PHPEmbed

Andrew Bosworth, Brian Shire, Facebook, inc

February 8, 2007

Contents

1 Motivation

PHP has solidified itself as the language of choice for many top internet properties largely due to its ease of use and seamless integration with apache and MySQL. It is also a fine scripting language for quick and easy data consumption or modification. Despite its suitability for web development and scripting, PHP is generally not the best choice for most standalone programs or servers. Unfortunately, when there are many developers actively maintaining a PHP code base for a website it creates a dangerous dependency to have a separate code path to the data layer in another language. If the cache keys or database schema were to change in the PHP code base then the corollary compiled program would be broken (and potentially corrupting data) until the code was updated and the program recompiled and restarted.

Faced with this problem we endeavored to embed the PHP Interpreter into C++ binaries. This isn't a novel endeavor by any means; the aforementioned seamless Apache integration uses the PHP Server API (SAPI) to accomplish exactly that. However, when we began attempting integration of our own we found that the SAPI was less than user friendly. There is almost no documentation for most of the functionality provided in the SAPI and the code base is large and often difficult to follow.

In order to make embedding PHP truly simple for all of our developers (and indeed the world) we developed this PHPEmbed library which is just a more accessible and simplified API built on top of the PHP SAPI.

2 Design

The design of the PHPEmbed library centers on the principle of simplicity. The problem with the existing PHP SAPI is that it is complicated and requires a substantial amount of understanding to accomplish even basic tasks. The goal was to build a library that required no knowledge of the inner workings of PHP and provided the most useful functionality "out of the box."

The PHPEmbed library also provides relatively complete functionality. In addition to providing clients the ability to call any PHP function from C++ and get back return values, all with automatic translation between basic PHP types and C++ POD types, PHPEmbed also provides an optional Standard Template Library (STL) extension to support a number of more complex type conversions such as converting PHP arrays to vectors, sets, or even hash maps. In order to support arbitrarily nested and weakly typed

PHP arrays there is a simple `php_array` class and iterator in C++ which can be used both to navigate PHP arrays internally but also to create arrays to pass to PHP functions. Finally, there is a PHP tokenizing library included although there is no example usage or support of any kind for that code at this time.

In general, this implementation errs towards copying data rather than mutating it in place or incrementing ref counts without regard to the longevity of the embedded object. Macros have been avoided since they tend make code harder to understand and debug and have no impact on the performance in this case.[1] The library is intended to remain easy to use and understand above all else.

3 Download

Download PHPEmbed at http://www.phpembed.org/download/

4 Installation

TODO

5 Usage

This section walks through a simple (and tragically pointless) application to demonstrate common usage of the PHPEmbed library. This example will work specifically with the `php_stl` object since it includes all the functionality in the library, but using the `php` object (which doesn't use the STL) would work similarly. References to specific functions or code will be typeset in `courier` font. Additionally, please note that going forward references to the PHPEmbed libraries and their specific instances will be indicated by lowercase courier font (`php`) and the actual programming language, code written in that language, and the interpreter for that language will be indicated with normal upper case font (PHP). So, for example, `php` functions are the member methods belonging to the `php` class written in C++ and defined in `php_cxx.h` while PHP functions are methods written in the PHP programming language and evaluated by the PHP interpreter.

[1] Bjarne Stroustrup would approve: http://www.research.att.com/~bs/bs_faq2.html

5.1 Initializing

Initializing the object is as simple as including the code and instantiating it.

```
#include "php_stl.h"
...
php_stl p;
```

Passing `true` as an optional argument to the `php_stl` creator will enable warnings when the `php` object has to type cast a result returned from PHP evaluation. It is generally recommended to only turn those warnings on for debugging or if it is always an error condition for data of non-exact type to be returned by the PHP functions being called. The behavior of these type mismatch errors can also be controlled by changing error output behavior as described in the next paragraph.

The `php` libraries have three different classes of output:

- *Operational Output* – This class contains all the strings produced in PHP by such commands as `echo`, `print`, or `printf`. The default behavior of the `php` libraries is to print these strings to stdout prepended with the string "PHP OUTPUT:". *NOTE:* Due to its primary application as a hypertext processor the default behavior in PHP is to only flush the output buffer when the PHP interpreter is destroyed; to get the output flushed sooner use the PHP function `ob_flush()`.

- *Messages* – This class contains all the strings which represent errors or warnings produced in the interpretation of PHP but *not* fatal errors for the PHP interpreter itself or the `php` object. Good examples are calling an undefined function in PHP or using a non-array in the PHP `foreach` construct. The default behavior of the `php` libraries is to print these strings to stderr prepended with the string "PHP MESSAGE:."

