PHP 5

Marcus Börger
Sterling Hughes

Overview

- ☑ PHP5 vs PHP4
- ☑ Is PHP5 revolutionary?
- ☑ PHP 5 OO
  - ☑ Why is OO a good thing?
- ☑ PHP5 and Databases
- ☑ PHP5 and XML

$E = mc^2$

- PHP5 is “faster” than PHP4
  - Speed by design
  - Nitty gritty engine improvements
    - Faster callbacks
    - Faster comparisons
    - Faster Harder Stronger
  - New extensions that eliminate userspace code overhead
    - PDO
    - SQLite
- PHP4 executes code faster
  - New execution architecture slows things down
  - Execution architecture isn’t terribly important though

Revamped OO Model

- ☑ PHP5 has really good OO
  - ☑ Better code reuse
  - ☑ Better for team development
  - ☑ Easier to refactor
  - ☑ Some patterns lead to much more efficient code
  - ☑ Fits better in marketing scenarios

PHP 4 and OO ?

- Poor Object model
  - ☑ Methods
    - ☒ No visibility
    - ☒ No abstracts, No final
    - ☒ Static without declaration
  - ☑ Properties
    - ☒ No default values
    - ☒ No static properties
  - ☑ Inheritance
    - ☒ No abstract, final inheritance, no interfaces
  - ☑ Object handling
    - ☒ Copied by value
    - ☒ No destructors

The Solution to all your problems

- ☑ PHP4’s XML was pathetic
  - ☑ SAX was OK
  - ☑ DOM was crappy, DOM was fake
  - ☑ There was nothing else
- ☑ PHP5 XML is brilliant
  - ☑ **Bow**
  - ☑ SAX is OK
  - ☑ DOM is functional
  - ☑ SimpleXML is the solution to all your problems

Other Stuff

- ☑ PHP5 has much improved streams support
  - ☑ Stream filters
  - ☑ Engine level integration
  - ☑ Stream “Servers”
- ☑ PHP5 will have a packaging system
- ☑ PHP5 has completely new XML support
- ☑ PHP5 has a new database abstraction api
- ☑ PHP5 supports embedded databases
- ☑ PHP5 has an improved CLI
- ☑ PHP5 has a new imaging system (PIMP)

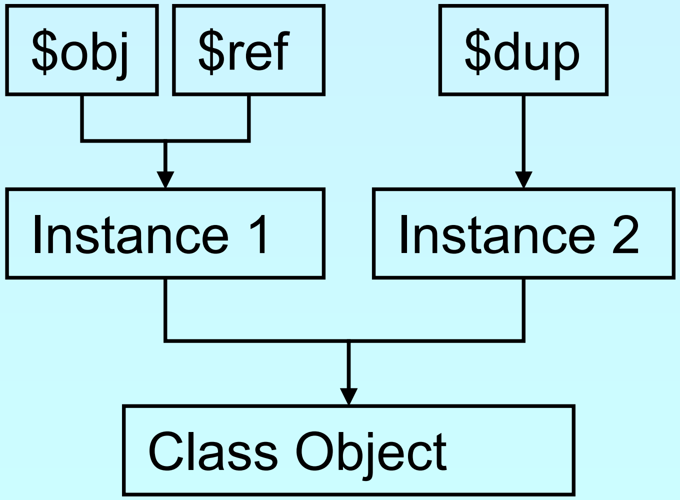
ZE2's revamped object model

- ☑ Objects are referenced by identifiers
- ☑ Constructors and Destructors
- ☑ Static members
- ☑ Default property values
- ☑ Constants
- ☑ Visibility
- ☑ Interfaces
- ☑ Final and abstract members
- ☑ Interceptors
- ☑ Exceptions
- ☑ Reflection API
- ☑ Iterators

Objects referenced by identifiers

- ☑ Objects are no longer copied by default
- ☑ Objects may be copied using __clone()

```php
<?php

class Object {};

$obj = new Object();

$ref = $obj;

$dup = $obj->__clone();

?>
```



Constructors and Destructors

☑ Constructors/Destructors control object lifetime

- ☑ Constructors may have both new OR old style names
- ☑ Destructors are called when deleting last reference

```php
<?php

class Object {
  function __construct() {}
  function __destruct() {}
}
$obj = new Object();
unset($obj);

?>
```

Constructors and Destructors

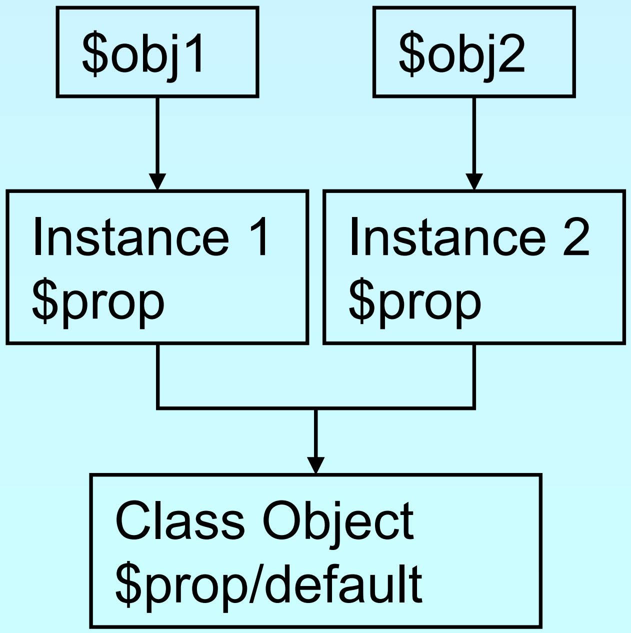
☑ Parents must be called manually

```php
<?php
class Base {
   function __construct() {}
   function __destruct() {}
}
class Object extends Base {
   function __construct() {
      parent::__construct();
   }
   function __destruct() {
      parent::__destruct();
   }
}
$obj = new Object();
unset($obj);
?>
```

Default property values

- ☑ Properties can have default values
  - ☑ Bound to the class not to the object
  - ☑ Default values cannot be changed but overwritten

