

```
<?php
/**
 * Proxy Class for all classes
 *
 * The proxy pattern was born out of the need for speed by avoiding unnecessary class
 * initiation, initiating the class only when its functionality is required. To achieve
 * this you needed to create a proxy pattern class for every one of your classes; not
 * any more, MasterProxyClass assures us of this. MasterProxyClass was developed to save
 * time and effort in creating a new proxy pattern class for every class in your project;
 * it even comes with ease of use to boot.
 *
 * @name      Master Proxy Class
 * @author    David Branco <David@NeoliteUSA.com>
 * @link      http://www.NeoliteUSA.com/
 * @version   0.7.6
 * @since     Jan 04, 2007
 * @license   http://creativecommons.org/licenses/by/2.5/
 */
class masterProxyClass {

    /**
     * Holds class file location
     */
    private $_classFile = null;

    /**
     * Holds class name
     */
    private $_className = null;

    /**
     * Holds class arguments for call
     */
    private $_classArgs = null;

    /**
     * Holds loaded class
     */
    private $_class = null;

    /**
     * Sets the class to initiate and the file it can be found in.
     * Any other variables passed will be sent to the class created.
     *
     * Example:
     * If you wanted:
     *     $Tank = new Phishtank('MyAppKey', 'MySharedSecret', 'MyUsername',
     *         'MyAPIKey');
     *
     * You would call it by:
     *     $TankProxy = new masterProxyClass('Phishtank', './phishtank.class.php',
     *         'MyAppKey', 'MySharedSecret', 'MyUsername', 'MyAPIKey');
     *
     * @return void
     */
    public function __construct($className, $classFile, array $classArgs = array())
    {
        $this->_className = $className;
        $this->_classFile = $classFile;
        $this->_classArgs = $classArgs;
    }

    /**
     * Sends variable reading over to the real variables wanted.
     *
     * @return mixed
     */
}
```

```
*/
public function __get($variable)
{
    $this->__createClass();
    return $this->_class->$variable;
}

/**
 * Sends variable writing over to the real variables wanted.
 *
 * @return void
 */
public function __set($variable, $value)
{
    $this->__createClass();
    $this->_class->$variable = $value;
}

/**
 * Sends the isset function over to the real variables wanted.
 *
 * @return boolean
 */
public function __isset($variable)
{
    $this->__createClass();
    return isset($this->_class->$variable);
}

/**
 * Sends the unset function over to the real variables wanted.
 *
 * @return void
 */
public function __unset($variable)
{
    $this->__createClass();
    unset($this->_class->$variable);
}

/**
 * Redirects all functions called over to the real functions wanted.
 *
 * @return mixed
 */
public function __call($methodName, $args)
{
    $this->__createClass();
    return call_user_func_array(array(&$this->_class, $methodName), $args);
}

/**
 * Checks to see if the proxied class has been created. If not
 * the class is created.
 *
 * Thank you, ole for the ReflectionClass idea.
 *
 * @return void
 */
private function __createClass() {
    if($this->_class == null){
        require_once($this->_classFile);
        $src = new ReflectionClass($this->_className);
        $this->_class = $src->newInstanceArgs($this->_classArgs);
    }
}
```

```
}  
  
class testProxy {  
  
    public function __construct($one = 'blank', $two = 'blank') {  
        echo 'Var 1 = ' . $one . '<br />';  
        echo 'Var 2 = ' . $two . '<br /><br />';  
    }  
  
    public function kill($who) {  
        echo 'You killed ' . $who . '! <br /><br />';  
    }  
  
    public function create($who, $mother, $father) {  
        echo 'You created ' . $who . '! <br />';  
        echo 'Mother: ' . $mother . '<br />';  
        echo 'Father: ' . $father . '<br /><br />';  
    }  
}  
  
$masterProxy = new masterProxyClass('testProxy', './masterProxyClass.php', array('Value One Works', 'Value Two Works'));  
  
$masterProxy->create('Kenny', 'Daisy', 'Bob');  
  
$masterProxy->kill('Kenny');
```