ionCube PHP Encoder 15

Evaluation User Guide



ioncube.com

ionCube and the ionCube logo are registered trademarks of ionCube Ltd.

CONTACTS AND LINKS

Contacting ionCube

Please see our contact details at https://www.ioncube.com/contact.php

FAQ

Find answers to common questions in our FAQ at https://www.ioncube.com/faq.php

Support

For online support please visit https://support.ioncube.com

Purchasing Products

To purchase ionCube products please visit https://www.ioncube.com/purchase.php

Latest Loaders

To obtain the latest Loaders or the Loader Wizard please visit https://loaders.ioncube.com

CONTENTS

1 INTRODUCTION....8

1.1 Encoder Outline....9

1.2 Using Encoded Files / the ionCube Loader....10

1.3 User Guide Notation....11

1.3.1 Command Examples....11

1.3.2 Hints and Tips....11

1.4 The Supplied Software....12

1.4.1 Software Versions....12

1.4.2 Running the Software....12

1.4.3 Using on 64-bit Systems....12

1.4.4 References to Unix....13

1.4.5 Encoder PHP Language Coverage....13

2 GETTING STARTED....15

2.1 PHP Compatibility of Encoded Files....15

2.1.1 Bundling multiple encodings....15

2.2 Running the Encoder....16

2.3 Differences between the Evaluation and Full Version....17

2.4 Command Line Basics....18

2.4.1 Command Line Format....18

2.4.2 Passing Command Line Options....18

2.4.3 Filename, Directory and Wildcard Pattern Matching....19

2.4.4 Using Wildcard Characters on Unix....20

2.4.5 Source directory substitution using @....21

2.5 Quick-Start Encoding Examples....22

2.5.1 Encoding Single Files....22

2.5.2 Encoding Directories....22

2.5.3 Encoding Files with non-default File Extensions....23

2.5.4 Encoding PHP Shell Scripts....23

2.5.5 Encrypting Templates and other Files....23

2.5.6 Leaving Files non-encoded....23

2.5.7 Omitting Files from the Encoding Target....24

2.5.8 Adding Copyright and License Details to Encoded Files....24

3 ENCODER COMMAND LINE OPTIONS....25

3.1 Specifying the Source and Target....25

3.1.1 Source Items....25

3.1.2 The Encoder Target [-o, --into]....25

3.2 Encoded File Format....26

3.2.1 ASCII Format [--ascii]....26

3.2.2 Binary Format [--binary]....26

3.3 Encoding to an Existing Directory Target....27
3.3.1 Replacing the Target [--replace-target]....27
3.3.2 Merging into the Target [--merge-target]....27
3.3.3 Renaming the Target [--rename-target]....27
3.3.4 Updating the Target [--update-target]....27
3.3.5 Create Target Path [--create-target]....27
3.3.6 Target Options and Bundled Encodings....27

3.4 Selecting Files to be Encoded, Encrypted, Copied or Ignored....28
3.4.1 Encoding Specific PHP Files [--encode]....28
3.4.2 Encrypting Files [--encrypt]....28
3.4.3 Copy Files without Encoding or Encrypting [--copy]....29
3.4.4 Skipping Items in a Project [--skip]....29
3.4.5 Excluding Files from the Target [--ignore]....29
3.4.6 Including Ignored Files [--keep]....29
3.4.7 Including only Encoded Files into the Target [--only-include-encoded-files]....30

3.5 Bytecode Obfuscation....31
3.5.1 Obfuscating Compiled Bytecode Symbols [--obfuscate]....31
3.5.2 Specifying an Obfuscation Key [--obfuscation-key]....31
3.5.3 Specifying Obfuscation Exclusions [--obfuscation-exclusion-file]....32

3.6 File Based Server Restrictions (Pro and Cerberus Editions)....33
3.6.1 Expiring Files after a Period [--expire-in]....33
3.6.2 Expiring Files from a Date [--expire-on]....33
3.6.3 Locking Files to Specific Domains and Servers [--allowed-server]....33
3.6.4 Using domain restricted scripts with PHP CLI [--trust-unnamed-servers]....35
3.6.5 Exclude checking of interface aliases [--ignore-interface-aliases]....35

3.7 License Based Server Restrictions (Pro and Cerberus Editions)....36
3.7.1 Specifying a License File [--with-license]....36
3.7.2 Specifying a Passphrase [--passphrase]....36
3.7.3 License Check Mode [--license-check]....37

3.8 Target File Attributes....38
3.8.1 Copying with Hard Links [--use-hard-links]....38
3.8.2 Using Default File Permissions [--without-keeping-file-perms]....38
3.8.3 Updating File Times [--without-keeping-file-times]....38
3.8.4 File Ownership [--without-keeping-file-owner]....38
3.8.5 Setting File Ownership [--apply-file-user, --apply-file-group]....38

3.9 Language Options....39
3.9.1 Ignoring Short Open Tags [--no-short-open-tags]....39
3.9.2 Ignoring Strict Language Warnings [--ignore-strict-warnings]....39
3.9.3 Ignoring Deprecated Feature Warnings [--ignore-deprecated-warnings]....39
3.9.4 Register Custom Auto Globals [--register-autoglobal]....39

3.10 Encoded File Header Customisation....40
3.10.1 Generating Files with no PHP Header [--without-loader-check]....40
3.10.2 Customising the ‘no Loader installed’ Message [--message-if-no-loader]....40
3.10.3 Customising the ‘no Loader installed’ Action [--action-if-no-loader]....40
3.10.4 Setting the Run-Time Loader Path [--loader-path]....40
3.10.5 Setting the Header Code [--preamble-file]....41
3.10.6 Header Comments [--add-comment, --add-comments]....41

3.11 Customising Loader Behaviour....42
3.11.1 Loader Event Messages [--loader-event]....42
3.11.2 Callback Files [--callback-file]....43
3.11.3 Loader Event Constants....44
3.12 File Properties and Include Attack Prevention....45
3.12.1 Setting Properties [--property, --properties]....45
3.12.2 Include Attack Prevention [--include-if-property]....46
3.12.3 Preventing Prepend and Append File Usage [--disable-auto-prepend-append]....46
3.13 Project Handling....47
3.13.1 Specifying the Project File [--project-file]....47
3.13.2 Creating the Project File [--create-project]....47
3.13.3 Update a Project File [--update-project]....47
3.14 Bundled Encodings....48
3.14.1 Starting a Bundled Encoding [--bundle, -B] 48
3.14.2 Adding to a Bundled Encoding [--add-to-bundle, -A]....48
3.14.3 Bundled encoding and the --*-target options....48
3.14.4 Adding header comments with extra encodings [--keep-comments]....49
3.14.5 Bundled Encodings and Encryption of Non-PHP Files....49
3.14.6 Limits on Bundled Encoding Files....49
3.14.7 Bundled Encodings and Encoded File Format....49
3.15 Miscellany....50
3.15.1 Encoding and Bytecode Optimisation [--optimise, --optimize]....50
3.15.2 Allowing Encoding into the Source Tree [--allow-encoding-into-source]....50
3.15.3 Omitting Documentation Comments [--no-doc-comments]....51
3.15.4 Setting an alternate Shell Script Line [--shell-script-line]....51
3.15.5 Enforce Minimum Loader Version [--min-loader-version]....51
3.15.6 Check for Program Updates [--check-version]....51
3.15.7 Program Version [-V, --version]....51
3.15.8 Verbose Mode [-v, --verbose]....51
3.15.9 File Verify [--verify] (legacy option)....51
3.15.10 Help [-h, --help]....51

4 EXTERNAL AND DYNAMIC KEYS....52
4.1 Introduction....52
4.2 External Keys....53
4.2.1 File System External Keys [--encoding-key file:...]....53
4.2.2 Using Configuration Properties With External Keys [--encoding-key ini:...]....54
4.2.3 Where External Key ini Properties Can be Set....55
4.2.4 Using License Properties With External Keys [--encoding-key lic:...]....55
4.2.5 External Keys and Security....56
4.2.6 Uniqueness of External Key Values....56
4.2.7 Errors That May Occur When Encoding With External Keys....57
4.2.8 Errors That May Occur With External Keys at Runtime....57

4.3 Dynamic Keys....58
4.3.1 Function Annotations to Specify Dynamic Keys....58
4.3.2 Rules for Expressions Used to Get the Decoding Key....60
4.3.3 Available Encryption Methods....62
4.3.4 Dynamic Keys for Whole Scripts....62
4.3.5 Dynamic Keys for Primary Scripts....62
4.3.6 Dynamic Keys and Closures....62
4.3.7 Chains of Functions Having Dynamic Keys....63
4.3.8 Performance Versus Security When Using Dynamic Keys....63
4.3.9 Using the Reflection API with Dynamic Decoding [--allow-reflection-all, --allow-reflection] 64
4.3.10 Errors & Warnings Related to Dynamic Keys Produced by the Encoder....65
4.3.11 Errors Related to Dynamic Keys That May Occur At Runtime....66
4.3.12 Controlling How Dynamic Decoding Errors Are Presented [--dynamic-key-errors]....66
4.3.13 Getting Dynamic Key Information With Verbose [--verbose]....66

5 LICENSE FILE GENERATION (Pro and Cerberus Editions)....67

5.1 Introduction to License Files....67

5.2 Creating License Files....69
5.2.1 Command Line Usage....69
5.2.2 Help [--help, -h]....69
5.2.3 Program version [--version, -v]....69
5.2.4 Output path [--output, -o]....69
5.2.5 Using Passphrases to Differentiate Products [--passphrase]....69
5.2.6 Setting License Restrictions [--allowed-server, --expose-server-restrictions]....69
5.2.7 Exclude checking of interface aliases [--ignore-interface-aliases]....70
5.2.8 Setting License Restrictions from Server Data [--use-server-file]....70
5.2.9 Selecting Adapters [--select-server-adapter, --select-adapters]....70
5.2.10 License Expiry [--expire-in, --expire-on, --expose-expiry]....71
5.2.11 License Properties [--property, --properties, --expose-property]....71
5.2.12 License Property Checking [--enforce-property]....71
5.2.13 Customising the Header Block [--header-line]....71
5.2.14 Viewing Server Data Files [--decode-server-file]....71
5.2.15 Decoding a license [--decode-license, --decode-format]....73
5.2.16 Legacy compatibility [--compat]....75
5.2.17 Troubleshooting License Problems....75

6 LOADER API....76

6.1 File Information and Execution....76
6.1.1 Checking for an Encoded File [ioncube_file_is_encoded()]....76
6.1.2 General Encoded File Information [ioncube_file_info()]....76
6.1.3 Retrieving Properties Stored in an Encoded File [ioncube_file_properties()]....76
6.1.4 Retrieving the Loader String Version [ioncube_loader_version()]....76
6.1.5 Retrieving the Loader Integer Version [ioncube_loader_iversion()]....76

6.2 License and Server Information....77
6.2.1 Retrieving Properties Stored in a License [ioncube_license_properties()]....77
6.2.2 Retrieving the List of Permissioned Servers [ioncube_licensed_servers()]....77
6.2.3 Creating a Server Data Block [ioncube_server_data()]....77

6.3 License Validation....78
6.3.1 Validating License Properties [ioncube_check_license_properties()]....78
6.3.2 Validating Licensed Servers [ioncube_license_matches_server()]....78
6.3.3 Validating License Expiry [ioncube_license_has_expired()]....78

6.4 Encrypted File Support....79
6.4.1 Reading Encrypted Files [ioncube_read_file()]....79
6.4.2 Writing Encrypted Files [ioncube_write_file()]....79

6.5 Error codes....80

7 ERROR REPORTING....81

8 TROUBLESHOOTING....82

8.1 Unable to Start the Encoder....82
8.1.1 On Unix (Linux, FreeBSD, MacOS)....82
8.1.2 Using a 64 bit system....82
8.1.3 On Windows....82

9 LOADER INSTALLATION....83

9.1 Loader Naming....83

9.2 Installation in a php.ini File....84

1 INTRODUCTION

The ionCube PHP Encoder is a powerful, high performance solution for encoding and licensing PHP scripts, plus encrypting files of any type.

Encoding and Encryption Total Solution

The Encoder protects PHP /HTML scripts with obfuscated bytecode protection and a custom execution engine. In addition, any other project files can be automatically encrypted if required, which is ideal for protecting files such as templates or XML documents. This is complemented by Loader API functions for reading and writing encrypted files. For most existing template engines, a small change is all that would be required to add the ability to read encrypted templates.

Bytecode Compilation and Obfuscation

The Encoder achieves script protection by first compiling and then optimising PHP scripts to highly efficient binary data. The compilation process replaces source with virtual-machine instructions and then applies several layers of encoding and transformations to produce the final platform independent files. Optional class, method, function name and local variable obfuscation adds extra protection, with further internal obfuscations applied at compile and runtime. This approach has the advantage that files are never restored to PHP source code, compiled code is changed and hidden from the Open Source PHP engine, and run-time performance is comparable to source due to the parsing and compilation taking place at encoding time. Other Encoder features offer further benefits, such as the easy addition of tamper resistant plain text to the start of files, which is ideal for including custom copyright or license details.

License File Creation

License files can be created for your projects with the Pro and Cerberus Encoders that lock your projects to particular machines. License files can also have an optional time expiry, and can store arbitrary key/value data that can be read at runtime by the licensed application.

ionCube Loader

The ionCube Loader handles execution of encoded PHP files, encrypting or decrypting non-PHP files, validating licenses, and so on. This component is easily installed into a `php.ini` file, and a free tool called the Loader Wizard is available to assist installation by end-users. Loaders are available for a wide range of common and less-common platforms, and a service is also available for producing Loaders on platforms outside of the standard range.

Windows and MacOS GUI

For Windows and MacOS users, a powerful GUI makes setting up projects simple. As well as encoding features, integration with Explorer adds usability features such as dynamic icons to distinguish between encoded and non-encoded files at a glance, and quick right click encoding with a cut-down GUI. Source files can also be launched in a user's preferred editor straight from the GUI, and also at a specific line if supported by the editor; this is great for quickly squashing syntax errors in rarely used code. Additional features are available in a Special Edition GUI upgrade, such as automatic archive creation, FTP, and a unique *dynamic fields* feature that dramatically simplifies data entry for custom encoding and license creation through a dynamically created custom interface. Together with other features, the GUI helps to maximise productivity.

1.1 Encoder Outline

The Encoder is driven by a command line interface allowing for easy automation and integration into project build, test and release environments, and for calling from UNIX shell scripts or Windows batch files. Encoding is quick, making it practical to perform “just in time” (JIT) encoding if required.