- *Errors* – This class contains all the error strings generated by the `php` object itself as it attempts to process the commands given. Such errors could result from improper argument passing, illegal function calling, or any generic malfunction in the core PHP interpreter code. The default behavior of the `php` libraries is to print this to stderr prepended with the string "PHP ERROR:." *NOTE:* The aforementioned optional type mismatch errors fall into this class of output.

The client can alter the default behavior by providing an alternative function to handle the output:

```
void print_null(const char *str) {}
void print_mine(const char *str) { printf("My Output: %s", str); }
...
p.set_message_function(print_null);
p.set_output_function(print_mine);
```

This code directs the `php` object to discard all message output and prepend all operational output with the string "My Output:" before printing it to stdout. Default error output behavior remains in place although it could have been changed as well by passing `print_null` or `print_mine` to `set_error_function`. It is not a requirement to change any of output behaviors.

The final phase of initialization for most clients would be loading the PHP code into the interpreter:

```
if(SUCCESS != p.load("usage.php")){
  printf("load failed\n");
  exit(1);
}
```

This also isn't a requirement; clients could define functions just by creating strings in C++ and calling `eval_string` with them. In most cases the most flexible thing to do is keep that functionality in a separate file so it can be changed without having to recompile the binary. Note the check for success at the end, this is important so the program doesn't get too far before realizing the `php` object is broken. This check can be done after any function call to detect if the `php` object has gotten into a bad state.

5.2 Calling functions

C++ is a strongly typed language so the interface with a weakly typed language like PHP requires that types be called out explicitly. Polymorphism in C++ requires that functions of the same name vary in argument signature (just changing the return type isn't enough). As a consequence there are a host of functions where the return type expected is specified in the function name such as `call_void`, `call_bool`, `call_double_arr`, and `call_string_string_hash_map`. Here is an example of usage:

```
long memused = p.call_long("memory_get_usage");
hash_set<string> ex;
ex = p.call_string_hash_set("get_loaded_extensions");
```

In the first call the PHP builtin function `memory_get_usage` is invoked which takes no arguments and returns the number of bytes currently allocated internally by the PHP interpreter. Observe that there are no parens in the function name. The return is given in a `long`, which is the default numeric type for PHP. *NOTE:* Any function calls that return bool, int, or unsigned int are really returning a long and the `php` object is just typecasting in C++.

The second call demonstrates that it is no more difficult to take advantage of the more complex return types. The PHP builtin `get_loaded_extensions` will return an array of the names of all the extensions loaded into the current build of PHP which is then converted into a `hash_set`. See the Function Reference section for a list of all the functions available.

5.3 Passing Arguments

Passing C++ arguments into PHP in PHPEmbed works much like passing format arguments into `printf`. The caller must provide a format string which identifies the type of each argument to follow and then follow it with arguments matching in number and type to those specified. Table 1 below provides the argument specifiers for each C++ data type. *Note that C-style strings (`char *`) must be null terminated and that `php_array` objects are passed by reference.)*

Table 1: PHPEmbed argument specifiers

`s`	`char *`
`i`	`int`
`l`	`long`
`d`	`double`
`b`	`bool`
`a`	`php_array *`

In the following example, two strings and an integer are passed into PHP and a long is returned from the PHP builtin `strncmp` which behaves like the standard C function of the same name.

```
char *one = "test1";
int comp = 4;
long match = p.call_long("strncmp", "ssi", one, "test2", comp);
```

The result should be 0 (indicating a match) even through the strings are different because this code compares only the first four characters.

5.4 Creating PHP Arrays

As long as the program has included either `php_cxx.h`, `php_stl.h`, or `php_arr.h`, a `php_array` object can be instantiated as follows:

```
php_array a;
php_array b(a);
```

The second `php_array` object `b` is created as a deep copy of the first object `a` although it is a bit pointless in this example as `a` is still empty.

There are three ways to add elements to a `php_array` in C++ (although just as with PHP, clients aren't limited to using any single method for any `php_array` object):

- *Associative* These entries map string values to arbitrary PHP values just as `$arr["blah"] = $foo` maps the string "blah" to the value of the variable `$foo` in a PHP script.
- *Indexed* These entries map long values to arbitrary PHP values just as `$arr[5] = $foo` maps the long 5 to the value of the variable `$foo` in a PHP script.
- *Enumerated* These entries insert an arbitrary value into the array with the next available index just as `$a[] = 5` would do with the value 5 in a PHP script.

Inserting data into the `php_array` uses the same mechanism as passing arguments to PHP functions. Refer to Table 1 on page 8 for the list of argument specifiers.

```
a.add("l", 5);
a.add_assoc("slsd", "one", 1, "two", 2.5);
a.add_index("lsll", 6, "six", 128, 129);
```

It is important to observe that when adding associative elements each even argument (starting with the 0th element) must be a string. Similarly, when adding indexed elements each even argument (again starting with the 0th element) must be a `long`.

