Tutorial: Inheritance of xoopsForm - XOOPS

NEWS_PDF_AUTHOR: Mamba

NEWS_PDF_DATE: 2012/3/7 2:10:00

French XOOPS user br_750 *recently published a* nice article *about inheritance of* ***xoopsForm*** *class. Here is the English translation:*

What is inheritance?

Inheritance is a concept specific to Object-Oriented Programming (OOP), to create a new class based on an existing class. The new class "inherits" the properties and methods of the class it inherits from, called "parent" class. The inheriting class is called a "child" class.

This definition is very short and you can enhance your knowledge of object-oriented programming on the Web:

- Wikipedia - French article with nice UML images - Tutorial for PHP - PHP docs **XOOPS and Inheritance** Like any good IT project, XOOPS uses OOP. Indeed its developers (whom I thank by the way) wrote a set of classes that govern this project. So we can use inheritance to add and modify the behavior of XOOPS. **All this is fine, but why?** - you can modify any classes directly in the project. These changes are called " HACKS ". At the very moment you enter your code, it might become a grain of sand that might undermine your life forcing you to constant development and maintenance of your site. And what will happen in the next release arriving in few days? In a year? Nothing is going to run correctly, you'll have to start all changes one by one. Let's hope that as a competent developer, you have recorded all changes somewhere to be able to repeat them again, if needed. Otherwise I wish you good luck! **The alternative is to use inheritance, let's go!** Consider an example with a form in XOOPS. Forms in XOOPS, though often enough, can not always meet all your needs. Here we will see two applications of inheritance with XOOPS and more specifically with **xoopsThemeForm** and **xoopsElement**. These are only examples and you're not limited to them: - change the URL of the form's action on the fly - add a new item, not available in the original project We assume that we have a module called MyModule, and that this module has a standard directory for classes in which we can place our new classes. **The first thing is to understand how classes are structured in form inheritance tree.** XoopsForm class that is the base class of the form, is an abstract class, which means that this class should not be used directly, but must be extended and generalized. This is done in a kind of template, properties and methods. I let you immerse yourself in OOP for more details. XoopsThemeForm class is the class that extends XoopsForm and that'll display the form as a table. It is this a class which we inherit, to change the general behavior of the form. **Note:** if we want our form to be no longer displayed in tabular form, we would inherit directly from XoopsForm and rework our source code; For here, it will be enough to just inherit from xoopsThemeForm and change the render() method which is responsible for creating the form. As you may have noticed, in XoopsForm (see API) the *render()* method is empty, and

that's correct. This is an abstract class, and it just tells us that this method should be implemented in the class that extends it. So in XoopsThemeForm the method is implemented to display the form as a table:

```
 function render()
{
    $ele_name = $this->getName();
    $ret      = '

            ' . $this->getTitle() . '

        ';
    $hidden   = '';
    $class    = 'even';
    foreach ($this->getElements() as $ele) {
        if (!is_object($ele)) {
            $ret .= $ele;
        } else {
            if (!$ele->isHidden()) {
                if (!$ele->getNocolspan()) {
                    $ret .= '';
                    if (($caption = $ele->getCaption()) != '') {
                        $ret
                            .=
                            '';
                        $ret .= '' . $caption . '';
                        $ret .= '*';
                        $ret .= '
';
                    }
                    if (($desc = $ele->getDescription()) != '') {
                        $ret .= '' . $desc . '
';
                    }
                    $ret .= '
                            ' . $ele->render() . '
';. NWLINE;
                } else {$ele->render() . '
            } else{{$ret .= '';
                $hidd(($caption = $ele->getCaption()) != '') {
            }           $ret
        }                   .=
                            '';
    }                   $ret .= '' . $caption . '';" . NWLINE;
    $ret .= $this->renderValidationJS(true);  $ret .= '
    return $ret;
};
```

Let's put it into practice In our first example we want our form to create a different view, depending on a user action. Our form is as follows:

```
// create form object
$my_form = new ThemeForm("my form", 'Form_bien', "handle_bien.php?action=save");

// create form elements
$reference = new XoopsFormText("Reference", "reference", 50, 100);

$prix= new XoopsFormText("Price","price",50,100);

// add elements to the form
$my_form->addElement($reference,true);
$my_form->addElement($price, true);

// create/add buttons
$button = new XoopsFormButton('', 'post', _SEND, 'submit');
$my_form->addElement($button);
// display the form
$my_form->display();
```