Major features include:

- Current software version PLUS the two previous versions.
- Encoding of PHP 4 up to PHP 8.4 language syntax.
- Optional external and dynamically created keys, eliminating the need for keys within encoded files.
- Bytecode encoding of PHP for optimal performance, reliability and protection.
- File encryption feature plus Loader API support for reading/writing encrypted data, ideal for encrypting templates and XML documents.
- Choice of an ASCII encoded file format for reliable cross platform transfers with FTP or binary file format.
- Optional compiled-code obfuscation of class, method, function and local variable names.
- Single file or recursive directory encoding.
- Replication of source file attributes, i.e. times and permissions.
- Full control over the project items to be encoded, encrypted, copied or ignored.
- Custom text such as copyright details may be added to encoded files.
- Optional *include-attack* protection to block interaction with unauthorised files.
- Associating key/value *properties* with files.
- Projects feature to encode projects with pre-configured options.
- Custom Loader event messages or error handler callback.
- Automatic syntax checking and error reporting of encoded files.
- Syntax-check only operation.
- Bundled encodings where a file can include encodings from multiple encoders.

Features of the Pro Encoder include Basic Encoder features plus:

- Generating files to expire after a time period or on a specific date.
- Generating files restricted by IP addresses and/or domain names.
- Generating files to work only with a valid license file.
- License generator program for creating license files with time expiry, machine restrictions and custom key/value properties.
- License generator for Linux included with the Windows Encoder.

Features of the Cerberus Encoder include Pro Encoder features plus:

- Generating files restricted to Ethernet (MAC) addresses.
- Creating license files with MAC address restrictions.

1.2 Using Encoded Files / the ionCube Loader

ionCube Encoded files contain compiled code with various layers of encoding, and a special component must be available to process encoded files when required. The component for this task is called the ionCube Loader, which is an engine extension to PHP. Loaders for common platforms and architectures are provided as standard, and a porting service for producing bespoke Loaders to target less common platforms is also available.

ionCube is the most widely used and recognised encoding technology, and Loaders are often already installed by hosting providers, however a Loader Wizard PHP script is also available that greatly assists with installation guidance when necessary.

See chapter 9 for more detailed information about Loader installation.

1.3 User Guide Notation

1.3.1 Command Examples

Command or other computer related text in the user guide is printed in `monospace` type. Command examples are either written in the form:

```
ioncube_encoder source.php -o target.php
```

or in some places to illustrate command usage and output, as:

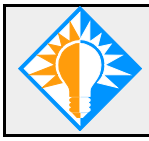
```
$ ./ioncube_encoder.sh -84 -V

ionCube PHP Encoder Version 15.0
Language Support: PHP 4, 5, 7, 8.1, 8.2, 8.3, 8.4
Copyright (c) 2002-2025 ionCube Ltd.
```

In this case, shell input is shown in **`bold`**, and program output is `plain`. **`$`** is an example shell prompt for command entry and may be different on your own system.

1.3.2 Hints and Tips

Look out for hints and tips in the guide, for example:



You can use the bookmarks feature in Acrobat to quickly navigate the user guide, or click on references in the text.

1.4 The Supplied Software

1.4.1 Software Versions

As well as the latest software, the two prior versions are included in the package. The previous version (version 14) is termed *legacy*, and the one before that (version 13) we refer to as *obsolete*. We include these so older versions of encoded files can be produced if desired. Example use cases for this would be if having upgraded but wanting to decide when to start creating files with the newer software, or if wanting to maintain a package built with an older Encoder.

1.4.2 Running the Software

There are two ways to run the Encoder software. On Linux and FreeBSD the shell script `ioncube_encoder.sh` can be used with the PHP version as an option, e.g.:

```
./ioncube_encoder.sh -84 hello.php -o hello_enc.php
```

The shell script selects the actual Encoder program to use, and in the example above, the PHP 8.4 Encoder would be used to encode the `hello.php` script and generate the encoded file `hello_enc.php`.

By default the latest Encoder version will be selected, however the previous version (*legacy*) can be selected by passing `-L` and the one before that (*obsolete)* with `-O`. For completeness, `-C` can be used to select the current version, but this is optional as it's the default.

Passing `-h` to the script without specifying a PHP language will display help about the `ioncube_encoder.sh` script. Used after a PHP version selection will pass the option to the Encoder instead.

The alternative is to run a specific Encoder program directly. The Encoder programs themselves are in the `bin` sub-directory, and suffixed with the PHP language number and the Encoder version. For example, `ioncube_encoder84_15.0` will be the PHP 8.4 Encoder of version 15.0 (the current version). If there are 64 bit versions, these would additionally have `_64` in the name. OnWindows, the program names have the usual `.exe` extension.

Some User Guide examples will refer to the shell script, however for convenience going forward we shall mostly refer to the Encoder command line program simply as `ioncube_encoder`. Replace this with your preferred choice of actual program depending on your requirements. Examples that reference a specific PHP version will include an option such as `-85` that the shell script recognises to denote an Encoder for PHP 8.4.

1.4.3 Using on 64-bit Systems

64-bit Encoders are included on Linux as 32 bit support with distributions is increasingly being phased out. Running the Encoder shell script on a 64-bit Linux system will automatically select the 64-bit version, but if required, you can force use of the 32 bit version by adding the `-x86` option to the command line. Using `-x86-64` can be used to select the 64 bit version, but as the script auto-detects, this should not be required.

On 64-bit FreeBSD systems you should install the 32-bit compatibility libraries to run the FreeBSD Encoder as 64-bit Encoder binaries are not currently available for FreeBSD.

The Encoders for Windows are compatible with both 32 and 64 bit systems to avoid unnecessary package bloat as there is no advantage of a 64 bit Encoder over a 32 bit one.

1.4.4 References to Unix

Aside from Microsoft Windows, the other supported operating systems for the ionCube Encoder (Linux, FreeBSD and MacOS) are all UNIX®-like. For simplicity, the User Guide uses Unix to collectively refer to those platforms.

1.4.5 Encoder PHP Language Coverage

The language coverage of the various Encoders is listed below.

`ioncube_encoder84` is the PHP 8.4 Encoder, and should be used for encoding projects that make use of PHP 8.4 language features. Files can run on PHP 8.4. This is the same as running `ioncube_encoder.sh -84`

`ioncube_encoder83` is the PHP 8.3 Encoder, and should be used for encoding projects that make use of PHP 8.3 language features. Files can run on PHP 8.3 and 8.4. This is the same as running `ioncube_encoder.sh -83`

`ioncube_encoder82` is the PHP 8.2 Encoder, and should be used for encoding projects that make use of PHP 8.2 language features. Files can run on PHP 8.2, 8.3 and 8.4. This is the same as running `ioncube_encoder.sh -82`

`ioncube_encoder81` is the PHP 8.1 Encoder, and should be used for encoding projects that make use of PHP 8.1 language features. Files can run on PHP 8.1. This is the same as running `ioncube_encoder.sh -81`

`ioncube_encoder74` is the PHP 7.4 Encoder, and should be used for encoding projects that make use of PHP 7.4 language features. Files can run on PHP 7.4. This is the same as running `ioncube_encoder.sh -74`

`ioncube_encoder72` is the PHP 7.2 Encoder, and should be used for encoding projects that make use of PHP 7.2 language features. Files can run on PHP 7.2 and 7.4. This is the same as running `ioncube_encoder.sh -72`

`ioncube_encoder71` is the PHP 7.1 Encoder, and should be used for encoding projects that make use of PHP 7.1 language features. Files can run on PHP 7.1, 7.2 and 7.4. This is the same as running `ioncube_encoder.sh -71`

`ioncube_encoder56` is the PHP 5.6 Encoder, and should be used for encoding projects that make use of PHP 5.6 language features. Files can run on PHP 5.6 and 7.0. This is the same as running `ioncube_encoder.sh -56`

`ioncube_encoder55` is the PHP 5.5 Encoder, and should be used for encoding projects that make use of PHP 5.5 language features. Files can run on PHP 5.5 and 5.6. This is the same as running `ioncube_encoder.sh -55`

`ioncube_encoder54` is the PHP 5.4 Encoder, and should be used for encoding projects that make use of PHP 5.4 language features. Files can run on PHP 5.4, 5.5 and 5.6. This is the same as running `ioncube_encoder.sh -54`

`ioncube_encoder53` is the PHP 5.3 Encoder, and should be used for encoding projects that make use of PHP 5.3 language features. Files can run on PHP 5.3, 5.4, 5.5 and 5.6. This is the same as running `ioncube_encoder.sh -53`

`ioncube_encoder5` is the Encoder for the first PHP 5 language, and should be used when a project uses language features added to PHP 5, and none from the PHP 5.3 or subsequent languages. Files can run on PHP 5.0.3 and higher. This is the same as running `ioncube_encoder.sh -5`

`ioncube_encoder4` is the PHP 4 Encoder, and should be used if encoding a project that is required to run on servers using PHP 4.0.6 and higher but excluding PHP 5.4 onwards. This is the same as running `ioncube_encoder.sh -4`

2 GETTING STARTED

This chapter introduces some of the most common features of the ionCube command line Encoder, with quick-start examples for typical scenarios. For Windows and MacOS users, an additional document describes the GUI. Chapter 3 describes all command line options in detail, and will be a useful reference guide to features even if using the GUI. The License generation program that comes with the Pro and Cerberus editions is described in chapter 5, and chapter 6 describes the Loader API, including features for reading and writing encrypted files.

If reading this guide using Acrobat, you can also use the Acrobat bookmarks feature to quickly jump to different sections, and in the document you can click on hyperlinks, entries in the contents section, and on any section references.

2.1 PHP Compatibility of Encoded Files

To achieve the best performance and security we recommend encoding using PHP 8.4 or later.

With ionCube, it is often possible to encode files for one version of PHP and run them on a range of subsequent versions of the same PHP generation (i.e., 4, 5, 7, 8). For example, files encoded for PHP 5.5 can run on systems using versions 5.5 and 5.6. Files encoded for PHP 7.1 can run on PHP 7.1 to 7.4.

Note that files encoded for PHP 8.1 can only run on PHP 8.1, however files encoded for PHP 8.2 can run on PHP 8.2, PHP 8.3 and PHP 8.4. Note too that files cannot be encoded for a version of PHP that does not support the syntax used; so for example, if using language features from PHP 8.3, those files cannot be encoded to run on PHP 8.2.

2.1.1 Bundling multiple encodings

To support a range of PHP versions that is *wider* than supported by a single encoding, a bundling feature is available that combines multiple encodings into a single file. For example, encodings for PHP 7.1, 8.1 and 8.2 can be bundled together so that files that can work on targets from PHP 7.1 up to PHP 8.4.

We recommend that an up-to-date ionCube Loader is used for the latest and widest compatibility.

2.2 Running the Encoder

For convenience we suggest to ensure that the installation location for the ionCube Encoder software has first been added to the user environment on Windows, or the Unix shell `PATH` variable (e.g. in `.bashrc` or `.profile`). Otherwise you should specify the location of the program with an absolute or relative appropriate path.

For example, if the Encoder software has been installed on a Unix system in `/usr/local/ioncube`, the `ioncube_encoder.sh` program could be run as follows:

From any location as:

```
/usr/local/ioncube/ioncube_encoder.sh
```

Within the current directory of `/usr/local/ioncube` as:

```
./ioncube_encoder.sh
```

Or if the `PATH` variable contains `/usr/local/ioncube` then simply as:

```
ioncube_encoder.sh
```

Unless otherwise stated, command examples in this User Guide omit any path prefix and assume that the `PATH` variable has been set correctly. If there is a failure to launch the software, please check/use an appropriate path to the program, and see chapter 8 for troubleshooting information if necessary.

2.3 Differences between the Evaluation and Full Version

The Evaluation version includes all features of the Cerberus edition, allowing the license file creation facilities to be fully tried. The standard evaluation period is 14 days. Encoded files are fully functional but will expire after 36 hours, and also contain text indicating that they were produced by the evaluation version. This does not affect operation of the files, and the text is not present in files produced by the licensed product. With the Windows and MacOS evaluations, the Special Edition GUI upgrade is provided, adding some extra features to the default GUI.

2.4 Command Line Basics

2.4.1 Command Line Format

The general form for running the Encoder to encode is either

```
ioncube_encoder [options] source -o target
```

or

```
ioncube_encoder [options] sources --into target
```

The `-o` option encodes `source` as `target`, where `source` and `target` are both either single files or directories. When using `--into` the Encoder sources can be one or more files or directories that are encoded *into* a target directory, keeping the same names. See sections 2.5 and 3.1 for more details.

As mentioned earlier, examples in the User Guide mostly use `ioncube_encoder` for the program name, but remember that on Linux and FreeBSD you would ether use `ioncube_encoder.sh` with the desired PHP target version, e.g. `-84` (see section 1.4), or the specific Encoder program directly. On Windows, the program would be `ioncube_encoder84.exe`

To syntax check use `-S` and specify one or more files or directories.

```
ioncube_encoder [options] -S files_and_directories
```

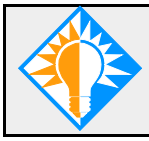
2.4.2 Passing Command Line Options

Most Encoder options use descriptive names, starting with `--`. Options requiring a value may use either an = character or white space to separate the option from its value.

Examples:

```
--add-comment="Encoded by ionCube"
```

```
--add-comment "Encoded by ionCube"
```



Encoder options can be abbreviated to the shortest string that makes them unique. The Encoder will report an error if there are multiple choices.

2.4.3 Filename, Directory and Wildcard Pattern Matching

For options related to files, e.g. `--encode` and `--encrypt`, the Encoder provides flexible pattern processing to match files and directories by name or wildcard patterns.

Matching Files

Names with no trailing path separator are considered to be filenames.

Examples:

Encode all files with default extensions and any files named `x.inc`

```
--encode x.inc
```

Encode all files with default extensions and any files named `config/config.inc`

```
--encode config/config.inc
```

Directories

Appending a file separator specifies directories.

Examples:

Ignore files in directory `config` and subdirectories

```
--ignore config/
```

Note: a similar option `--skip` is available from version 11 and offers improved performance compared to `--ignore`.

Encrypt all files in any directories named `templates`

```
--encrypt templates/
```

Wildcard Matching

The Encoder also supports wildcard matching using the wildcard characters `*`, `?` and `[]`.