Data can also be removed from the array by associative or numerical index

```
a.remove(6);
a.remove("two");
```

Passing these arrays into PHP as the argument to a function then is as simple as using the `php_array` argument specifier and passing a reference to the object.

```
p.call_void("print_r", "a", &a);
```

This particular call, in the context of the other code above, produces the following operational output:

```
My Output: Array
(
    [0] => 5
    [one] => 1
    [128] => 129
)
```

5.5 Navigating PHP Arrays

Exploring the contents of `php_array` objects in C++ requires the use of the `php_iterator` class. This class must be initialized with a specific array and has an additional optional boolean argument for type warnings much like the base `php` class.

```
php_iterator it(a);
```

Just initializing the iterator causes it to jump to the first element in the array with a call to `go_to_start` but it is also possible to start at the end by calling `go_to_end`. These functions can also be used to reset the position of the iterator at any point during execution.

The `++` and `--` operators are used to navigate forward and backwards in the array a single element at a time and the `done` function is used to check for when the iterator has gone beyond the boundary of the `php_array` in any direction.

```
int count = 0;
while(!it.done()){
  count++;
  it++;
}
```

At the end of this loop the value of `count` should be the same as the return of a call to the `size` function of the `it` object.

Navigation wouldn't be very useful without the ability to get data back out of the array. However, since any entry could contain data of any type, there are functions for checking the type of key and type of data at each element. See Table 2 for a list of all supported and unsupported types.

Table 2: `php_type` values

Supported	Unsupported
`IS_LONG`	`IS_NULL`
`IS_DOUBLE`	`IS_OBJECT`
`IS_STRING`	`IS_RESOURCE`
`IS_BOOL`	`IS_CONSTANT`
`IS_ARRAY`	anything else

By using the type information the current position of the iterator, the data can be safely interpreted. *NOTE:* While the data at any array element could be any of the types listed in Table 2, keys can only be of type `IS_STRING` or `IS_LONG`. Consider the following code which prints out the key and value types and data for the last object in a iterator (doing nothing if the array is empty).

```
it.go_to_end();
if(!it.done()){
  switch(it.get_key_type()){
    case IS_LONG:
      printf("long %ld => ", it.get_key_long());
      break;
    case IS_STRING:
      printf("string %s => ", it.get_key_c_string());
      break;
    default:
      printf("??? %s => ", it.get_key_c_string());
      break;
    }

  switch(it.get_data_type()){
    case IS_LONG:
```

```
      printf("long %ld\n", it.get_data_long());
      break;
    case IS_STRING:
      printf("string %s\n", it.get_data_c_string());
      break;
    case IS_DOUBLE:
      printf("double %f\n", it.get_data_double());
      break;
    case IS_BOOL:
      printf("bool %s\n", it.get_data_c_string());
      break;
    case IS_ARRAY:
      printf("Array\n");
      {
        php_array suba = it.get_data_array();
        php_iterator subit(suba);
        // now iterate on the sub array
      }
      break;
    default:
      printf("??? %s\n", it.get_data_c_string());
      break;
  }
}
```

Note that in several cases PHP type conversion is used to get the data in a C-Style string even when it has a known type. For boolean values this makes sense for output since C++ would only print 0 or 1 if the data were printed as an integer. It also works with the data of unknown type since those must include a mechanism for printing as a string and will provide better insight into what data was provided such as a resource of or an object name. Of course, since it is only printing the data underneath it anyways this code could also just ignore the type of the data underlying entirely and always print the C style string data, but that wouldn't make a very informative example.

Observe also in the above example that in the case that the data is an array a new `php_array` object is created and initialized to that sub array. Unlike the majority of operations in PHPEmbed, the sub array `suba` is actually a reference to the same data that exists in the `a` object so mutating it by using the `add` or `remove` methods of `php_array` will affect both `suba`

and `a`.

5.6 Additional Examples

The example discussed in this section is implemented in the provided source code as `usage.cpp`. The source code also includes `example.cpp` and `example.php` which implement similar simple examples. In addition to that, there is a (relatively superficial) unit testing framework in `test.cpp` and `test.php` which will provide a working example of nearly every function available in the API.

6 Frequently Asked Questions

6.1 *Why not use macros instead of repeating so much code?*

Macros can be convenient but they can also make code difficult to read and understand. Furthermore, Macros only provide a potential improvement in performance when inlining otherwise large and unwieldy sections of code which doesn't apply to this library. We hate code replication as much as anyone but we believe in clarity above all else.

