Updating Custom PHP Libraries used by 4D

By Timothy Aaron Penner, Technical Services Engineer, 4D Inc.

Technical Note 19-03

Table of Contents

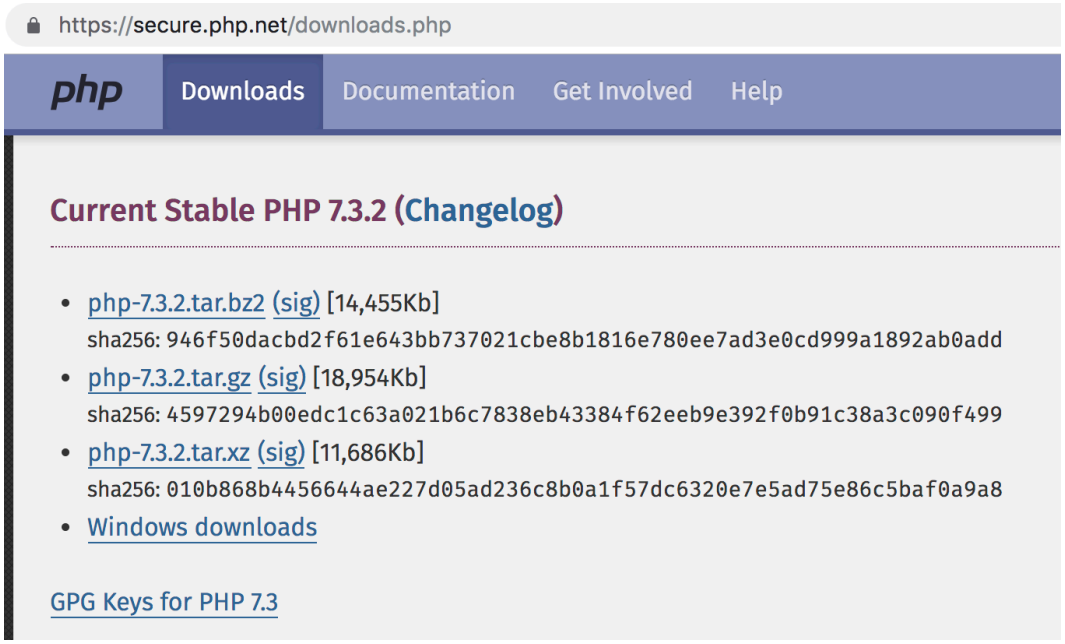
Introduction

In the next R-release of 4D, the version of PHP has been updated to be 7.3 x64 bit. The PHP interpreter included with 4D contains a certain subset of modules and extensions that have been tested by 4D. Adding additional modules or extensions is not supported. So, if the deployment requires a module that is not included by default then the developer will need to build a custom PHP only includes specific modules; adding modules is not supported.

If additional modules or extensions are needed beyond th3e basic set provided within 4D the developer would need to build a custom PHP engine. This documentation would be interesting for our customers who use or want to use a customer PHP engine.

Updating PHP on Mac

To upgrade PHP on Mac the developer will have to build it from the sources. The sources can be obtained from the official php website (php.net/downloads.php).



1) Compile extensions dependencies

Some extensions depend on external libraries, these must be compiled beforehand. Their sources can be found on their official website and sometimes their packages are also on pecl.php.net – in this case the extension can be downloaded and placed in the ***ext*** folder of PHP source code.

Generally, their compilations are similar:

```
./configure --prefix=INSTALL_DIR --options
make
make install
```

*Note: use **./configure --help** to see the available options*

Sometimes libraries compile using different commands, for that refer to their documentation.

The installation folder of these libraries (INSTALL_DIR) is important, the developer will need to inform it to the PHP configure script before the compilation.

2) Compilation of PHP

Once the extensions are compiled and installed, we can compile PHP. The compilation of PHP on UNIX systems is also done this way:

```
./configure --options
make
make install
```

Now let's look at the different options to compile a PHP that work with 4D and integrate the desired extensions.

First, we need to compile the FastCGI (Fast Common Gateway Interface) version of PHP, this is this version that allows communication between 4D and PHP. To activate it just use the option --enable-cgi. The CLI version being useless for 4D, it can be disabled with the ***--disable-cli*** option.

If the developer wishes to make every extension statics, they should use these options:
--enable-static=yes --enable-shared=no

And vice versa if they want to make every extension shared, then they should use:
--enable-static=no --enable-shared=yes

The next step is to tell the script which extensions to integrate.

There are two possible cases:

- The extension does not depend on an external library, so it can simply be enabled with the --enable-extname option.
- The extension depends on an external library, in this case it must be activate with the --with-extname="INSTALL_DIR" option, INSTALL_DIR is the library installation path that was defined during compilation and installation of these.

Note: (run ./configure --help to see which option are available)

If precisely one extension is to be shared or static, then the suffix shared or static could be added like this:
--enable-extname=shared --with-extname="shared,INSTALL_DIR"

Some extensions are added by default, they can be removed with the respective options ***--disable-extname*** and ***--without-extname***.

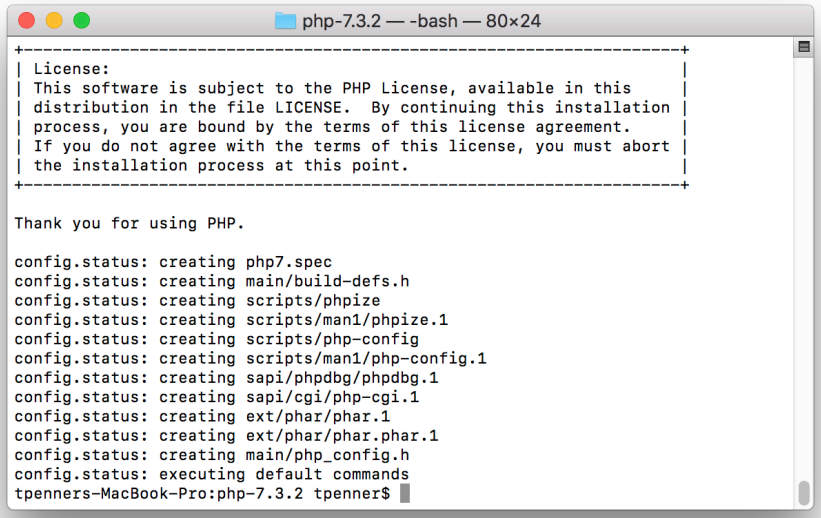
For some extensions (eg iconv or ldap), informing INSTALL_DIR with the ***--with-extname="DIR"*** option does not work. This problem can be solved by adding this option:
CFLAGS="-LINSTALL_DIR/lib -IINSTALL_DIR/include -Qunused-arguments"