Wildcards are interpreted as follows:

Wildcard Character	*Matches*
`*`	Zero or more characters, e.g. `*.inc`
`?`	Any single character, e.g. `*.php?`
`[]`	Any character from a set, e.g. `*.php[78]`
`[!]`	Any character not in the set, e.g. `*.php[!4]`
`@`	Substitute with the source directory for the project

Examples:

Encode files with default extensions and any files ending in `.inc`

```
--encode "*.inc"
```

Encode all files inside any directories named `config`

```
--encode "config/*"
```

Ignore all directories inside any directories named `config`

```
--ignore "config/*/"
```

Encode any files named `config` *having one character after the name*

```
--encode "config?"
```

Encode any files named `doc0`, `doc1`, `doc2`, ... `doc7`

```
--encode "doc[0-7]"
```

Encode any files named `configx`, `configy` *and* `configz`

```
--encode "config[xyz].php"
```

2.4.4 Using Wildcard Characters on Unix

When specifying an option value containing a wildcard character, be sure to use quotes or to escape the wildcard to prevent unintended shell expansion. Remember too that the Encoder can process entire directories. *It is generally not necessary to use wildcards to select multiple files to be processed.* The containing directory can be named instead.

Examples:

Use:

```
--encode "*.ini"
```

```
--encode \*.ini
```

Instead of:

```
--encode *.ini
```

2.4.5 Source directory substitution using `@`

An `@` character appearing at the front of a pattern will be replaced by the first source directory specified on the Encoder command line. This ensures that a pattern has an absolute path, avoiding any possible undesired matching.

For example, given a project structure containing directories `/project/config/config.php` and `/project/live/config/config.php`, using the option

```
--ignore config/config.php
```

would ignore both of `config.php` files when encoding, whereas

```
--ignore @/config/config.php
```

would ignore only `/project/config/config.php`. The benefit over hardcoding the full path is a shorter command line with no dependency on the source location.

2.5 Quick-Start Encoding Examples

This section provides examples of how to use the Encoder to handle some common encoding requirements. Filenames and directory paths are examples only.

2.5.1 Encoding Single Files

Examples:

Files can be encoded as another by specifying the file as the source and using `-o` to name the target file. The target can be any filename.

Encode `/project/file1.php` as `/encoded-project/file1.php`

```
ioncube_encoder /project/file1.php -o /encoded-project/file1.php
```

Files can also be encoded *into* a directory using `--into` to name the target directory. The encoded file is will be given the same name as the source file.

Encode `/project/file1.php` into directory `/encoded-project`

```
ioncube_encoder /project/file1.php --into /encoded-project
```

More than one source item can be specified when using `--into` but giving a directory as the source is usually more convenient.

Encode `/project/file1.php` and `/extra/file2.php` into `/encoded-project`

```
ioncube_encoder /project/file1.php /extra/file2.php --into
/encoded-project
```

2.5.2 Encoding Directories

Entire directory hierarchies can be recursively encoded by specifying a directory as the source. Files are either encoded or copied into the target. File attributes are also copied if possible, and on Unix, any symbolic links will be replicated.

Examples:

Recursively encode `/project` as `/encoded-project`

```
ioncube_encoder /project -o /encoded-project
```

or

```
ioncube_encoder /project --into /encoded-projects
```

If the target directory already exists then you must specify how the Encoder should proceed. The available choices are:

Encoder Option	*Action*
`--replace-target`	Replace the target directory.
`--merge-target`	Merge files into the target directory.
`--rename-target`	Rename the existing target by appending a unique number.
`--update-target`	Similar to merge but only process the source file if its modified time is later than the target's modified time.

The `--replace` option is most commonly used.

2.5.3 Encoding Files with non-default File Extensions

By default the Encoder will encode files matching the pattern `*.php` and `*.phtml`, however files and directories matching any pattern or name can be encoded by using `--encode`. This option may be used as many times as required.

Example:

Encode files with default extensions and also any ending in `.inc`

```
ioncube_encoder --encode "*.inc" /project --into /encoded-projects
```

2.5.4 Encoding PHP Shell Scripts

The Encoder will encode shell scripts having PHP as their interpreter unless explicitly directed otherwise with `--copy` or `--ignore`. By default, the first line of the source shell script will be copied to the target, but a custom line can be used with the following option

```
--shell-script-line '#!/usr/bin/php -q'
```

where the text inside the quotes is an example shell script line. Any encoded script can be used as a shell script by manually adding a shell script line to the encoded file after the encoding process is complete, and similarly an encoded PHP shell script will remain a valid encoded PHP script if the shell script line is removed. On Unix it is important to enclose the option argument in single quotes as the '`!`' character has a special meaning for most shells.

2.5.5 Encrypting Templates and other Files

As well as protecting PHP files, the Encoder can encrypt other files too, such as templates and images. Using the Loader API, encrypted files can then be decrypted only by an encoded file that was encoded by the same Encoder.

As an example of working with encrypted templates, a simple patch to the popular Smarty template engine is available on the ionCube website, allowing the Smarty engine to process both plain text and encrypted templates.

Example:

Encode PHP files with default extensions and encrypt files ending in `.tpl`

```
ioncube_encoder --encrypt "*.tpl" /project --into /encoded-projects
```

2.5.6 Leaving Files non-encoded

Files that would normally be encoded can be left non-encoded and just copied by using `--copy`. This option may be used as many times as required.

Examples:

Encode `/project` into `/encoded-projects` leaving `config/config.inc.php` non-encoded

```
ioncube_encoder --copy config/config.inc.php  /project --into
/encoded-projects
```

Exclude all files matching `*_config.php` from being encoded

```
ioncube_encoder --copy "*_config.php"  /project --into /encoded-
projects
```

Exclude files in `config` and subdirectories from being encoded

```
ioncube_encoder --copy config/ /project --into /encoded-projects
```

Exclude files in `config` from being encoded but not subdirectories

```
ioncube_encoder --copy "config/*" /project --into /encoded-projects
```

2.5.7 Omitting Files from the Encoding Target

When encoding a directory the Encoder will usually fully replicate the directory hierarchy. It can sometimes be necessary to ignore some items from the source tree and this is handled with the `--ignore` option. This option may be used as many times as required.

Example:

Ignore `.svn` files and backup files

```
ioncube_encoder --ignore .svn/ --ignore "*~" /project --into
/encoded-projects
```



Use the `-v` option to see which files are being encoded, encrypted, copied, or ignored when using `--encode`, `--encrypt`, `--copy`, `--ignore` and `--keep`.

2.5.8 Adding Copyright and License Details to Encoded Files

Custom text can be added to the start of encoded PHP files, and this is useful for incorporating your own license or copyright messages. Files are protected so that any changes to an encoded file would prevent it from functioning, and so this text cannot be successfully removed.

Example:

```
ioncube_encoder --add-comment "Software written by FooSystems"
      --add-comment "Licensed to SomeCompany Inc." /project
      --into /encoded-projects
```

3 ENCODER COMMAND LINE OPTIONS

This chapter describes all available command line options, grouped by their purpose.

3.1 Specifying the Source and Target

3.1.1 Source Items

The Encoder can be used to either encode single files or to recursively encode directories. Items to encode are passed to the Encoder without any associated option.

3.1.2 The Encoder Target [`-o, --into`]

The Encoder target may be specified in two ways. The `-o` option encodes a single source file or directory as a new name, and the `--into` option encodes one or more items into an existing directory.

Examples:

Encode file `hello.php` as `hello-encoded.php`

```
ioncube_encoder hello.php -o hello-encoded.php
```

Encode directory `/projects/test` as `/encoded-projects/test`

```
ioncube_encoder /projects/test -o /encoded-projects/test
```

Encode project `/projects/test` into `/encoded-projects`

```
ioncube_encoder /projects/test --into /encoded-projects
```

Encode projects `project1` and `project2` into `/encoded-projects`

```
ioncube_encoder /projects/project1 /projects/project2 --into
/encoded-projects
```

Encode `file1.php` and `file2.php` into `/home/encoded-files`

```
ioncube_encoder file1.php file2.php --into /home/encoded-files
```

To protect against accidental error and possible loss of source files, the Encoder checks for being asked to encode directories that lie within the target tree or encoding into a directory that lies within the source tree. If encoding into the source tree is required then the option `--allow-encoding-into-source` may be used, see 3.15.2 below.

3.2 Encoded File Format

The Encoder can produce files in both binary and ASCII format. These formats are discussed below.

3.2.1 ASCII Format [`--ascii`]

By default, the Encoder produces encoded files in ASCII format. These files contain only printable characters, and may be safely transferred using FTP clients operating in either ASCII or binary mode without being corrupted.

3.2.2 Binary Format [`--binary`]

Binary format files are marginally more efficient than files produced with the default ASCII format. The advantages can be slightly improved runtime performance and a smaller file size, however a significant *disadvantage* is that some file transfer and archive programs, particularly on Windows, may corrupt binary encoded files during processing. Corruption can occur due to different characters being used to represent line breaks on different operating systems, and programs trying to convert these characters when crossing different operating systems. Examples are the CuteFTP file transfer program, WinZip if the *TAR smart cr/lf conversion* option is enabled (which it is by default), and FTP programs transferring in ASCII mode. Some PHP IDE's with FTP features only offer ASCII file transfers, and so guarantee problems.

Using Binary format files if installation of files is performed correctly using tools that will not corrupt the files should be fine. However if this cannot be guaranteed, the default ASCII format is recommended as it still offers good performance, and will greatly reduce the chance of users accidentally corrupting files during installation.

Note that the binary format cannot be used with bundled encodings (section 3.14).

3.3 Encoding to an Existing Directory Target

When an encoding target directory already exists, one of the following options must be used to tell the Encoder what you want it to do. There are four choices, to replace the target, update the target with changed files, merge files into the target, or to rename the target.

3.3.1 Replacing the Target [`--replace-target`]

This option replaces an existing target, ensuring that the target contains no files from a previous encoding. This is the most common option to use when the target already exists.

3.3.2 Merging into the Target [`--merge-target`]

This will preserve the existing target, and all files processed from the source will either be created in the target or overwrite any existing files. Note that any files already part of the target but not present in the source project are preserved.

3.3.3 Renaming the Target [`--rename-target`]

This will rename the target directory by appending a number to it. The number used will be the smallest number to produce a directory name that does not already exist. It is effectively a backup option.

3.3.4 Updating the Target [`--update-target`]

This option is similar to `--merge-target` except that files are only processed if the modified time of the source file is greater than the modified time of the existing target file, or if the target file does not exist.

3.3.5 Create Target Path [`--create-target`]

Create any missing directory components of the target path.

3.3.6 Target Options and Bundled Encodings

Please note that the `--replace-target`, `--merge-target`, `--rename-target` and `--update-target` options should not be used with the `--bundle` and `--add-to-bundle` options (see section 3.14). If they are used together then an error will result.

3.4 Selecting Files to be Encoded, Encrypted, Copied or Ignored

By default, the Encoder will encode files having names ending with `.php` and `.phtml`, and encrypt any files specified with `--encrypt`. All other files will be copied. This section explains how to encode and encrypt files with other extensions, how to prevent files from being encoded, and how to exclude files from being part of the encoding target. For more examples, please see section 2.5. Note that you can use the options described here multiple times and in any order to describe precisely how you require the Encoder to process your project.

3.4.1 Encoding Specific PHP Files [`--encode`]

Use `--encode` to specify additional file patterns, files or directories to encode, or to reverse the effect of `--copy` (see below).

Examples:

Encode all files ending in `.inc` as well as the default extensions

```
--encode "*.inc"
```

Also encode the file `licenses/license.key`

```
--encode licenses/license.key
```

Encode files in directory `tests/encoded`

```
--encode tests/encoded/
```

This last example would be useful if you had used `--copy tests` to tell the Encoder to copy the directory `tests`, but where you want the subdirectory `encoded` to be encoded. Note that this does not request *all* files to be encoded, but ensures that any files matching the default extensions or other file patterns will be.

Encode all files in directory `tests/encoded` but not subdirectories

```
--encode "tests/encoded/*"
```

3.4.2 Encrypting Files [`--encrypt`]

Use `--encrypt` to specify files or directories that are to be encrypted. Encrypted files can be decrypted by the Loader API `ioncube_read_file()` function, see section 6.4.1, and only when called from a file encoded by the same purchased Encoder installation. Typical examples would be encoding template or XML files.

Examples:

Encrypt files ending in `.tpl` or `.xml`

```
--encrypt "*.tpl" --encrypt "*.xml"
```

3.4.3 Copy Files without Encoding or Encrypting [--copy]

Use `--copy` to copy files to the target directory without encoding or encrypting.

Examples:

Copy `user_config.php` instead of encoding it

```
--copy user_config.php
```

Copy files in directory `config` and subdirectories

```
--copy config/
```

Copy files in directory `config` but still encode files in any subdirectories

```
--copy "config/*"
```

3.4.4 Skipping Items in a Project [--skip]

Use `--skip` to skip files and directories in a project. This acts as if the matched items do not exist, and should offer improved performance over the option to ignore files described below.

Example:

Skip `.svn` directories

```
--skip .svn/
```

3.4.5 Excluding Files from the Target [--ignore]

Similar to the skip option, `--ignore` will ignore files and directories and exclude them from the target directory. The difference is that ignored items can still become part of the project target if they match a subsequent `--keep` option, or are a directory that is part of the path to a kept item. To support this, ignored directories are still recursively scanned to discover any items that match a keep rule, which adds overhead. Unless taking advantage of this feature, using `--skip` is therefore recommended instead for performance reasons.

3.4.6 Including Ignored Files [--keep]

Mentioned above, the effect of `--ignore` can be reversed by using `--keep`.

Example:

Ignore contents of directory `project/tmp` but keep files called `README.txt`

```
--ignore project/tmp/ --keep README.txt
```

In this example, `project/tmp` would be ignored and excluded from the target. *However*, if it or any subdirectory contains a file called `README.txt`, the `README.txt` file would be kept and still appear in the target. For this to be possible, the ignored parent directories in the path would still be created to form a valid path.



The Encoder applies the options above in the order they appear. Combining them can achieve precise control over which files are to be encoded, encrypted, copied, or excluded. The `-v` option is useful to see the effect of these options, and will show how files were processed.

3.4.7 Including only Encoded Files into the Target [`--only-include-encoded-files`]

This option will produce a target containing only encoded files. Files that would otherwise be copied are ignored.

3.5 Bytecode Obfuscation

The ionCube PHP Encoder achieves a high level of protection by compiling PHP source code into PHP bytecode, storing this in a proprietary format, and executing code inside the Loader component. This is an analogous operation to compiling a C program into native machine code. Although the entire source code is eliminated during this process, as with compiled C programs, some symbols must remain. As a further level of protection, the ionCube Encoder can apply a *one-way binary transformation* to obfuscate certain data, and elements such as function names will then not be exposed in their original form by PHP features such as `get_defined_functions()` and `get_declared_classes()`. This is different to simple source code obfuscation that tends to be easily reversible, and in addition to user controlled obfuscation, further obfuscations are automatically applied during compilation and at runtime.

3.5.1 Obfuscating Compiled Bytecode Symbols [`--obfuscate`]

The Encoder can obfuscate the names of global functions, local variables, class names, method names, and line numbers. Note that class related obfuscation is not available with the PHP 4 Encoder.