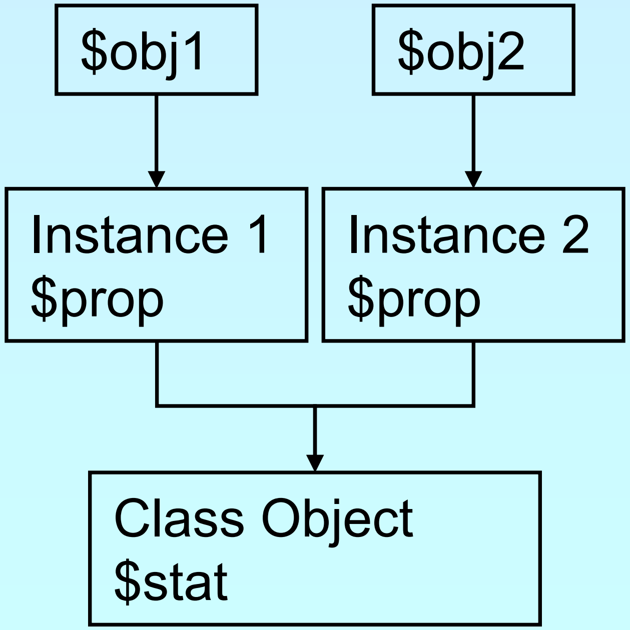
```
<?php

class Object {
  var $prop = "Hello\n";
}

$obj1 = new Object;
$obj1->prop = "Hello World\n";

$obj2 = new Object;
echo $obj2->prop; // Hello

?>
```



Static members

☑ Static methods and properties
- ☑ Bound to the class not to the object
- ☑ Can be initialized

```
<?php
class Object {
  var $pop;
  static $stat = "Hello\n";
  static function test() {
    echo self::$stat;
  }
}
Object::test();
$obj1 = new Object;
$obj2 = new Object;
?>
```



New pseudo constants

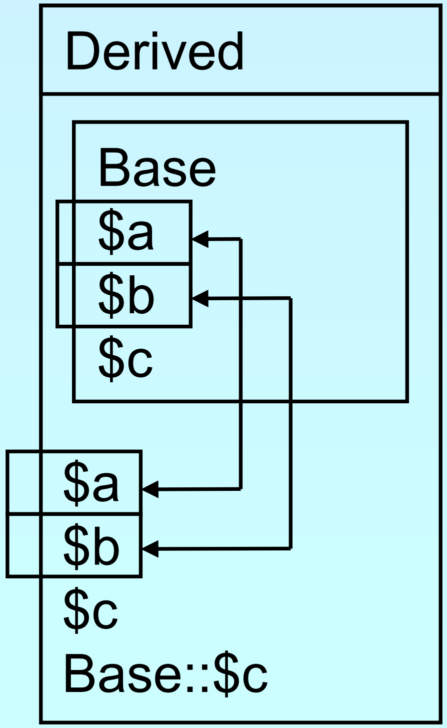
☑	__CLASS__	shows the current class name
☑	__METHOD__	shows class and method or function
☑	Self	references the class itself
☑	Parent	references the parent class
☑	$this	references the object itself

```
<?php
class Base {
    static function Show() {
        echo __FILE__.'('.__LINE__.'):'.__METHOD__."\n";
    }
}
class Object extends Base {
    static function Use() {
        Self::Show();
        Parent::Show();
    }
    static function Show() {
        echo __FILE__.'('.__LINE__.'):'.__METHOD__."\n";
    }
}
?>
```

Visibility

- ☑ Controlling member visibility / Information hiding
  - ☑ A derived class does not know inherited privates
  - ☑ An inherited protected member can be made public

```php
<?php
class Base {
  public $a;
  protected $b;
  private $c;
}
class Derived extends Base {
  public $a;
  public $b;
  private $c;
}
?>
```



Constructor visibility

☑ A protected constructor prevents instantiation

```
<?php
class Base {
   protected function __construct() {
   }
}
class Derived extends Base {
   // constructor is still protected
   static function getBase() {
      return new Base; // Factory pattern
   }
}
class Three extends Derived {
   public function __construct() {
   }
}
?>
```

Clone visibility

- ☑ A protected __clone prevents external cloning
- ☑ A private final __clone prevents cloning

```
<?php
class Base {
   protected function __clone($that) {
   private final function __clone($that) {
   }
}
class Derived extends Base {
   public function __clone($that) {
   // public function __clone($that) {
      return new Base;
   // return new Base;
   }
   // }
   public static function copyBase($that) {
   // public static function copyBase($that) {
      return Base::__clone($that);
   // return Base::__clone($that);
   }
   // }
}
?>
```

Constants

- ☑ Constants are read only static properties
- ☑ Constants are always public

```
<?php
class Base {
  const greeting = "Hello\n";
}
class Dervied extends Base {
  const greeting = "Hello World\n";
  static function func() {
    echo parent::greeting;
  }
}
echo Base::greeting;
echo Derived::greeting;
Derived::func();
?>
```

Abstract members

- ☑ Properties cannot be made abstract
- ☑ Methods can be abstract
  - ☑ They don’t have a body
  - ☑ A class with an abstract method must be abstract
- ☑ Classes can be made abstract
  - ☑ The class cannot be instantiated

```php
<?php
abstract class Base {
  abstract function no_body();
}
class Derived extends Base {
  function no_body() { echo "Body\n"; }
}
?>
```

Final members

- ☑ Methods can be made final
  - ☑ They cannot be overwritten
  - ☑ They are class invariants
- ☑ Classes can be made final
  - ☑ They cannot be inherited

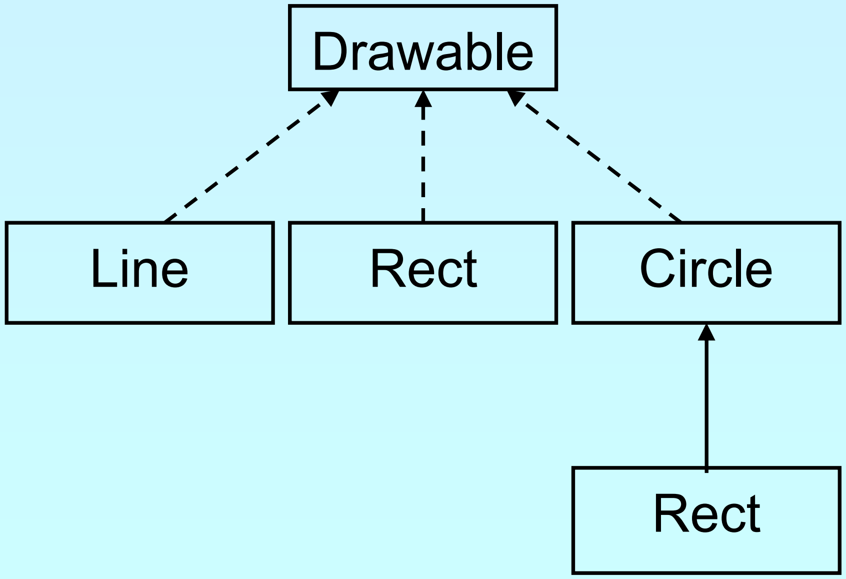
```php
<?php
class Base {
  final function invariant() { echo "Hello\n"; }
}
class Derived extends Base {
}
final class Leaf extends Derived {
}
?>
```

