have a PDO connection available. PDO is a database independent access layer in PHP. To setup a PDO connection pick the PDO tab:



Add A PDO name in the first column, select the database to use, and PHsPeed will add a default connection string for you to modify if needed. If you are using Oracle, SQLServer etc, then it is possible that the installation requires different settings. The PDO settings here are based upon a local installation of your database.

How to setup a database connection is described in the manual. Basic setup for the different database flavors can be found there as well. In short (for a MariaDB/MySQL connection)

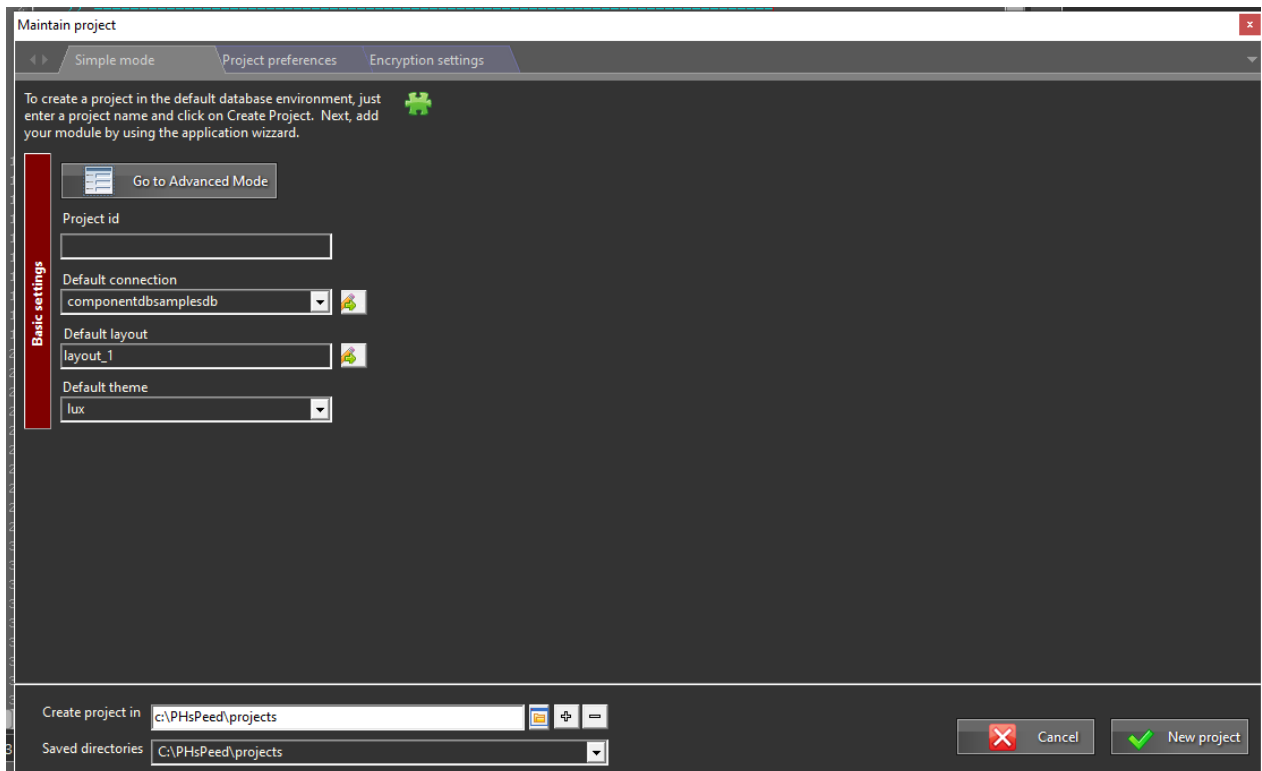


Apply a connection name and click on +. Enter the required info in the first column and then test the connection. The database drop down field will contain the databases of the connection. Select the desired one. On other databases you must select the applicable PDO connection.

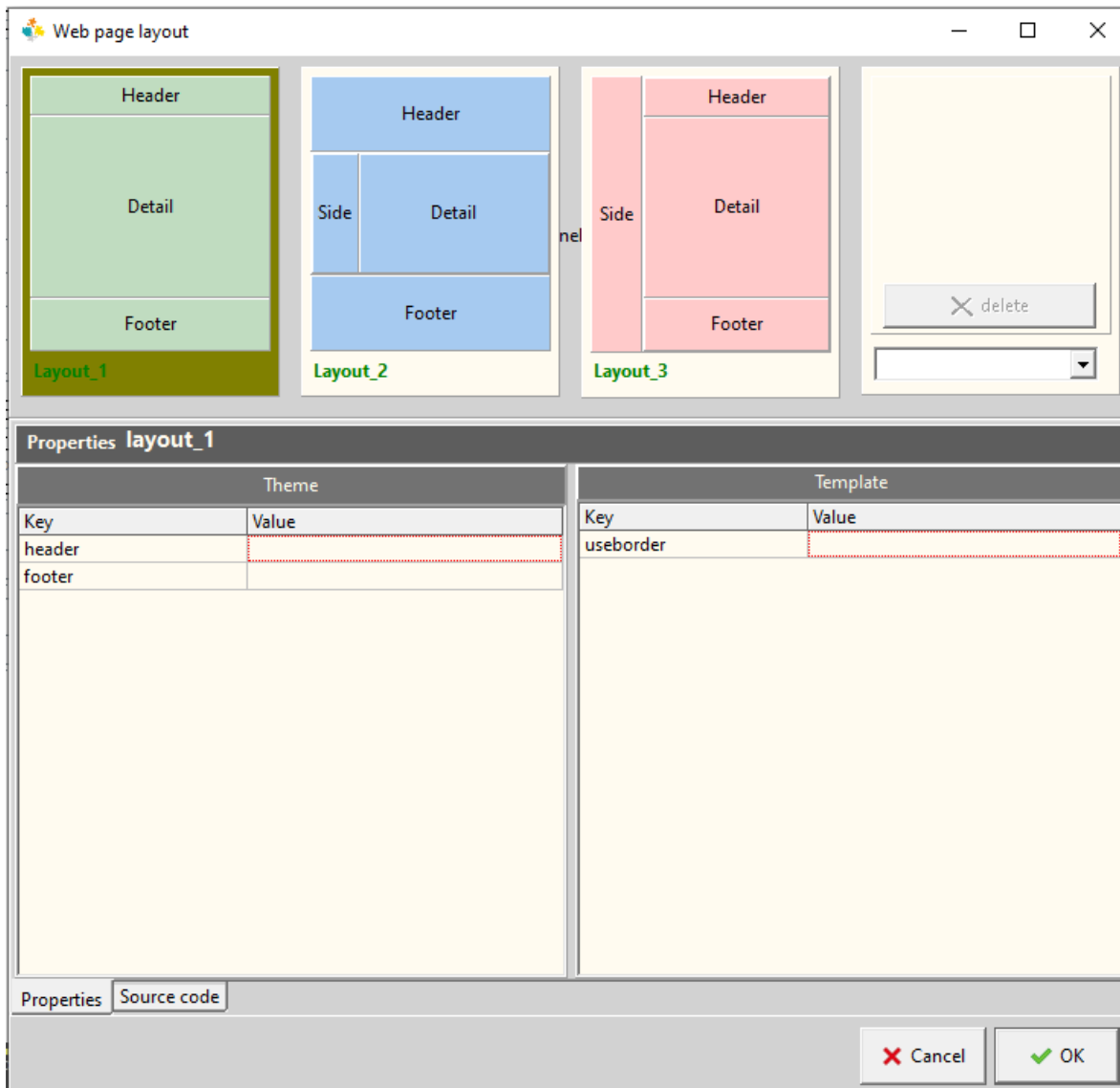
## Create project

The next step is to create your project.

The project can be created in a simplified mode where you only need to provide a project id, the default connection to use, which layout and which theme. By default PHsPeed will create an empty landing page called main.php, that you can use to convert into a login page.



The layout is standard 'layout\_1', but you can choose another:

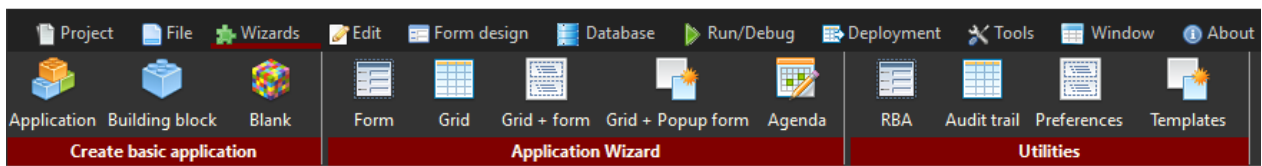


The header and footer are created later in the project (if you want to make use of them of course).