The easiest way to use this option is shown in the following example of enabling all features:

Obfuscate fully

```
--obfuscate all
```

This option can also be used with any combination of the tokens `classes`, `methods`, `functions`, `locals`, `linenos` to allow the obfuscation of class/method/function names, local variable names and line numbers. Separate tokens with a comma and no whitespace. Note that *variable variable assignment* (e.g. `$$keyName = $value`) may not work as expected if local variable obfuscation is used.

Example:

Obfuscate line numbers, class names and function names, but not local variables or method names

```
--obfuscate linenos,functions,classes
```

3.5.2 Specifying an Obfuscation Key [`--obfuscation-key`]

Even though the obfuscation uses a one-way, non reversible algorithm, a custom obfuscation key prevents the possibility of reversal by chance. The obfuscation key must be supplied, and the Encoder will generate an error if it is missing.

Example:

```
--obfuscate all  --obfuscation-key "an apple a day"
```

3.5.3 Specifying Obfuscation Exclusions [`--obfuscation-exclusion-file`]

While it is desirable to obfuscate an application, it is sometimes necessary to prevent certain program elements from being obfuscated. For example, functions and methods external to the application and called by the application should not be obfuscated. Similarly, application interface functions called by external scripts should not be obfuscated.

To provide these exceptions, a text file can be used to specify any functions, classes and methods whose names should not be obfuscated. The option `--obfuscation-exclusion-file` should be used to pass the name of the file to the Encoder, and the file should have sections `[functions]` `[classes]` and `[methods]`, followed by the names of the items to be excluded. The sections may appear in any order, and may be used more than once.
A # character can be used to introduce a comment.

As an example, the following file contents would exclude the class name `GlobalModule`, the class name `Module` within the namespace `Provider`, the method `getName()`, and the global functions `fn1()` and `fn2()`.

```
[classes]
# Exclude our GlobalModule class for introspection purposes
GlobalModule
Provider\Module
[methods]
getName
[functions]
fn1 # used with preg_replace so we mustn't obfuscate
fn2
```

For backwards compatibility, any names appearing before the first section name will be interpreted as function names.

Note 1: excluding a function will also disable obfuscation of any local variables within that function.

Note 2: excluding a class name will exclude just the name of the class from being obfuscated and not any *contents* of the class, such as methods.

Note 3: for security reasons, excluding a method name will exclude it from being obfuscated in all classes having a method of the same name, which avoids needing a reversible obfuscation technique.

Note 4: variable variable assignment (e.g. `$$keyName = $value`) may not work as expected if local variable obfuscation is used. Excluding the function from being obfuscated where such assignments are used will handle this case.

3.6 File Based Server Restrictions (Pro and Cerberus Editions)

The Pro and Cerberus Encoders can optionally encode files with server restrictions such that the files will stop functioning beyond some point in time, and can also restrict the machines on which files can be run. The options for restricting files are described below. As an alternative, files can be restricted to require a license file containing time and server restrictions. The License file approach is recommended and generally preferable, and is described in section 3.7 below.

3.6.1 Expiring Files after a Period [`--expire-in`]

Files can be set to expire after a given number of seconds, minutes, hours or days by using:

```
--expire-in <period>
```

where `<period>` is a number followed by either `s`, `m`, `h` or `d` to denote a period in seconds, minutes, hours or days.

Examples:

Expire files in 7 days

```
--expire-in 7d
```

Expire files in 8 hours

```
--expire-in 8h
```

Note that the expiry period is relative to the time that files are encoded.

3.6.2 Expiring Files from a Date [`--expire-on`]

Files can be set to expire from a specific date by using:

```
--expire-on <yyyy-mm-dd>
```

Example:

Expire files from 2023-07-27

```
--expire-on 2023-07-27
```

3.6.3 Locking Files to Specific Domains and Servers [`--allowed-server`]

Encoded files can be restricted to load only on machines with specific IP addresses and/or domain names. The domain name is also referred to as the server name[1]. Using the Cerberus version, encoded files can also be restricted by MAC (Ethernet) address.

The complete syntax for server restricting files is:

```
--allowed-server [<domain names>][@[<IP addresses>]][{<MAC
address>}]
```

Each server specification can contain as many domain names and IP addresses as desired. If using Cerberus, a MAC address restriction may also be given. At least one type of restriction must be specified, but items are optional. The option may also be used more than once to specify multiple restrictions. **Note:** There is a limit of 250 file based restrictions for an encoded file.

[1] The server name is an attribute set by the web server for each domain or virtual host, and is the value accessed in PHP as `$_SERVER['SERVER_NAME']`.

Restricting by Domain Name

Specify domain names separated by commas and optionally ending the list with an `@` character.

Examples:

Restrict files to `www.foo.com`

```
--allowed-server www.foo.com
```

Restrict files to `www.foo.com` and `www.bar.com`

```
--allowed-server www.foo.com,www.bar.com
```

Restrict files to name `1.2.3.4`

```
--allowed-server 1.2.3.4@
```

Note the trailing `@` after the name in this example. An `@` after a list of domain names indicates that all items before should be treated as domain names, even if they look like IP addresses. Although rare, some domain names can look like the start of an IP address, and if writing a script to automatically process domain names, it is recommended always to add an `@` to the end of the server names to avoid any misinterpretation.

Wildcard Domain Names

Domain name restrictions may contain wildcard characters. The wildcard characters have the following interpretations:

Wildcard Character	*Matches*
`*`	Zero or more characters, e.g. `*.foo.com`
`?`	Any single character, e.g. `site?.foo.com`
`[]`	Any character from a set, e.g. `site[123].foo.com`
`[!]`	Any character not in the set, e.g. `site[!89].foo.com`

Restricting by IP Address

IP addresses may be specified as a single address, a range of addresses or a subnet. Multiple addresses should be specified separated by commas.

Note:

1) When files are accessed via a web server, IP address restrictions are tested against the server IP addressed reported by the web server. When accessed directly, IP restrictions are tested against the network interfaces.

2) For security reasons it is not possible to lock the loopback, 127.0.0.1 localhost address.

Examples:

Restrict files to `192.168.1.4`

```
--allowed-server 192.168.1.4
```

Restrict files to `192.168.1.4` and `192.168.1.20`

```
--allowed-server 192.168.1.4,192.168.1.20
```

Restrict files to `192.168.1.20` through `192.168.1.25`

```
--allowed-server 192.168.1.20-192.168.1.25
```

or

```
--allowed-server 192.168.1.20-25
```

Restrict files to `192.168.1` subnet

```
--allowed-server 192.168.1
```

Restrict files to subnet with 28 significant bits

```
--allowed-server 192.168.1.255/28
```

Restricting by MAC Address

MAC addresses are composed of 6 bytes and should be specified using hex notation as follows.

Example:

Restrict to MAC `00:01:02:06:DA:5B`

```
--allowed-server '{00:01:02:06:DA:5B}'
```

Combining Restrictions

Examples:

Restricting files to a domain name and an IP address

```
--allowed-server www.foo.com@192.168.1.2
```

Restricting files to a domain name and specific MAC address

```
--allowed-server 'www.foo.com{00:02:08:02:e0:c8}'
```

Restricting to either of two domains on either of two IP addresses

```
--allowed-server www.foo.com,www.bar.com@192.168.1.1,192.168.1.3
```

Restricting to a domain name, IP address and MAC address

```
--allowed-server 'www.foo.com@192.168.1.1{00:02:08:02:e0:c8}'
```

3.6.4 Using domain restricted scripts with PHP CLI [`--trust-unnamed-servers`]

By default, scripts restricted by domain will only run via a webserver, and fail with `php-cli` because there is no domain name. Encoding files with the `--trust-unnamed-servers` option will cause the Loader to *ignore* domain restrictions if run with `php-cli` (`php-cgi` still enforces domain checks).
Note: domain name checks with license files are always ignored by `php-cli`.

3.6.5 Exclude checking of interface aliases [`--ignore-interface-aliases`]

Ignore interface aliases when checking IP restrictions, e.g. `eth0:1`

3.7 License Based Server Restrictions (Pro and Cerberus Editions)

A flexible alternative to file based restrictions are license based restrictions. With this mechanism, files are encoded to need a license file, and it is the license file that contains time and server based restrictions rather than the encoded files themselves. A major advantage compared to file based restrictions is that the encoded files may be encoded just once, with a license file containing custom restrictions created for each installation. This is especially beneficial when producing software updates as a single encoded update may be made available to all users, whose existing license files will continue to control the updated encoded files. With file based restrictions, it would be necessary to produce an update encoded for each installation with the same restrictions as the original installation.

Note: License file restrictions override file based restrictions. To avoid accidentally setting file based restrictions when files are encoded to use a license file, the Encoder will generate an error if both file restrictions and license files are used.

3.7.1 Specifying a License File [`--with-license`]

To restrict files to require a license file, use the option

```
--with-license <path>
```

to specify the path of the license file that should be used to restrict the execution of the encoded script. At runtime, the Loader will search for the license file relative to the location of the encoded script, so a relative path should be used when specifying the license. The path will often be just the name of the license file, e.g. `license.txt`

Typically an application will have a single top level directory. In this case the license file could be saved into this top level directory, and the filename of the license could be used on the command line instead of a more complicated relative path. If the Loader cannot locate the license file relative to the PHP script that needs it, the Loader will search parent directories until either the License file is found or there are no further parent directories.

3.7.2 Specifying a Passphrase [`--passphrase`]

License files are encrypted using industry proven algorithms, and with the encryption keyed with a passphrase. Use the command line option

```
--passphrase <key>
```

to specify a passphrase. The passphrase used when encoding files must match the passphrase used to generate the corresponding license file in order for the Loader to successfully decrypt the license file when the script is executed. See section 5.2.5 below for more details. If the Loader cannot decrypt the license file it will prevent execution of the script.

3.7.3 License Check Mode [`--license-check`]

When an encoded file restricted by a license is read by the Loader, there are two methods by which the license restrictions can be enforced.

Automatic License Checking

With automatic checking, the Loader will ensure that all server restrictions are matched, that the license has not expired, and that all enforced properties are matched in the license file (see section 5.2.12 below). This is the default method, but it can also be specified by encoding with the option

```
--license-check auto
```

Script Based License Checking

Script based checking is an alternative that encodes files so that they will still run even if their license file has expired or is not matched to their server. A script can then use the Loader API to validate properties, server restrictions, and any expiry date contained in the license. In order to implement a manual license check, and so prevent the Loader from automatically validating the license, encode files with the option

```
--license-check script
```

Several Loader API functions useful when implementing a manual license check are described in chapter 6. Be careful to ensure code security when implementing a license validator in PHP. In particular if a validator is contained in a file which will be included in each top-level script, then it is necessary to use Include File Protection to ensure that a hostile user cannot replace the validator with a different file.

3.8 Target File Attributes

3.8.1 Copying with Hard Links [`--use-hard-links`]

The Encoder will normally copy any files into the target that are neither encoded nor encrypted. This is fast, but performance can be improved and disc space saved by using the `--use-hard-links` option to replicate by using hard links. This feature is only available with Unix Encoders, and only if the source and target files are on the same filesystem.

3.8.2 Using Default File Permissions [`--without-keeping-file-perms`]

This option applies the default file permissions to target files instead of copying permissions of the corresponding source file.

3.8.3 Updating File Times [`--without-keeping-file-times`]

This option creates target files with a current timestamp instead of copying times from the corresponding source file.

3.8.4 File Ownership [`--without-keeping-file-owner`]

When running as root on Unix the Encoder will usually copy file ownership from source files and directories. This option will create target items as the user running the Encoder.

3.8.5 Setting File Ownership [`--apply-file-user`, `--apply-file-group`]

When running the Encoder as root on Unix, different user and group IDs can be set for target files with:

`--apply-file-user` *<user id/name>*

`--apply-file-group` *<group id/name>*

The `id` or `name` may be either a numeric id or a name.

3.9 Language Options

3.9.1 Ignoring Short Open Tags [`--no-short-open-tags`]

Use this option to recognise only PHP files that use `<?php ?>` style tags. By default the Encoders for PHP 4, 5 and 7 recognise both `<?php ?>` and `<? ?>` style. Note that support for short open tags was removed in PHP 8, so they are unrecognised even if this option is not used. Note also that this option will be removed in a future version of the software, so its use is discouraged.

3.9.2 Ignoring Strict Language Warnings [`--ignore-strict-warnings`]

This option ignores warnings generated by the compiler from the use of language features that go against strict language usage rules. Resolving source code issues rather than using this option to hide them is recommended in the long term.

3.9.3 Ignoring Deprecated Feature Warnings [`--ignore-deprecated-warnings`]

This option ignores warnings generated by the compiler from the use of deprecated language features. Resolving source code issues rather than using this option to hide them is recommended in the long term.

3.9.4 Register Custom Auto Globals [`--register-autoglobal`]

Specify the names of custom variables that are to be treated as if they were *autoglobals* (also known as *superglobals*).

For example:

```
--register-autoglobal MYAUTO
```

would encode access to `$MYAUTO` as if it were a global even if not explicitly made global with the `global` keyword. It is not necessary to use this option for standard autoglobals, but this feature may be useful if encoding files containing references to variables that are declared as autoglobals by a PHP module that will be used in the target system but that is unknown to the Encoder.

3.10 Encoded File Header Customisation

NOTE: Runtime (on demand) install for older versions of the Loader as mentioned below is a feature that uses the `dl()` function in PHP, which in PHP 5.3 became restricted and less flexible, and will be removed completely from PHP in the future. Features related to runtime install of the Loader are described for completeness as they still exist, however installing of the Loader should generally be achieved via a `php.ini` file.

Encoded files contain a PHP header (the *preamble*) that, if required, will perform the run-time installation of the Loader. It will also produce an error message if no Loader could be installed or in some cases of file corruption. The default header is ideal for most cases, however Encoder options support customising parts of the header and for setting an entirely new header.

Files may also have custom comments added. This feature offers the addition of plain text such as copyright messages, a product version number, contact details, and so on. Embedded digital signatures protect encoded files from tampering, and any modifications to an encoded file will render it useless so that such messages cannot be successfully changed or removed.

3.10.1 Generating Files with no PHP Header [`--without-loader-check`]

This option produces files with no PHP header and only the encoded file data. When running files without a header there will be no error produced if there is no Loader installed and the Loader must be installed in the `php.ini` file.

3.10.2 Customising the 'no Loader installed' Message [`--message-if-no-loader`]

To customise the message produced if no Loader is installed, use:

```
--message-if-no-loader <text>
```

`<text>` must be a valid PHP expression and is passed to the PHP `die()` function.

