# 솔루션 상호 운용성 인증지원
# 공개SW 솔루션 개발자들을 위한 PHP Convention Guide

한국소프트웨어진흥원
공개SW기술지원센터





## <Revision 정보>

| 일자 | VERSION | 변경내역 | 작성자 |
|---|---|---|---|
| 2007. 11. 09 | 0.1 | 초기 작성 | 박준완 |
| | | | |

# 목 차







## 1. 문서 개요

본 문서는 KIPA 공개S/W 기술지원 센터에서의 솔루션 상호운용성 인증지원 결과를 보고하기 위해 제작되었다.

### 가. 문서의 목적

다음과 같은 세부적인 목적을 달성하기 위하여 작성되었다.

0 PHP 웹 프로그램이 대중화 되고 널리 사용되면서 사용자가 증대 됨에 따라 개발자들이 개인적인 규칙을 적용하여 프로그램을 작성하는 사례가 늘고 있어, 차후 수정이나 내용 파악이 어려운 경우가 많이 발생하고 있다. 이에 따라 프로그램을 작성하면서 지켜야 할 기본적인 규칙을 지킴으로써, 다른 사람들이 보다 편하게 이해하고 수정할 수 있는 환경을 조성한다.

### 나. 본 문서의 사용방법

다음과 같은 방법으로 사용할 수 있다.

0 PHP를 이용해 개발시 기본적인 규약으로 참조한다.

0 초급 프로그래머들이 참조하여 활용할 수 있다.

### 다. 참고사항

0 개발지원시 시스템 관련사항, 기술적 배경 등 참고사항들을 기술한다.

## 2. GForge: PHP 코드 표준 규약



**0. 소개글**

코드 표준 규약.

==========

**1. 주석**

**가이드라인**

● C++ 스타일 주석(/* */)과 표준 C 주석 (//) 두 개를 사용합니다.
● perl/shell 타입의 주석인 (#)은 금지합니다.

**PHPdoc 태그**

클래스 내의 주석은 PHPdoc 주석을 따릅니다. 자세한 사항은 아래 페이지를 참조하세요

http://www.phpdoc.de/

**파일 주석:**

모든 파일은 주석 내에 버전, 저자, copyright 메시지 등을 포함하여 시작합니다. 주석은 CVS Id 태그와 함께 JavaDoc 포맷 표준을 따릅니다. 모든 JavaDoc 태그들은 허용됩니다.

GForge는 혼합된 copyright를 가질 수 있습니다.

```
/**
 *
 * brief description.
 * long description.  more long description.
 *
 * Portions Copyright 1999-2001 (c) VA Linux Systems
 * The rest Copyright 2002 (c) their respective authors
 *
 * @version    $Id: coding_standards.html,v 1.3 2001/05/17 17:19:37 dbrogdon Exp $
 *
 */
```

**함수와 클래스 주석:**

비슷하게도, 모든 함수는 주석에 특별한 이름, 파라미터, 리턴 값, 그리고 마지막 변경 일자를 가집니다.

```
/**
 * brief description.
 * long description.  more long description.
 *
 * @author      firstname lastname email
 * @param       variable  description
 * @return      value       description
 * @date        YYYY-MM-DD
 * @deprecated
 * @see
 *
 */
```

==========

**2. 포멧**

**들여쓰기**

모든 들여쓰기는 탭을 사용합니다. CVS를 이용해 파일을 보내기 전에, 먼저 탭으로 들여쓰기를 했는지 여부를 확인하세요
(참고 : pear 표준의 경우 탭 대신 4개의 공백을 사용합니다.)

**PHP Tags**

PHP 코드에는 <?php ?>를 사용합니다. <? ?>는 사용하지 않습니다. 이것은 php 코드를 다른 OS나 웹서버 셋업시에 이식할 수 있는 가장 정확한 방법입니다. 또한 XML parsers와 혼동되지

않습니다.

