

Rapid Prototyping with PEAR

...using DataObject and
FormBuilder

Markus Wolff



Rapid Prototyping with PEAR

The principles of Rapid Prototyping

- Show results to the customer quickly
- Discuss necessary changes early
- Refine the prototype until customer is happy with the application
- Either refactor until a clean codebase is reached or reprogram cleanly based on the approved prototype's features



Rapid Prototyping with PEAR

The PEAR packages that aid us

- DB (database API abstraction layer)
- DB_DataObject (object-relational mapping)
- HTML_QuickForm (building and validating of HTML forms)
- DB_DataObject_FormBuilder (auto-generates forms from DataObjects)



The concept

- What must my application do?
- What kind of data entities do I need?
- How are the entities related?
- GOAL: Reach an object-oriented approach to the data – think of every data entity as a class

Example: Bookstore application

- What it must do:
 - Store book data (titles, authors, formats)
 - Store customer data (name, adress)
 - Store customer reviews (did it suck?)



Bookstore data entities

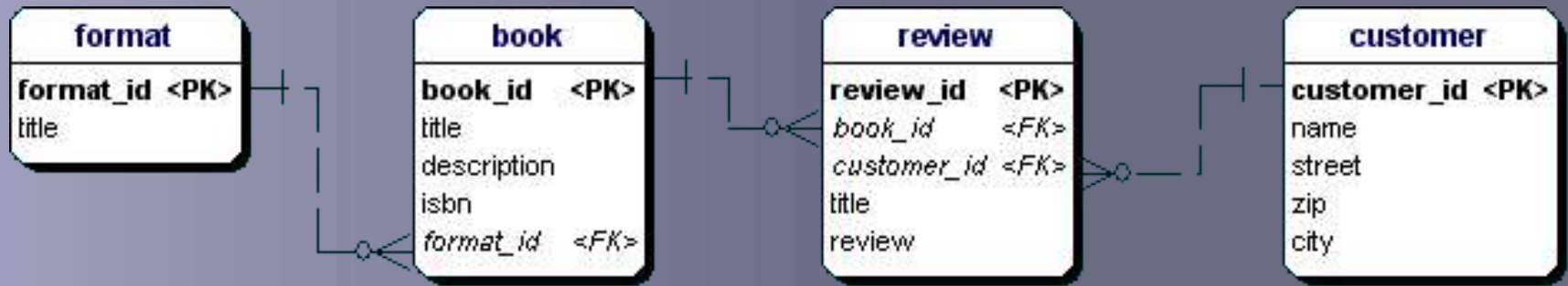
- ...this translates to the following entities:
Book, Format, Customer, Review
- Rule of thumb:
 - If one entity has a property that can have more than two values, make this another entity (especially if values can change)!
 - Your database models should always be properly normalized.



Use an ERD tool

- Better understanding of relations between entities
- Changes can be done quickly
- Database is always documented
- DBMS independence (depending on the tool)

The entity relationship model



- Keep naming scheme consistent wherever possible
- Use pl. txt fld & tbl names, av. abbr.!

Introducing DataObject

- PEAR::DB_DataObject...
 - maps database tables to PHP classes
 - provides easy access to common SQL functions like select, insert, update, delete...
 - allows developers with weak knowledge of SQL to write database-aware code
 - encourages clean distinction between presentation and business logic



Configuring DataObject

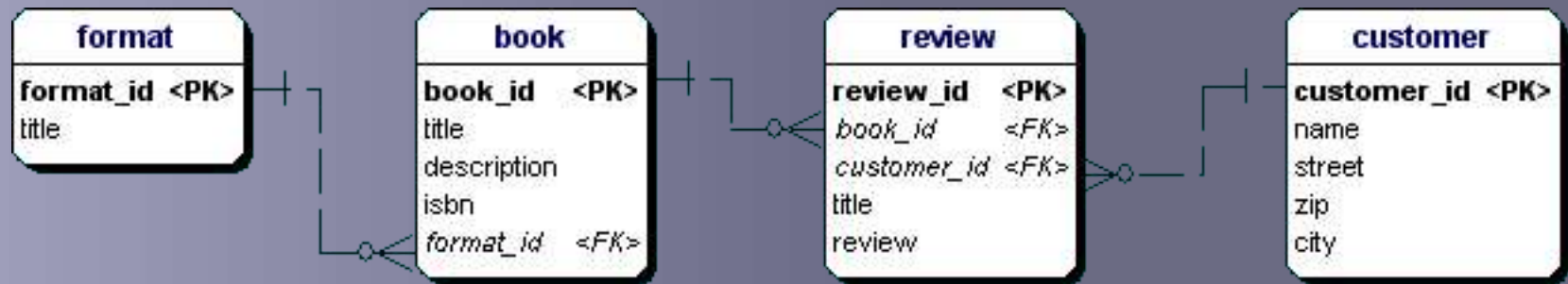
- Uses one simple .ini file:

```
[DB_DataObject]
database = mysql://user:pw@localhost/demo
schema_location = /dataobjects/schema/
class_location = /dataobjects/
require_prefix = /dataobjects/
extends_location = DB/DataObject.php
extends = DB_DataObject
```