Example:

```
--message-if-no-loader "'No Loader is installed. Please contact
support.'"
```

Note the use of single quotes around the message because a string is being passed to the `die()` function.

3.10.3 Customising the 'no Loader installed' Action [`--action-if-no-loader`]

To customise the action when no Loader is installed, use:

```
--action-if-no-loader <php code>
```

3.10.4 Setting the Run-Time Loader Path [`--loader-path`]

To change the run-time Loader path, use:

```
--loader-path <path>
```

The current default setting performs selection of the Loader based on operating system type and PHP version, and is:

```
'/ioncube/ioncube_loader_'.$__oc.'_'.substr(phpversion(),0,3).
(($__oc=='win')?'.dll':'.so')
```

`$__oc` is predefined in the header as:

```
strtolower(substr(php_uname(),0,3))
```

Changing the Loader path may be useful if you wish to distribute Loaders with your application but in a different directory, or if you wish to use run-time Loading but do not need to use the dynamic selection of the Loader.

3.10.5 Setting the Header Code [`--preamble-file`]

The entire PHP header may be set by using:

```
--preamble-file <file>
```

`<file>` should be the path to a file containing PHP code to place at the start of the encoded files.

3.10.6 Header Comments [`--add-comment`, `--add-comments`]

To add text to appear as comments at the start of encoded files, use:

```
--add-comment <text>
```

The option may be used as many times as required.

To add comments from a file, use:

```
--add-comments <file>
```

Examples:

```
--add-comment "Copyright New FooBar Inc. 2023" --add-comment "All
Rights Reserved"
```

```
--add-comments custom/comments.txt
```

3.11 Customising Loader Behaviour

3.11.1 Loader Event Messages [`--loader-event`]

Loader messages generated by run-time events can be customised at encoding time to those of your own choice.

For each web request, Loader messages customised by an encoded file will take effect for all other encoded files unless a later included file provides a different message.

To customise an event message, use:

```
--loader-event "<event>=<message>"
```

`<event>` should be the event type to customise and `<message>` the text to associate with the event.

Event types are:

Event Type	*Triggered When…*
`corrupt-file`	An encoded file has been corrupted.
`expired-file`	An encoded file has reached its expiry time.
`no-permissions`	An encoded file has a server restriction and is used on a non-authorised system.
`clock-skew`	An encoded file is used on a system where the clock is set more than 24 hours before the file was encoded.
`license-not-found`	The license file required by an encoded script could not be found.
`license-corrupt`	The license file has been altered or the passphrase used to decrypt the license was incorrect.
`license-expired`	The license file has reached its expiry time.
`license-property-invalid`	A property marked as ‘enforced’ in the license file was not matched by a property contained in the encoded file.
`license-header-invalid`	The header block of the license file has been altered.
`license-server-invalid`	The license has a server restriction and is used on a non-authorised system.
`unauth-including-file`	The encoded file has been included by a file which is either non-encoded or has incorrect properties.
`unauth-included-file`	The encoded file has included a file which is either non-encoded or has incorrect properties.
`unauth-append-prepend-file`	The `php.ini` has either the `--auto-append-file` or `--auto-prepend-file` setting enabled.

Example:

```
--loader-event "expired-file=This software has expired."
```

Custom messages may also contain display formats. Each format is replaced with specific text as follows.

Format	*Replaced With*
`%f`	The path of the file generating the event.
`%i`	Server IP address ('no-permissions' event only).
`%h`	Server name ('no-permissions' event only).
`%n`	The path of the unauthorised including or included file.

3.11.2 Callback Files [`--callback-file`]

To implement a more elaborate error handling mechanism than with Loader Events, a *callback file* can be specified to handle Loader error cases. Use the option

```
--callback-file <relative-path>
```

to specify a callback file. The path should be relative to the top-level directory of the PHP application. In particular, if the callback file is contained in the top-level directory, then specify the filename rather than a full path.

The callback file should contain a function with the signature:

```
function ioncube_event_handler($err_code, $params)
```

The error code will be passed as the first argument, and an associative array of context-dependent values will be passed as the second argument. The error code is an integer, and for convenience the Loader defines constants for all event error codes. See section 3.11.3 for a list of all constants.

The name of the file that caused the error is always passed as a parameter with key `current_file`, and for server restriction errors parameters are passed with keys `domain_name` and `ip_address`. The path to the expected license file is passed with key `license_file`, and for errors related to include file restrictions, the file that included the encoded file, or was included by the encoded file, is passed with key `include_file`.

3.11.3 Loader Event Constants

The Loader defines PHP constants corresponding to the supported error codes. These codes correspond to the Loader events described in the previous section follow:

Value	*PHP constant*	*Loader Event*
1	ION_CORRUPT_FILE	corrupt-file
2	ION_EXPIRED_FILE	expired-file
3	ION_NO_PERMISSIONS	no-permissions
4	ION_CLOCK_SKEW	clock-skew
5	(constant intentionally unused)	
6	ION_LICENSE_NOT_FOUND	license-not-found
7	ION_LICENSE_CORRUPT	license-corrupt
8	ION_LICENSE_EXPIRED	license-expired
9	ION_LICENSE_PROPERTY_INVALID	license-property-invalid
10	ION_LICENSE_HEADER_INVALID	license-header-invalid
11	ION_LICENSE_SERVER_INVALID	license-server-invalid
12	ION_UNAUTH_INCLUDING_FILE	unauth-including-file
13	ION_UNAUTH_INCLUDED_FILE	unauth-included-file
14	ION_UNAUTH_APPEND_PREPEND_FILE	unauth-append-prepend-file

3.12 File Properties and Include Attack Prevention

File *properties* are key-value pair data items that are securely stored as metadata in encoded files, separate to the PHP code. Properties can be read by a Loader API function, and also used with the *include attack* prevention system. Include attacks are where program scripts are replaced by unauthorised ones in an attempt to change program behaviour. To guard against this, encoded files can be protected so that they can only be included by a file with specific properties defined, and so that they can only include a file if it has certain properties. This powerful feature can also help prevent unauthorised use of libraries by allowing your included files to only be included by your own application, and not by someone else's program.

3.12.1 Setting Properties [`--property`, `--properties`]

To define one or more properties, use:

```
--property "<name>[=<value>]"

--properties "<name>[=<value>][, ...]"
```

`name` is the name of the property to be defined and `value` is the property value. Use a comma to separate multiple properties.

Values can be numeric, a string that is optionally delimited by `' '` or `" "`, or an array delimited by `{ }`. Array elements may optionally have keys.

Examples:

Define a property `version` with integer value `5`

```
--property version=5
```

Define a property `version` with string value `"2.0"`

```
--property "version='2.0'"
```

Define several properties

```
--properties "version=1,company='Foo Technologies'"
```

Define an array

```
--property "features={options=>{'save','load'},licensed_to=>'Foo
Tech Inc.'}"
```

3.12.2 Include Attack Prevention [`--include-if-property`]

To restrict which files can be included by a file and also the files that can include a file, use:

```
--include-if-property "<name>=[<value>][, ...]"
```

Property `name` and `value` are as defined in section 3.12.1 above.

This option may be used more than once to define multiple sets of required properties.

Examples:

Include files if property `program_name` has value `"my app"`

```
--include-if-property "program_name='my app'"
```

Include files if property `pname` is either `"app1"` or `"app2"`

```
--include-if-property "pname='app1'" --include-if-property
"pname='app2'"
```

3.12.3 Preventing Prepend and Append File Usage [`--disable-auto-prepend-append`]

By utilising the `auto_prepend_file` and `auto_append_file` settings in a `php.ini` file, it is possible to specify a PHP file which should run before or after any other scripts. This may undermine the security of encoded PHP scripts, and can be disabled using the option

```
--disable-auto-prepend-append
```

If this option is used and a server has either the append or prepend `php.ini` setting enabled, the encoded scripts will not run. Some servers may have a legitimate reason for enabling these settings, so this Encoder option is not enabled by default.

3.13 Project Handling

Setting up the Encoder for single-command repeat encoding of projects can easily be performed using Unix shell scripts or Windows batch files, however the Encoder also has a built-in projects handling feature that may usefully be used as well or instead.

The projects feature uses a project file to store and provide command line options. Project file options are merged with any additional options passed to the Encoder, and the project file may be updated or recreated when required. As the project file is a plain text file, it can also be edited if necessary.

3.13.1 Specifying the Project File [`--project-file`]

The project file to use is set with:

```
--project-file <file>
```

Once a project file has been created, to encode with the given project options, only this option need be used.

3.13.2 Creating the Project File [`--create-project`]

This option creates the project file named with `--project-file`. The file is created or overwritten, and is set with whatever other options are used to the Encoder.

Examples:

Create and initialise project file `p1`

```
ioncube_encoder --project-file p1 --create-project /project1 --into
/encoded-apps
```

Repeat encoding based on project file `p1`

```
ioncube_encoder --project-file p1
```

Repeat encoding based on project file `p1` but with verbose mode enabled

```
ioncube_encoder --project-file p1 --verbose
```

3.13.3 Update a Project File [`--update-project`]

This option updates a project file by merging in any new options.

Example:

Repeat encoding based on project file `p1` but permanently add the `--replace` option

```
ioncube_encoder --project-file p1 --replace --update-project
```

3.14 Bundled Encodings

Multiple encodings may be added to one encoded file to cover cases where an encoded file is not supported by all of the desired target possibilities. This is useful for the transition between generations of PHP, such as PHP 5 to PHP 7 and PHP 7 to PHP 8, and for PHP 8.1 and 8.2.

3.14.1 Starting a Bundled Encoding [`--bundle, -B`]

A bundled encoding can be started by using the `--bundle` option (short form: `-B`) to the Encoder. We could start a bundle with the Encoder for PHP 7.1 :

```
ioncube_encoder -71 --bundle example_dir -o example_dir_encoded
```

which will use the PHP 7.1 Encoder to encode all files in the `example_dir` to produce the files in `example_dir_encoded`. As those stand, the encoded files can run on PHP 7.1 to 7.4. The files in `example_dir_encoded` can then have other encodings added to them by using the `--add-to-bundle` option.

3.14.2 Adding to a Bundled Encoding [`--add-to-bundle, -A`]

Extra encodings can be added to an existing bundled encoding with the `--add-to-bundle` option (short form: `-A`):

```
ioncube_encoder -81 --add-to-bundle example_dir -o
example_dir_encoded
```

That adds PHP 8.1 encoding to the encoded files already in `example_dir_encoded`. The resulting encoded files will then run on PHP 7.1 to 7.4 and 8.1 with the appropriate ionCube Loader installed. Additionally, adding a further encoding for PHP 8.2 would cover PHP 8.2, 8.3 and 8.4. This is possible because the Loaders for PHP 8.3 and 8.4 and can process files encoded for PHP 8.2, as well as those encoded specifically for PHP 8.3 or 8.4.

A file in the target *must* exist if a file with the same name in the source is being encoded with the `--add-to-bundle` option. If the file in the target does not exist then an error will result.

Adding to an encoding to a file having that encoding already will also produce an error. e.g.

```
$ ioncube_encoder -81 --add-to-bundle example_dir -o
example_dir_encoded

Error: Encoding with PHP source language 8.1 already exists
```

3.14.3 Bundled encoding and the `--*-target` options

The `--replace-target`, `--merge-target`, `--rename-target` and `--update-target` options should not be used with the `--bundle` and `--add-to-bundle` options. This is because the `--bundle` option will replace the existing directory, and `--add-to-bundle` will add encodings to those files. Each file to be encoded in the source must have a corresponding file in the target for adding to a bundle to work. So the following will produce an error:

```
$ ioncube_encoder -81 --add-to-bundle --merge-target example_dir -o
example_dir_encoded

Error: Only one of --update-target, --rename-target, --replace-
target, --merge-target, --bundle and --add-to-bundle should be given.
```

3.14.4 Adding header comments with extra encodings [--keep-comments]

Header comments are normally only added to the first encoding in a bundle as otherwise the comments would appear down in the body of the file just before the start of the encoding being added. So if you have give an --add-comment option along with the --add-to-bundle option then the comment will just be ignored rather than added before the extra encoding. If you really do want the comment to be added before the new encoding then please use the --keep-comments option:

```
ioncube_encoder -81 --add-to-bundle --keep-comments --add-comment "A
comment for the extra encoding" example_dir -o example_dir_encoded
```

In the resulting encoded files the comment "A comment for the extra encoding" will appear just before the start of the PHP 8.1 encoding that has been added.

So in the resulting encoded file you would see something like this at the point where the latest encoding has been added:

```
AfqqdUbOzrICH8asIGy6/FERcTNRy8sfholmrB/aiQJrfm/7q9/EBV04MsDcoQ8q8xVReaL
2bXnvggG1OYXQ5mAFAzBTN3YSGcx5aYs0yvHjMgT2DO2HgMdOFGePRTDZ3rp/24PZefUofH
fcl00hMcMjAXtyKszeTf0HoTXSJl/p6/kUBs6grf8Kp4zOFPRINQVBdTGoXKsGFmsN9tm3W
O8RDhU+82TPj6PLTkSlycK65AE4Fkyl0BXVWR60FlVFHTbwIqRiqTxY+SibTXtZJz9kb2w9
t5x7hgg6IkNZ=
<?php //00032

// A comment for the extra encoding

?>
HR+cPuepB5Wa0PMVCOsvU+8+sBZoYulWKypbDgkuoRL2Eoa/4nbupwJV/weQVVOeaw3xS3V
B7UzKZU2LnaERgnll7uWv/zJwSrHArdntAt3e7ihdGwrElUtBiCKWEFVSufwYhDnopqsqg5
D04Vn1O8BzbqUpU+N1DCqx7hGmlfQ58dtjOtz2dmdMSJgMbGONefqu704H6TXlTsSBaTGDG
QqrfYnGG6tJoM8tR1aCO2zCBIE/t0e4k62yH67QPVZVysMAdfUeJ2y1ppBpwd2s1s+C9dWL
wwXeLTpwcmp1P+9QC14A5zP70fWkXs21O0Bx92iKiCS4tw/H+HOXAtxVyE+qXUqYn0NOx+L
qt36kTnnDvEqGvTNpqoBkNyXq
```

3.14.5 Bundled Encodings and Encryption of Non-PHP Files

Bundled encoding has no effect on the encryption of non-PHP files. So if a new encoding is being added to a bundle then the non-PHP files are simply re-encrypted each time.

3.14.6 Limits on Bundled Encoding Files

Bundled encodings cannot go beyond 268MB. If an attempt is made to add an encoding beyond that limit then an error will result.

3.14.7 Bundled Encodings and Encoded File Format

Bundled encodings can only be used with the ASCII encoding format. They cannot be used with the binary encoding format.