## Create module

Now you have created a project, you have an initial web page that can be used to start your application. In this course we will leave this page alone for now as we will use it later for our login page.

We will start by showing you how to create modules in your project. There are a few different ways, you can add a new 'free' module. This is comparable to your main page that has been created when you created the project. But as you most likely want to create crud applications and reports there is a faster way to get started, using wizards.



## Creating basic applications

A basic application is actually a blank page where you can put your components to build your form from scratch:

**Application:** This is a web page with direct interaction with the end-user. It is a 'run-able' application.

**Building Block:** This is a reusable module which can be used in applications. They cannot run independent.

**For PHP developers:** a building block is a 'PHP class'.

**Blank:** This is a web application without a form, and can be used for implementing API's or landing pages for external instances. I.e. return pages from payment providers.

## Application Wizards

Application wizards allows you to generate basic applications without writing code. You assign a database table, the type of application you want and PHsPeed will generate a working application for you. You find the wizards by selecting the item from the top menu or by clicking on the 'apps' item in the project manager.

### Form

This is a standard crud page with a toolbar to maintain the records of the defined dataset (table)

### Grid

This is a standard report of data from a table in Grid form. It is possible to make an editable grid.

### Grid+Form

This is a grid with an editable form. If you click in a row, this data will be shown in the form for crud operations.

### Grid+PopupForm

This is a grid with a popup form. You can click on an icon to popup a crud form, or have direct actions like update or delete a record.

### Agenda

This is a calendar application, we will not describe this application type in this basic level 0 manual.

## Form

The screenshot shows the 'Create...' wizard interface for generating a Form application. The interface is divided into several sections:

- Top Navigation:** A horizontal bar with icons for 'Form', 'Grid view', 'Form with Grid', 'Grid with popup form', 'Agenda', and 'Bulk'. The 'Form' option is selected.
- Basics Section:**
  - Module id:** A dropdown menu.
  - Sequence:** A dropdown menu.
  - Connection:** A dropdown menu with 'componentdbsamplesdb' selected.
  - Primary table (master):** A dropdown menu.
  - Primary index:** A dropdown menu.
  - Tables:** Two checkboxes: 'Default, select all fields' (checked) and 'Editable grid' (unchecked).
- Responsive Design Section:**
  - Number of columns:** A dropdown menu set to '2'.
  - Device Classes:** A list of checkboxes for different device classes:
    - Small devices: Extra small <576px (col-) DeviceClassMobile
    - Phone: Small ≥576px (col-sm-) DeviceClassPhone
    - Tablet: Medium ≥768px (col-md-) DeviceClassTablet (checked)
    - Laptop: Large ≥992px (col-lg-) DeviceClassDesktop
    - Desktop: Extra Large ≥1200px (col-xl-) DeviceClassDesktopXL
- Crud and Icons Section:**
  - Crud type:** A dropdown menu set to 'crud (ignore form initial mode)'.
  - Edit icon:** A dropdown menu with 'fas fa-pen' and 'dark' selected.
  - Icon position:** A dropdown menu set to 'front'.
  - Delete icon:** A dropdown menu with 'fas fa-trash' and 'dark' selected.
- Bottom Section:**
  - Buttons for 'Use DBTable' and 'Use DBQuery'.
  - Navigation buttons: 'Previous', 'Create module', and 'Next'.

The wizard has a number of forms that you can follow to define the application that you want. In this sample we show you the most basic form and the fastest way to generate a crud form:

### Module id

A unique name of the module to create. You can use an existing module but then the application will be appended (handy for master-detail relationship forms that will be explained in other course-ware).

### Connection

By default this element will be filled by the connection that is defined in your project. It is possible to make more connections to different databases (of different brands), so if necessary, you need to select the correct one.

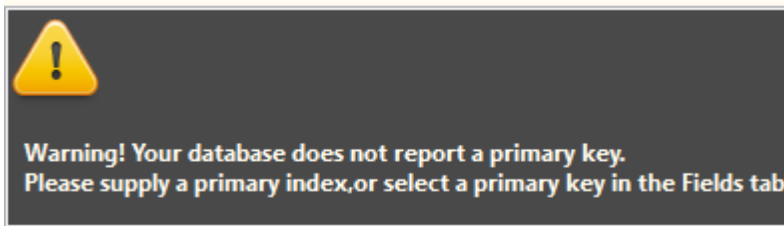
### Primary table

The table from the database you have selected

(Primary index is rarely used, you can leave this empty)

Then click 'create module'.

Important! If you generate a crud application, then you must have a defined primary key. If not you can get a warning:



But if all goes well, then the module will be created.

Select applicationtype to generate

Form Grid view Form with Grid Grid with popup form Agenda Bulk

Basics

Module id departments

Sequence

Connection componentdbsamplesdb

Primary table (master)

Primary index current\_dept\_emp

Tables departments dept\_emp dept\_emp\_latest\_date dept\_manager employees salaries titles