- ...or, you can use plain PHP arrays.



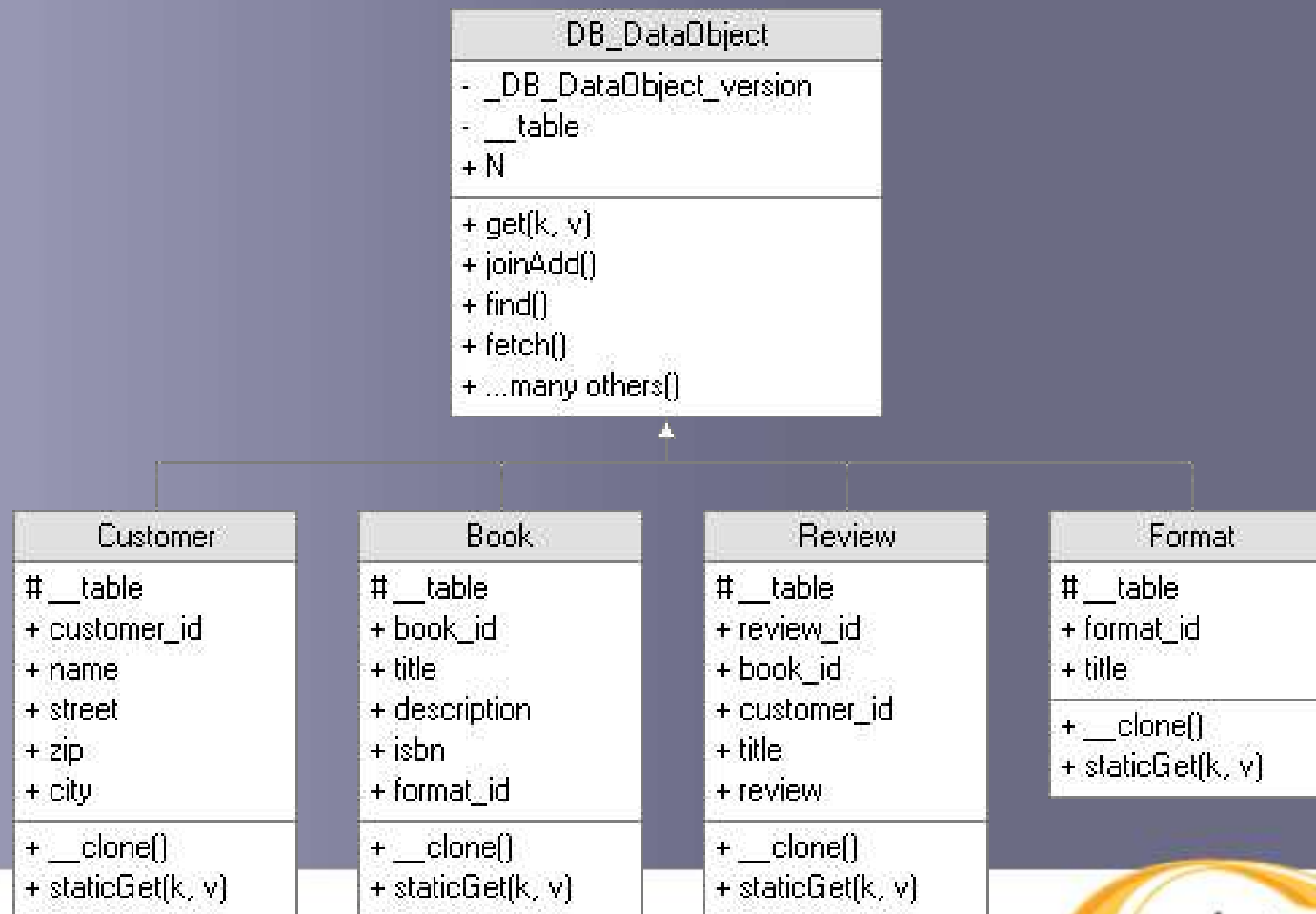
Creating classes from tables...