3.15 Miscellany

3.15.1 Encoding and Bytecode Optimisation [`--optimise, --optimize`]

By default, the Encoder uses an encoding format that encodes with the best Encoder performance and good run-time performance. At the expense of increased encoding time, smaller files with possibly marginally better run-time performance may be obtained by increasing the optimisation level.

The options:

```
--optimise more

--optimise max
```

increase optimisation to either an intermediate or a maximum level.

The option `--optimize` is an alias for `--optimise`

3.15.2 Allowing Encoding into the Source Tree [`--allow-encoding-into-source`]

By default, the Encoder prevents a target directory to be within the source tree or for the source directory to be within the target tree. This is to prevent accidental overwriting of source files or unexpected results. The `--allow-encoding-into-source` option allows this. To avoid the target being treated as part of the source tree use the `--ignore` option to ignore the target.

Examples:

In these examples we have a simple project in a directory called `test`. The project contains the file `helloworld.php`

Encoding with the default safety check

```
$ ioncube_encoder test -o test/encoded

Error: Can't encode to a directory within the source tree
```

The Encoder safety check prevented encoding because the target is within the source tree.

Encoding with the default safety check disabled

```
$ ioncube_encoder test -o test/encoded --allow-encoding-into-source --ignore encoded/

$ ls -R test

test:

encoded/  helloworld.php

test/encoded:

helloworld.php
```

Source directory with source file and encoded target

Target directory with encoded result

3.15.3 Omitting Documentation Comments [`--no-doc-comments`]

Documentation comments are comments with the following syntax:

```
/**
My code comment
*/
```

These comments are exposed by the PHP 5 reflection API, and are preserved by the PHP 5 Encoder by default. In order to omit these documentation comments from encoded files specify the `--no-doc-comments` option.

3.15.4 Setting an alternate Shell Script Line [`--shell-script-line`]

The Encoder will encode shell scripts having PHP as their interpreter unless explicitly directed otherwise with `--copy` or `--ignore`. By default, the first line of the source shell script will be copied to the target, but a custom line can be used with the following option

```
--shell-script-line '#!/usr/bin/php -q'
```

where the text inside the quotes is an example shell script line. Any encoded script can be used as a shell script by manually adding a shell script line to the encoded file after the encoding process is complete, and similarly an encoded PHP shell script will remain a valid encoded PHP script if the shell script line is removed. On Unix it is important to enclose the option argument in single quotes as the '`!`' character has a special meaning for most shells.

3.15.5 Enforce Minimum Loader Version [`--min-loader-version`]

This option encodes files with a requirement that the Loader version is of the specified version or greater. This can be useful to ensure that a particular feature of the Loader is supported. The option should be specified as a version string in the format `major[.minor[.revision]]`.

Example:

```
--min-loader-version 12.0.0
```

3.15.6 Check for Program Updates [`--check-version`]

Check for the availability of a program update.

3.15.7 Program Version [`-V, --version`]

Displays the program version.

3.15.8 Verbose Mode [`-v, --verbose`]

Verbose mode will produce details of Encoder operations and progress.

3.15.9 File Verify [`--verify`] (*legacy option*)

If run-time loading is to be used then files must be able to be read and parsed by the PHP engine as valid PHP files. This will increase encoding time, and is a legacy option that would only be necessary if you had customised the PHP header and wish to check it.

3.15.10 Help [`-h, --help`]

This option displays a summary of Encoder options.

4 EXTERNAL AND DYNAMIC KEYS

4.1 Introduction

As part of a protection strategy, compiled code encoding systems may protect the compiled code (bytecode) with some kind of key in order to present challenges to a would-be hacker intent on analysing the files. Such a key must be present when files are processed otherwise running the protected code would not be possible, and a key will normally be stored somewhere within the protected file itself. This can work well, but if an attacker has learnt how keys are stored, their effectiveness is inevitably diminished.

A more powerful approach is to have keys that are not within the files that they protect, and the ionCube Encoder has two innovative mechanisms that can be used to present significant challenges to would be attackers.

External Keys allows for static keys to be located externally to the encoded files, and is particularly useful where the end user is trusted and the key file(s) can be kept away and protected with file permissions more than the encoded files themselves. You might choose to protect files of your own website this way.

Dynamic Keys is a radical concept whereby decoding keys do not exist statically at all, and instead, only exist at runtime and as a result of running part of the protected program itself. This is described in depth from section 4.3.

Please note that these features are only available for the PHP 5.3 and above Encoders, so `ioncube_encoder53`, `ioncube_encoder54`, etc. up to `ioncube_encoder84`. They cannot be used with the `ioncube_encoder` (for PHP 4) and the `ioncube_encoder5` (for PHP 5.0) binaries.

Both features mean that all or part of an encoded file cannot be decoded using just the encoded file contents. External keys will require additional information from some other file location and dynamic keys will only be generated during the actual execution of the file.



Note that if an external or dynamic key is not correct at runtime then unpredictable behaviour may occur. Whilst the ionCube Loader tries to catch such errors in decoding, in some situations they may not be caught and this may cause PHP to fail.

4.2 External Keys

External keys are encoding keys located outside of an encoded file. A key may be a string value, or the contents of a file. Their use means that anybody trying to reverse engineer an encoded file will need not only the encoded file itself, but also the separately stored external key. External keys affect how the entire file is encoded. There are three types of location for an external key:

- As a file on the system.
- As a PHP configuration (`php.ini`) property.
- As a property of a license file required by an encoded file.

4.2.1 File System External Keys [`--encoding-key file:...`]

For file system keys, the format of the option is:

```
--encoding-key file:<RUNTIME FILE PATH>=<ENCODING-TIME PATH>
```

In the above, the `<ENCODING-TIME PATH>` will be the absolute path to the file whose contents will be used as the encoding key.

The `<RUNTIME FILE PATH>` will be the relative or absolute path to the file used by the ionCube Loader to decode the encoded file at runtime.

As an example, suppose we have a file called `test.php` that we want to encode to produce a file called `test_enc.php`. We will use the contents of a file `/home/dev/mytext.txt` as the encoding key. When the encoded file is run by the Loader, it should read the same key from the file `dk.txt` that is located in a directory called `xyz`. To do this, we would encode as follows:

```
$ ioncube_encoder test.php -o test_enc.php --encoding-key
'file:xyz/dk.txt=/home/dev/mytext.txt'
```

If the file `/home/dev/mytext.txt` does not exist when encoding, the Encoder will produce error: `Error in accessing encoding key file`.

When the Loader processes `test_enc.php` at runtime, it will find that decoding depends on the contents of an external key file. As the expected path is relative, the Loader will append that path to the location of the script being decoded. So in this example, if the encoded file is at `/var/www/html/test_enc.php`, the Loader would attempt to read the key from `/var/www/html/xyz/dk.txt`. When encoded with version 14 or later and run using Loader 14.4.0 and above, the Loader will also search for a relative key path in parent directories of the script until it is found, i.e. `/var/www/xyz/dk.txt`, `/var/xyz/dk.txt` and `/xyz/dk.txt`. If the path for a key file is absolute, the Loader will just attempt to read that file.

If a key file exists but the contents do not match the encoding key then a runtime error will occur indicating that a decoding problem has occurred.

Using this method for external keys means that the location of the decoding key file will be included within the encoded file. Although convenient and ideal for many cases, this may make it slightly less secure than the other two methods described next; PHP configuration (`ini`) properties and using license properties (available with the Pro and Cerberus editions).

4.2.2 Using Configuration Properties With External Keys [`--encoding-key ini:...`]

For PHP configuration (`php.ini`) properties the format of the option is either

```
--encoding-key ini:<INI PROPERTY>="<INI STRING VALUE>"
```

or

```
--encoding-key ini:<INI_PROPERTY>=<FILE LOCATION>
```

In the above, the `<INI PROPERTY>` is the base name of the ini property. At runtime the Loader will expect that name to be prefixed by `ioncube.loader.key`. The `<INI STRING VALUE>` will give the encoding key as a string value whilst `<FILE LOCATION>` will give a path to a file whose contents will be the encoding key. The double quotes (`" "`) indicate to the Encoder that the value is to be used directly as the key rather than being interpreted as a file name.

Unix: Use single quotes around the entire option value or escape double quotes with \ to ensure that the quote is not interpreted by the shell.

Those values will also indicate the type of values that the Loader will expect to see when decoding. So, if a string is provided as the encoding key then the Loader will expect the ini property to have the same string as a decoding property. Similarly, if a file location is provided as the encoding key then the Loader will expect the configuration property to have a file location value.

Where the encoding key ini property can be set is detailed in section 4.2.3.

Instead of using the file system external key to encode our example `test.php`, suppose we decide to specify that an ini property that will give the decoding key. To do that we would use:

```
$ ioncube_encoder test.php -o test_enc.php --encoding-key
'ini:myprop=/home/dev/mytext.txt'
```

This means that again the contents of the file `/home/dev/mytext.txt` will be used to encode. However, at runtime, the Loader will look for the ini property `ioncube.loader.key.myprop`. Suppose that property does exist in the main `php.ini` file as follows:

```
; External key setting with file value

ioncube.loader.key.myprop = subdir/kf.txt
```

Then, to decode the `test_enc.php` file the Loader finds the property `ioncube.loader.key.myprop` together with its value `subdir/kf.txt`. The Loader will then use the contents of `subdir/kf.txt` as the decoding key.

Again, if the encoding key file does not exist then the Encoder will produce an error at encoding time. Similarly, the Loader will produce an error if the `ini` property cannot be found or if its file value does not exist. If the file does exist but has the incorrect contents (that is, not the same as the encoding file) then a runtime error will be produced indicating that a decoding error has occurred.

As `ini` properties separate from the encoded file itself it is possible to use a string value directly as the encoding key. Going back to our example that would be done as follows:

```
$ ioncube_encoder test.php -o test_enc.php --encoding-key
'ini:myprop="encoding key phrase"'
```

Note that the encoding key is included in double-quotes (`" "`) to indicate that it is a string key rather than a file path. The Encoder will use that string as the encoding key directly. At runtime the Loader will look for the value of the ini property `ioncube.loader.key.myprop`. For decoding to be successful the value of the key must be the same as that used when encoding. So if we had the following set in the `php.ini` file then decoding would be successful and `test_enc.php` would be executed:

```
; External key setting with string value

ioncube.loader.key.myprop = "encoding key phrase"
```

Using a string as the value is slightly less secure than using a file since there is one less step required to obtain the key and, in addition, the `php.ini` file may be generally accessible. On the other hand, using a string value means that no extra key file needs to be provided.

4.2.3 Where External Key ini Properties Can be Set

As well as being set in `php.ini` files, configuration properties for encoding keys can be set in `.htaccess` files (when using Apache as the web server), `.user.ini` files (when PHP is run in CGI mode) and even using `ini_set()` in a PHP file. In the latter case the PHP file itself be encoded to depend upon the same external key that is set with `ini_set()` in its body.

4.2.4 Using License Properties With External Keys [`--encoding-key lic:...`]

The Pro and Cerberus editions of the ionCube PHP Encoder can produce license files which can be used to limit the usage of an encoded application to a domain, IP address or, in the case of Cerberus, MAC address. Properties, as key-value pairs, can also be set in a license. Further details on license generation may be found in section 5.2.

License properties can be used with encoding keys and the format of the option is:

```
--encoding-key lic:<LIC PROPERTY>="<LIC STRING VALUE>"
```

or

```
--encoding-key lic:<LIC_PROPERTY>=<FILE LOCATION>
```

Those options are very similar to the case for the PHP configuration properties except that the keys will be obtained from license properties rather than `ini` file properties. So the `<LIC PROPERTY>` will be the license property that the Loader will need to find in the license when decoding. The `<LIC STRING VALUE>` will be the encoding key as a simple string value whilst the `<FILE LOCATION>` will give the path to a file whose contents will be used as the encoding key.

As with PHP configuration properties, if a string is given as the encoding key then the Loader will expect to find a simple string as the value of the license property but if a file path is given as the encoding key then the Loader will expect to find a file path as the value of the license property.

As an example of using properties of licenses with external keys, suppose we encode our `test.php` file to depend on a license called `mylic.txt`. That license file can contain a property whose value gives the decoding key. That can be done as follows:

```
$ ioncube_encoder test.php -o test_enc.php --encoding-key
'lic:licprop=/home/dev/mytext.txt' --with-license mylic.txt
```

The `mylic.txt` license file may be generated using the `make_license` program as follows:

```
$ make_license -o mylic.txt --property 'licprop=kprop.txt' --allowed-
server www.example.com
```

That will create a license restricted to the domain `www.example.com` and containing a property called `licprop` that gives the location of the external decoding key, a file called `kprop.txt`.

At runtime, apart from checking that the license is present and valid for the server, the Loader will obtain the value of the `licprop` property of the license. The Loader will then get the contents of `kprop.txt` and use that to produce the decoding key for `test_enc.php`. Again, if either the license property or the file `kprop.txt` cannot be found then the Loader will indicate that a decoding error has occurred.

As with configuration properties, a simple string may be used as the key instead of the contents of a file. In that we case we would have the following:

```
$ ioncube_encoder test.php -o test_enc.php --encoding-key
'lic:licprop="secret prop value"' --with-license mylic.txt
```

The license property would have to have a string value to match:

```
$ make_license -o mylic.txt --property 'licprop="secret prop value"'
--allowed-server www.example.com
```

The method of using license properties for external keys has the advantage that the license file itself is encrypted whereas an `ini` property may be in a readable, plain text configuration file. Furthermore, the Loader will stop execution if the license file is not valid for the server on which files are being run.

4.2.5 External Keys and Security

For string value external encoding keys (in the `ini` and license property cases), longer keys should provide greater security. We recommend when using string value keys that the length is 16 or more characters. When using file contents we suggest that they contain at least 64 characters but not more than 256K.

4.2.6 Uniqueness of External Key Values

The license property used for an encoding key must have a unique value across your encoded projects. You should not, for example, have an encoding key property with the name `foo` that has one value in one license (e.g. `"secret prop value 1"`) and a different value (`"secret prop value 2"`) in another license. If you need to use different encoding key values be sure to use different names for your encoding key properties, such as `foo1` and `foo2`.

4.2.7 Errors That May Occur When Encoding With External Keys

The ionCube Encoder may stop with the following errors:

- `Cannot have more than one encoding-key argument`
  Only one `--encoding-key` option may be given to encode a file or directory.
- `Error in accessing encoding key file [file name]`
  In this case the encoding key file cannot not be found.
- `Cannot open encoding key file [file name]`
  The Encoder cannot read the specified file in this case.
- `Error attempting to allocate [number] bytes when reading encoding key file [file name]`
  This indicates that the encoding key file is too large to be read into memory successfully.
- `Error when reading encoding key file`
  This indicates some other error when reading the encoding key file.
- `Unable to allocate memory for hash of encoding key`
  This indicates excessive memory use when generating a hash of a string or of a file contents.
- `Error hashing encoding key file [file name]`
  This indicates some other error when generating a hash.
- `Error in hashing encoding key string [string]`
  This indicates an error during the hashing of an encoding string.