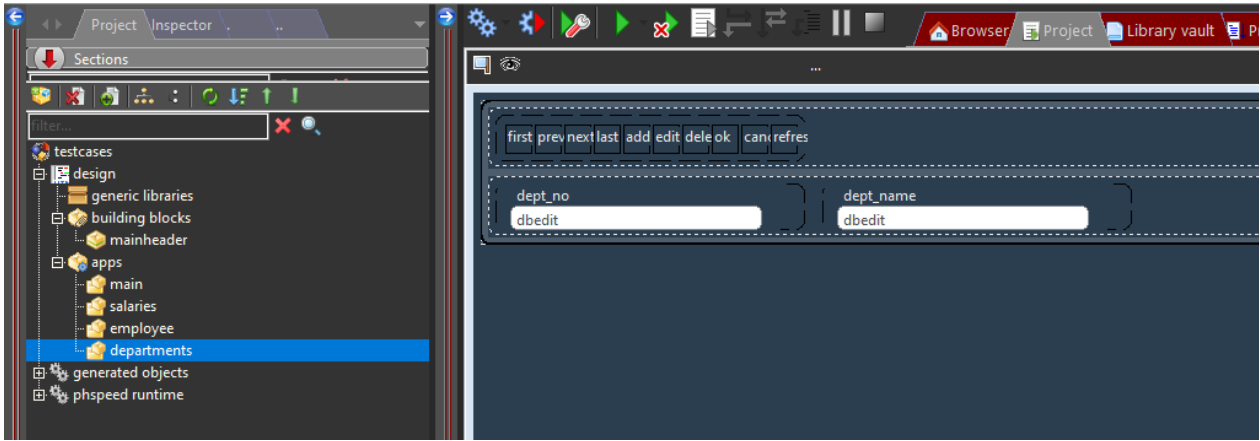
Number of columns

Phone: Small  $\geq 576$ px (col-sm-) DeviceClassPhone

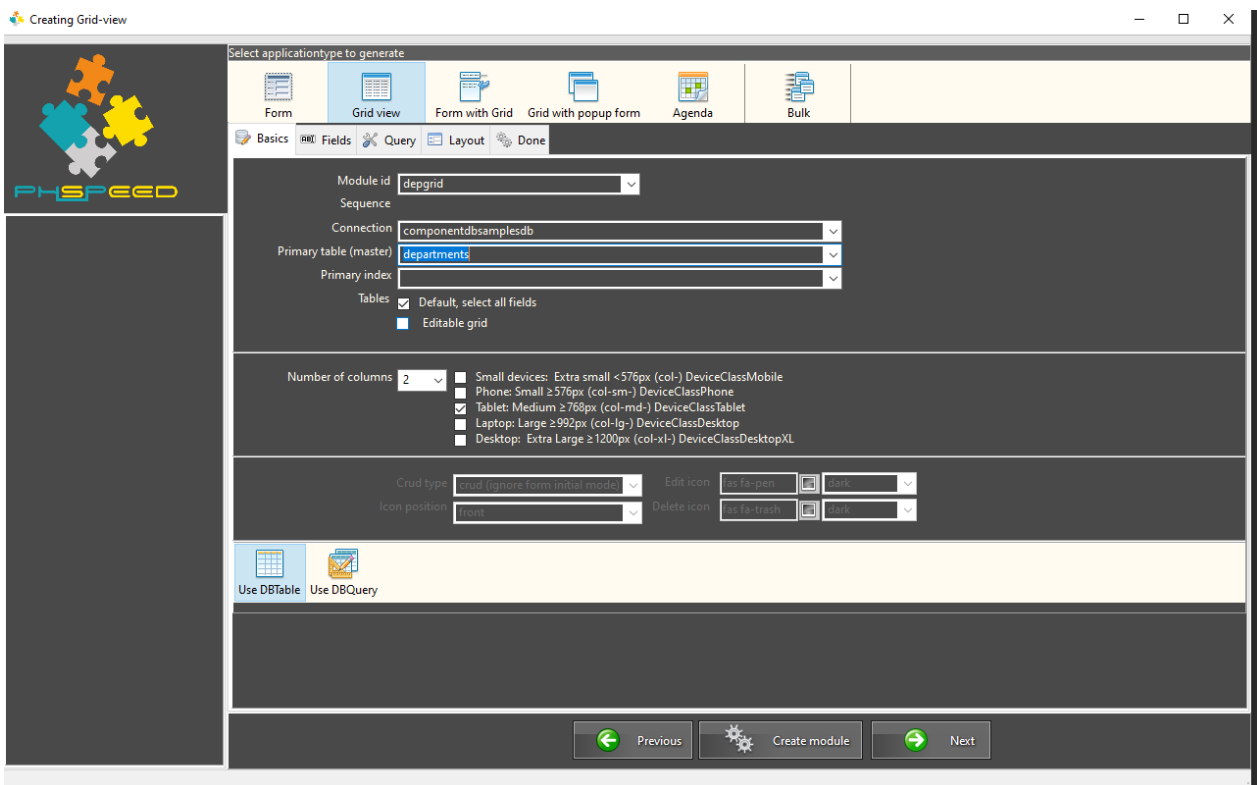
Tablet: Medium  $\geq 768$ px (col-md-) DeviceClassTablet

Laptop: Large  $\geq 992$ px (col-lg-) DeviceClassDesktop

Desktop: Extra Large  $\geq 1200$ px (col-xl-) DeviceClassDesktopXL



## Grid



The wizard has a number of forms that you can follow to define the application that you want. In this sample we show you the most basic form and the fastest way to generate a grid.

### Module id

A unique name of the module to create. You can use an existing module but then the application will be appended (handy for master-detail relationship forms that will be explained in other course-ware).

### Connection

By default this element will be filled by the connection that is defined in your project. It is possible to make more connections to different databases (of different brands), so if necessary, you need to select the correct one.

### Primary table

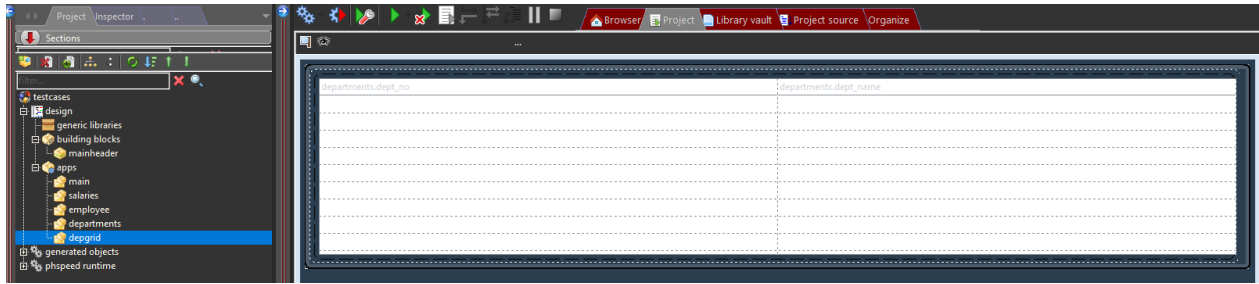
The table from the database you have selected

### Editable grid

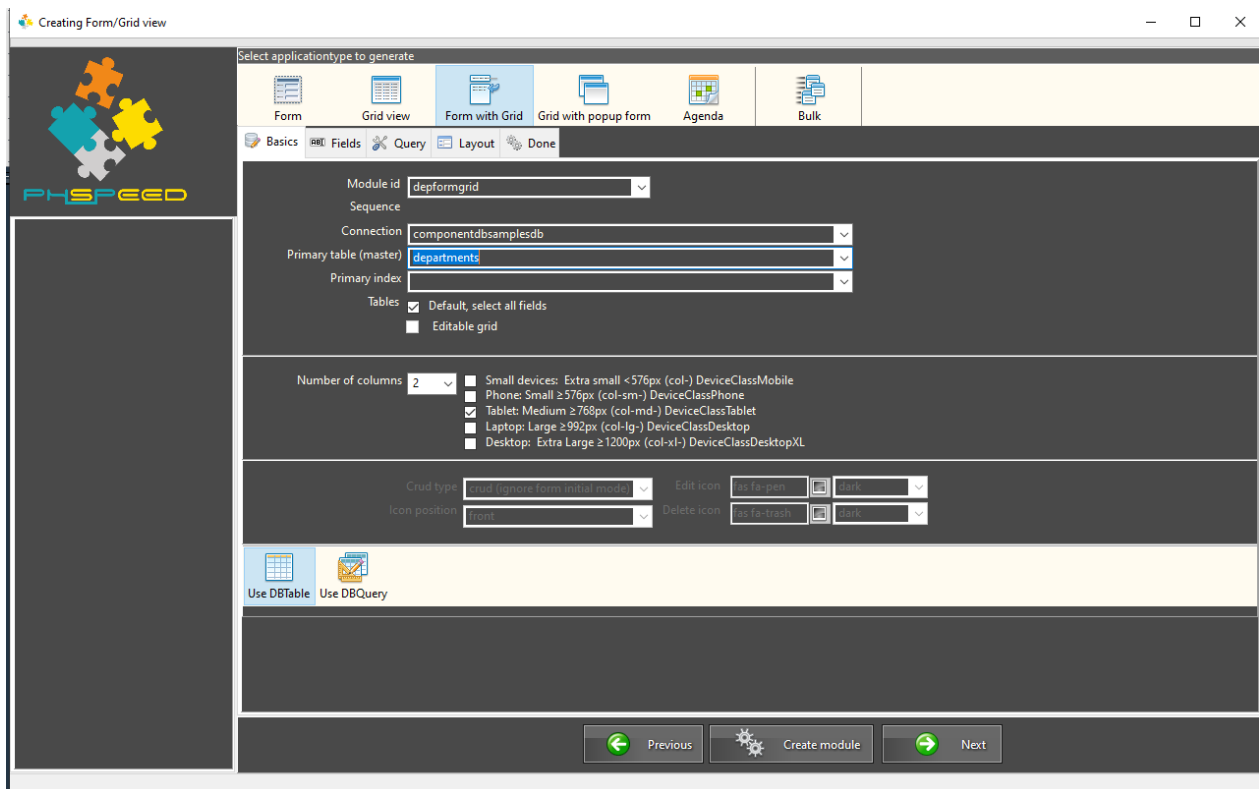
If you set this checkbox then the grid will be made 'editable', meaning that you can bulk-edit records.

(Primary index is rarely used, you can leave this empty)

Then click 'create module'.



## Grid+ Form



The wizard has a number of forms that you can follow to define the application that you want. In this sample we show you the most basic form and the fastest way to generate a crud form within a grid

### Module id

A unique name of the module to create. You can use an existing module but then the application will be appended (handy for master-detail relationship forms that will be explained in other course-ware).

### Connection

By default this element will be filled by the connection that is defined in your project. It is possible to make more connections to different databases (of different brands), so if necessary, you need to select the correct one.