6.2 *Why do I keep running out of memory?*

PHP rarely runs beyond its memory limit in apache because the instance only lives for the time it takes to serve one page and then that thread dies and the memory is freed. The embedded environment, however, sticks around and hence requires vigilance around declaring variables in global scope and failing to clean them up with unset. Additionally, it may help to increase the allowed operating memory by using the `ini_set` function before loading any PHP scripts, for example: `ini_set('memory_limit', '100M');`

There are also a few known leaks in the PHP core, using the builtin PHP function `memory_get_usage()` in conjunction with the `memory_limit` ini setting will enable any client of PHPEmbed to predict a memory problem before it affects program operation and handle it as appropriate, for example by destroying the PHP object and recreating it.

6.3 *Can I have multiple instances of PHPEmbed in one program?*

No. Although the SAPI provides thread safe resource management and access we haven't written this library to support it.

6.4 *Will this library work on my platform?*

We have only tested this software on 64-bit linux architectures, and there is no guarantee it will even work there. That being said, if you can compile PHP on your machine you should be able to compile and use PHPEmbed.

7 Function Reference

7.1 php

```
#include "php_cxx.h"

class php {
public:
  php(bool type_warnings = false);
  ~php();
  ...
};
```

Description: The `php` object is the center of access for the embedded PHP interpreter. The `type_warnings` value is optional and defaults to `false`. If set to `true` the interpreter will send a string to the error function (see `php::set_error_function`) when it is forced to use PHP type conversion to get an object into the required C++ data type.

Errors: Any errors during initialization will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.2 php::call_bool

```
#include "php_cxx.h"

bool
php::call_bool(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The value returned by PHP will be interpreted as a `bool`. If PHP type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error

for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.3 php::call_bool_arr

```
#include "php_cxx.h"

bool *
php::call_bool_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `bool` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a `bool` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not an array `call_bool_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.4 php::call_c_string

```
#include "php_cxx.h"

char *
php::call_c_string(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified

the function will be called without any arguments. The value returned by PHP will be interpreted as a C style string. If PHP type conversion is necessary it may generate a type mismatch warning if those warnings are enabled. **The memory for this string is maintained by the `php` object so if that object is destroyed the string is no longer valid; if the string value needs to persist beyond the life of the `php` object then it must be copied.**

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.5 `php::call_c_string_arr`

```
#include "php_cxx.h"

char **
php::call_c_string_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `char *` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a string it may generate a type mismatch warning if those warnings are enabled. **The memory for each individual string in the array is maintained by the `php` object so if that object is destroyed the pointers in the array are no longer valid; if the string values need to persist beyond the life of the `php` object then they must be copied.**

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not an

array `call_double_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.6 `php::call_double`

```
#include "php_cxx.h"

double
php::call_double(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The value returned by PHP will be interpreted as a `double`. If PHP type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.7 `php::call_double_arr`

```
#include "php_cxx.h"

double *
php::call_double_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `double` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not an array `call_double_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.8 `php::call_int_array`

```
#include "php_cxx.h"

int
php::call_int_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `int` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a `int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not an array `call_int_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.9 php::call_long

```
#include "php_cxx.h"

long
php::call_long(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The value returned by PHP will be interpreted as a `long`. If PHP type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.10 php::call_long_array

```
#include "php_cxx.h"

long *
php::call_long_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `long` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not

an array `call_long_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.11 php::call_php_array

```
#include "php_cxx.h"

php_array
php::call_php_array(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The value returned by PHP will be interpreted as an array. If PHP type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.12 php::call_uint_array

```
#include "php_cxx.h"

unsigned int *
php::call_uint_arr(size_t *size, char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put into a C array of type `unsigned int` large enough to hold them. The size of that array will be returned to the client via the `size` argument and **it is the responsibility of the client to free the array returned when they are done.** Each time PHP type conversion is necessary to convert any element of the returned array to a `unsigned int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric

values in PHP are of type `long`, this function simply typecasts each value from `long` to `unsigned int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). It is an error for the `size` argument to be passed in as `NULL`. If the value returned by PHP is not an array `call_uint_arr` will output a type mismatch error (regardless of whether type warnings are on or not) and return `NULL`. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.13 php::call_void

```
#include "php_cxx.h"

void
php::call_void(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. Any value returned by PHP will be ignored.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.14 php::eval_string

```
#include "php_cxx.h"

void
php::eval_string(const char *fmt, ...);
```

Description: Use the PHP interpreter to evaluate string `fmt` with any `printf` style format arguments substituted for the value in the additional arguments. *NOTE:* This function is made available for more advanced clients but its use is generally discouraged.