Interfaces

☑ Interfaces describe an abstract class protocol

☑ Classes may inherit multiple Interfaces

```
<?php
interface Drawable {
  function draw();
}
class Line implements Drawable {
 function draw() {};
}
class Rect implements Drawable {
 function draw() {};
}
class Circle implements Drawable {
 function draw() {};
}
class Ellipse extends Circle {
 function draw() {};
}
?>
```



Property types

- ☑ Declared properties
  - ☑ May have a default value
  - ☑ Can have selected visibility
- ☑ Implicit public properties
  - ☑ Declared by simply using them in ANY method
- ☑ Virtual properties
  - ☑ Handled by interceptor methods
- ☑ Static properties

Object to String conversion

☑ __toString(): automatic object string conversion

```
<?php
class Object {
    function __toString() {
        return 'Object as string';
    }
}

$o = new Object;

echo $o;

$str = (string) $o;
?>
```

Interceptors

- ☑ Allow to dynamically handle non class members
  - ☑ Lazy initialization of properties
  - ☑ Simulating Object aggregation, Multiple inheritance

```php
<?php
class Object {
  protected $virtual;
  function __get($name) {
     return @$virtual[$name];
  }
  function __set($name, $value) {
     $virtual[$name] = $value;
  }
  function __call() {
     echo 'Could not call ' . __CLASS__ . '::' . $func . "\n";
  }
}
?>
```

Exceptions

☑ Respect these rules

1. Exceptions are exceptions
2. Never use exceptions for control flow
3. Never ever use exceptions for parameter passing

```
<?php
try {
    // your code
    throw new Exception();
}
catch (Exception $e) {
    // exception handling
}
?>
```

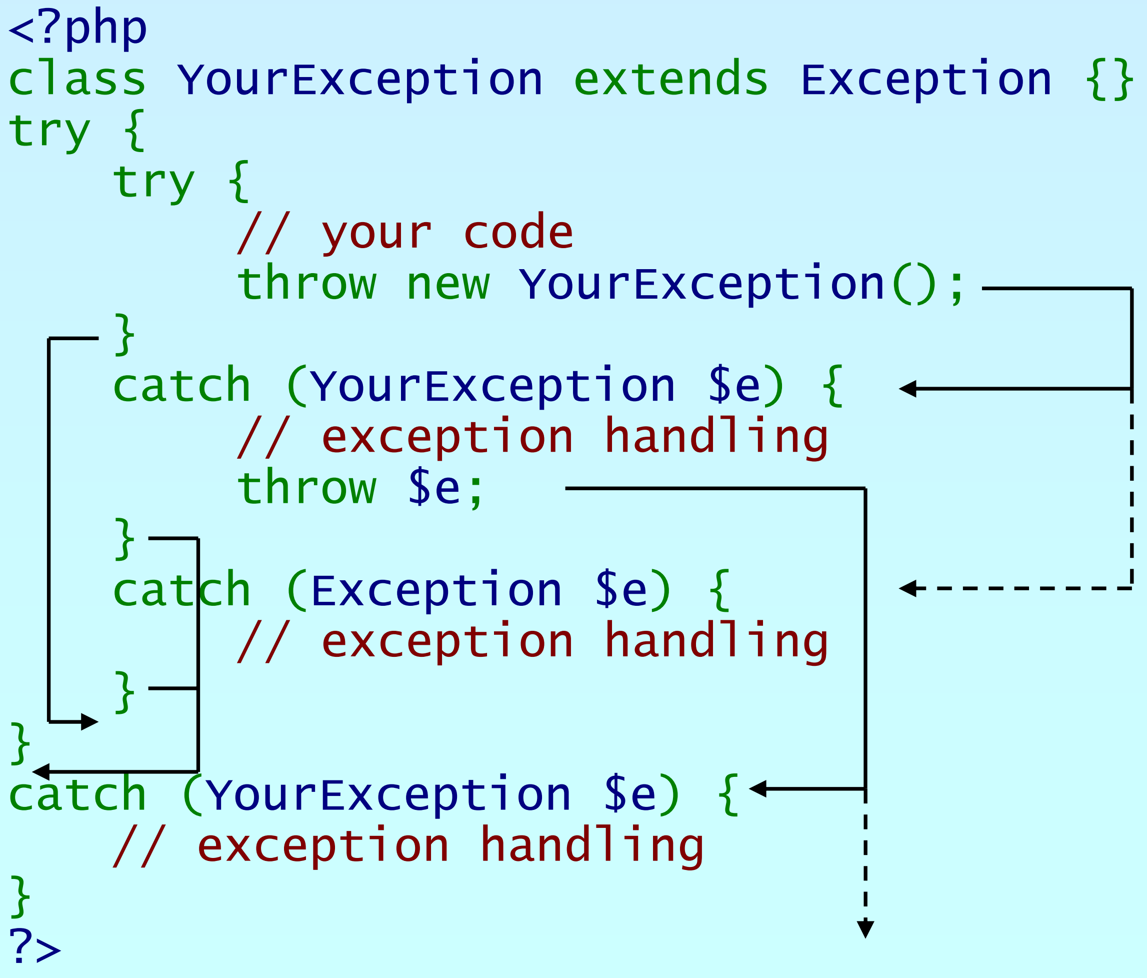
Exception specialization

☑ Exceptions should be specialized

```php
<?php
class YourException extends Exception {
}
try {
    // your code
    throw new YourException();
}
catch (YourException $e) {
    // exception handling
}
catch (Exception $e) {
    // exception handling
}
?>
```

Exception specialization

- ☑ Exception blocks can be nested
- ☑ Exceptions can be rethrown



```
<?php
class YourException extends Exception {};
try {
    try {
        // your code
        throw new YourException();
    }
    catch (YourException $e) {
        // exception handling
        throw $e;
    }
    catch (Exception $e) {
        // exception handling
    }
}
catch (YourException $e) {
    // exception handling
}
?>
```

Constructor failure

- ☑ Constructors do not return the created object
- ☑ Exceptions allow to handle failed constructors

```
<?php
class Object {
    function __construct() {
        throw new Exception;
    }
}
try {
    $o = new Object;
}
catch (exception $e) {
    echo "Object could not be instantiated\n";
}
?>
```

Reflection API

☑ Can reflect nearly all aspects of your PHP code

- ☑ Functions
- ☑ Classes, Methods, Properties
- ☑ Extensions

```php
<?php
class Foo {
    public $prop;
    function Func($name) {
        echo "Hello $name";
    }
}

reflection_class::export('Foo');
reflection_object::export(new Foo);
reflection_method::export('Foo', 'func');
reflection_property::export('Foo', 'prop');
reflection_extension::export('standard');
?>
```

Iterators

- ☑ Some objects can be iterated
- ☑ Others show their properties

```
<?php

class Object {
	public $prop1 = "Hello";
	public $prop2 = "World\n";
}

foreach(new Object as $prop) {
	echo $prop;
}

?>
```