The Encoder will also give the warning `--encoding-key option only applies for PHP 5.3+ encoding` if an encoding-key option is used with the Encoders for the PHP 4 or PHP 5.0 source languages.

4.2.8 Errors That May Occur With External Keys at Runtime

If an external key cannot be found at runtime then the ionCube Loader will produce the error `The file [file name] could not be decoded as an encoding key was not found`.r

If an inconsistency due to an incorrect run time key is found in the byte code during execution, the Loader will produce the error `Unable to call protected script`. The format of these errors can be specified by the `--dynamic-key-errors` option to the Encoder that is detailed in section 4.3.12.



Note that if an external key is not correct at runtime then unpredictable behaviour may occur. Whilst the ionCube Loader tries to catch such errors in decoding (producing the `unable to call protected script` error), in some situations they may not be caught and this may cause PHP to fail.

4.3 Dynamic Keys

While the external key mechanism described above does provide an enhanced level of protection, it should be noted that the key is still "static" in the sense that it is either a simple string value or the contents of a file which must have the same contents as that used when encoding. Moreover, the decoding key will be obtained at the start of decoding and will apply to all the byte code in the file. Potentially, therefore, the key could be obtained without actually running the encoded application.

With ionCube encoded files, functions are only decoded when they are first called in the application. This "dynamic decoding" approach means that the decoded byte code only exists at the last possible moment. The only exception to this is for closures, which are decoded at the point that they are read from an encoded file.

In addition, *Dynamic Keys* takes the idea of dynamic decoding a significant step further. The basis of this feature is that a decoding key can only be obtained by actually running the PHP application, and will be the result of some internal computation.

Unlike external encoding keys, dynamic keys are specified as *annotations* within the source code, and can be specified on a per function or method basis rather than for the entire file. Functions are only decoded at the point that they are first called, and the dynamic key may be obtained in one of the following ways:

- The contents of a *file* or *URL* that may be either local or remote. The file only has to exist at the point the particular function is being first called and decoded.
- The value of a *variable* at the point the function is first called.
- The computed value of a different *function* at the point that the original function is being called and decoded.

4.3.1 Function Annotations to Specify Dynamic Keys

Dynamic key specifiers must be added as single line PHP comments starting with `//` and having a tag `@ioncube.dynamickey`. To illustrate, in the following example we use a value obtained from a URL as the dynamic key.

```
// Using a URL to get the dynamic key.
// @ioncube.dynamickey https://www.example.com/secret?key=4 -> "encoding key
value"

function myfn($a)
{
   [Code body for myfn]
}

myfn($v);  // myfn is decoded only when first called here
```

In the above, the Encoder will use the string value `"encoding key value"` to the right of the arrow as the encoding key for the function `myfn` immediately below it. The encoding key must always be a constant string value. The function called `myfn` will only be decoded by the Loader when it is first called and that decoding will depend upon successfully getting the response from `https://www.example.com/secretkey?key=4`. Decoding will then only succeed if the response value matches that of the encoding key string, `"encoding key value"`. The arrow `->` separates the runtime expression used to obtain the key and the expected value as used at encoding time value. The encoding time value `"encoding key`

value" will be used as the encryption key used to encrypt the byte code of myfn(). An encryption method may also be specified after the key, as illustrated in the next example, however in this case no method was given and a default encryption method would be used.

In the following example the key is obtained from a PHP *variable* at runtime. This also shows that instead of @ioncube.dynamickey the shorter form @ioncube.dk can be used as the tag.

```
// Using the value of a variable as the dynamic key.
// @ioncube.dk $x -> "58" RANDOM

function myfn($a)
{
   [Code body for myfn]
}

$x = 53;
$x += 5;

myfn($v);  // First call to myfn – successful decode depends on value of $x
```

In the example above, the decoding of myfn depends upon the value of the variable $x at the point that myfn is called. The Loader will convert the value of $x to be a string and use that as the decoding value. If $x does not exist or does not have a value equivalent to "58" then a runtime error will occur. In this example we have used RANDOM as the encryption algorithm to secure the byte code of myfn. That means the actual encryption method used will be selected from a number of built-in encryption methods.

The value of function calls can also be used as the expression to be evaluated to produce the decoding key, allowing for more sophisticated approaches as seen in the following example:

```
// Using the result of a function to get the dynamic key.
function concatfn($first,$second)
{
   $phrase = ucfirst($first) . ' ';
   $phrase .= ucfirst($second) . ' again!';
   return $phrase;
}

// @ioncube.dk concatfn("hello","world") -> "Hello World again!"
function myfn($a)
{
   [Code body for myfn]
}

// First call to myfn - decoding depends on evaluating
concatfn("hello","world")
myfn($v);
```

In the above, the Loader will call `concatfn()` with the string arguments `"hello"` and `"world"`. Doing so should evaluate to the string `"Hello World again!"` or a runtime error will occur. Please note that the arguments to the function must be constant strings and it must return a string value or one that can be converted to a string.

4.3.2 Rules for Expressions Used to Get the Decoding Key

There are the following restrictions on the expressions used to obtain the decoding key at runtime:

- The result of the expression must be a string or convertible to a string, so an object, array or resource cannot be the result of such an expression. Note that the Encoder cannot check this itself as it cannot know to what type an expression will evaluate.
- Variable expressions must be scalar variables of the form `$name`. Variable variables (such as `$$x`), array elements (such as `$a["apple"]`) or object expressions cannot be used. Any variable given must be in scope at the point a function being decoded is called.
- File expressions can be a full path on the system prefixed with `file://` or an external URL with the `http`, `https` or `ftp` transport types. e.g.
  `file:///var/www/html/app/favicon.ico`
  `https://www.example.com/get-app-key.php`

  Notice that *three* `/` characters are used at the start of an absolute file path. Be aware also that URLs should only be used if it is known that the systems on which the encoded files will run have the PHP configuration value `allow_url_fopen` set to be `on`.
- For the contents of files and URLs, any starting white space (spaces, newlines, tabs) in the contents will be ignored. So, suppose the response to requesting the URL `https://www.example.com/secretkey` was `"  encoding key value"` then that would be trimmed to `"encoding key value"`.

- For function expressions, the function used must be a user-defined one in the global namespace and must return a string value (or a value that can be converted to a string). Using an internal PHP function or operator will not work, although the Encoder cannot detect such use. So the following would fail when the Loader attempted to decode the function:

```
// Cannot use built-in PHP functions.
// @ioncube.dk strlen("my secret phrase") -> "16"
```

- User-defined functions used to obtain the key can use internal PHP functions and methods, including the functions of the Loader API (see section 6). Other user-defined functions and methods can also be used within the definition. So the following will be fine:

```
function addon($str)
{
   return ' ' . lcfirst($str) . ' ';
}

function use_funcs($a, $b)
{
   // Get the properties in a Pro or Cerberus license.
   // The value of mprop should be "propv" for decoding to
   // work in this example.

   $licprops = ioncube_license_properties();
   $retval = str_repeat($a,2) . addon($b) . $licprops['mprop']['value'];
   return $retval;
}

// @ioncube.dk use_funcs("say ","Again") -> "say say  again propv"
function myfn($arg) // ...
```

- Function expressions cannot be method calls or calls to variables that have function variables. So for example, the following expressions would be disallowed by the Encoder:

```
// Attempt to use a static method will be disallowed.
// @ioncube.dk MyClass::StaticMethod("a") -> "my static res"

// Attempt to use an instance method will be disallowed.
// @ioncube.dk $obj->mymethod("arg1","arg2") -> "obj res"

// Cannot use variables for function calls.
// @ioncube.dk $f("pear") -> "apples and pears"
```

- Any variables passed to a function used to get a decoding key must all be string constants, not containing any variables. So the following would be disallowed by the Encoder:

```
// Attempt to use variables as arguments will be disallowed.
// @ioncube.dk decodefn($v) -> "my string"

// Cannot use function calls in arguments.
// @ioncube.dk myfn(ucfirst("my str")) -> "res"
```

4.3.3 Available Encryption Methods

The following encryption methods may be used when encrypting the compiled byte code of a function. When included at the end of a dynamic key annotation they will be recognised in a case-insensitive way. So "`random`", "`RANDOM`" and "`Random`" will all be recognised as the random method.

- `Random` - A random method is chosen from a set of strong encryption methods when encoding.
- `Basic` - This is a non-cryptographic byte code obfuscation method that may be used if more application speed is required.

If neither `Random` nor `Basic` is stipulated then a default internal encryption algorithm will be used. The default encryption method will not vary between encodings, unlike `Random`. This may give slightly better performance than a random choice.

4.3.4 Dynamic Keys for Whole Scripts

We have seen above how dynamic keys can be used to encrypt the byte code of functions. Dynamic keys can also be used for entire scripts. A dynamic key specifier will apply to the entire script if it occurs before all other key specifiers in the script and, in addition, another dynamic key specifier occurs before the first function. It must occur also in the body of the script and not in any class. Below is an example:

```
<?php

// The dynamic key specifier immediately below will apply
// to the script as there is another specifier before
// the first function.

// @ioncube.dk https://www.example.com/special -> "ghjvbbn444646"

$x = 'original text';
$x .= ' here';

// First function in the script.
// The dynamic key specifier below applies to g.
//@ioncube.dk $x -> "original text here"
function g($a,$b) // ...
```

4.3.5 Dynamic Keys for Primary Scripts

In a dynamic key specifier for a script, any variables or functions in the specification must be defined by the time the script is accessed. This means that unless a prepend script is in place to provide the necessary variables or functions, a primary script (the one first called in a request) can only be protected by a file or URL dynamic key as there cannot be any user-defined variables or functions defined at that point.

4.3.6 Dynamic Keys and Closures

Dynamic keys may be applied to closures, although with some differences. Closures are decoded at the time that the encoded file is read rather than at the point of the first call. This means that any variable or function used in the dynamic key specifier for the closure must exist at the point that the file is read rather than when the closure is first called. Due to this restriction the Encoder will produce a warning if the dynamic key specifier for a closure is a variable.

4.3.7 Chains of Functions Having Dynamic Keys

If a dynamic key specifier depends on the result of a function (`fn`, say) then that function `fn()` may itself have a dynamic key as can be seen in the following example:

```
function g($v, $w)
{
   $x = $v . ' yes';
   $x .= ' ' . str_repeat($w, 2);
   $x .= ' ' . substr($w, 0, -2);
   return $x;
}

// @ioncube.dk g("'say'","again") -> "'say' yes againagain aga"

function fn($p)
{
   return('HEY ' . $p);
}

// Dynamic key specifier has a call to function fn
// which itself has a dynamic key specifier.

//  @ioncube.dynamickey fn("world") -> "HEY world" RANDOM
function first($v)
{ // …
}

// Below the call to first will require fn to be
// decoded before first's key can be obtained.
// Decoding fn will require a call to g.
first($vn);
```

Care should be taken when using such chains. In particular, circular chains of key specifiers must be avoided, e.g. where a function `f()` has a key specifier that requires `g()` to be called, and `g()` has a key specifier that uses `f()`.

4.3.8 Performance Versus Security When Using Dynamic Keys

The significant benefits of dynamic keys to secure functions and methods should be balanced against a possible loss of performance as true encryption/decryption is relatively slow by design. For a function that has been encoded with a dynamic key, the Loader must first either read the contents of a file, access a URL, examine the value of a variable, or use the result of a function call. The Loader will then decrypt the byte code of the function using the key and the appropriate decryption method. Full encryption and decryption are computationally intensive procedures, and all together, these steps inevitably add some overhead to the first call to a function.

We would recommend that the use of dynamic keys with true encryption be kept for certain central functions or methods upon which the application as a whole depends. If those functions cannot be decoded then the rest of the application will not run and the remainder of the application will not be decoded, even if the remainder of the application is not protected by dynamic keys. One possible strategy would be to secure the files of an application with the external keys described in the Section External Keys above and to add dynamic key protection to a few key functions or methods within each file. In addition, the Basic encryption method can be used with dynamic keys to improve performance as it is not a true encryption method and therefore faster.

4.3.9 Using the Reflection API with Dynamic Decoding [`--allow-reflection-all`, `--allow-reflection`]

Certain methods of PHP's reflection API that may act on functions or methods require the byte code of the function or method to exist in order to operate. Thus simply using the methods of the reflection API with a function that had not yet been decoded would not work with dynamic decoding. On the other hand, if the ionCube Loader decoded functions automatically when Reflection API methods were called, this would provide a loophole allowing the decoding of functions to be forced without them being called during normal execution.

To get around this issue, there are two options to the Encoder that allow the reflection API to be used with user-defined functions and methods. These are `--allow-reflection-all` and `--allow-reflection` that allow the following reflection API methods to be applied to encoded functions:

- `getDefaultValue()`
- `getDocComment()`
- `getEndLine()`
- `getFileName()`
- `getStartLine()`
- `getStaticVariables()`
- `isDefaultValueAvailable()`
- `__toString()`

Other reflection API methods are not affected by the `--allow-reflection-all` and `--allow-reflection` options.

The `--allow-reflection-all` option allows the reflection API to be applied to any function or method within encoded files. Thus if this option is used, the Loader will decode any function or method to which the reflection API was being applied. If there is an `--allow-reflection` option then that will override `--allow-reflection-all` as `--allow-reflection` is more specific and therefore safer. The Encoder will produce a warning to the effect that `--allow-reflection` overrides `--allow-reflection-all`.

The `--allow-reflection` option allows the reflection API to be applied to functions or methods matching a specific pattern. There may be multiple `--allow-reflection` options when encoding. The possible specifier patterns that can be used with `--allow-reflection` are as follows:

- `<function name>`
  The name of a specific function (not a method). For example,
  `--allow-reflection myfn`
  will allow reflection to be applied to the `myfn()` function.
- `<class name>::<method name>`
  The name of a specific method in a specific class. For example,
  `--allow-reflection MyClass::foo`
  will allow the reflection API to be applied to the method `foo()` of the class `MyClass`.

- `<class name>::*`
  All methods within a specific class. For example,
  `--allow-reflection MyClass::*`
  will allow the reflection API to be applied to all the methods of the class `MyClass`.

- `<namespace>\*`
  All functions and methods within the given namespace. For example,
  `--allow-reflection Lang\Translate\*`
  will allow every method and function within the `Lang\Translate` namespace to have the reflection API applied to it.