The configure command should look like something like this:
./configure --prefix="PHP_INSTALL_DIR" --diasble-cli --enable-cgi --enable-shared=no --enable-static=yes --enable-extname --with-extname="EXT_INSTALL_DIR"

For example:

```
./configure --prefix="/opt" --disable-cli --enable-cgi --enable-shared=no --
enable-static=yes
```

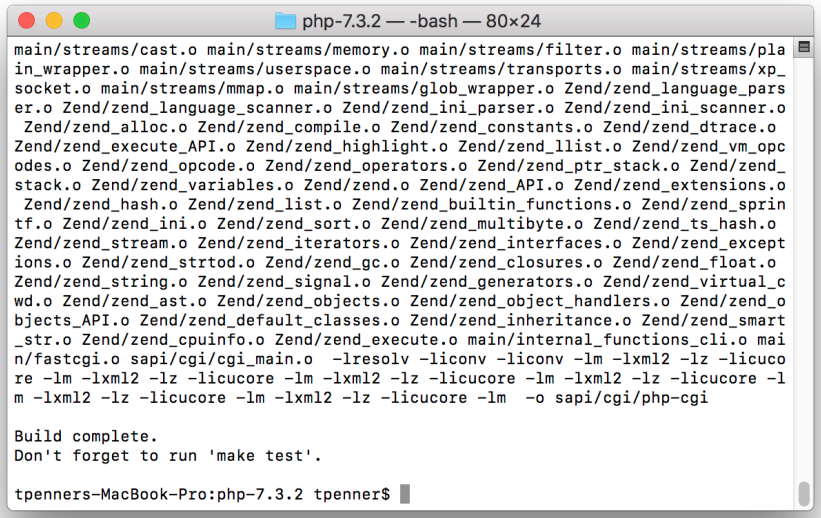
Once this command is executed, it may look like this:



Next, run make:

```
make
```

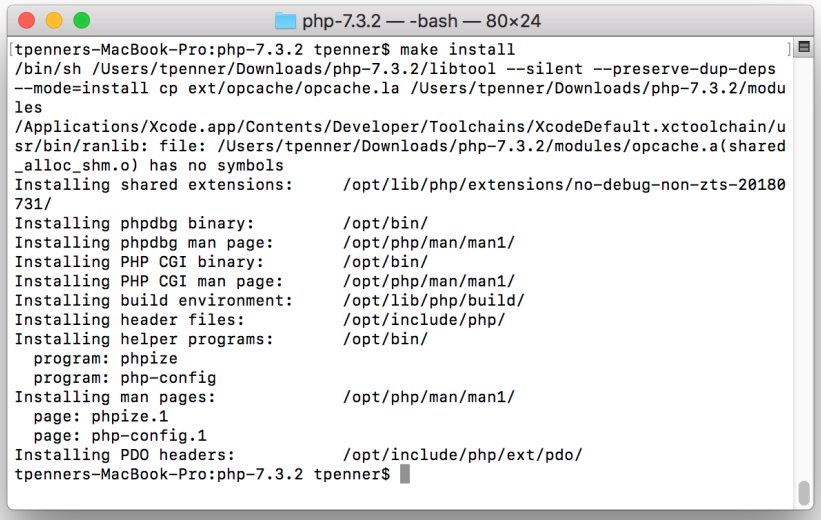
It will take a while for the make command to complete, and it may look like this once complete:



Once it finishes, run make install:

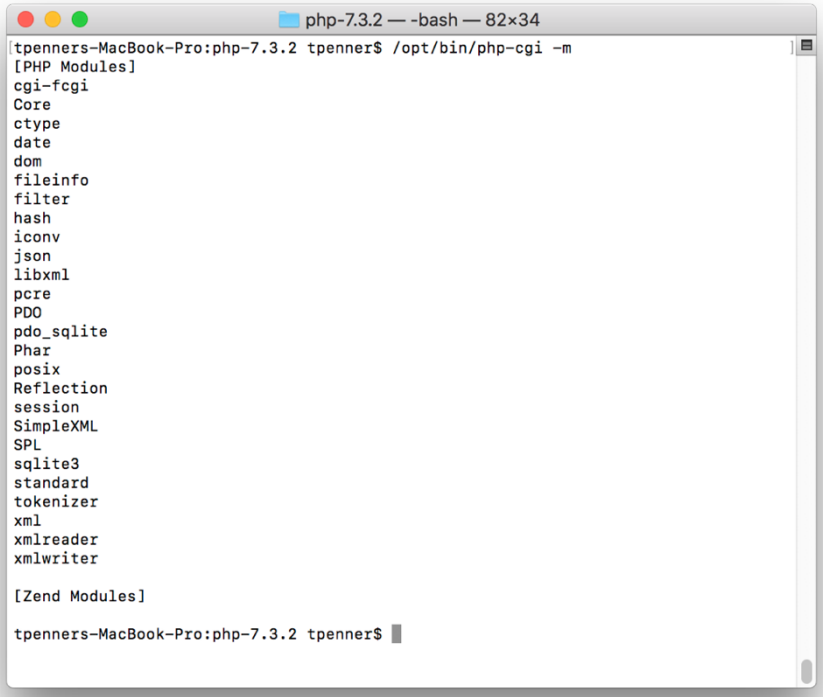
```
make install
```

Once complete, the make install command will likely output something like this:



The php-cgi binary file can be located in the PHP_INSTALL_DIR/bin folder, running this one with the m option (./php-cgi -m) displays the list of extensions loaded (only the static one for the moment). The output of that command may

look like this:



This file must be renamed to php-fcgi-4d and replace the one present in the 4D package (display the contents of the package, it is located in Contents/Resources/php/Mac/).

