



Getting Started with PHP-FFI

Thomas Bley, June 2021 @ ipc

About me

- Senior PHP Developer
- Linux, PHP, MySQL since 2001
- studied at TU München
- working for Bringmeister in Berlin



What is PHP-FFI?

- PHP Foreign Function Interface
- PHP extension
- requires PHP 7.4
- currently experimental
- allows
 - loading of **shared libraries** (.so)
 - **calling C functions** and accessing C data structures
 - **in pure PHP** during runtime
- requires converting PHP types to C types

source:

php.net/manual/en/intro.ffi.php

phpconference.com/blog/php-ffi-and-what-it-can-do-for-you/



Your Code Runs in Docker, Why Not Your IDE?

PHP-FFI use cases

- call C/C++ as shared libraries
 - allows direct hardware access (e.g. webcams, GPU)
 - gives more performance (e.g. image or video rendering)
 - integrate databases that have no PHP integration (e.g. DuckDB)
- call Go / Rust compiled as shared libraries
- easier than writing PHP extensions

source: github.com/gabrielrcouto/awesome-php-ffi



What is Go?

- programming language
- designed by Google
- statically typed
- memory safety, fast garbage collection
- more performance and concurrency
- can be compiled as a shared library

→ combining PHP and Go gives a lot of new opportunities

source: [en.wikipedia.org/wiki/Go_\(programming_language\)](https://en.wikipedia.org/wiki/Go_(programming_language))



PHP-Example: Ackermann function

```
<?php
```

```
function ackermann(int $n, int $m): int {  
    if ($n == 0) {  
        return $m + 1;  
    } else if ($m == 0) {  
        return ackermann($n - 1, 1);  
    } else {  
        return ackermann($n - 1, ackermann($n, $m - 1));  
    }  
}  
echo ackermann(3, 11);
```

```
// time php -dopcache.enable_cli=0 ackermann.php # 12.8s
```

```
// time php -dopcache.enable_cli=1 ackermann.php # 4.8s
```

```
// time php -dopcache.enable_cli=1 -dopcache.jit_buffer_size=32M ackermann.php # 3.2s
```

source: en.wikipedia.org/wiki/Ackermann_function
rosettacode.org/wiki/Ackermann_function



Go-Example: Ackermann function

```
package main
import "C"

func main() {
    println(ackermann(3, 11));
}

//export ackermann
func ackermann(n int, m int) int {
    if n == 0 {
        return m + 1
    } else if m == 0 {
        return ackermann(n-1, 1)
    }
    return ackermann(n-1, ackermann(n, m-1))
}

// time go run ackermann.go # 0.7s
// go build ackermann.go && time ./ackermann # 0.5s (C/C++/Rust: 0.2s)
```



Calling Go from PHP

```
// create ackermann.so and ackermann.h:
```

```
// go build -o ackermann.so -buildmode=c-shared ackermann.go
```

```
<?php
```

```
error_reporting(E_ALL);
```

```
$ffi = FFI::cdef('long ackermann(long n, long m);', __DIR__ . '/ackermann.so');
```

```
echo $ffi->ackermann(3, 11);
```

```
unset($ffi);
```

```
// time php -dffi.enable=1 ackermann.php # 0.5s
```

```
// cat ackermann.h | grep -E "ackermann|GoInt;|GoInt64"
```

```
// typedef long long GoInt64;
```

```
// typedef GoInt64 GoInt;
```

```
// extern GoInt ackermann(GoInt n, GoInt m);
```



Using complex data types

```
// php
$ffi = FFI::cdef('long ackermann(char* input);', __DIR__ . '/ackermann.so');
$result = $ffi->ackermann_json(json_encode([3, 11]));
echo json_decode(FFI::string($result));
FFI::free($result); // avoid memory leak
unset($ffi);

// go
import "encoding/json"
//export ackermann_json
func ackermann_json(input *C.char) *C.char {
    var params [2]int
    err := json.Unmarshal([]byte(C.GoString(input)), &params)
    if err != nil { return nil }

    data, err := json.Marshal(ackermann(params[0], params[1]))
    if err != nil { return nil }

    return C.CString(string(data))
}
```



Checking memory leaks

```
// php
$ffi = FFI::cdef('long ackermann(char* input);', __DIR__ . '/ackermann.so');

for ($i=0; $i < 100000000) {
    $result = $ffi->ackermann_json(json_encode([0, 1]));
    FFI::free($result);
}
unset($ffi);
```

```
php -dffi.enable=1 ackermann.php &
```

```
while :; do pmap -x `pgrep php` | grep total; sleep 1; done
total kB          1489596    44040    16884
total kB          1489596    43864    16708
total kB          1489596    44568    17412
total kB          1489596    44048    16892
total kB          1489596    43896    16740
...
```



What is DuckDB?