==========

### 3. 함수

#### 함수 부르기

함수를 부를때는 함수이름, 괄호 열기, 그리고 첫 번째 파라미터 사이에는 공백을 두지 않고 콤마(,)와 파라미터간 사이에는 공백을 두며, 마지막 파라미터와 괄호 닫기, 그리고 세미콜론 사이에는 공백을 두지 않습니다. 아래에 예가 있습니다.

```
$var = foo($bar, $baz, $quux);
```

위와 같이, 함수의 리턴값을 변수에 대입할 때 사용하는 등호의 좌우에는, 공백을 1개씩 사용합니다. 관련된 일련의 대입문에 대해서는 가독성을 향상시키기 위해 공백을 여러개 두어도 좋습니다.

```
$short        = foo($bar);
$long_variable = foo($baz);
```

#### 함수 정의

함수 정의는 유닉스 관례에 따릅니다.

```
function fooFunction($arg1, $arg2 = '') {
    if (condition) {
        statement;
    }
    return $val;
}
```

기본 값을 가지는 인수는 인수 리스트의 마지막에 둡니다. 특별한 경우를 제외하고, 함수는 의미가 있는 값을 되돌려 줍니다. 아래에는 약간 긴 예가 있습니다.

```
function connect(&$dsn, $persistent = false) {
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }
```





```
    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
```

==========

### 4. 명명규약

- 상수는 항상 대문자를 사용하며, 아래바와 함께 분할된 단어를 사용합니다. 일반적으로 상수 이름은 클래스나 패키지의 이름을 사용합니다. 예를 들어, DB:: 패키지에서 사용하는 모든 상수는 "DB_" 로 시작합니다.
- True와 false는 PHP 언어에서 상수처럼 사용할 수 있습니다. 하지만 사용자 정의 상수로 구분되어 소문자로 사용합니다.
- 함수 이름은 진행형이나 동사를 제안합니다: updateAddress, makeStateSelector
- 변수 이름은 부사나 명사를 제안합니다: UserName, Width
- 데이터베이스 테이블은 아래바와 함께 이름을 사용합니다: my_table
- 그리고 테이블의 인덱스 이름은 테이블의 언더바를 제외한 테이블의 이름을 포함합니다: mytable_field1field2
- 전역 변수는 설명하는 이름을 사용하며, 지역변수는 이름을 짧게 사용합니다.

```
$AddressInfo = array(...);

for($i=0; $i < count($list); $i++)
```

==========

### 5. 제어구조

제어 구조에는 if, for, while, switch 등이 있습니다. 아래는 가장 복잡한 제어구조인 if 문장의 예를 나타냅니다.

```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
```

제어 구조에서는 함수 콜과 구별하기 위해서, 제어 키워드와 괄호 열기 사이에 한 개의 공백을 둡니다. 가능한 일반적인 경우에도, 중괄호({})를 사용하는 것을 추천합니다. 중괄호를 붙이는 것만으로도 가독성이 향상되어, 새롭게 행을 추가했을 때에 논리적인 에러를 발생 시킬 확률이 줄어듭니다. switch 문장의 경우에는 다음과 같습니다.

For switch statements:

```
switch (condition) {
    case 1: {
        action1;
        break;
    }
    case 2: {
        action2;
        break;
    }
    default: {
        defaultaction;
        break;
    }
}
```

==========

**6. PHP 파일을 포함하는 경우**

무조건적으로 포함되는 클래스 파일이 있기 때문에 언제나 require_once가 사용됩니다. 조건적으로 포함하는 클래스 파일(예를 들어, factory methods)이 있기 때문에 include_once가 사용됩니다. 이들은 클래스 파일을 오직 한번 포함하는 것을 결정합니다. 같은 파일이 공유되었을 때 이것들이 혼합되는 것을 걱정할 필요가 없습니다. 파일이 포함될때 require_once 가 사용될 경우 include_once로 다시 포함하지는 않습니다. 노트: include_once와 require_once는 함수가 아니라 키워드입니다. 사용할 때는 쿼트(")가 아닌 어퍼스트로피(')를 사용합니다.

```
include('pre.php');
```





## 3. GForge: PHP 코드 표준 규약 원본 파일

**GForge: PHP Coding Standards**

---

0. Introduction
1. Comments
2. Formatting
3. Functions
4. Naming
5. Control Statements
6. Including PHP Files

---

**0. Introduction**

Coding Standards.