Now, if using shared extensions, go in the Database/Resources directory. There will be a php.ini file found here. To activate the extensions, these directives must be added:

extension_dir = PATH_TO_EXTENSION_DIR (after the compilation it should be at PHP_INSTALL_DIR/lib/php/extensions/...).
extension = extname.so

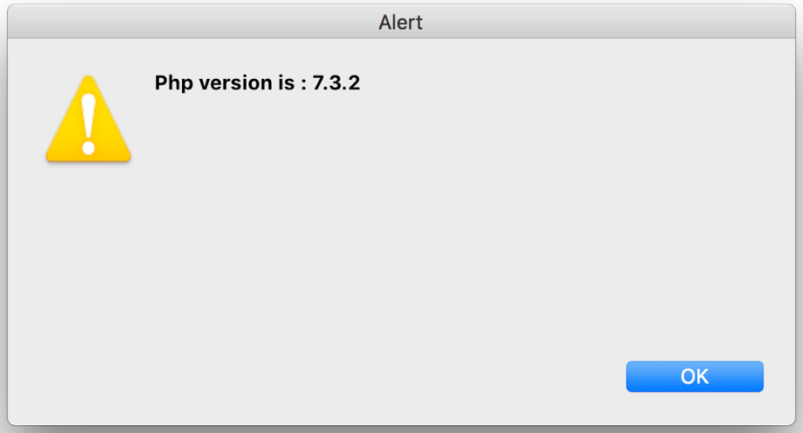
Execute this 4D method to check that the good version of PHP is installed and the good modules are loaded:

After restarting 4D, execute this 4D method to check that the good version of PHP is installed, and the correct modules are loaded:

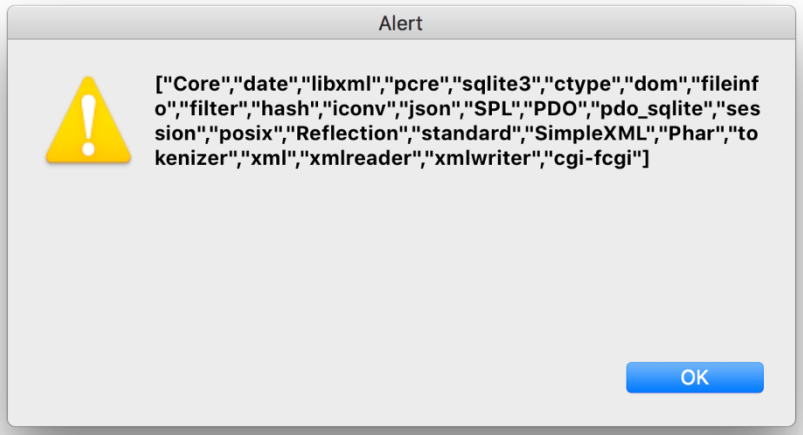
```
C_TEXT($result)
C_BOOLEAN($phpOK)
$phpOK:=PHP Execute("";"phpversion";$result)
ALERT("Php version is : "+$result)
$phpOK:=PHP Execute("";"get_loaded_extensions";$result)
```

```
ALERT($result)
```

The output should look like the following dialogs:



And



If the alert messages shown above are displayed, then the installation was successful.

Updating PHP on Windows

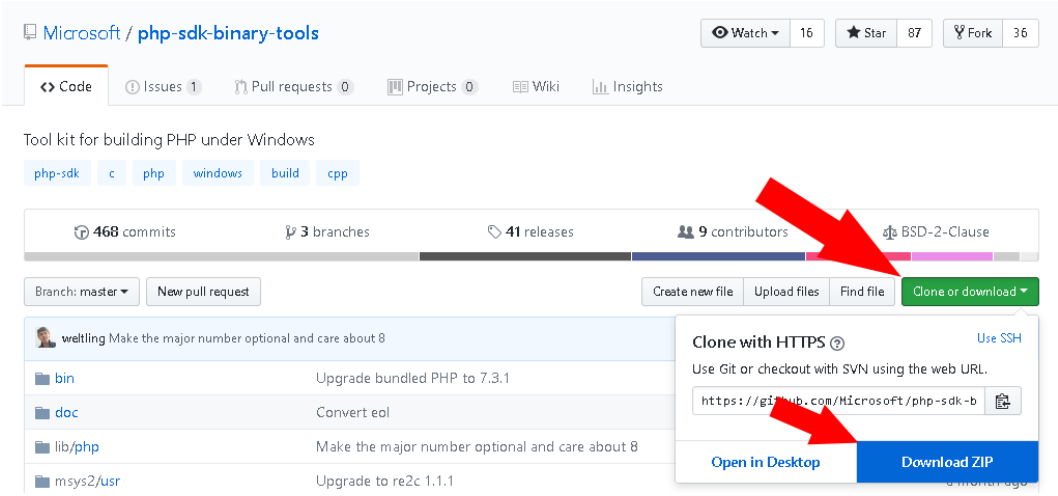
To upgrade PHP on windows the developer can directly get the binaries on php.net or compile it from the source code. The following section describes the steps needed to build it from source.

1) Prepare the environment

First, make sure to have either Visual C++ 2015 or Visual C++ 2017 installed before proceeding. The version required depends on the version of PHP being built.

- PHP 7.0 needs Visual C++ **14**.0 (Visual Studio 20**15**).
- PHP 7.1 needs Visual C++ **14**.0 (Visual Studio 20**15**).
- PHP 7.2 needs Visual C++ **15**.0 (Visual Studio 20**17**)
- PHP 7.3 needs Visual C++ **15**.0 (Visual Studio 20**17**)