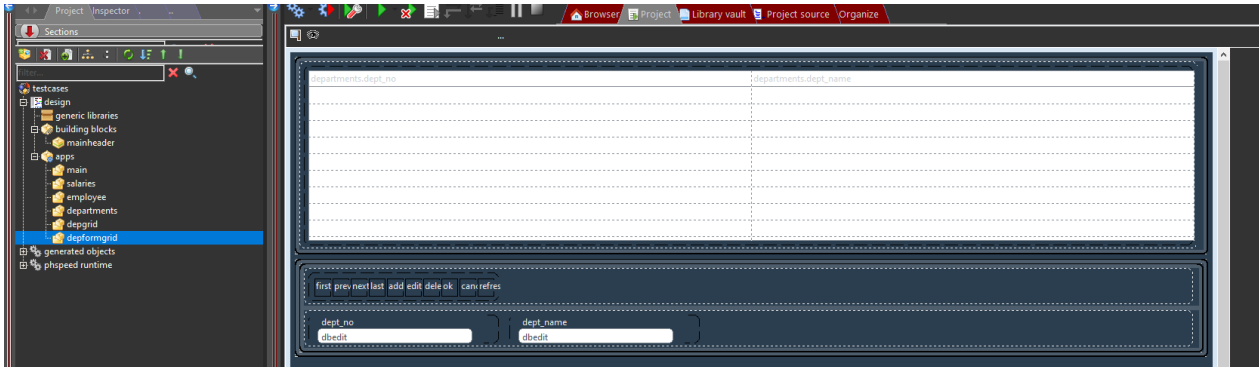
### Primary table

The table from the database you have selected

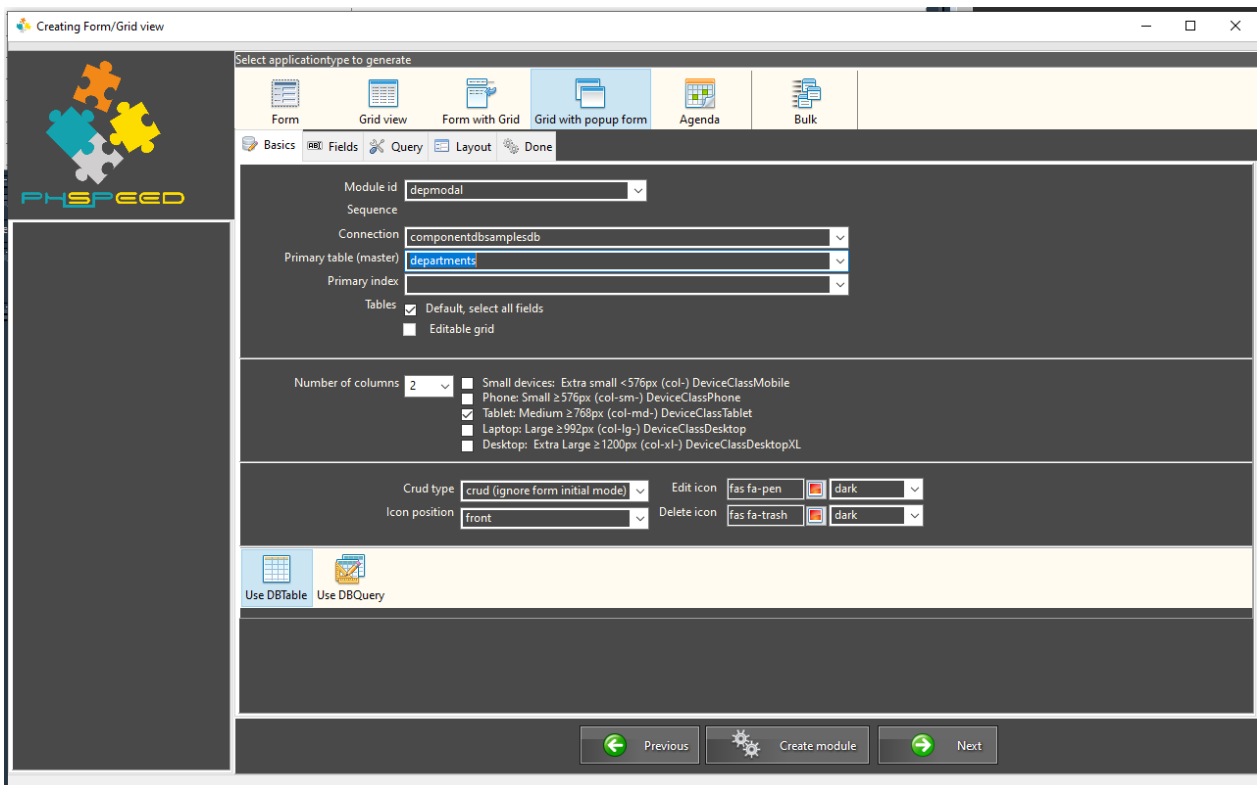


(Primary index is rarely used, you can leave this empty)

Then click 'create module'.



## Grid + popup Form



The wizard has a number of forms that you can follow to define the application that you want. In this sample we show you the most basic form and the fastest way to generate a popup crud form, based on a grid.

### Module id

A unique name of the module to create. You can use an existing module but then the application will be appended (handy for master-detail relationship forms that will be explained in other course-ware).

### Connection

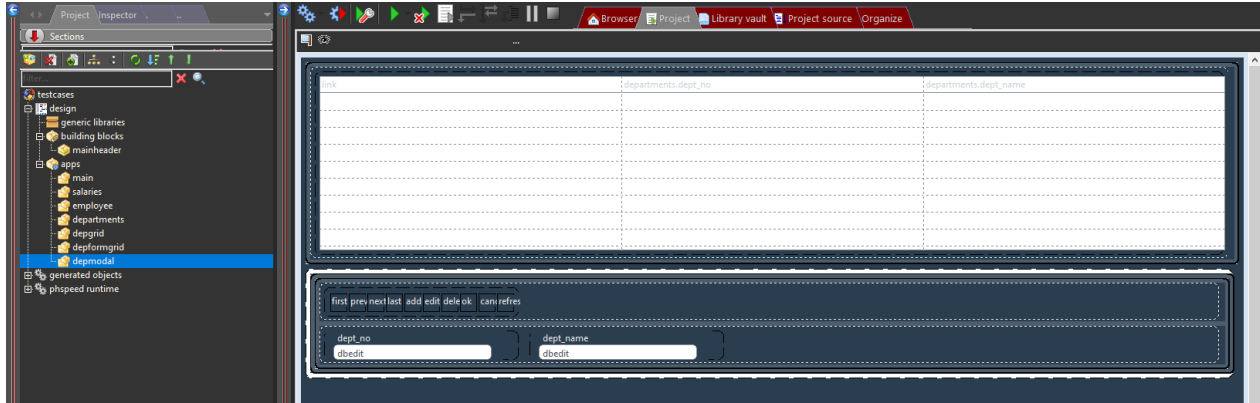
By default this element will be filled by the connection that is defined in your project. It is possible to make more connections to different databases (of different brands), so if necessary, you need to select the correct one.

## Primary table

The table from the database you have selected

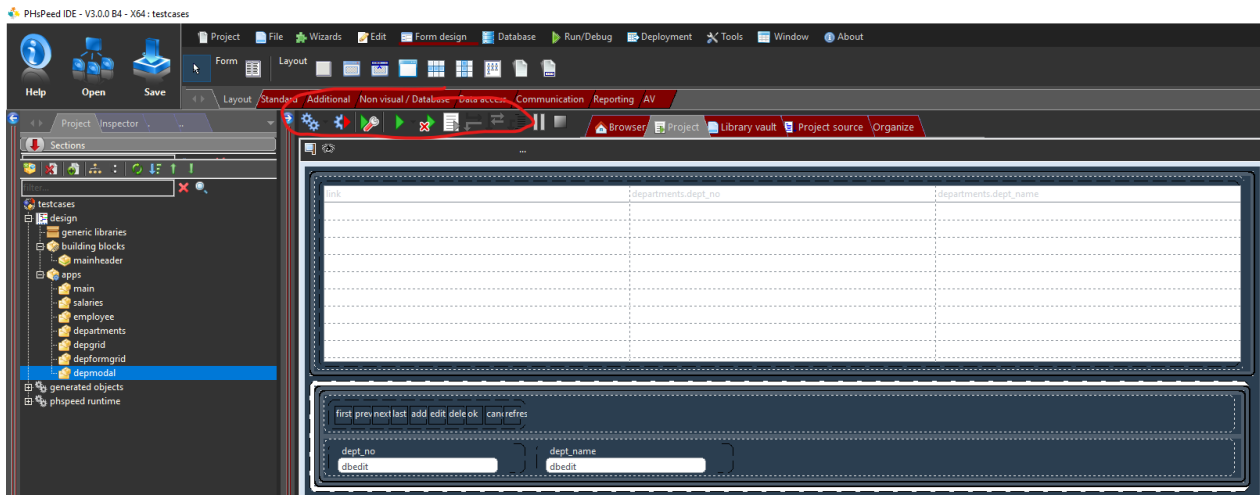
(Primary index is rarely used, you can leave this empty)

Then click 'create module'.