Example: The `php::load` function is actually implemented very simply as:

```
php_ret php::load(const char *filename)
{
  return eval_string("include_once('%s');", filename);
}
```

Note the inclusion of the semicolon as the interpreter will treat the string being evaluated as a line of PHP code.

Errors: It is an error for the types or count of format specifiers in `fmt` to differ from the additional arguments passed. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.15 php::load

```
#include "php_cxx.h"

void
php::load(const char *filename);
```

Description: Direct the PHP interpreter to parse all the code in `filename`. This function is really just a wrapper around the PHP builtin `include_once`.

Errors: Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.16 php::set_error_function

```
#include "php_cxx.h"

void
set_error_function(void (*error_function)(const char *));
```

Description: This function controls the behavior of error output from the `php` object. This class of output contains all the error strings generated by the `php` object itself as it attempts to process the commands given. Such errors could result from improper argument passing, illegal function calling, or any generic malfunction in the core PHP interpreter code. The default behavior of the `php` libraries is to print this to stderr prepended with the string "PHP ERROR:." *NOTE:* The optional type mismatch errors in the `PHP` object fall into this class of output. If set, `php` will call `error_function` with a `const char *` string containing the error.

7.17 php::set_message_function

```
#include "php_cxx.h"

void
set_message_function(void (*message_function)(const char *));
```

Description: This function controls the behavior of message output from the `php` object. This class of output contains all the strings which represent errors or warnings produced in the interpretation of PHP but *not* fatal errors for the PHP interpreter itself or the `php` object. Good examples are calling an undefined function in PHP or using a non-array in the PHP `foreach` construct. The default behavior of the `php` libraries is to print these strings to stderr prepended with the string "PHP MESSAGE:." If set, `php` will call `message_function` with a `const char *` string containing the message.

7.18 php::set_output_function

```
#include "php_cxx.h"

void
set_output_function(void (*output_function)(const char *));
```

Description: This function controls the behavior of operational output from the `php` object. This class of output contains all the strings produced in PHP by such commands as `echo`, `print`, or `printf`. The default behavior of the `php` libraries is to print these strings to stdout prepended with the string "PHP OUTPUT:". *NOTE:* Due to its primarily application as a hypertext processor the default behavior in PHP is to only flush the output buffer when the PHP interpreter is destroyed; to get the output flushed sooner use the PHP function `ob_flush()`. If set, `php` will call `output_function` with a `const char *` string containing the output.

7.19 php::status

```
#include "php_cxx.h"

class php{
public:
   ...
   php_ret status;
};
```

Description: This is a public variable of the `php` class that indicates the status of the class, generally in reference to the most recently executed command. If the `status` is ever nonzero the class should be destroyed. Although nearly every operation in the `php` and `php_stl` classes has the potential to affect `status`, it may be sufficient to rely on error output to detect problems and only check `status` after initialization.

7.20 php_array

```
#include "php_arr.h"

class php_array{
public:
   php_array();
   ~php_array();
};
```

Description: The `php_array` object is an object which implements the creation and storage of nested and arbitrarily typed key-value pairs for transporting complex data to and from PHP functions.

7.21 php_array::add

```
#include "php_arr.h"

void
php_array::add(char *argspec, ...);
```

Description: Add each value described by `argspec` and included as subsequent arguments to the `php_array` at the next available index. This is logically equivalent to taking each value passed in and calling `$php_array[] = $value;` in PHP.

Errors: It is an error for `argspec` to be undefined. It is an error for the number of values described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec` (see Table 1 on page 8 for details). Any errors during execution may result in fewer values than expected being inserted into the array.

7.22 php_array::add_assoc

```
#include "php_arr.h"

void
php_array::add_assoc(char *argspec, ...);
```

Description: Add each pair of values described by `argspec` and included as subsequent arguments to the `php_array` as key value pairs. This is logically equivalent to taking each key-value pair passed in and calling `$php_array[$key] = $value;` in PHP. Since this function adds associative keys to the array, all even values in `argspec` must be specified as string values (beginning with the 0th index). If a key provided as an argument already exists in the array, its value will be overwritten with the new value provided.

Errors: It is an error for `argspec` to be undefined. It is an error for the number of values described in `argspec` to differ from the number of arguments provided thereafter. It is an error for an odd number of values to be passed in. It is an error for the types of additional arguments to differ from their description in `argspec` (see Table 1 on page 8 for details). It is an error for any even values to have type other than `char *` since associative keys must be strings. Any errors during execution may result in fewer values than expected being inserted into the array.

7.23 php_array::add_index

```
#include "php_arr.h"

void
php_array::add_index(char *argspec, ...);
```

Description: Add each pair of values described by `argspec` and included as subsequent arguments to the `php_array` as key value pairs. This is logically equivalent to taking each key-value pair passed in and calling `$php_array[$key] = $value;` in PHP. Since this function adds indexed keys to the array, all even values in `argspec` must be specified as `long` values (beginning with the 0th index). If a key provided as an argument already exists in the array, its value will be overwritten with the new value provided.