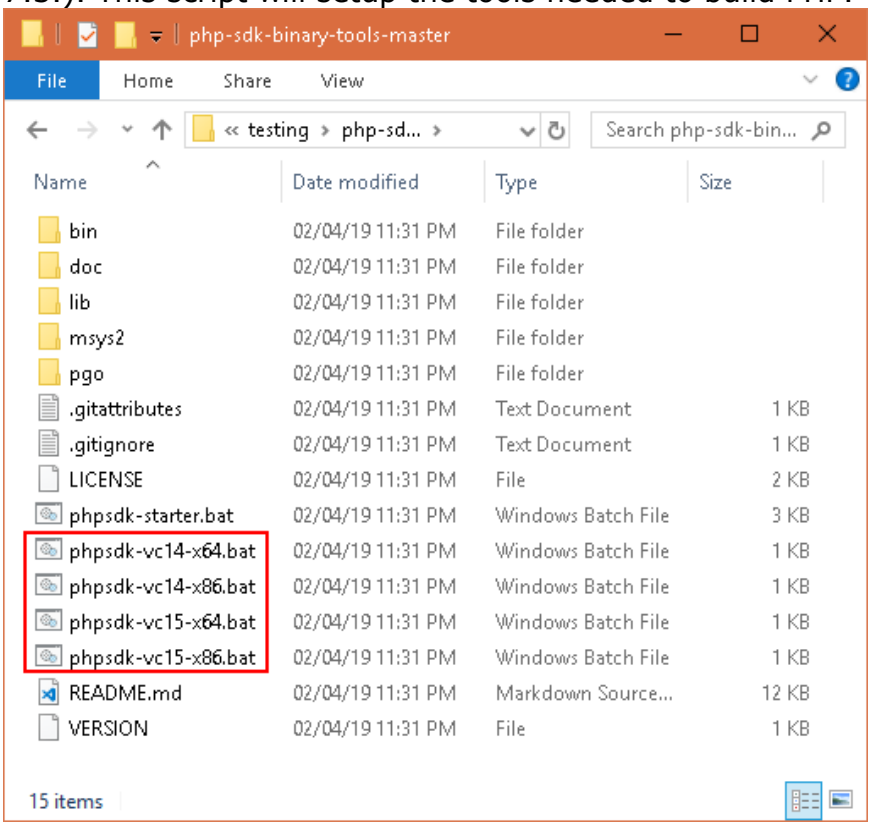
Download the PHP-SDK-binary-tools from Microsoft github page: https://github.com/Microsoft/php-sdk-binary-tools



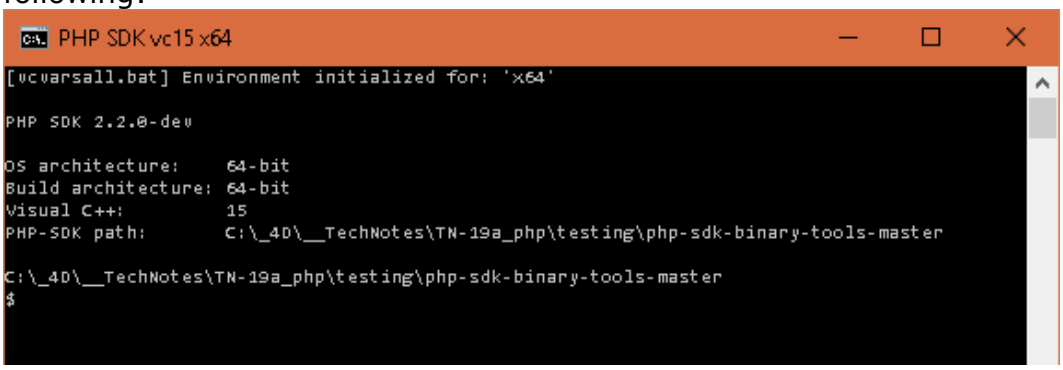
Extract php-sdk-binary-tools-master.zip after downloading it.

Then go into the new directory and invoke the phpsdk-vc##-x##.bat script corresponding to the environment (Visual C++ 14.0 (Visual Studio 2015) for PHP 7.0 or PHP 7.1. Visual C++ 15.0 (Visual Studio 2017) for PHP 7.2 or PHP

7.3.). This script will setup the tools needed to build PHP.

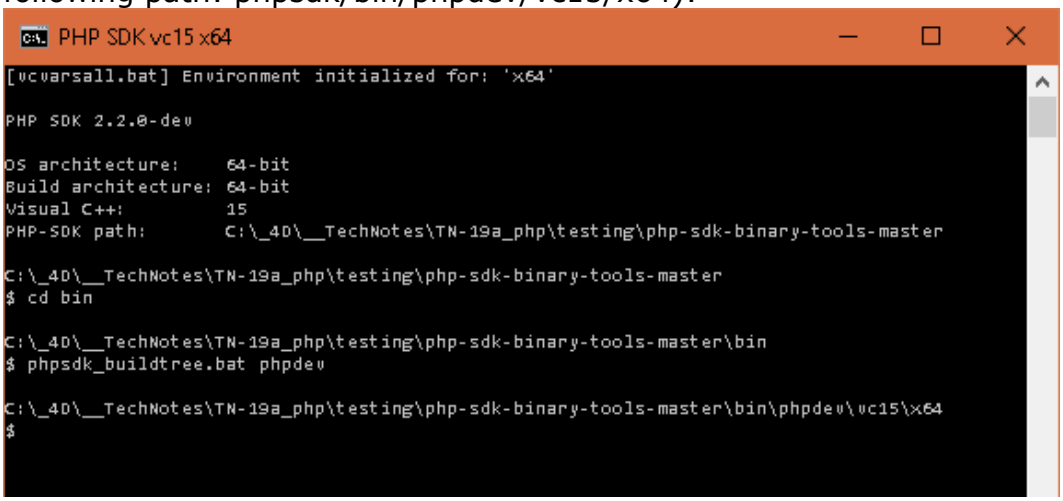


For example, if using Visual Studio 2017 x64 bit then double-clicking on the phpsdk-vc15-x64.bat would be the appropriate action and would produce the following:



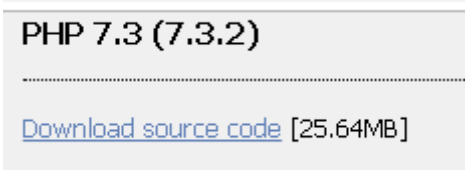
From the command prompt, navigate to the bin directory using “cd bin” and then call the "phpsdk_buildtree.bat phpdev" script. This will create some directories (eg if using phpsdk-vc15-x64.bat, the script should have created the

following path: phpsdk/bin/phpdev/vc15/x64):



Now, obtain the source code for the PHP version being built – this can be obtained from php.net/downloads.php by clicking on the Windows download link:

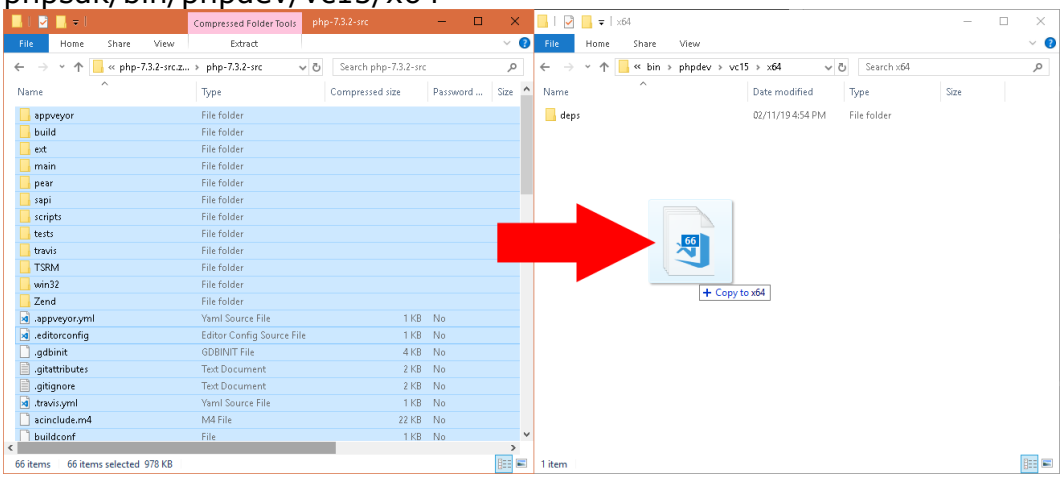
On the "Windows Downloads" page click on the "Download source code" option:



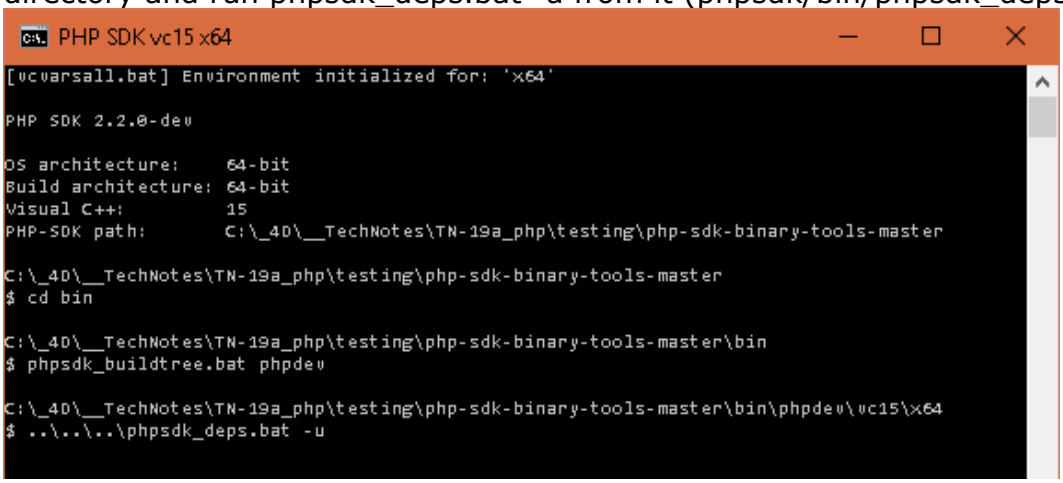
In the example of download 7.3.2 the downloaded file is php-7.3.2-src.zip

Extract the contents of the zip file into the appropriate folder matching this path:
phpsdk/bin/phpdev/vc##/x##

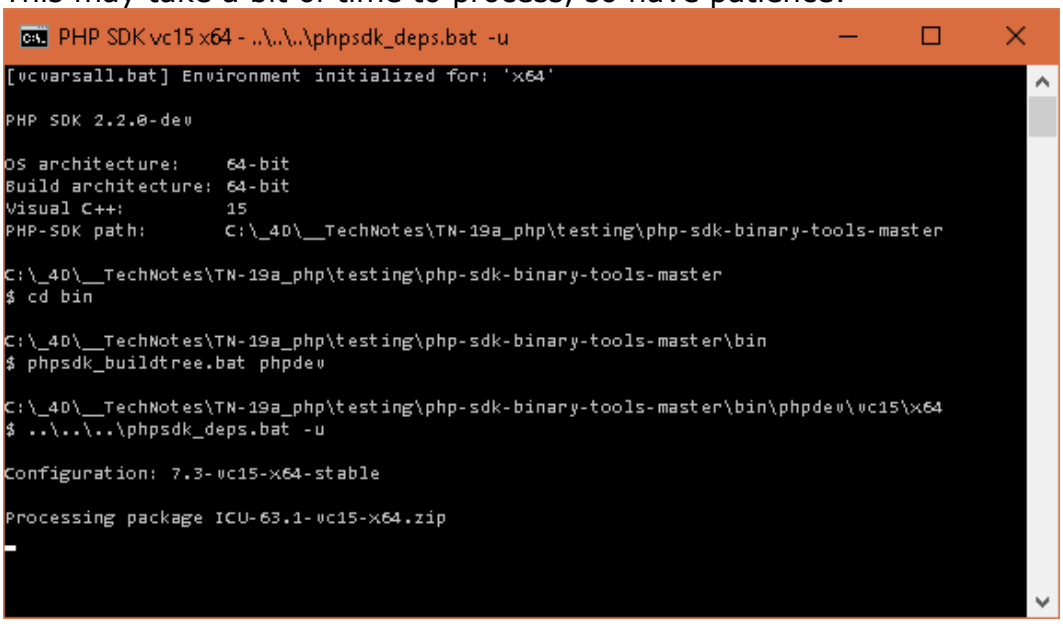
In this example the source files will be extracted to:
phpsdk/bin/phpdev/vc15/x64