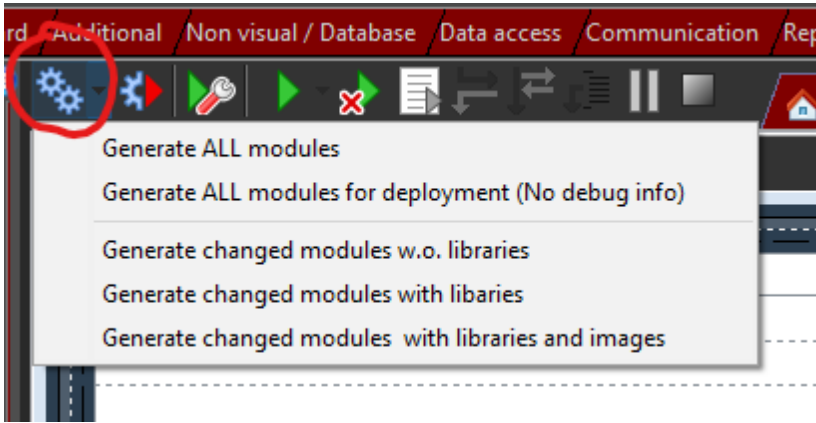


## Generate code

If you create a module, then PHsPeed will create a form with basic functionality. It is very likely that you want to generate code and review the module as soon as you have created one, modify it until done and then start with the next one. Another approach is that you create a basic prototype, adjust it with your customer and then apply your own code to finish the application. Either way, you need to generate code, in order to let your application work.



There are several ways to generate code. The first button allows you to generate full code.

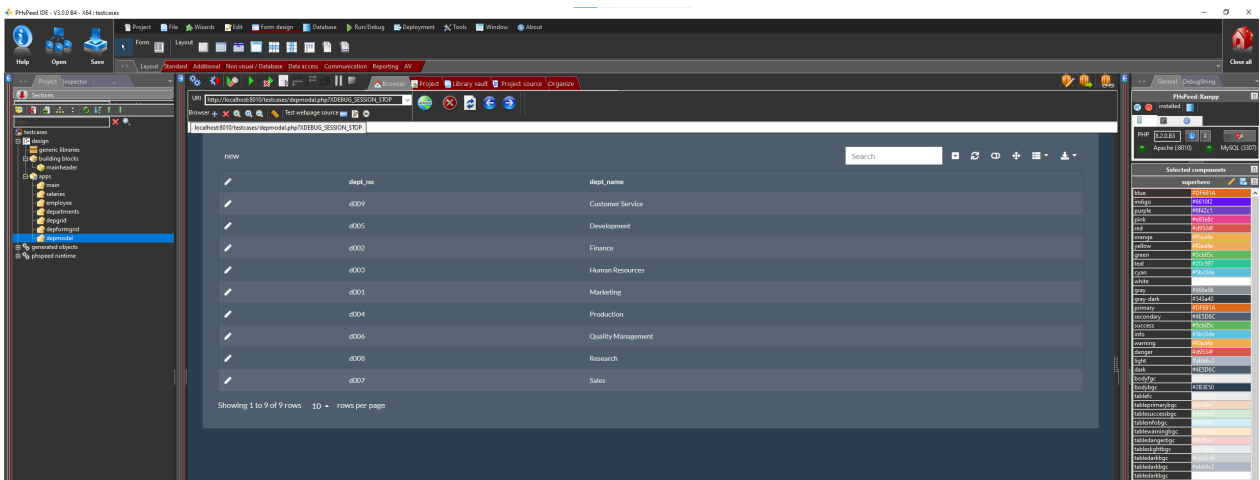



The first option will generate code of all the modules that are in your project. Although it is a fast process, usually you want to generate the code of the module that you are working on. However, when you load a new project, copy it from someone else, you initially need to generate the full code in order to have headers and footers to work etc. After that you can generate the code that you have been working on.

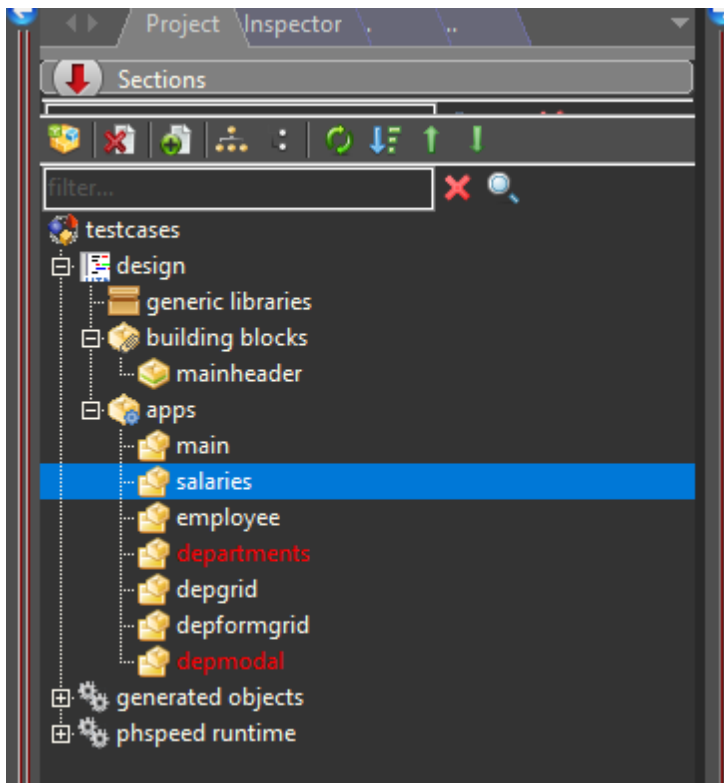
That is done by the second icon.



If you use this button then the system will generate the code and immediately will run it in the internal webbrowser. It will generate and run the code of the *selected* module, in above sample, depmodal:



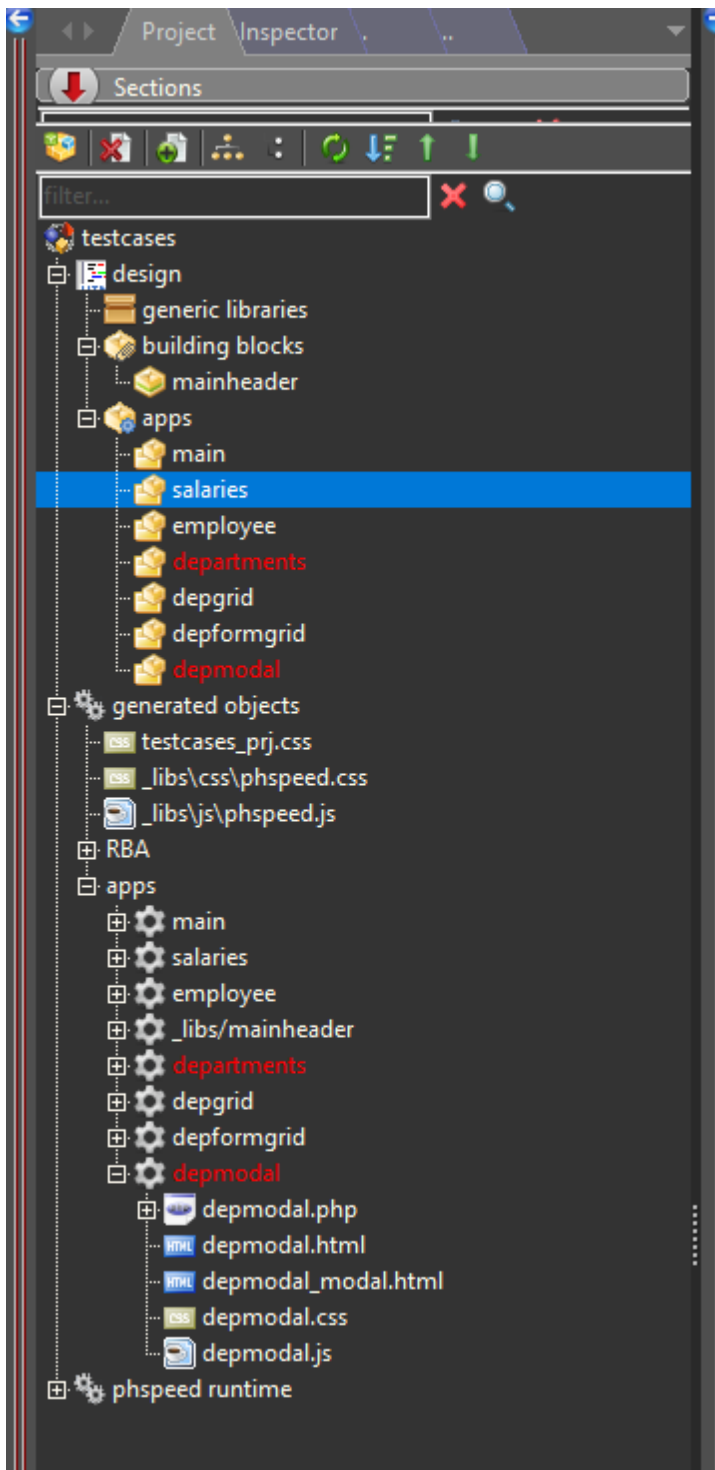
A third option is to use the  button. In this case, the system will generate code for all the changed modules. These are recognized by a difference in color.