Errors: It is an error for `argspec` to be undefined. It is an error for the number of values described in `argspec` to differ from the number of

arguments provided thereafter. It is an error for an odd number of values to be passed in. It is an error for the types of additional arguments to differ from their description in `argspec` (see Table 1 on page 8 for details). It is an error for any even values to have type other than `long` since indexed keys must be numeric. Any errors during execution may result in fewer values than expected being inserted into the array.

7.24 php_array::remove

```
#include "php_arr.h"

void
php_array::remove(char *key);

void
php_array::remove(long index);
```

Description: Remove `key` or `index` and its related value from the `php_array`. This is logically equivalent to calling either `unset($php_array[$key]);` or `unset($php_array[$index]);` in PHP. Calling remove on a key or an index that does not exist in the array does nothing.

7.25 php_iterator

```
#include "php_arr.h"

class php_iterator {
public:
  php_iterator(php_array &a, bool type_warnings = false);
  ~php_iterator();
  ...
};
```

Description: The `php_iterator` object provides access to the keys and values stored in a `php_array` object. The required `php_array a` argument is the array this iterator will provide access to. The `type_warnings` value is optional and defaults to `false`. If set to `true` the iterator will write an error to stderr when it is forced to use convert a key or value into the required C++ data type.

7.26 php_iterator::done

```
#include "php_arr.h"

bool
php_iterator::done();
```

Description: Returns `true` if and only if the iterator no longer points to a valid `php_array` element, either because the array is empty or the iterator has been advanced beyond the bounds of the array in either direction.

7.27 php_iterator::get_data_type

```
#include "php_arr.h"

php_type
php_iterator::get_data_type();
```

Description: Returns the data type of the value at the current position of the iterator. See `php_ret` for the list of types.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.28 php_iterator::get_data_c_string

```
#include "php_arr.h"

char *
php_iterator::get_data_c_string();
```

Description: Return the value of the element at the current iterator position as a `char *`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.29 php_iterator::get_data_double

```
#include "php_arr.h"

double
php_iterator::get_data_double();
```

Description: Return the value of the element at the current iterator position as a `double`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.30 php_iterator::get_data_long

```
#include "php_arr.h"

long
php_iterator::get_data_long();
```

Description: Return the value of the element at the current iterator position as a `long`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.31 php_iterator::get_data_bool

```
#include "php_arr.h"

bool
php_iterator::get_data_bool();
```

Description: Return the value of the element at the current iterator position as a `bool`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.32 php_iterator::get_data_array

```
#include "php_arr.h"

php_array
php_iterator::get_data_array();
```

Description: Return the value of the element at the current iterator position as a `php_array`. This function can be used in conjunction with

an additional iterator to navigate nested arrays. If the value at the current iterator position isn't of type `IS_ARRAY` then this function will return a new `php_array` with a single element whose key is 0 and whose value is the same value and type of the data at the current iterator position and may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.33 php_iterator::get_key_type

```
#include "php_arr.h"

php_type
php_iterator::get_key_type();
```

Description: Return the data type of the key at the current position of the iterator. See `php_ret` for the list of types.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.34 php_iterator::get_key_c_string

```
#include "php_arr.h"

char *
php_iterator::get_key_c_string();
```

Description: Return the key of the element at the current iterator position as a `char *`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.35 php_iterator::get_key_long

```
#include "php_arr.h"

long
php_iterator::get_key_long();
```

Description: Return the key of the element at the current iterator position as a `long`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error to call this function on an iterator in an invalid position (see `php_iterator::done`).

7.36 php_iterator::go_to_end

```
#include "php_arr.h"

void
php_iterator::go_to_end();
```

Description: Reset the position of the iterator to the last element in the `php_array`.

7.37 php_iterator::go_to_start

```
#include "php_arr.h"

void
php_iterator::go_to_end();
```

Description: Reset the position of the iterator to the first element in the `php_array`.

7.38 php_iterator::operator--

```
#include "php_arr.h"

void
php_iterator::operator--(int ignore);
```

Description: Move the iterator to the previous element in the `php_array`. If it was previously on the first element the iterator will no longer be valid after this operation. The client should always call `php_iterator::done` before trying to access data at the new iterator position. The `ignore` parameter just indicates that this is a postfix expression.

7.39 php_iterator::operator++

```
#include "php_arr.h"

void
php_iterator::operator++(int ignore);
```

Description: Move the iterator to the next element in the `php_array`. If it was previously on the last element the iterator will no longer be valid after this operation. The client should always call `php_iterator::done` before trying to access data at the new iterator position. The `ignore` parameter just indicates that this is a postfix expression.