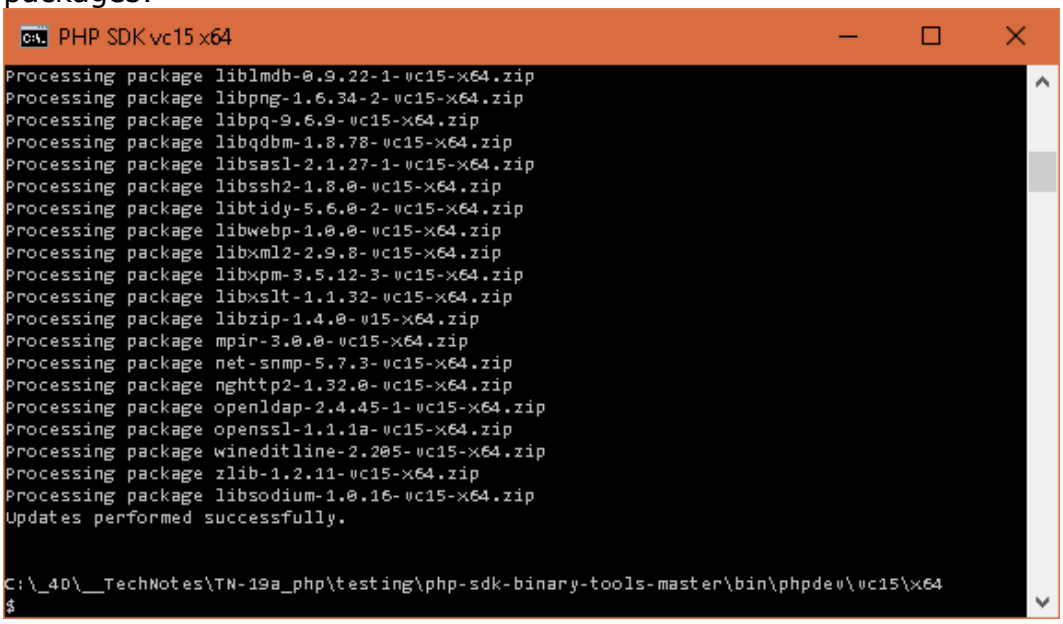
After extracting the files from the PHP Sources ZIP file go into the php sources directory and run phpsdk_deps.bat -u from it (phpsdk/bin/phpsdk_deps.bat):



This may take a bit of time to process, so have patience:



The PHP SDK Window should look similar to this once finished processing all the packages:

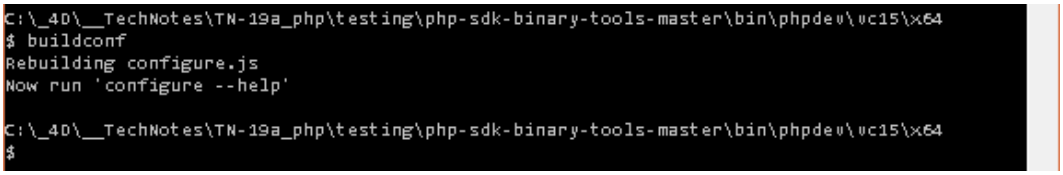


Now there is a deps directory near the php sources directory.

If using extensions that are not present on the sdk, they can usually be compiled from their sources or download their package on pecl.php.net that will have to be put into the php-sources/ext folder or sometimes pre-compiled version can be found on PECL that can be added dynamically after the compilation of PHP.

2) Build PHP

After preparing the environment is complete, the next step is to run buildconf.bat:



Now we will see how to use the configure.bat script to build a PHP that works with 4D and has some extension.

First, we need to compile the FastCGI (Fast Common Gateway Interface) version of PHP, this is this version that allows communication between 4D and PHP. To activate it just use the option --enable-cgi. The CLI version being useless for 4D, it can be disabled with the --disable-cli option.

Then tell the script which extensions to integrate.

There are two possible cases:

- The extension does not depend on an external library, so it can simply be enabled with the --enable-extname option.
- The extension depends on an external library, in this case it must be activated with the --with-extname="INSTALL_DIR" option, INSTALL_DIR is the library installation path that should be the deps folder near the php sources folder

Extensions can be toggled to be shared or static using the suffix shared and static:

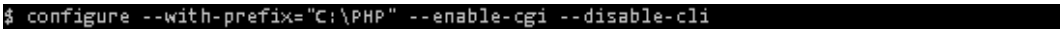
```
      --enable-extname=shared --with-extname="shared,INSTALL_DIR"
```

Some extensions are added by default, they can be removed with the respective options --disable-extname and --without-extname.

The configure command should look something like this:

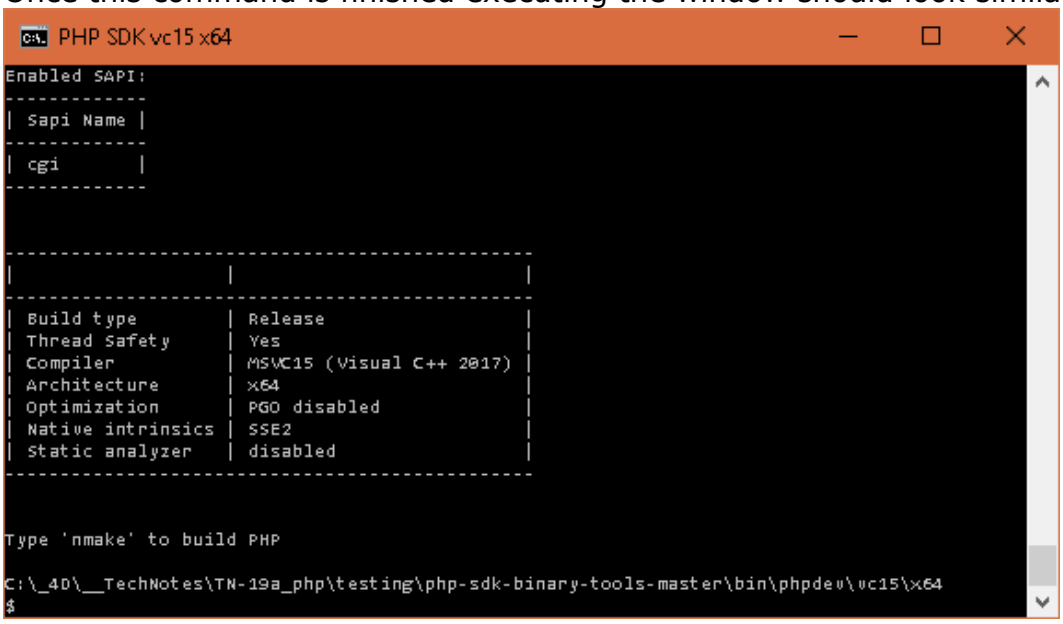
```
    ./configure --with-prefix="PHP_INSTALL_DIR" --diasble-cli --enable-cgi--
enable-extname --with-extname="EXT_INSTALL_DIR"
```