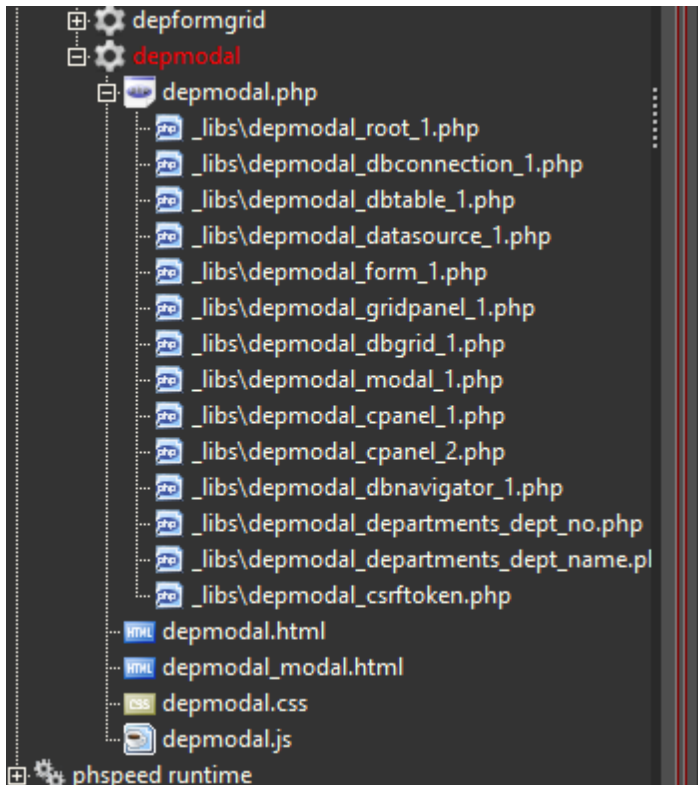


In this case salaries, departments and depmodal will be generated.

[Locate the generated code.](#)

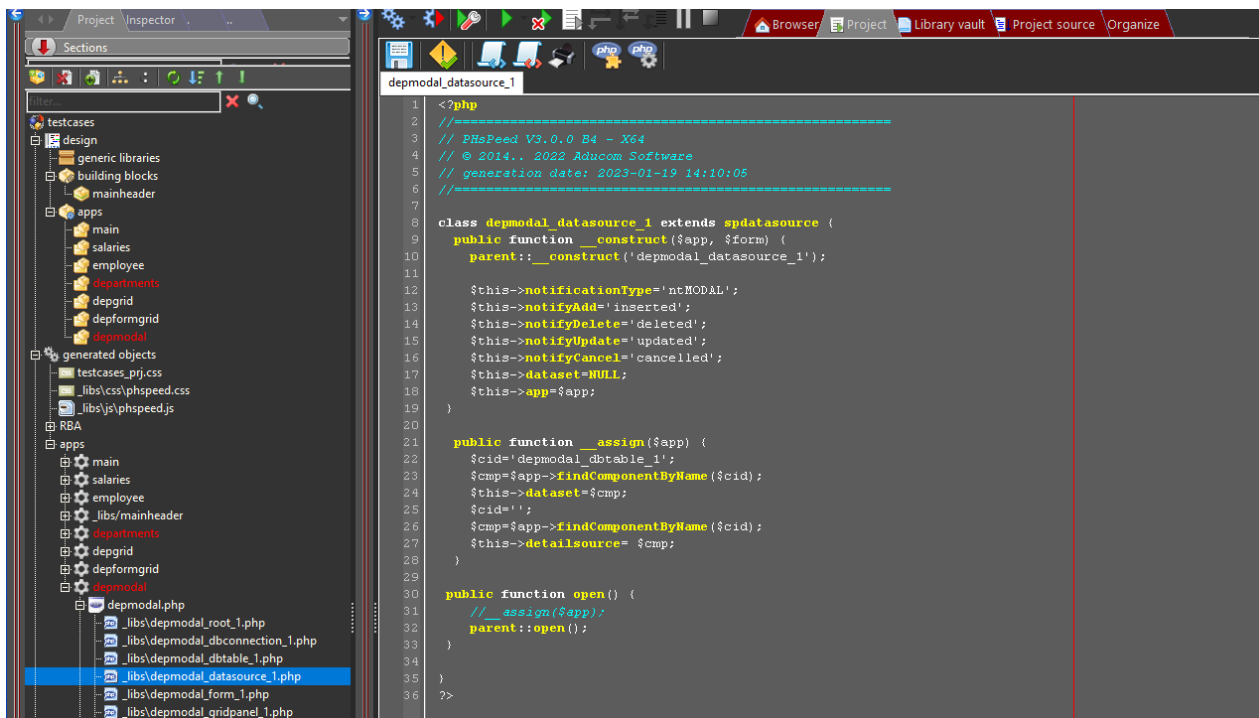
When the system generates the code, you can access this, using the project manager.





Under 'apps' you will find the generated code for your applications, if you open the app, then you will find the separate parts. In this basic training we will not talk too deeply about the structure of the application itself, but there are some things that you need to know.

The main application is the application that is 'runnable'. Here depmodal.php. Below this module, you will see the generated HTML templates, the style-sheet and java-script that are required as part of the application. If you open depmodal.php, then you will see the underlying PHP modules. In fact, you will find a PHP module for each component on your form. If you are not a developer, then you will not need to look at this much, but if you are, then this is the way to access your code. To view it, click on the module:




Each component is generated by extending a basic PHsPeed class. If you apply code to an event then you will find this code as part of the component you have used the event of.

## Run the code

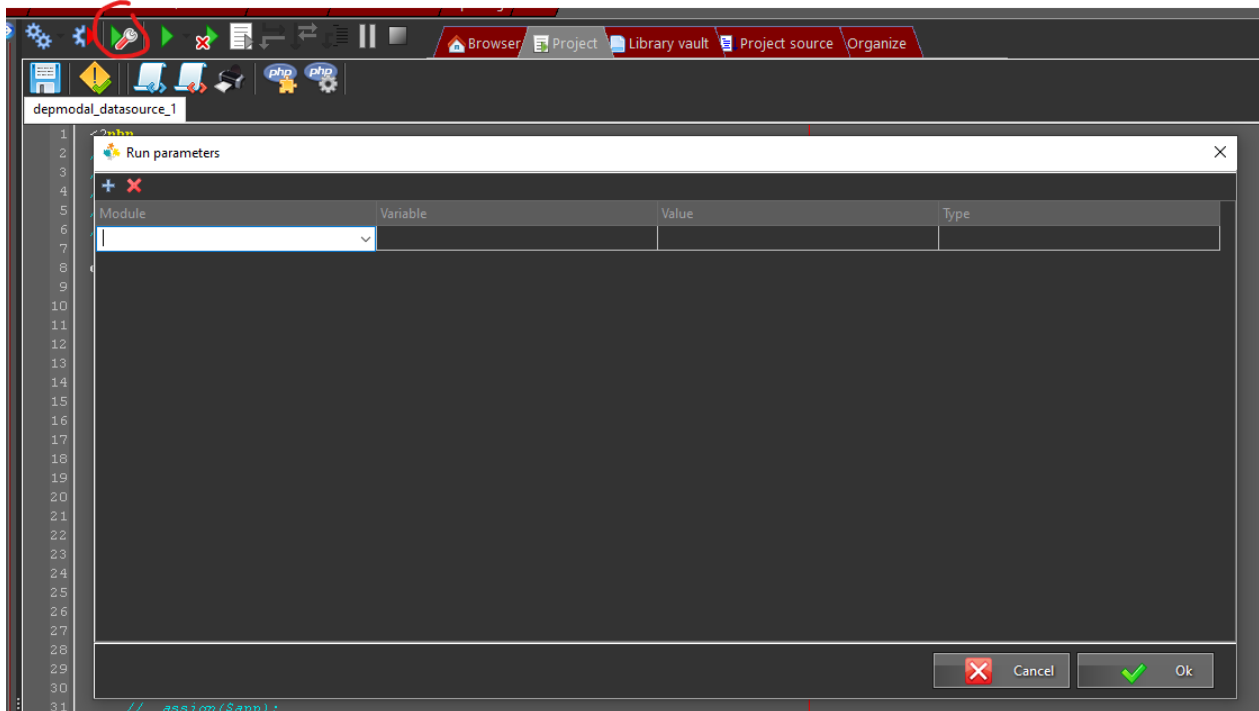
### ***Run without generating (new) code***

You can run the code by the buttons described in the previous chapter. Usually the buttons will also generate code before execution of the program starts. Sometimes that is not what you want. Suppose you want to try something out before putting that in the real code. In that case you change the generated code. However, as soon as PHsPeed generates code it will overwrite your changes. So if you need to run the

code without generating, use the  button.