- embedded database (similar to SQLite)
- optimized for analytical queries (OLAP)
- optimized for large tables, large resultsets
- column storage, 1 database = 1 single file
- vectorized query execution engine
- open source
- very fast

but: no PHP extension available

source: duckdb.org/docs/why_duckdb



Getting started with DuckDB

- download [libduckdb-linux-amd64.zip](#)
- extract [duckdb.h](#) and [libduckdb.so](#)
- C preprocessor directives are not supported in header files
 - remove “`#ifdef __cplusplus...#endif`” from [duckdb.h](#)
 - remove “`DUCKDB_API` ” from [duckdb.h](#)
 - remove lines with “`duckdb_bind_uint8`”, “`duckdb_bind_uint16`”
- `FFI::cdef($header, $soLibrary)` — Creates a new FFI object
- `FFI::new($type)` — Creates a C data structure
- `FFI::addr($ptr)` — Creates an unmanaged pointer to C data



How to integrate DuckDB?

master

duckdb / examples / embedded-c / main.c

48 lines (46 sloc) 1.38 KB

```
1 #include "duckdb.h"
2 #include <stdio.h>
3
4 int main() {
5     duckdb_database db = NULL;
6     duckdb_connection con = NULL;
7     duckdb_result result;
8
9     if (duckdb_open(NULL, &db) == DuckDBError) {
10         fprintf(stderr, "Failed to open database\n");
11         goto cleanup;
12     }
13     if (duckdb_connect(db, &con) == DuckDBError) {
14         fprintf(stderr, "Failed to open connection\n");
15         goto cleanup;
16     }
17     if (duckdb_query(con, "CREATE TABLE integers(i INTEGER, j INTEGER)")
18         fprintf(stderr, "Failed to query database\n");
19         goto cleanup;
20     }
21     if (duckdb_query(con, "INSERT INTO integers VALUES (3, 4), (5, 6),")
22         fprintf(stderr, "Failed to query database\n");
23         goto cleanup;
24     }
25     if (duckdb_query(con, "SELECT * FROM integers", &result) == DuckDBError)
26         fprintf(stderr, "Failed to query database\n");
27     goto cleanup;
```

<?php

```
$ffi = FFI::cdef(
    file_get_contents(__DIR__ . '/duckdb.h'),
    __DIR__ . '/libduckdb.so');

$db = $ffi->new('duckdb_database');
$con = $ffi->new('duckdb_connection');
$result = $ffi->new('duckdb_result');

$ffi->duckdb_open(null, FFI::addr($db));
$ffi->duckdb_connect($db, FFI::addr($con));

$query = 'select current_date;';
$ffi->duckdb_query(
    $con, $query, FFI::addr($result));

$val = $ffi->duckdb_value_varchar(
    FFI::addr($result), 0, 0);
echo FFI::string($val);
```



Your Code Runs in Docker, Why Not Your IDE?

Cleanup resources

```
25     if (duckdb_query(con, "SELECT * FROM integers", &result) == D
26         fprintf(stderr, "Failed to query database\n");
27         goto cleanup;
28     }
29     // print the names of the result
30     for (size_t i = 0; i < result.column_count; i++) {
31         printf("%s ", result.columns[i].name);
32     }
33     printf("\n");
34     // print the data of the result
35     for (size_t row_idx = 0; row_idx < result.row_count; row_idx+
36         for (size_t col_idx = 0; col_idx < result.column_count; c
37             char *val = duckdb_value_varchar(&result, col_idx, ro
38             printf("%s ", val);
39             duckdb_free(val);
40         }
41         printf("\n");
42     }
43     // duckdb_print_result(result);
44     cleanup:
45     duckdb_destroy_result(&result);
46     duckdb_disconnect(&con);
47     duckdb_close(&db);
48 }
```

<?php

FFI::free(\$val);

\$ffi->duckdb_destroy_result(
 FFI::addr(\$result)
);

\$ffi->duckdb_disconnect(FFI::addr(\$con));
FFI::free(\$con);

\$ffi->duckdb_close(FFI::addr(\$db));
FFI::free(\$db);

unset(\$ffi);



DEMO



Thanks for listening!

Questions?

slides and sources:

github.com/thomasbley/php-go-integration

github.com/thomasbley/php-duckdb-integration



Your Code Runs in Docker, Why Not Your IDE?