\$> createTables.php /path/to/DataObject.ini

- ...creates five files:
Format.php, Book.php, Review.php,
Customer.php, demo.ini

Class anatomy



Generated sourcecode

```
<?php
/**
 * Table Definition for format
 */
require_once 'DB/DataObject.php';

class Format extends DB_DataObject
{

    ###START_AUTOCODE
    /* the code below is auto generated do not remove the above tag */

    var $__table = 'format';           // table name
    var $format_id;                     // int(4)  not_null primary_key unique_key
    unsigned auto_increment
    var $title;                         // string(40)

    /* ZE2 compatibility trick*/
    function __clone() { return $this;}

    /* Static get */
    function staticGet($k,$v=NULL) { return DB_DataObject::staticGet('Format',$k,$v);
    }

    /* the code above is auto generated do not remove the tag below */
    ###END_AUTOCODE
}
?>
```



Generated database file

- DO puts the database model into simple config file named „demo.ini“

```
[book]
book_id = 129
title = 130
description = 66
isbn = 130
format_id = 129
```

```
[book__keys]
book_id = N
```

```
[customer]
customer_id = 129
name = 130
street = 2
zip = 2
city = 2
```

```
[customer__keys]
customer_id = N
```

```
[format]
format_id = 129
title = 2
...
```



Working with DataObjects (1)

- How to get a book record by primary key:

```
<?php
require_once('DB/DataObject.php');
require('config.php');

$book = DB_DataObject::factory('book');
$book->get(42);
echo 'This book is called: '.$book->title;
?>
```



Working with DataObjects (2)

- Getting all books having a specific format:

```
$book = DB_DataObject::factory('book');  
$book->format_id=23;  
$num = $book->find();  
$i = 1;  
while ($book->fetch()) {  
    echo "Book: $i of $num: {$book->title}<br>";  
    $i++;  
}
```



Working with DataObjects (3)

- Adding your own WHERE clauses:

```
$book = DB_DataObject::factory('book');  
$book->whereAdd("book.title LIKE 'PHP%'");  
$num = $book->find();  
$i = 1;  
while ($book->fetch()) {  
    echo "Book: $i of $num: {$book->title}<br>";  
    $i++;  
}
```



Working with DataObjects (4)

- Insert, update, delete...

```
$book = DB_DataObject::factory('book');  
$book->title = 'Advanced PHP Programming';  
$book->format_id = 23;  
$book_id = $book->insert();
```

```
$book->description = 'Great book ;-)';  
$book->update();
```

```
$aBook = DB_DataObject::factory('book');  
$aBook->book_id = $book_id;  
$aBook->delete();
```



Working with DataObjects (5)

- Set up table relations using another config file: demo.links.ini

```
[book]  
format_id = format:format_id
```

```
[review]  
book_id = book:book_id  
customer_id = customer:customer_id
```



Working with DataObjects (6)

- After setting up relationships, you can join table objects:

```
$review = DB_DataObject::factory('review');  
$customer = DB_DataObject::factory('customer');  
$review->book_id=4711;  
$review->joinAdd($customer, 'INNER');  
$review->selectAdd('customer.name');  
$review->find();  
while ($review->fetch()) {  
    echo $review->title.' (a review by '.  
        $review->name.')<br>';  
    echo $review->description.'<hr>';  
}
```



Working with DataObjects (7)

- Emulating triggers, encapsulating business logic, embracing inheritance:

```
function delete()  
{  
    if ($GLOBALS['user']->checkRight(MAY_DELETE_HERE)) {  
        $this->cleanUpStuff();  
        return parent::delete();  
    }  
    return false;  
}
```



Adding forms – the simple way

- DB_DataObject_FormBuilder...
 - ...creates forms from DataObjects
 - ...allows configuring form generation through setting reserved properties in DataObjects
 - ...triggers callback methods for further form manipulation
 - ...handles form processing and



Creating a form for a review

```
require_once('DB/DataObject.php');
require_once('DB/DataObject/FormBuilder.php');
require('config.php');

$review = DB_DataObject::factory('review');
$builder =& DB_DataObject_FormBuilder::create($review);
$form =& $builder->getForm();

if ($form->validate()) {
    $form->process(array(&$builder, 'processForm'),
        false);
    echo 'New review ID: ' . $review->review_id;
}
echo $form->toHtml();
```



The generated form



The screenshot shows a Mozilla Firebird browser window with a menu bar (Datei, Bearbeiten, Ansicht, Gehe, Lesezeichen) and a toolbar. The address bar shows a local file path. The main content area displays a form titled "review" with the following fields:

- Book_id**: A dropdown menu.
- Customer_id**: A dropdown menu.
- Title**: A text input field.
- Review**: A text input field.
- Submit**: A button.