Iterators

- ☑ Internal Iterators
- ☑ User Iterators

```
<?php
interface Iterator {
  function rewind();
  function hasMore();
  function current();
  function key();
  function next();
}
?>
```

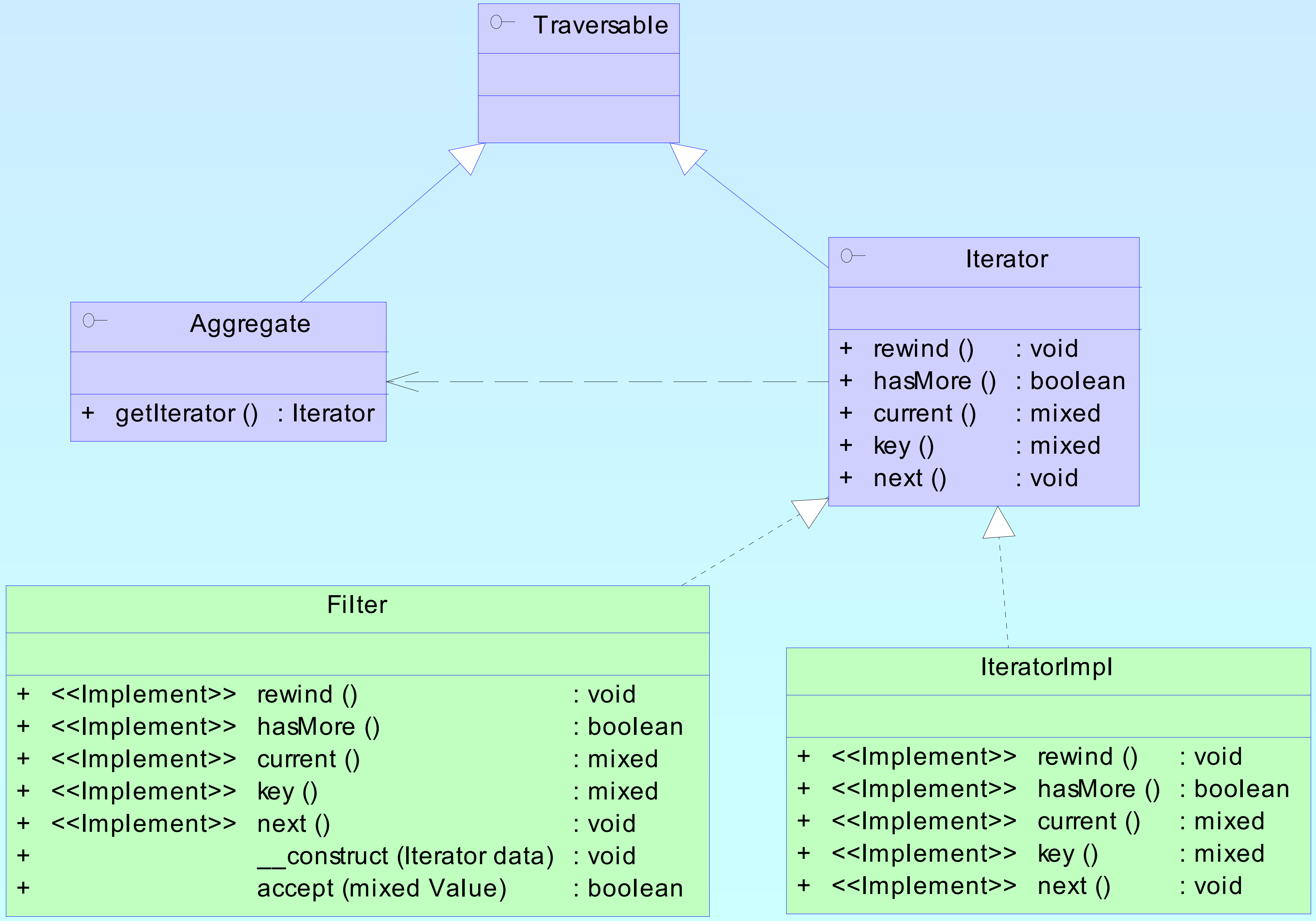
```
<?php
class Filter implements Iterator {
  function __construct(Iterator $input)...
  function rewind()...
  function accept($value)...
  function hasMore()...
  function current()...
  function key()...
  function next()...
}
?>
```

```
<?php
$it = get_resource();
foreach($it as $key=>$val) {
  // access data
}
?>
```

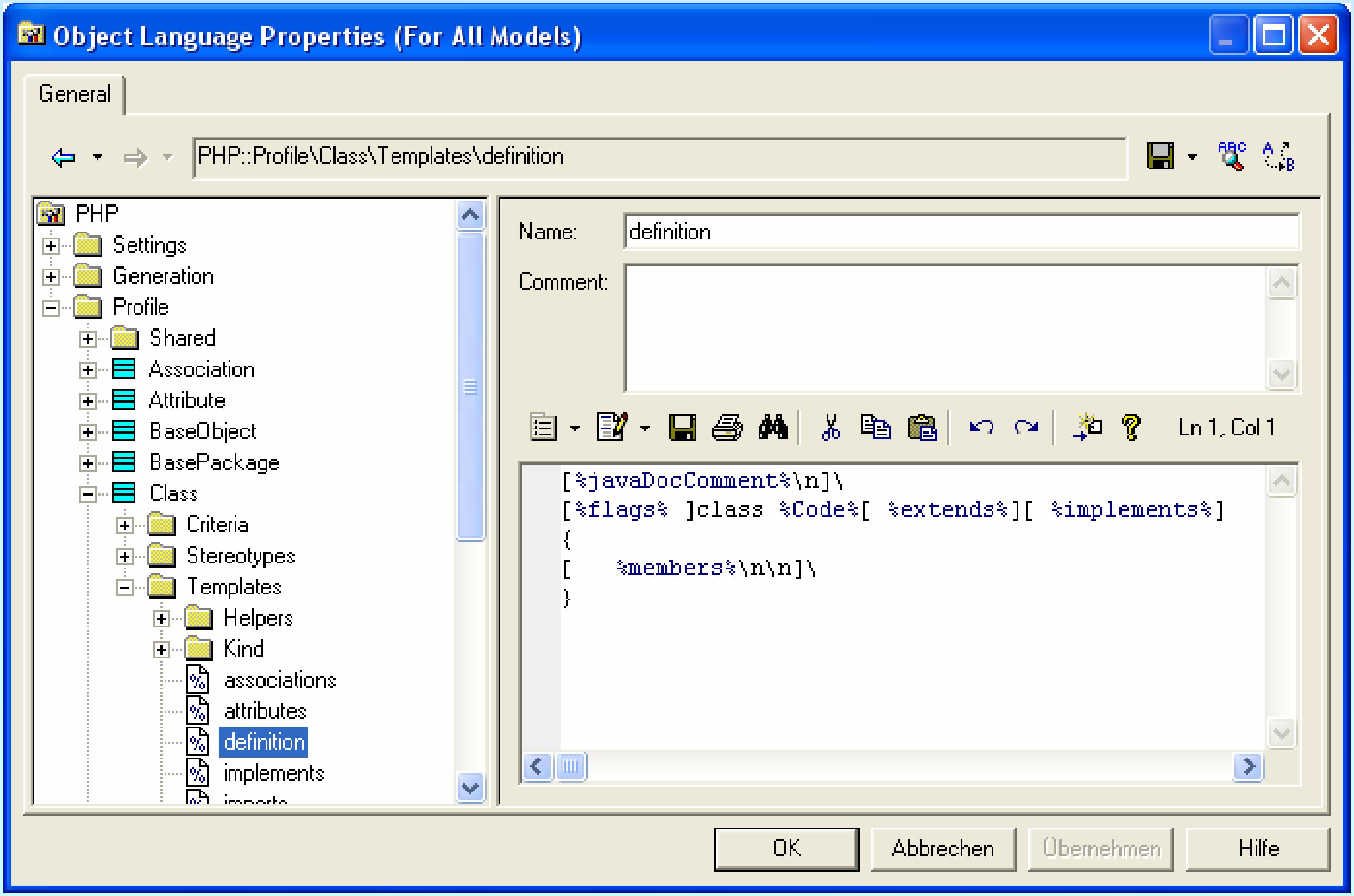
```
<?php
$it = get_resource();
for ($it->rewind(); $it->hasMore(); $it->next()) {
  $value = $it->current(); $key = $it->key();
}
?>
```

```
<?php
$it = get_resource();
foreach(new Filter($it, $filter_param) as $key=>$val) {
  // access filtered data only
}
?>
```