### ***Run with parameters***

If you write an API or need to read data from the url in get or post mode, you must be able to test this. To start a program with preset values you can use the 'set parameters' button.



If you run the application then the variables will be supplied automatically (`$_GET`). If you have declared a variable as 'Post' then PHsPeed will popup a form to enter the values as the application will then receive data as `$_POST`.

## Debug code

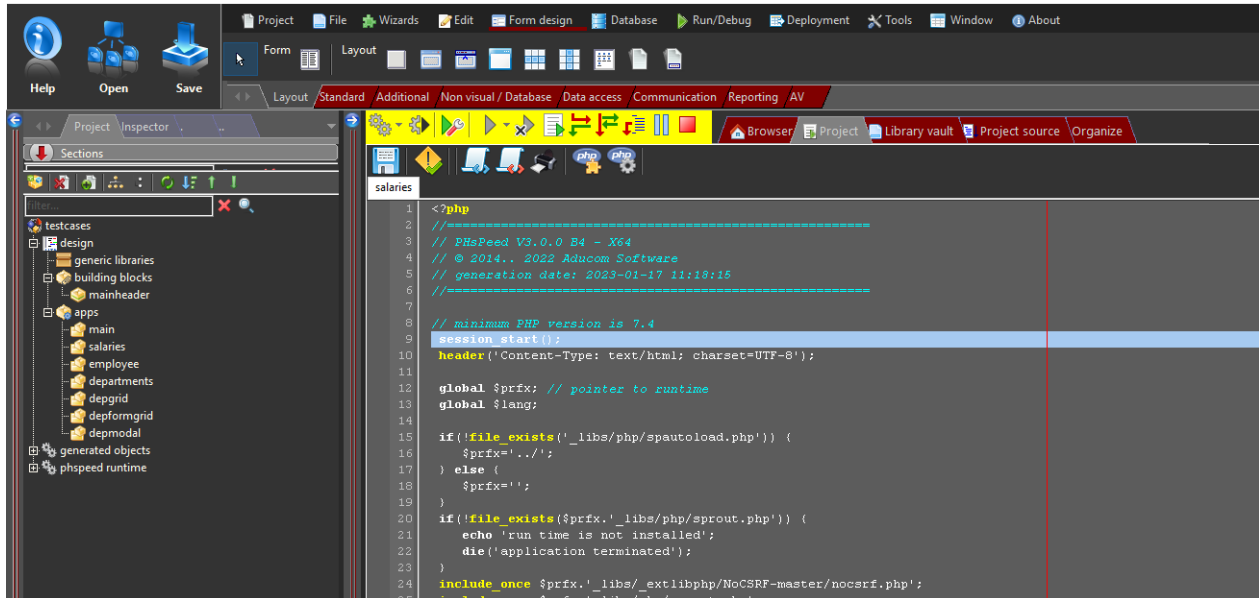
In PHsPeed you can debug your code. To be able to do so, you must set the IDE in 'debug mode'.



For PHP use



If you have the application in debug mode, and run the application, then the application will stop at the first executable line:



You can now choose to continue the application by stepping through the code (F7) or (F8), or to continue (F9). If you have set breakpoints then the application will stop when it reaches that breakpoint. You can watch data, evaluate, look at the used json objects etc. Very handy for developers. The difference of F7 vs F8 is that F8 will jump over functions, where F7 will jump into functions. Of course you can also use the (yellow) toolbar and the mouse if you prefer that.

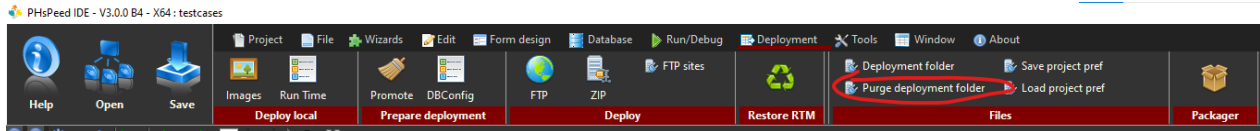
## Finalize development

While developing your application, there is a stage where you want to add headers, footers, menus etc. The advise is to wait with this until you are pretty sure that the structure of your application is not going to change too much. Especially when you create menus, apply RBA, etc., it is inconvenient if you have to modify your application. Suppose that you have an application structure, applied, users, usergroups, applications, applicationgroups and then change structure, then you might have to redo a lot of your work. It is evident that you have to change your code over time as you will probably add functionality which need to be applied in your application, but then you are in a maintenance state. During initial development, you might develop new ideas that you want to try out, some modules are deleted and perhaps replaced, etc. All that will cause you to make changes to your menu structure and/or configuration.

After a while developing your application, you will find out that there are modules in your web environment that are removed from your development. The reason is that if you generate code, PHsPeed will generate this in your local web environment. If you remove a module from your development, then the generated code in your web environment is not removed automatically. This can lead to orphan files or other 'junk' that you don't want to move to your production environment. For that reason it is a good practice to purge your deployment environment, deploy your images and regenerate your application fully from time to time, at least before you move the project to your acceptance environment. This will remove all leftovers and avoid potential security risks of users accessing modules that were created as a 'trial' or other reason.

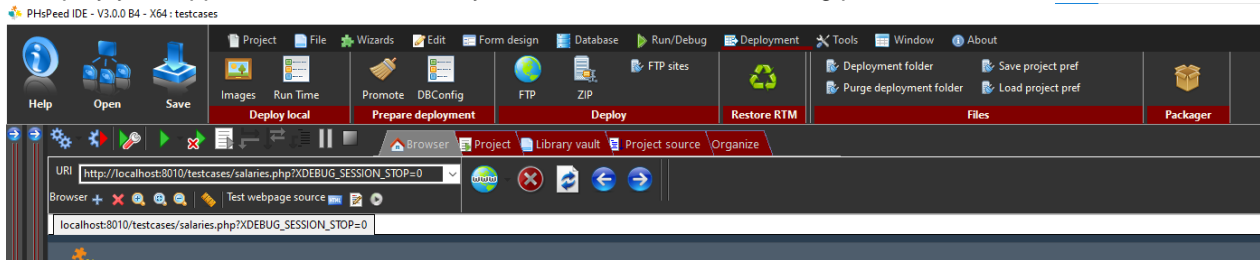
To do this use the 'purge deployment folder' option. Then follow the deployment procedure.





## Deploy application

When your application is done, you want to run in a real life situation. That will be a separate server, outside your development environment, usually a server that runs a separate webserver and database. So you need to deploy your application. To be able to you need to follow the following procedure:



Purge your local deploy environment as described in the previous chapter  
This will prevent orphan files and unnecessary security risks.

## Deploy the images of your project

While you are developing you have to manually deploy your images. This is implemented this way as you only need to deploy the images once. But while developing, you might have changed icons, images and this will remove all unused images.

## Do NOT use the Run Time button

PHsPeed has the ability to use a private runtime library. This is beyond the purpose of this courseware.

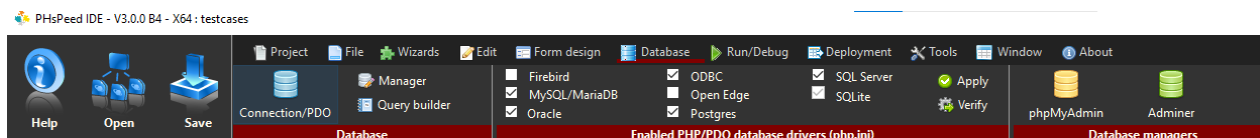
## Promote your code for production

To be able to debug your code, PHsPeed adds code to your project to make this possible. Promoting code for production will remove this code from your sources, and also removes remarks, empty lines etc. to make your code smaller and faster. After all, the code will run in a separate environment. The Shared Runtime is also promoted.