7.40 php_iterator::size

```
#include "php_arr.h"

int
php_iterator::size();
```

Description: Return the number of elements in the `php_array` associated with this iterator.

7.41 php_ret

```
#include "php_cxx.h"

#define SUCCESS 0
#define FAIL 1

typedef unsigned int php_ret;
```

Description: This type is an unsigned integer value which represents a failure when given a nonzero value.

7.42 php_stl

```
#include "php_stl.h"

class php_stl : public php {
public:
  php(bool type_warnings = false);
```

```
  ~php();
  ...
};
```

Description: The `php_stl` object extends the basic `php` object and provides support for Standard Template Library (STL) types in addition to all the basic `php` functionality. The `type_warnings` value is optional and defaults to `false` If set to `true` the interpreter will send a string to the error function (see `php::set_error_function`) when it is forced to convert a value into the required type for the given STL data structure.

Errors: Any errors during initialization will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.43 php_stl::call_string

```
#include "php_stl.h"

string
php::call_string(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The value returned by PHP will be interpreted as a `string`. If type conversion is necessary it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.44 php_stl::call_string_vector

```
#include "php_stl.h"

vector<string>
php::call_string_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified

the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be pushed in order onto an STL vector of type `string`. Each time it is necessary to convert a value in the array to a `string` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.45 php_stl::call_double_vector

```
#include "php_stl.h"

vector<double>
php::call_double_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values of that array will be put in order an STL vector of type `double`. Each time it is necessary to convert a value in the array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_double_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.46 php_stl::call_long_vector

```
#include "php_stl.h"
```

```
vector<long>
php::call_long_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be pushed in order onto an STL vector of type `long`. Each time it is necessary to convert a value in the array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.47 `php_stl::call_bool_vector`

```
#include "php_stl.h"

vector<bool>
php::call_bool_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be pushed in order onto an STL vector of type `bool`. Each time it is necessary to convert a value in the array to a `bool` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_bool_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.48 php_stl::call_int_vector

```
#include "php_stl.h"

vector<int>
php::call_int_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be pushed in order onto an STL vector of type `int`. Each time it is necessary to convert an value in the array to an `int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_int_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.49 php_stl::call_uint_vector

```
#include "php_stl.h"

vector<unsigned int>
php::call_uint_vector(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be pushed in order onto an STL vector of type `unsigned int`. Each time it is necessary to convert a value in the array to an `unsigned int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `unsigned int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_uint_vector` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty vector. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.50 php_stl::call_string_set

```
#include "php_stl.h"

set<string>
php::call_string_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL set of type `string`. Each time it is necessary to convert a value in the array to a `string` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_set` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty set. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.51 php_stl::call_double_set

```
#include "php_stl.h"

set<double>
php::call_double_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified

the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL set of type `double`. Each time it is necessary to convert a value in the array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_double_set` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty set. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.52 php_stl::call_long_set

```
#include "php_stl.h"

set<long>
php::call_long_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL set of type `long`. Each time it is necessary to convert a value in the array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_set` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty set. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.53 php_stl::call_int_set

```
#include "php_stl.h"
```

```
set<int>
php::call_int_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL set of type `int`. Each time it is necessary to convert a value in the array to an `int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_int_set` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty set. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.54 php_stl::call_uint_set

```
#include "php_stl.h"

set<unsigned int>
php::call_uint_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL set of type `unsigned int`. Each time it is necessary to convert a value in the array to an `unsigned int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `unsigned int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in

argspec(see Table 1 on page 8 for details). If the value returned by PHP is not an array call_uint_set will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty set. Any internal errors during execution will result in the status flag (see php::status) being set to a nonzero value.

7.55 php_stl::call_string_hash_set

```
#include "php_stl.h"

hash_set<string>
php::call_string_hash_set(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function fn with the arguments described by argspec and included as subsequent arguments. If argspec is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and the values in that array will be inserted into an STL hash set of type string. Each time it is necessary to convert a value in the array to a string it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in argspec to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in argspec(see Table 1 on page 8 for details). If the value returned by PHP is not an array call_string_hash_set will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty hash set. Any internal errors during execution will result in the status flag (see php::status) being set to a nonzero value.

7.56 php_stl::call_string_string_map

```
#include "php_stl.h"

map<string, string>
php::call_string_string_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function fn with the arguments described by argspec and included as subsequent arguments. If argspec is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion

is necessary to convert a key or value in the returned array to a `string` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_string_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.57 php_stl::call_string_double_map

```
#include "php_stl.h"

map<string, double>
php::call_string_double_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_double_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.58 php_stl::call_string_long_map