The status bar at the bottom indicates "Fertig".

Using the form to update data

```
require_once('DB/DataObject.php');  
require_once('DB/DataObject/FormBuilder.php');  
require('config.php');  
  
$review = DB_DataObject::factory('review');  
$review->get(13);  
$builder =& DB_DataObject_FormBuilder::create($review);  
$form =& $builder->getForm();  
  
if ($form->validate()) {  
    $form->process(array(&$builder, 'processForm'),  
false);  
    echo 'Review updated!';  
}
```



Configuring FormBuilder

- Add some lines to DataObject.ini:

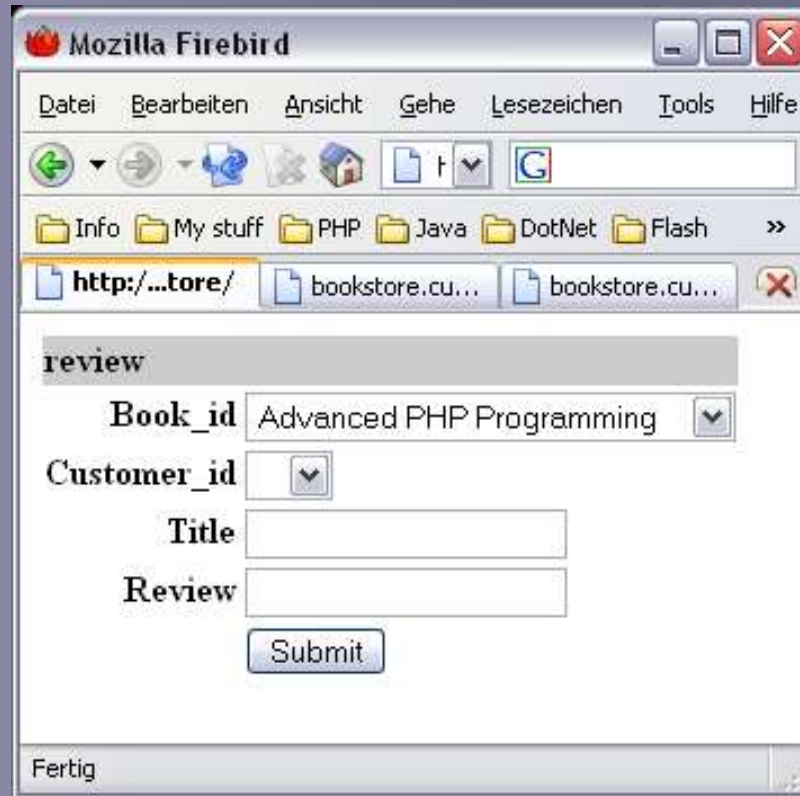
```
[DB_DataObject_FormBuilder]  
select_display_field = title
```

- Add one line to the config include file:

```
$_DB_DATAOBJECT_FORMBUILDER['CONFIG'] =  
    $config['DB_DataObject_FormBuilder'];
```



The new form output



- Customer still not displayed... no „title“!

Tweaking the customer class

```
class Customer extends DB_DataObject
{
    ###START_AUTOCODE
    /* some code omitted */
    ###END_AUTOCODE

    // Use the 'name' property to display records
    // whenever this class is used as the source for
    // a select box!
    var $select_display_field = 'name';
}
```



Customers on display



The screenshot shows a Mozilla Firebird browser window. The title bar reads "Mozilla Firebird". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Gehe", "Lesezeichen", "Tools", and "Hilf". The address bar shows a Google search icon and a search bar. Below the address bar is a folder bar with icons for "Info", "My stuff", "PHP", "Java", "DotNet", and "Flash". The main content area displays a form titled "review". The form contains the following fields:

- Book_id**: A dropdown menu with "Advanced PHP Programming" selected.
- Customer_id**: A dropdown menu with "Arthur Dent" selected.
- Title**: A text input field.
- Review**: A text input field.
- Submit**: A button.

The status bar at the bottom of the browser window displays the word "Fertig".