---

**1. Comments**

**Guidelines**

- C++ style comments (/* */) and standard C comments (//) are both acceptable.
- Use of perl/shell style comments (#) is prohibited.

**PHPdoc Tags**

Inline documentation for classes should follow the PHPDoc convention, similar to Javadoc. More information about PHPDoc can be found here:

http://www.phpdoc.de/

**File Comments:**

Every file should start with a comment block describing its purpose, version, author and a copyright message. The comment block should be a block comment in standard JavaDoc format along with a CVS Id tag. While all JavaDoc tags are allowed, only the tags in the examples below will be parsed by PHPdoc.

GForge contains a mixed copyright. For files that have been changed since the GForge fork, the following header should be used:

```
/**
 *
 * brief description.
 * long description.  more long description.
 *
 * Portions Copyright 1999-2001 (c) VA Linux Systems
 * The rest Copyright 2002 (c) their respective authors
 *
 * @version    $Id: coding_standards.html,v 1.3 2001/05/17 17:19:37 dbrogdon Exp $
 *
 */
```

**Function and Class Comments:**

Similarly, every function should have a block comment specifying name, parameters, return values, and last change date.

```
/**
 * brief description.
 * long description.  more long description.
 *
 * @author      firstname lastname email
 * @param       variable   description
 * @return      value      description
 * @date        YYYY-MM-DD
 * @deprecated
 * @see
 *
 */
```

---

**2. Formatting**

**Indenting**

All indenting is done with TABS. Before committing any file to CVS, make sure you first replace spaces with tabs and verify the formatting.





**PHP Tags**

The use of <?php ?> to delimit PHP code is required. Using <? ?> is not valid. This is the most portable way to include PHP code on differing operating systems and webserver setups. Also, XML parsers are confused by the shorthand syntax.

---

**3. Functions**

**Function Calls**

Functions shall be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo($bar, $baz, $quux);
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

```
$short              = foo($bar);
$long_variable      = foo($baz);
```

**Function Definitions**

Function declarations follow the unix convention:

```
function fooFunction($arg1, $arg2 = '') {
    if (condition) {
        statement;
    }
    return $val;
}
```

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate. Here is a slightly longer example:

```
function connect(&$dsn, $persistent = false) {
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
```

---

**4. Naming**

- Constants should always be uppercase, with underscores to separate words. Prefix constant names with the name of the class/package they are used in. For example, the constants used by the DB:: package all begin with "DB_".
- True and false are built in to the php language and behave like constants, but should be written in lowercase to distinguish them from user-defined constants.
- Function names should suggest an action or verb: updateAddress, makeStateSelector
- Variable names should suggest a property or noun: UserName, Width
- Database tables should be named with underscores _ between words like: my_table
- And indexes on tables should be named with the table name first with the underscores removed, then field names mytable_field1field2.
- Use descriptive names for variables used globally, use short names for variables used locally.

```
$AddressInfo = array(...);

for($i=0; $i < count($list); $i++)
```

---

**5. Control Structures**

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated form:





```
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
```

Control statements shall have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You should use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

```
switch (condition) {
    case 1: {
        action1;
        break;
    }
    case 2: {
        action2;
        break;
    }
    default: {
        defaultaction;
        break;
    }
}
```

---

**6. Including PHP Files**

Anywhere you are unconditionally including a class file, use require_once. Anywhere you are conditionally including a class file (for example, factory methods), use include_once. Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with require_once will not be included again by

include_once. Note: include_once and require_once are keywords, not functions. You don't need parentheses around the filename to be included, however you should do it anyway and use ' (apostrophes) not " (quotes):

include('pre.php');