```
#include "php_stl.h"

map<string, long>
php::call_string_long_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_long_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.59 php_stl::call_string_bool_map

```
#include "php_stl.h"

map<string, bool>
php::call_string_bool_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `bool` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_bool_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.60 php_stl::call_string_int_map

```
#include "php_stl.h"

map<string, int>
php::call_string_int_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to an `int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_int_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.61 php_stl::call_string_uint_map

```
#include "php_stl.h"

map<string, unsigned int>
php::call_string_uint_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to an `unsigned int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `unsigned int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_uint_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.62 php_stl::call_long_string_map

```
#include "php_stl.h"

map<long, string>
php::call_long_string_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `long` or a value in the array to a `string` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_string_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.63 php_stl::call_long_double_map

```
#include "php_stl.h"

map<long, double>
php::call_long_double_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `long` or a value in the array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_double_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.64 php_stl::call_long_long_map

```
#include "php_stl.h"

map<long, long>
php::call_long_long_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key or value in the array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_long_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.65 php_stl::call_long_bool_map

```
#include "php_stl.h"

map<long, bool>
php::call_long_bool_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `long` or a value in the array to a `bool` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_bool_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.66 php_stl::call_long_int_map

```
#include "php_stl.h"

map<long, int>
php::call_long_int_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `long` or a value in the array to an `int` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an er-

ror for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_int_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.67 php_stl::call_long_uint_map

```
#include "php_stl.h"

map<long, unsigned int>
php::call_long_uint_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL map as such. Each time PHP type conversion is necessary to convert a key in the array to a `long` or a value in the array to an `unsigned int` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_long_uint_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.68 php_stl::call_string_string_hash_map

```
#include "php_stl.h"

map<string, string>
php::call_string_string_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value

returned from PHP must be an array and each key-value pair in that array will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key or value in the returned array to a `string` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_string_hash_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.69 php_stl::call_string_double_hash_map

```
#include "php_stl.h"

map<string, double>
php::call_string_double_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `double` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_double_hash_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.70 php_stl::call_string_long_hash_map

```
#include "php_stl.h"
```

```
map<string, long>
php::call_string_long_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `long` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_long_hash_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.71 php_stl::call_string_bool_hash_map

```
#include "php_stl.h"

map<string, bool>
php::call_string_bool_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to a `bool` it may generate a type mismatch warning if those warnings are enabled.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_bool_hash_map` will output a type mismatch

error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.72 php_stl::call_string_int_hash_map

```
#include "php_stl.h"

map<string, int>
php::call_string_int_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to an `int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_int_hash_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.73 php_stl::call_string_uint_hash_map

```
#include "php_stl.h"

map<string, unsigned int>
php::call_string_uint_hash_map(char *fn, char *argspec = "", ...);
```

Description: Call the PHP function `fn` with the arguments described by `argspec` and included as subsequent arguments. If `argspec` is not specified the function will be called without any arguments. The type of the value returned from PHP must be an array and each key-value pair in that array

will be inserted into an STL hash map as such. Each time PHP type conversion is necessary to convert a key in the array to a `string` or a value in the array to an `unsigned int` it may generate a type mismatch warning if those warnings are enabled. *NOTE:* Since all numeric values in PHP are of type `long`, this function simply typecasts each value from `long` to `unsigned int` in C++.

Errors: It is an error for the number of arguments described in `argspec` to differ from the number of arguments provided thereafter. It is an error for the types of additional arguments to differ from their description in `argspec`(see Table 1 on page 8 for details). If the value returned by PHP is not an array `call_string_uint_hash_map` will output a type mismatch error (regardless of whether type warnings are on or not) and return an empty map. Any internal errors during execution will result in the `status` flag (see `php::status`) being set to a nonzero value.

7.74 php_type

```
#include "php_arr.h"

typedef zend_uchar php_type;
```

Description: This is just a wrapper for an unsigned char type and it represents the possible data types for values in PHP. This is particularly useful when working with PHP arrays since any key or value can be of nearly any type. The following table (reproduced from Table 2) details the possible return types as well as which types are supported in C++ and which are not.

Supported	Unsupported
IS_LONG	IS_NULL
IS_DOUBLE	IS_OBJECT
IS_STRING	IS_RESOURCE
IS_BOOL	IS_CONSTANT
IS_ARRAY	anything else

7.75 php_tok

This library could be used to tokenize an arbitrary number of PHP files hierarchically for preprocessing in C++, but there is no example usage,

active development, or support for it at this time.

8 License and Copyright

9 Contact

Post feedback, patches, or bug reports to the discussion board at
http://www.phpembed.org/bboard/