Tweaking the form (labels)

- To define custom labels, use the 'fieldLabels' property:

```
class Customer extends DB_DataObject
{
    ###START_AUTOCODE
    /* some code omitted */
    ###END_AUTOCODE

    var $fieldLabels = array('customer_id' =>
        'Customer',
                                'book_id'      => 'Book' );
}
```



Tweaking the form (labels)



The screenshot shows a Mozilla Firebird browser window with a menu bar (Datei, Bearbeiten, Ansicht, Gehe, Lesezeichen, Tools, Hilf) and a toolbar with navigation icons. The address bar shows a Google search icon. The bookmarks bar contains folders for Info, My stuff, PHP, Java, DotNet, and Flash. The main content area displays a form titled "review" with the following fields:

- Book**: A dropdown menu with "Advanced PHP Programming" selected.
- Customer**: A dropdown menu with "Arthur Dent" selected.
- Title**: A text input field.
- Review**: A text input field.
- Submit**: A button.

The status bar at the bottom of the browser window shows the word "Fertig".

Tweaking the form (elements)

- Defining textareas... the old-fashioned way:

```
class Customer extends DB_DataObject
{
    ###START_AUTOCODE
    /* some code omitted */
    ###END_AUTOCODE

    var $textFields = array('review');
}
```



Tweaking the form (elements)



The screenshot shows a Mozilla Firebird browser window with a menu bar (Datei, Bearbeiten, Ansicht, Gehe, Lesezeichen, Tools, Hilfe) and a toolbar with navigation and search icons. The address bar shows a Google search page. The browser's file manager shows folders for Info, My stuff, PHP, Java, DotNet, and Flash. The main content area displays a form titled "review" with the following elements:

- Book**: A dropdown menu with "Advanced PHP Programming" selected.
- Customer**: A dropdown menu with "Arthur Dent" selected.
- Title**: A text input field.
- Review**: A large text area for writing the review.
- Submit**: A button to submit the form.

The status bar at the bottom of the browser window shows the word "Fertig".

Tweaking the form (elements)

- Use the `preGenerateForm()` callback method and 'preDefElements' property to define your own element types for specific fields:

```
function preGenerateForm(&$fb) {  
    $el = HTML_QuickForm::createElement('hidden',  
                                          'customer_id');  
    $this->preDefElements['customer_id'] = $el;  
}
```



Tweaking the form (elements)



The screenshot shows a Mozilla Firebird browser window with a menu bar (Datei, Bearbeiten, Ansicht, Gehe, Lesezeichen, Tools, Hilfe) and a toolbar with navigation and search icons. The address bar shows a search engine icon and a search box. The bookmarks bar contains folders for Info, My stuff, PHP, Java, DotNet, and Flash. The main content area displays a form titled "review" with the following elements:

- Book**: A dropdown menu with "Advanced PHP Programming" selected.
- Title**: A text input field.
- Review**: A large text area for writing the review.
- Submit**: A button to submit the form.

The status bar at the bottom indicates "Fertig".

Tweaking the form (rules)

- Use the `postGenerateForm()` callback method to add form validation rules and input filters:

```
function postGenerateForm(&$form) {  
    $form->addRule('title', 'Please enter a title',  
                  'required');  
    $form->addRule('review', 'Please at least try...',  
                  'minlength', 10);  
  
    $form->applyFilter('__ALL__', 'trim');  
}
```



Tweaking the form (rules)

review

Book

***Title** Please enter a title

Review Please at least try...

* denotes required field

Defining custom forms

- You can make your own forms if...
 - ...the form requires more than just „tweaking“
 - ...you want better model / view separation
- Just define your own `getForm()` method
- `pre-` and `postGenerateForm()` will still be triggered!
- FormBuilder can still process the input



Defining custom forms

```
class CustomerForm extends HTML_QuickForm {  
    function CustomerForm($formName='CustomerForm',  
        $method='post', $action='',  
        $target='_self', $attributes=null) {  
        parent::HTML_QuickForm($formName, $method, $action,  
            $target, $attributes);  
  
        $this->addElement('text', 'name', 'Customer name');  
        $this->addElement('text', 'street', 'Street');  
        $this->addElement('text', 'zip', 'ZIP');  
        $this->addElement('text', 'city', 'City');  
        $this->addElement('submit', 'submit', 'Submit');  
  
        $this->addRule('name', 'Please enter a name',  
            'required');  
        $this->applyFilter('__ALL__', 'trim');  
    }  
}
```