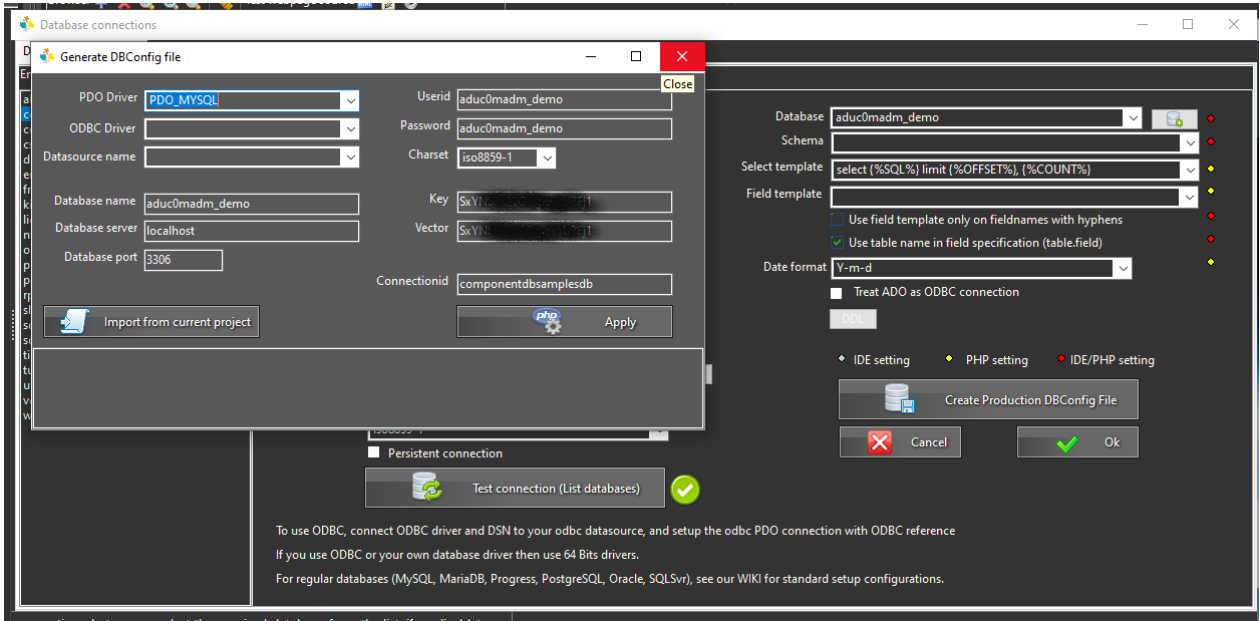
## Create a database config file.

It is very likely that the database of the environment where you deploy your application has different credentials. PHsPeed makes use of an encrypted config file that contains this information. There are several ways to create this file, but the DBConfig allows to create a file that can be deployed with your application. The other ways (like external logons as Saml, Ldap, or a configuration program to create the file dynamically, will be discussed in other courseware.).

To create the config file go to Database -> Connection/PDO, and on the connection screen choose Create Production DBConfig File. Other option is to use the DBConfig button on the toolbar that you find under the tab Deployment.



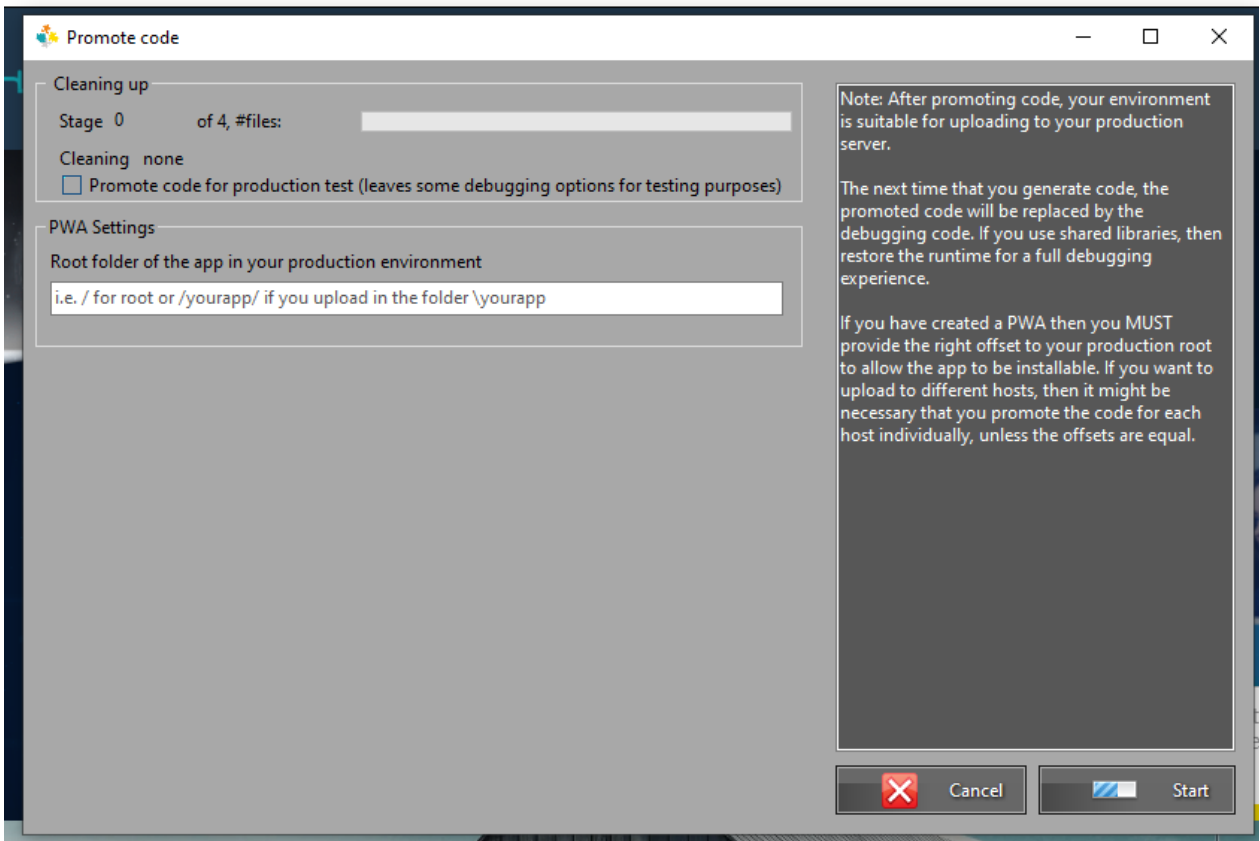
This will open the config setup:



The key and vector are set by default from the project options. When you click 'Apply' the config file is created in the project.

**Promote the code for production.**

This step is required to strip the code from debugging code, comments, empty lines, but it also prepares your database connection for production. While developing the properties of the connection component is used. But these properties are not allowed in a production environment.



Do not enter anything in the PWA root folder. This is only required if you create a Progressive Web Application ('a web application that behaves like an 'app)'). Click on start. PHsPeed will process all files

**including** the runtime. When done, you are ready to upload your project.

If you setup a database user in your development environment then you can test locally if the application works. Otherwise you need to upload first and then start your main application.

Note: This step is required for your first upload to your acceptance environment, if you have one. It is not uncommon to have end users test your application before you put it into production. As the production environment is a copy of your acceptance environment you can copy the folder or do a new upload. It is always possible to use the IDE to generate a new database config file for your production server if the database has different credentials for acceptance and production.

## Initial test / acceptance

To upload your project you can use a tool like FileZilla, or use the build-in FTP client. PHsPeed currently does not support SFTP or FTPS, so if you need that use a tool like FileZilla. This will also work faster.

You do not have to upload the full project if you change database credentials. The config file is created in the `htdocs\project\_libs\config`. If you upload this folder to the `_libs\folder` of your acceptance or production server then the system should work. If you have forgotten to create the config file then you can always create it later and upload.

## Production

In general copy the full project folder of your acceptance environment to your production server. Change the `dbconfig` if required.

## Evaluate

---

When you want to restart development or want to make changes to your project as you might have find issues in your acceptance environment you have to undo the promotion of your runtime code. Under deployment you will find the button 'Restore RTM':



This button will restore your runtime to its original state.

Next you need to fully regenerate your project to allow PHsPeed to put the debug code back in and reset the production state of your project.