As you can see, so far this is not a rocket science! Now we will modify the behavior of XoopsThemeForm - as first, we'll create a PHP file in the directory class, which we call a very originally: my_form.php Now we have following inheritance tree: For this class to inherit from XoopsThemeForm, also called extending it, we write in PHP:

Of course for this to work requires that the "child" class has its "mother" class (or knows where to find it) , so we must include the class XoopsThemeForm:

You can also use *xoops_load*, which helps if you don't know in which file is the class declared :

This new method (also called "setter") changes the URL to the new property $nowAction if the function parameter is passed. **Important Note:** Unfortunately, the class xoopsForm, was not optimized to give a UX developers the ability to do what they want. To do so would have required that the properties of the class xoopsForm be "protected" and not "private" (I refer you to OOP) or that each property is associated with a "setter" and a "getter". I do not know the reasons for this absence. This will force us to redefine the function *render()*. This would not have been required if we had had "setter" or if the properties had been declared as "protected". The class with the function render() redefined, we replace $this->getAction by our new property $newAction, which gives:

Now we can use it as follows:

```
 $my_form=new MyForm(parameter);

If(condition){
// Now we change the action URL of the form
   $MyForm->setNewAction('test.php')
}
```

You can make further updates on your own risk ..... In the second example we want to add an element that does not exist in the classes of XOOPS, for example we want our form to display images previously uploaded to the server, informing the user. There is no element in the forms of XOOPS to display an image. So we will create it by extending the class XoopsFormElement, which is the class responsible for creating a form element, as formcolorpicker etc. This will create following inheritance tree: Class:

The method *setContent($content)* will allow us to insert content. Use:

```
 $planchePhoto=new myElement();
$planchePhoto->setContent('photos
');
```

Now that we've created a new form element, let's add it to our form:

```
 $my_form->addElement($planchePhoto);
```

With this we have added a new item to our form.

French XOOPS user br_750 *recently published a* nice article *about inheritance of* ***xoopsForm*** *class. Here is the English translation:*

What is inheritance?

Inheritance is a concept specific to Object-Oriented Programming (OOP), to create a new class based on an existing class. The new class "inherits" the properties and methods of the class it inherits from, called "parent" class. The inheriting class is called a "child" class.

This definition is very short and you can enhance your knowledge of object-oriented programming on the Web:

- Wikipedia - French article with nice UML images - Tutorial for PHP - PHP docs **XOOPS and Inheritance** Like any good IT project, XOOPS uses OOP. Indeed its developers (whom I thank by the way) wrote a set of classes that govern this project. So we can use inheritance to add and modify the behavior of XOOPS. **All this is fine, but why?** - you can modify any classes directly in the project. These changes are called " HACKS ". At the very moment you enter your code, it might become a grain of sand that might undermine your life forcing you to constant development and maintenance of your site. And what will happen in the next release arriving in few days? In a year? Nothing is going to run correctly, you'll have to start all changes one by one. Let's hope that as a competent developer, you have recorded all changes somewhere to be able to repeat them again, if needed. Otherwise I wish you good luck! **The alternative is to use inheritance, let's go!** Consider an example with a form in XOOPS. Forms in XOOPS, though often enough, can not always meet all your needs. Here we will see two applications of inheritance with XOOPS and more specifically with **xoopsThemeForm** and **xoopsElement**. These are only examples and you're not limited to them: - change the URL of the form's action on the fly - add a new item, not available in the original project We assume that we have a module called MyModule, and that this module has a standard directory for classes in which we can place our new classes. **The first thing is to understand how classes are structured in form inheritance tree.** XoopsForm class that is the base class of the form, is an abstract class, which means that this class should not be used directly, but must be extended and generalized. This is done in a kind of template, properties and methods. I let you immerse yourself in OOP for more details. XoopsThemeForm class is the class that extends XoopsForm and that'll display the form as a table. It is this a class which we inherit, to change the general behavior of the form. **Note:** if we want our form to be no longer displayed in tabular form, we would inherit directly from XoopsForm and rework our source code; For here, it will be enough to just inherit from xoopsThemeForm and change the render() method which is responsible for creating the form. As you may have noticed, in XoopsForm (see API) the *render()* method is empty, and that's correct. This is an abstract class, and it just tells us that this method should be implemented in the class that extends it. So in XoopsThemeForm the method is implemented to display the form as a table:

```
 function render()
{
    $ele_name = $this->getName();
    $ret      = '
```

```
            ' . $this->getTitle() . '

        ';
    $hidden   = '';
    $class    = 'even';
    foreach ($this->getElements() as $ele) {
        if (!is_object($ele)) {
            $ret .= $ele;
        } else {
            if (!$ele->isHidden()) {
                if (!$ele->getNocolspan()) {
                    $ret .= '';
                    if (($caption = $ele->getCaption()) != '') {
                        $ret
                            .=
                            '';
                        $ret .= '' . $caption . '';
                        $ret .= '*';
                        $ret .= '
';
                    }
                    if (($desc = $ele->getDescription()) != '') {
                        $ret .= '' . $desc . '
';
                    }
                    $ret .= '
                            ' . $ele->render() . '
';. NWLINE;
                } else $ele->render() . '
            } else{ $ret .= '';
                $hid($caption = $ele->ele->getCaption()) != '') {
            }           $ret
        }                   .=
    }                       '';
    $ret .= '' . NWLINE .'' $caption .'';' . NWLINE;
    $ret .= $this->renderValidationJS(true);
    return $ret;  $ret .= '
};
```

Let's put it into practice In our first example we want our form to create a different view, depending on a user action. Our form is as follows:

```
 // create form object
$my_form = new ThemeForm("my form", 'Form_bien', "handle_bien.php?action=save");

// create form elements
$reference = new XoopsFormText("Reference", "reference", 50, 100);
```

```
$prix= new XoopsFormText("Price","price",50,100);

// add elements to the form
$my_form->addElement($reference,true);
$my_form->addElement($price, true);

// create/add buttons
$button = new XoopsFormButton('', 'post', _SEND, 'submit');
$my_form->addElement($button);
// display the form
$my_form->display();
```

As you can see, so far this is not a rocket science! Now we will modify the behavior of XoopsThemeForm - as first, we'll create a PHP file in the directory class, which we call a very originally: my_form.php Now we have following inheritance tree: For this class to inherit from XoopsThemeForm, also called extending it, we write in PHP:

Of course for this to work requires that the "child" class has its "mother" class (or knows where to find it) , so we must include the class XoopsThemeForm:

You can also use *xoops_load*, which helps if you don't know in which file is the class declared :

This new method (also called "setter") changes the URL to the new property $nowAction if the function parameter is passed. **Important Note:** Unfortunately, the class xoopsForm, was not optimized to give a UX developers the ability to do what they want. To do so would have required that the properties of the class xoopsForm be "protected" and not "private" (I refer you to OOP) or that each property is associated with a "setter" and a "getter". I do not know the reasons for this absence. This will force us to redefine the function *render()*. This would not have been required if we had had "setter" or if the properties had been declared as "protected". The class with the function render() redefined, we replace $this->getAction by our new property $newAction, which gives:

Now we can use it as follows:

```
 $my_form=new MyForm(parameter);

If(condition){
// Now we change the action URL of the form
   $MyForm->setNewAction('test.php')
}
```

You can make further updates on your own risk ..... In the second example we want to add an element that does not exist in the classes of XOOPS, for example we want our form to display images previously uploaded to the server, informing the user. There is no element in the forms of XOOPS to display an image. So we will create it by extending the class XoopsFormElement, which is the class responsible for creating a form element, as formcolorpicker etc. This will create following inheritance tree: Class:

The method *setContent($content)* will allow us to insert content. Use:

```
 $planchePhoto=new myElement();
$planchePhoto->setContent('photos
');
```

Now that we've created a new form element, let's add it to our form:

```
 $my_form->addElement($planchePhoto);
```

With this we have added a new item to our form.