PHP & UML


Traversable
Aggregate
+ getIterator () : Iterator
Iterator
+ rewind () : void
+ hasMore () : boolean
+ current () : mixed
+ key () : mixed
+ next () : void
Filter
+ <<Implement>> rewind () : void
+ <<Implement>> hasMore () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void
+ __construct (Iterator data) : void
+ accept (mixed Value) : boolean
IteratorImpl
+ <<Implement>> rewind () : void
+ <<Implement>> hasMore () : boolean
+ <<Implement>> current () : mixed
+ <<Implement>> key () : mixed
+ <<Implement>> next () : void

PHP & UML

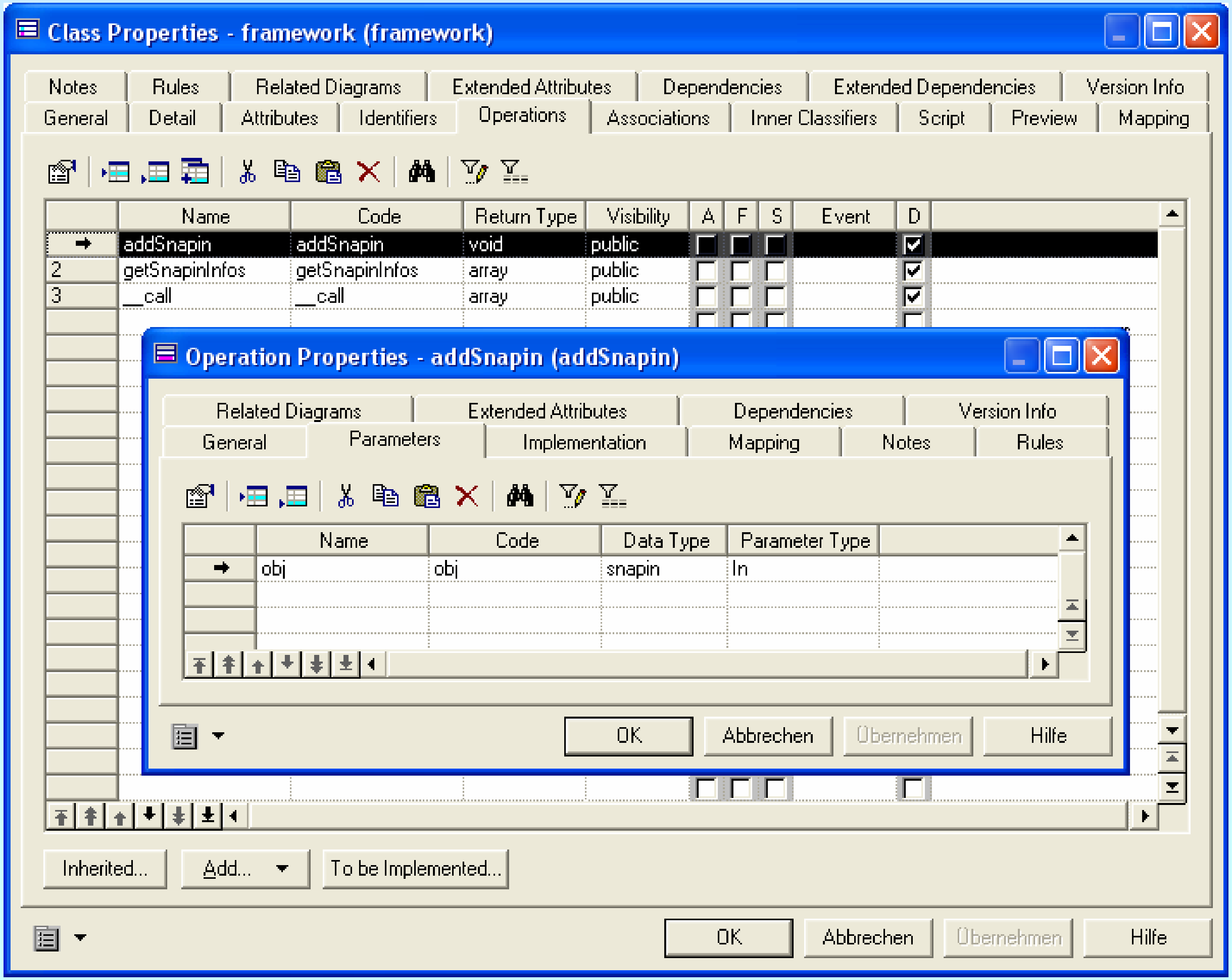


Name: definition

Comment:

```
[%javaDocComment%\n]\
[%flags% ]class %Code%[ %extends%][ %implements%]
{
[    %members%\n\n]\
}
```

PHP & UML



Class Properties - framework (framework)

	Name	Code	Return Type	Visibility	A	F	S	Event	D
→	addSnapin	addSnapin	void	public	☐	☐	☐		✓
2	getSnapinInfos	getSnapinInfos	array	public	☐	☐	☐		✓
3	__call	__call	array	public	☐	☐	☐		✓

Operation Properties - addSnapin (addSnapin)

	Name	Code	Data Type	Parameter Type
→	obj	obj	snapin	In

Typehinting

☑ PHP 5 allows to easily force a type of a parameter
  - ☑ Beta 1 and beta 2 allow NULL with typehints
  - ☑ Beta 3 will have a syntax to decide about NULL

```php
<?php
class Object {
   public function compare(Object $other) {
      // Some code here
   }
}
?>
```

CLI in PHP4

- Direct code execution

  php [*options*] –r *code* [[--] *args*...]