Defining custom forms

```
class Customer extends DB_DataObject
{
    function &getForm($action=false, $target='_self',
                    $formName='CustomerForm', $method='post')
    {
        if (!$action) {
            $action = $_SERVER['REQUEST_URI'];
        }
        include_once('forms/CustomerForm.php');
        $form =& new CustomerForm($formName, $method, $action,
                                $target);

        return $form;
    }
}
```



Processing the form

- Usually done automatically:

```
$form->process (array (&$builder, 'processForm') ,  
                  false) ;
```

- You can also force a specific data handling method:

```
$builder->forceQueryType (  
    DB_DATAOBJECT_FORMBUILDER_QUERY_FORCEINSERT  
    ) ;
```



Processing the form

- Callback methods available:
 - `preProcessForm()`
 - `postProcessForm()`
- Common uses:
 - Event notification
 - Generating changelogs (before / after)

Nested forms

- To make just one form for data from two different tables:

```
$review = DB_DataObject::factory('review');  
$customer = DB_DataObject::factory('customer');  
$customer->createSubmit = false;  
$reviewBuilder =&  
    DB_DataObject_FormBuilder::create($review);  
$customerBuilder =&  
    DB_DataObject_FormBuilder::create  
    ($customer);  
$customerForm =& $customerBuilder->getForm();  
$reviewBuilder->useForm($customerForm);  
$combinedForm =& $reviewBuilder->getForm();
```



Nested forms

customer
Name
Street
Zip
City

review
Book
*Title
Review

* denotes required field

The bleeding edge...

- Not yet in CVS:
 - Support for crosslink tables (m:n)
 - Support for custom elements as global replacements for standard ones:

```
elementTypeMap =
```

```
    date:jscalendar,longtext:htmlarea
```



Application template

```
// FILE: index.php
require('config.php');

if (!isset($_GET['table'])) {
    die ('Please specify "table" parameter');
}

$table = $_GET['table'];

$do = DB_DataObject::factory($table);
if (PEAR::isError($do)) {
    die($do->getMessage());
}
```



Application template

```
// Find primary key
$keys = $do->keys();
if (is_array($keys)) {
    $primaryKey = $keys[0];
}

// Find title field
$titleField = $_DB_DATAOBJECT_FORMBUILDER['CONFIG']

    ['select_display_field'];
if (isset($do->select_display_field)) {
    $titleField = $do->select_display_field;
}
```



Application template

```
$do->find();

while ($do->fetch()) {
    echo sprintf('<a href="details.php?id=%s&table=%s">
                %s</a><br>',
                $do->$primaryKey,
                $table,
                $do->$titleField);
}

echo '<hr><a href="details.php?table=' .
    $table . '">Add new</a>';

// EOF: index.php
```



Application template



Application template

```
// FILE: details.php
require('config.php');
if (!isset($_GET['table'])) {
    die ('Please specify "table" parameter');
}
$table = $_GET['table'];

$do = DB_DataObject::factory($table);
if (PEAR::isError($do)) {
    die($do->getMessage());
}
if (isset($_GET['id'])) {
    $do->get($_GET['id']);
}
```



Application template

```
$builder =& DB_DataObject_FormBuilder::create($do);  
$form = $builder->getForm($_SERVER['REQUEST_URI']);  
  
if ($form->validate()) {  
    $res = $form->process(array($builder,  
    'processForm'), false);  
    if ($res) {  
        header('Location: index.php?table='.$table);  
    }  
    echo "Something went wrong...<br>";  
}  
  
echo $form->toHtml();  
// EOF: details.php
```



Application template

Mozilla Firebird

Datei Bearbeiten Ansicht Gehe Lesezeichen Tools H

Info My stuff PHP Java DotNet Flash >>

book

***Title**

Description

Isbn

Format Hardcover

* denotes required field

Fertig

Mozilla Firebird

Datei Bearbeiten Ansicht Gehe Lesezeichen Tools H

Info My stuff PHP Java DotNet Flash >>

book

***Title** Advanced PHP Program

Description A great book by George Schlossnagle, it's a *must have*

Isbn 0-672-32561-6

Format Paperback

* denotes required field

Fertig

Famous last words

- To sum-up, aforementioned packages...
 - ...unify form and data handling
 - ...make it easy to train new project members
 - ...speed up application development by at least a factor of three
 - ...make it easy to create universal application frameworks, where new functionality can easily be plugged in



The end

- Thank you for your attention!
- ANY QUESTIONS ?