Here is a simple example of what the command may look like:

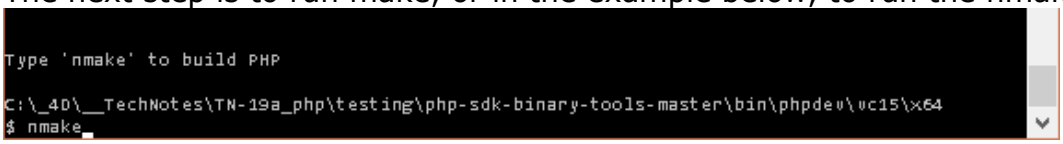



```
configure --with-prefix="C:\PHP" --enable-cgi --disable-cli
```

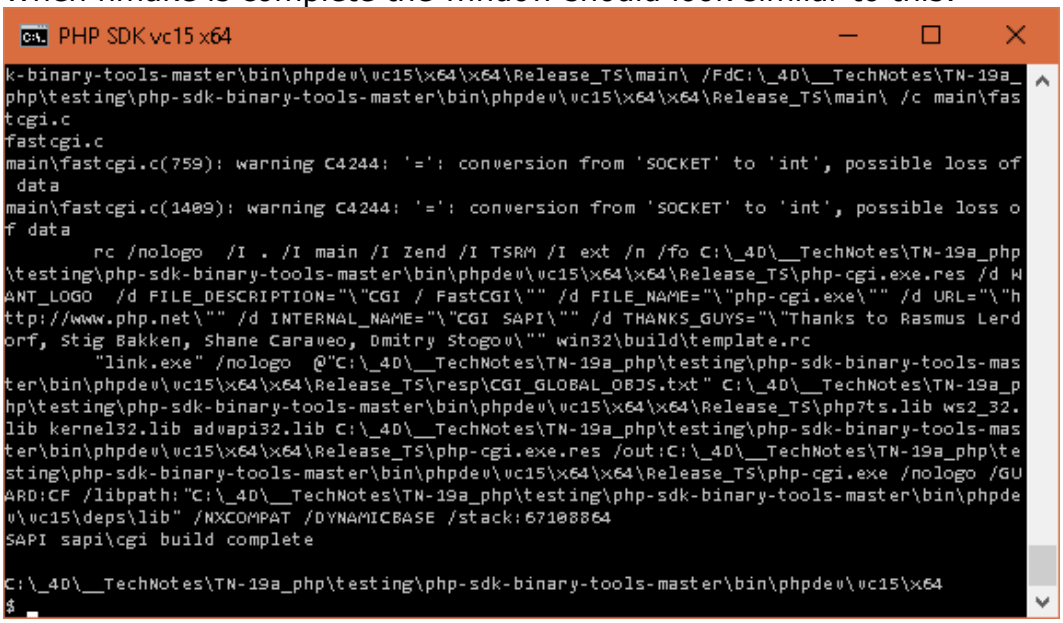
Once this command is finished executing the window should look similar to this:



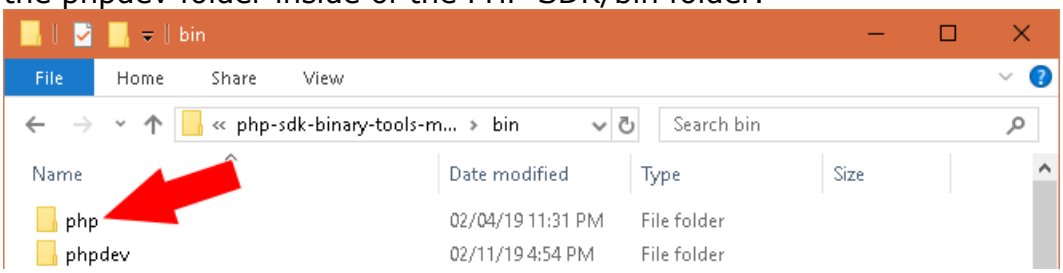
The next step is to run make, or in the example below, to run the nmake:



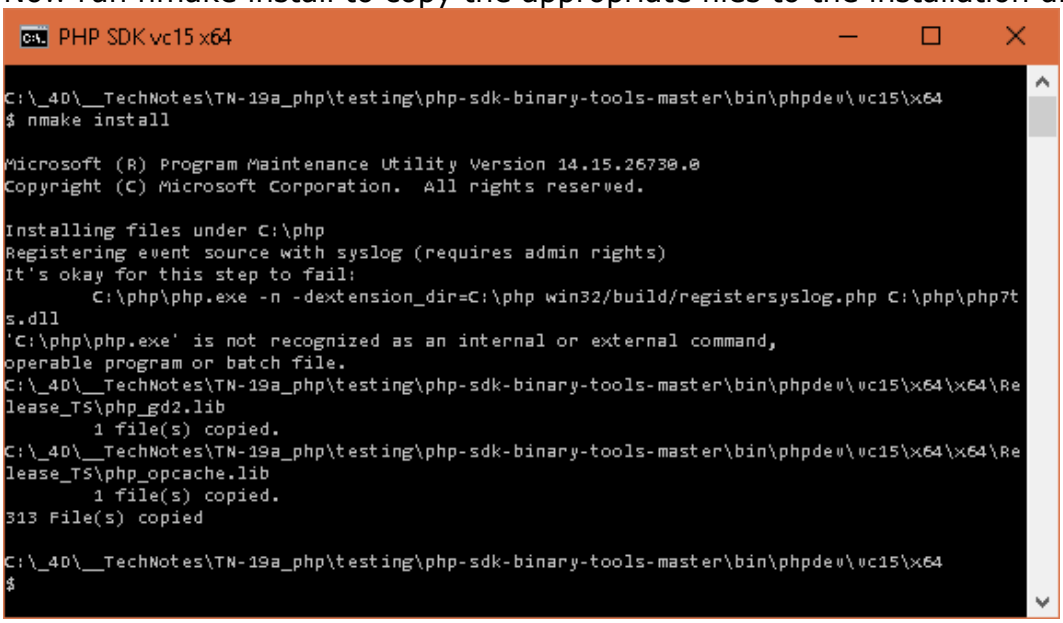
When nmake is complete the window should look similar to this:



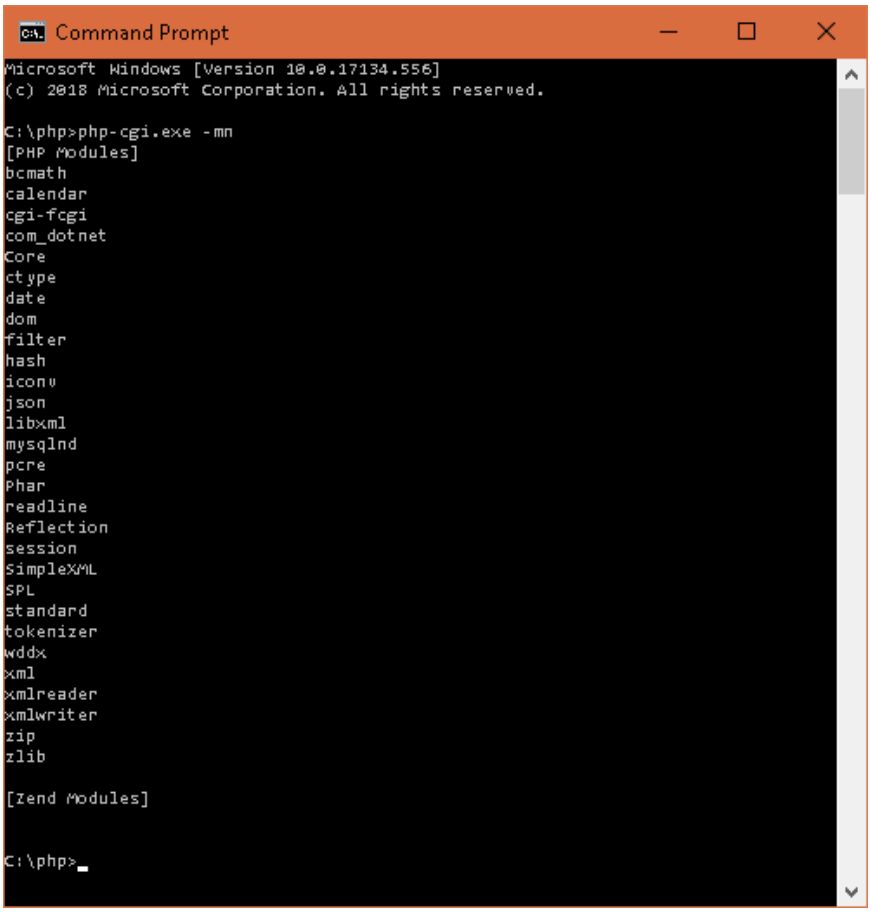
At this point the php interpreter should exist in the php folder located next to the phpdev folder inside of the PHP-SDK/bin folder.



Now run nmake install to copy the appropriate files to the installation directory:



Then run the php-cgi.exe binary with the -mn option (c:\php\php-cgi.exe -mn) to see a list of the extensions loaded. The list of extensions may look like this:



The php-cgi.exe file must be renamed to php-fcgi-4d.exe and replace the one present in the 4D installation (it is located in 4D/Resources/php/Winodws/).

Also put the .dll files with it. If using openssl, copy libsasl.dll, libssl-#_#-.dll and libcrypto-#_#.dll in the executable directory.

Now if using shared extensions, they must be activated by adding these directives to the php.ini that is in the Database/Resources folder:

```
extension_dir = PATH_TO_EXTENSION_DIR (should be PHP_INSTALL_DIR/ext)
extension = php-extname.dll
```

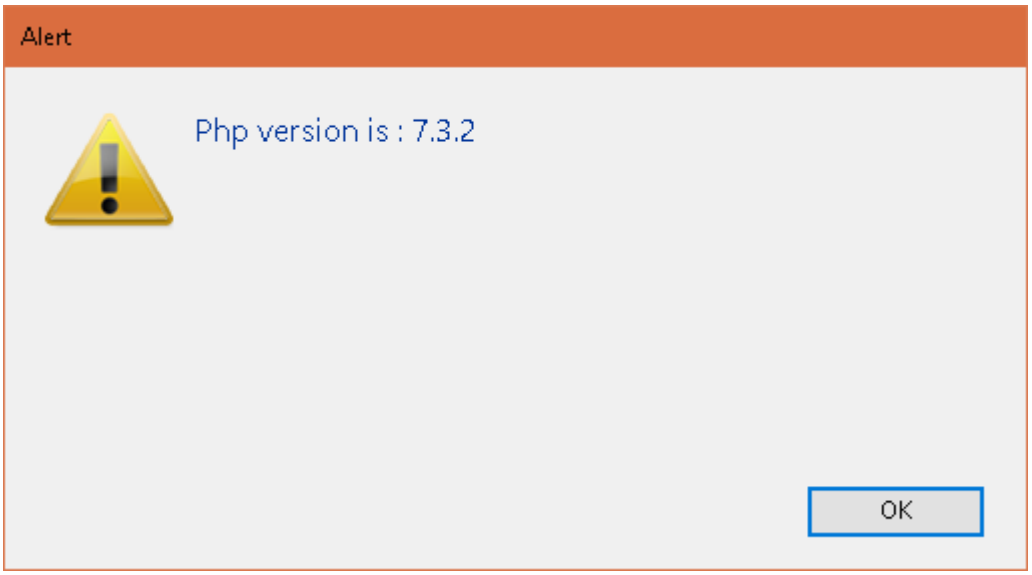
After putting the files in place remember to restart 4D (if it was already running).

After restarting 4D, execute this 4D method to check that the good version of PHP is installed, and the correct modules are loaded:

```
C_TEXT($result)
C_BOOLEAN($phpOK)
```

```
$phpOK:=PHP Execute("";"phpversion";$result)
ALERT("Php version is : "+$result)
$phpOK:=PHP Execute("";"get_loaded_extensions";$result)
ALERT($result)
```

The output should look like the following dialogs:



Followed by:

If the above alert windows (and not error messages) are displayed then the installation is complete!

Conclusion

This technical note described the general concepts of building a custom PHP engine for 4D. Platform options were presented for both MacOS and Windows. This information should allow the 4D developer to build their own custom PHP engine on whichever platform they prefer.