- Interactive mode

  php -a

Improved CLI in PHP5

- Line by line input processing

  php [-B *code*] –R *code* [-E *code*] [[--] *args*]

- Line by line input processing with scripts

  php [-B *code*] –F *code* [-E *code*] [[--] *args*]

Counting source lines

Try
```
find –regex '.*\.[ch]' –exec wc -l {} \;
```
Try
```
for i in `find -regex '.*\.[ch]' `;do wc –l $i;done;
```

Do
```
find –regex '.*\.[ch]' | xargs wc –l
```
Do
```
find –regex '.*\.[ch]' -exec wc –l {} \; |
          awk '{L=L+$1} END { print L }'
```
Do
```
find –regex '.*\.[ch]' |
          php –R '@$L+=count(file($argn));'
                    -E 'echo "$L\n";'
```

CLI meets CVS

- Search for locally modified files

```
cvs –n up 2>/dev/null |
        awk '/M\ / {print $2}'

cvs –n up 2>/dev/null |
        php –R 'ereg("^M ",$argn) &&
                print(substr($argn,2))."\n";'
```

CLI meets CVS

- CVS clean

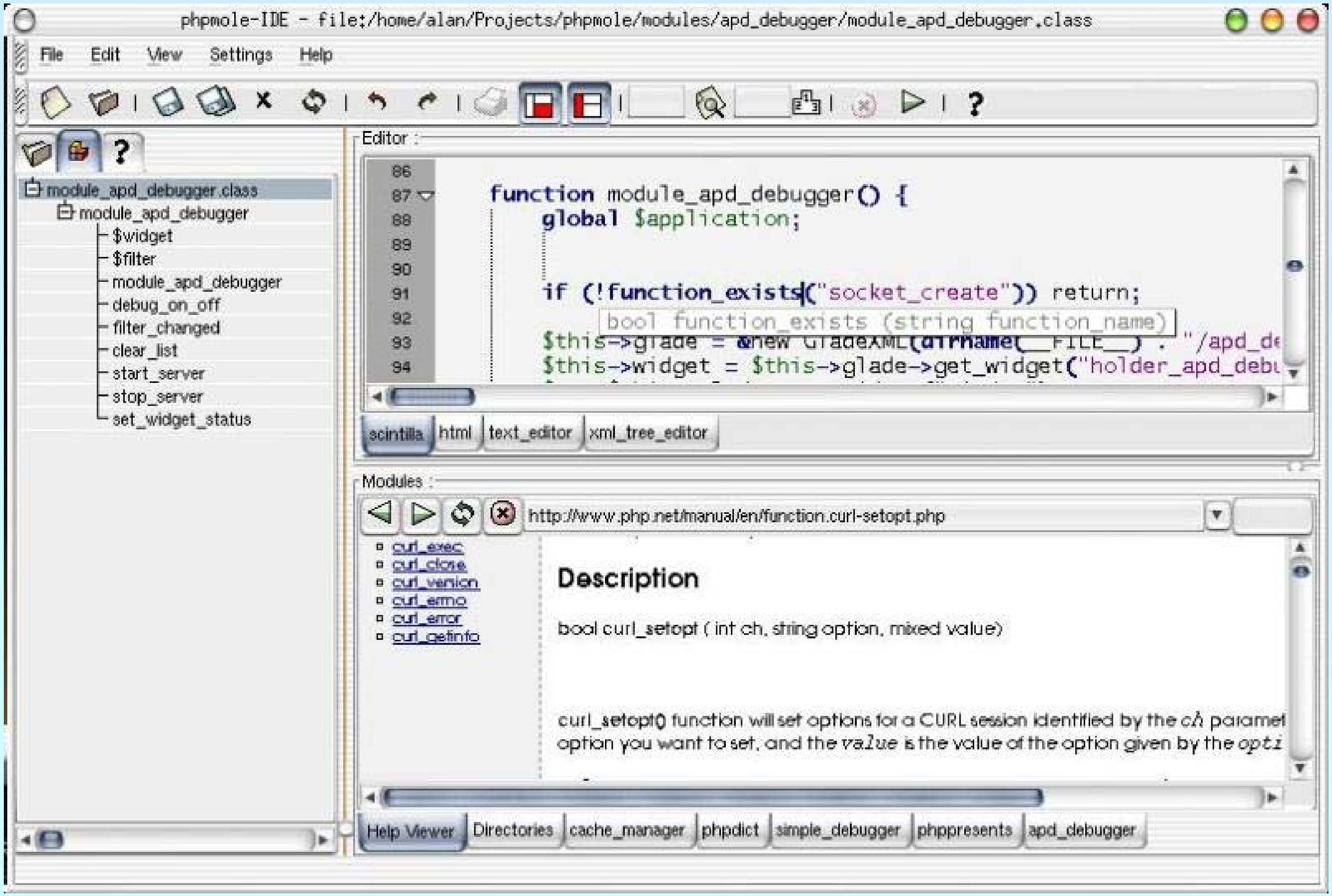
```
cvs up -C

cvs –n up 2>/dev/null |
        php –R 'ereg("^[MA] ",$argn) &&
                system("rm –f ".substr($argn,2));'
```

CLI and PHP5 ?

- New oo features provide new solutions

```
php –r 'foreach(new DirectoryIterator($argv[1])
                as $f) echo "$f\n";'

php –r 'include "dba.inc";
                $db=new dba($argv[1]);
                $db[$argv[2]=$argv[3];'
```

GTK: phpMole


phpmole-IDE - file:/home/alan/Projects/phpmole/modules/apd_debugger/module_apd_debugger.class
File Edit View Settings Help
module_apd_debugger.class
module_apd_debugger
$widget
$filter
module_apd_debugger
debug_on_off
filter_changed
clear_list
start_server
stop_server
set_widget_status
Editor :
scintilla html text_editor xml_tree_editor
Modules :
http://www.php.net/manual/en/function.curl-setopt.php
curl_exec
curl_close
curl_version
curl_errno
curl_error
curl_getinfo
Help Viewer Directories cache_manager phpdict simple_debugger phppresents apd_debugger

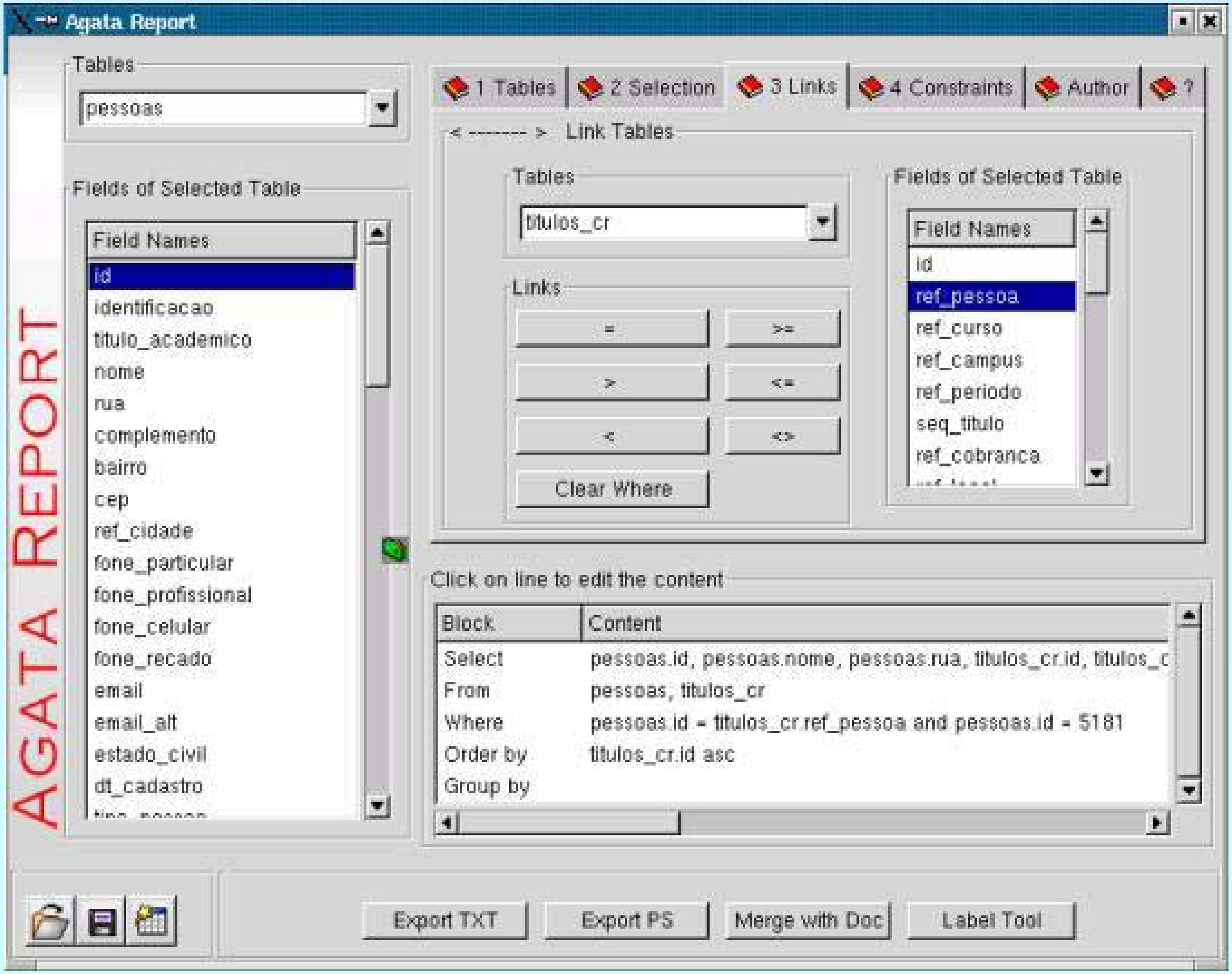
```
    function module_apd_debugger() {
        global $application;

        if (!function_exists("socket_create")) return;
                 bool function_exists (string function_name)
        $this->glade = &new GladeXML(dirname(__FILE__) . "/apd_de
        $this->widget = $this->glade->get_widget("holder_apd_debu
```

Description

bool curl_setopt (int ch, string option, mixed value)

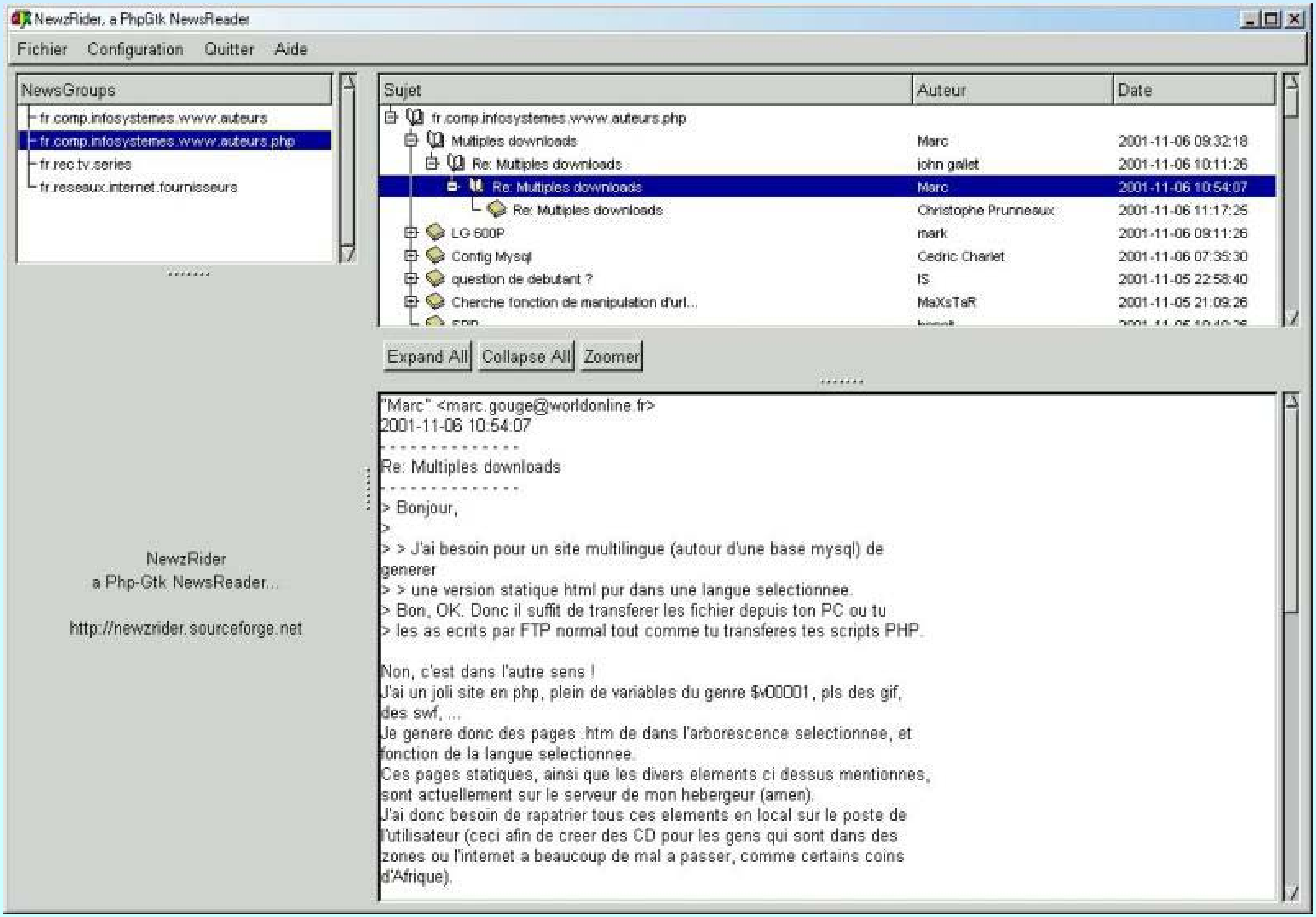
curl_setopt() function will set options for a CURL session identified by the *ch* paramet
option you want to set, and the *value* is the value of the option given by the *opti*

GTK: AgataReport



Block	Content
Select	pessoas.id, pessoas.nome, pessoas.rua, titulos_cr.id, titulos_c
From	pessoas, titulos_cr
Where	pessoas.id = titulos_cr.ref_pessoa and pessoas.id = 5181
Order by	titulos_cr.id asc
Group by	

GTK: NewzRider


NewzRider, a PhpGtk NewsReader
Fichier Configuration Quitter Aide
NewsGroups
Sujet Auteur Date
Expand All Collapse All Zoomer
NewzRider
a Php-Gtk NewsReader...
http://newzrider.sourceforge.net

NewsGroups:
- fr.comp.infosystemes.www.auteurs
- fr.comp.infosystemes.www.auteurs.php
- fr.rec.tv.series
- fr.reseaux.internet.fournisseurs

Sujet	Auteur	Date
fr.comp.infosystemes.www.auteurs.php		
Multiples downloads	Marc	2001-11-06 09:32:18
Re: Multiples downloads	john gallet	2001-11-06 10:11:26
Re: Multiples downloads	Marc	2001-11-06 10:54:07
Re: Multiples downloads	Christophe Prunneaux	2001-11-06 11:17:25
LG 600P	mark	2001-11-06 09:11:26
Config Mysql	Cedric Charlet	2001-11-06 07:35:30
question de debutant ?	IS	2001-11-05 22:58:40
Cherche fonction de manipulation d'url...	MaXsTaR	2001-11-05 21:09:26

```
"Marc" <marc.gouge@worldonline.fr>
2001-11-06 10:54:07
- - - - - - - - - - - - - -
Re: Multiples downloads
- - - - - - - - - - - - - -
> Bonjour,
>
> > J'ai besoin pour un site multilingue (autour d'une base mysql) de
generer
> > une version statique html pur dans une langue selectionnee.
> Bon, OK. Donc il suffit de transferer les fichier depuis ton PC ou tu
> les as ecrits par FTP normal tout comme tu transferes tes scripts PHP.

Non, c'est dans l'autre sens !
J'ai un joli site en php, plein de variables du genre $v00001, pls des gif,
des swf, ...
Je genere donc des pages .htm de dans l'arborescence selectionnee, et
fonction de la langue selectionnee.
Ces pages statiques, ainsi que les divers elements ci dessus mentionnes,
sont actuellement sur le serveur de mon hebergeur (amen).
J'ai donc besoin de rapatrier tous ces elements en local sur le poste de
l'utilisateur (ceci afin de creer des CD pour les gens qui sont dans des
zones ou l'internet a beaucoup de mal a passer, comme certains coins
d'Afrique).
```

Resources

☑ man php

☑ http://www.php.net/features.commandline

☑ http://gtk.php.net

New extensions

☑ New extensions

- ☑ FFI
- ☑ Date
- ☑ DOM
- ☑ MySQLi
- ☑ PDO
- ☑ PHILI
- ☑ SimpleXML
- ☑ SOAP
- ☑ SPL
- ☑ SQLite
- ☑ Tidy
- ☑ XML + XSL

FFI

☑ Native Function Call Interface
  ☑ Written by Wez Furlong…

```php
<?php
$lib = new FFI_Library('libc6.so');
echo $lib->strlen("Hello World");
?>
```

New extensions: DOM, SimpleXML, XSL

- ☑ PHP5 will use libXML2 instead of expat
- ☑ ext/DOM fully confirms to W3C standards and replaces ext/DOMXML
- ☑ ext/XSL is based on ext/DOM and replaces ext/XSLT
- ☑ ext/SimpleXML is the simple PHP way to access XML Data

New extensions: MySQLi

- ☑ Mysql grows to become more and more an enterprise ready DBMS but sticks to its origin fastness, easiness
- ☑ PHP5 reflects this development by providing a new extension named MySQLi
- ☑ Support for MySQL embedded into PHP
- ? Profiling

New extensions: SQLite

- ☑ Started in 2000 by D. Richard Hipp
- ☑ Single file database
- ☑ Subselects, Triggers, Transactions, Views
- ☑ Very fast, 2-3 times faster than MySQL, PostgreSQL for many common operations
- ☑ 2TB data storage limit

- ☒ Views are read-only
- ☒ No foreign keys
- ☒ Locks whole file for writing

New extensions: SQLite

- ☑ PHP extension bundled with PHP 5
- ☑ Available via PECL since PHP4.3
- ☑ Used on php.net
- ☑ SQLite library integrated with PHP extension
- ☑ API designed to be logical, easy to use
- ☑ High performance
- ☑ Convenient migration from other PHP database extensions
- ☑ Call PHP code from within SQL

New extensions: SPL

☑ SPL aka Standard PHP Library

☑ Iterators

☑ Filters

☑ Standard internal classes

New extensions: PDO

- ☑ PDO aka PHP Data Objects
- ☑ Provides an object oriented unified way of accessing data from different sources

- ☑ Limit support/emulation through ext/spl
- ☑ Profiling
- ☑ Precompiled statements

New extensions: PIMP

- ☑ A better version of GD
- ☑ Fast: 2X ... 100X
- ☑ Less memory usage and allocation calls
- ☑ Object oriented
- ☑ "old fashion" bitmap features (~90% compatible)
- ☑ Fast image filters
- ☑ Own plug-in mechanism
  - ☑ Libcairo
    - ☑ XWindow
    - ☑ PDF 1.4
    - ☑ Postscript