If a function matches a pattern then the Loader will decode that function and call the reflection API method being applied. If a function does not match the pattern then the Loader will not decode the function and a runtime error will occur.

Multiple `--allow-reflection` options can be used. For example, we could have the following when encoding:

```
$ ioncube_encoder -84 test.php -o test_enc.php --allow-reflection
MyClass:* --allow-reflection bar
```

which means that the reflection API can be applied to all methods of `MyClass` and also to the function `bar()`.

Note that no restriction is applied to the application of the reflection API upon a function after it has been first called, as the function body will have already been decoded.

4.3.10 Errors & Warnings Related to Dynamic Keys Produced by the Encoder

The Encoder may produce the following errors and warnings in relation to dynamic keys:

- `Warning: Invalid ioncube declaration at [character number] in [file name] at line [line number]`
  This means that a comment line has started with `@ioncube` but is not followed by `.dynamickey` or `.dk`. In this case the line is ignored.

- `Warning: Variable dynamic decoding key given for closure`
  The variable may not exist at the time the closure is decoded.
  Closures are decoded eagerly when a file is read rather than when they are first called. This means that it may not make sense for a closure to have a dynamic key specifier as the variable may not have been set at that point.

- `Warning: Orphan dynamic key specifier in file [file name]`
  This means that there is a dynamic key specifier but no function occurs after it, thus the key specifier cannot be bound to any function in the file.

- `Error: Invalid dynamic key specifier, [specifier string], in [file name] at line [line number]`
  This means `@ioncube.dynamickey` or `@ioncube.dk` has been seen but the rest of the dynamic key specifier does not make sense to the Encoder. The Encoder will exit if such an error occurs.

- `Warning: Option --allow-reflection-all ignored due to earlier --allow-reflection option`
  and
  `Warning: Option --allow-reflection-all overridden by --allow-reflection option`
  These two warnings indicate that the `--allow-reflection` option takes precedence over the `--allow-reflection-all option`.

- `Warning: --allow-reflection-all option only applies for PHP 5.3+ encoding`
  and
  `Warning: --allow-reflection option only applies for PHP 5.3+ encoding`
  These two warnings indicate that the `--allow-reflection` options cannot be used with `ioncube_encoder` and `ioncube_encoder5`.

4.3.11 Errors Related to Dynamic Keys That May Occur At Runtime

When dynamically decoding functions and methods, the ionCube Loader may produce an `Unable to call protected function` or `Unable to call protected script` error if a fault in the byte code occurs after dynamic decoding. The exact format of the error will depend on whether the `--dynamic-key-errors` option has been used. By default the Loader will give a message similar to standard PHP errors, including the function name, file name and line number.



Note that if a dynamic key is incorrect at runtime then unpredictable behaviour may occur. Whilst the ionCube Loader tries to catch such errors in decoding, in some situations they may not be caught and this may cause PHP to fail.

4.3.12 Controlling How Dynamic Decoding Errors Are Presented [`--dynamic-key-errors`]

The way that dynamic decoding errors are presented can be controlled by encoding with the `--dynamic-key-errors` option, which takes one of the following arguments:

- `simple`
  Error messages will just say `Unable to call protected function myfn` if `myfn` is the function being dynamically decoded. This option will give minimal information to anyone who might be trying to reverse engineer encoded files.

- `normal`
  This is similar to standard PHP error messages and gives the file name and line number, as well as the function name, in the message. This is the default.

- `backtrace`
  This gives a back trace of the execution, including functions executed to obtain the current dynamic key. This should be used when debugging dynamic key problems. This option also controls the format of errors produced for external keys where the external key was found but an incorrect value caused problems in the resulting byte code.

4.3.13 Getting Dynamic Key Information With Verbose [`--verbose`]

The dynamic key information discovered in a source file by the Encoder will be output when the `--verbose` option is used.

5 LICENSE FILE GENERATION (Pro and Cerberus Editions)

5.1 Introduction to License Files

The Pro and Cerberus editions of the ionCube Encoder support various features for license file creation. Using the included license generator program (`make_license`), various types of restrictions can be setup, with these either enforced automatically by the ionCube Loader at runtime, or in your own scripts using functions in the Loader API. With the Windows edition a license generator for Linux is also included, allowing license files to be created from a Linux server; this is located in a folder called `Linux` within the program installation folder.

Encoded files can be restricted to run only in the presence of a license file, properties can be set in the license file that must match properties set in the encoded files (useful to prevent use of an incorrect license file), and license files can be used to restrict encoded files to a particular machine. Licenses can also be set to expire at some point in the future.

Benefits of License Files

Using license files is an alternative to setting restrictions in the encoded files themselves. One benefit of this is that a project will not need to be encoded for each installation requiring different restrictions. Producing software updates to existing customers is also more straightforward as a single encoded update can be provided, whereas a unique one would be required for each user if the files themselves contained the restrictions.

Server Restrictions

License file server restrictions can be supplied to the license generator in one of two ways. If the details of the server to be licensed are known, such as IP address, then these can be used explicitly. Often, however, details are unknown, and the goal is simply to restrict to a server without knowing the actual parameters. To support this, a Loader API function can be used to generate server data containing information about the target server, and after being received via a method such as email or a web form, the data can be passed to the license generator to create a license file for that server. This can be ideal for automating license generation based on information supplied from the installed PHP scripts during a licensing procedure. The server data can also be decoded by the license generator into an easily parsed format, allowing a script to pick what server details are licensed.

Custom Properties

Custom key/value data called *properties* can also be added to license files and read via the Loader API at runtime. This feature might be used to customise product behaviour based on information read from the license file.

License File Contents

A license file consists of a header in plain text followed by an encrypted data block. Any properties or restrictions in the license data can be optionally exposed so that they also appear as plain text, e.g. the expiry time. Additional text can be added to the header if desired. Finally, license files are protected by signatures to prevent removal or changing of the plain text.

Locating License Files

Each encoded file contains the expected name or path to its associated license file. Typically this will be just the filename, e.g. `license.txt`. When accessing a license protected encoded file, the Loader first looks for the license file path relative to the same directory as the encoded script. If not found, parent directories will be searched until the license file is found or until the directory root is reached. This allows for easy installation and management of license files for both shared and dedicated servers.

5.2 Creating License Files

5.2.1 Command Line Usage

The general form for running the command line license generation tool is

```
make_license --passphrase <key> [options] -o <output-path>
```

When encoding files that require a license, it is also necessary to specify a passphrase. The passphrase is part of an encryption key and should agree with the passphrase specified when generating the corresponding license. It is advisable that passphrases are different for each project/product to prevent a license file for one project being used with another.

Please note that a license file is only compatible with encoded files from the related instance of the Encoder software. This ensures that even if another ionCube user generates a license file with the same passphrase as you have used for your files, their license file cannot be used with your encoded files.

The output path is the path to where the new license file will be saved.

Along with the passphrase and output path, various options can be used for setting restrictions and license properties. All available options are described below.

5.2.2 Help [`--help`, `-h`]

Show a summary of program options.

5.2.3 Program version [`--version`, `-v`]

Report the program version.

5.2.4 Output path [`--output`, `-o`]

Specify the output path for the license file.

Example:

```
-o product-license.txt
```

5.2.5 Using Passphrases to Differentiate Products [`--passphrase`]

As mentioned previously, it is recommended that a unique passphrase be used for each product or product variant that is encoded. This ensures that a license for one product will not unlock a second product. As an additional layer of security, a license created with one Encoder installation cannot unlock files encoded with a different Encoder installation, even if the passphrases were the same. The option `--passphrase` should be used to specify the passphrase.

5.2.6 Setting License Restrictions [`--allowed-server`, `--expose-server-restrictions`]

The `--allowed-server` option allows explicit setting of license file restrictions when target server details such as domain name or IP address are already known. The specification syntax is the same as for the similar option of the Encoder. Please see section 3.6.3 above for details of the syntax, examples and noteworthy points. This option may be used more than once if multiple restrictions are required.

Server restrictions are stored in the encoded part of the license file, but can also be exposed in the header block by using the `--expose-server-restrictions`.

Note: There is a limit of 250 server restrictions for a license file.

5.2.7 Exclude checking of interface aliases [`--ignore-interface-aliases`]

Ignore interface aliases when checking IP restrictions, e.g. `eth0:1`

5.2.8 Setting License Restrictions from Server Data [`--use-server-file`]

An alternative to setting license restrictions explicitly is to use server data from the target server that was collected by the application being licensed. Server data is obtained by calling the Loader API function `ioncube_server_data()`, which would then be passed back to the license provider be used for licensing. See section 6.2.3 for more details on this API function.

Once server data has been received, such as in an email or through a web form, and then written to a temporary file, the `--use-server-file` option can be used as follows

```
--use-server-file <path>
```

where *path* refers to the location of the server data file. As the machine may contain multiple network adaptors, it is also necessary to use either the option `--select-server-adapter` or `--select-adapters` to select which details to license to. These options are described in the next section.

5.2.9 Selecting Adapters [`--select-server-adapter`, `--select-adapters`]

Server data generated using the API function `ioncube_server_data()` includes all network interfaces. In addition, if the API function is called from a script via the web server, both the domain name and server IP address for the request will be stored if that information was available. This is the usual case, and it will generally be desirable to license to the information associated with the web request as this would usually be the same when accessing the main parts of the licensed product. In other cases, licensing to one or more adapters explicitly may be preferred. Both of these cases are catered for.

To license to the server information associated with the web request that called the server data API function, use the option

```
--select-server-adapter
```

Provided that the server data was requested from a page under the same domain as the application to be licensed, errors from mistaken domain names or IP addresses should not arise. When using this option, if no IP address or domain name was reported the `make_license` program will exit with code 2 to allow handling of this case.

To license one or more specific adapters, the option

```
--select-adapters <adapter list>
```

can be used to select the adapters. Here *adapter list* is either a comma separated list of numbers that refer to the position of the adapter in the server data file, or the `*` character. The first adapter is identified as `1`, and using `*` selects all adapters to be licensed.

5.2.10 License Expiry [`--expire-in`, `--expire-on`, `--expose-expiry`]

These options support the creation of time expiring licenses. Expiry information is stored in the encoded part of the license file, but can also be exposed in the header block by using the `--expose-expiry` option.

These options take arguments of the same format as the options in sections 3.6.1 and 3.6.2 for setting expiry time on files, and examples are included in those sections.

5.2.11 License Properties [`--property`, `--properties`, `--expose-property`]

Custom key/value pair property data can be stored in a license file in the same way that properties can be stored in encoded files. Use the option syntax

```
--property "<name>[=<value>]"
```

to specify a property. Multiple properties can be specified in this way. See section 3.12.1 for more details on the supported syntax. `--properties` is an alias for `--property`, and both options can be used to specify one or more properties. Properties can also be exposed as plain text in the license header block by using the option

```
--expose-property <name>
```

5.2.12 License Property Checking [`--enforce-property`]

Properties may be included in a license either as a convenient mechanism for securely accessing custom data from an encoded script, or in order to lock a license to an encoded file. If a property is to be used for the latter purpose, the following option should be used

```
--enforce-property <name>
```

By default, encoded files secured by such a license must have a property with a matching key and value. If the property is not found then the Loader will exit before execution of the script begins. See section 3.7.3 for details on how to customise this behaviour.

5.2.13 Customising the Header Block [`--header-line`]

The text that occurs before the encrypted license data is called the *header block*. The header block is protected from tampering, so it is important that this text is not edited after the license has been generated otherwise the license will become corrupted. The header block content is determined by those properties which have been exposed, whether there is an exposed expiry date, and any custom header lines. To add custom lines to the header block, for each line use the command line option

```
--header-line <text>
```

5.2.14 Viewing Server Data Files [`--decode-server-file`]

Once saved to a file, the contents of data generated by `ioncube_server_data()` can be viewed with the `make_license` option

```
--decode-server-file <path>
```

where `path` is the path to a file containing server data. The domain name and server IP address that were reported by the web server for the request calling the API function are output first, followed by the name, IP address, and MAC address of each adapter installed on the server. To be both human readable and easily parsed, each line has a field name and value separated by a `:` character. If there was no domain name or IP address stored for the request, the field value will be the token `none`.

5.2.15 Decoding a license [`--decode-license`, `--decode-format`]

Decodes a license file to human or machine readable form. Available formats are `legacy`, `json`, `json2`, `compact`

The legacy format is a human readable form, while `json` and `json2` are JSON. The two JSON formats may be combined together. JSON output is multiline, but if compact is specified, the JSON will be a single line. The formats produce the same output for properties and domain restrictions, but differ for IP and MAC addresses.

Example:

Given a license file locked to a domain with a single IP address and a range of IP addresses, all at the same MAC address, created as follows:

```
make_license --pass mypass -o lic.txt  \
      --allowed-server domain.com@1.2.3.4,10.11.12.13-
16{00:11:22:33:44:55}
```

The possible decoded JSON outputs are:

`--decode-format json`

```
{
  "ignore_interface_aliases" : false,
  "restrictions" :
  [
    {
      "domains" : [ "domain.com" ],
      "ip_rules" :
      [
        {
           "ip" : 16909060,
           "mask" : 4294967295,
           "numbits" : 32,
           "type" : "ip"
        },
        {
          "from" : 168496141,
          "to" : 168496144,
          "type" : "range"
        }
       ],
       "mac_addrs" :
      [
        [ 0, 17, 34, 51, 68, 85 ]
      ]
    }
  ]
}
```

```
--decode-format json2
```

```
{
  "ignore_interface_aliases" : false,
  "restrictions" :
  [
    {
      "domains" : [ "domain.com" ],
      "ip_rules" : [ "1.2.3.4", "10.11.12.13-16" ],
      "mac_addrs" : [ "00:11:22:33:44:55" ]
    }
  ]
}
```

```
--decode-format json,json2
```

```
{
  "ignore_interface_aliases" : false,
  "restrictions" :
  [
    {
      "domains" : [ "domain.com" ],
      "ip_rules" :
      [
        {
          "raw" :
          {
            "ip" : 16909060,
            "mask" : 4294967295,
            "numbits" : 32,
            "type" : "ip"
          },
          "text" : "1.2.3.4"
        },
        {
          "raw" :
          {
            "from" : 168496141,
            "to" : 168496144,
            "type" : "range"
          },
          "text" : "10.11.12.13-16"
        }
      ],
      "mac_addrs" :
      [
        {
          "raw" : [ 0, 17, 34, 51, 68, 85 ],
          "text" : "00:11:22:33:44:55"
        }
      ]
    }
  ]
}
```

As can be seen, the `json` decode format is oriented to program processing, whereas the `json2` format produces text that is suited to display purposes.

5.2.16 Legacy compatibility [`--compat`]

Prior to version 11, the unquoted property values `true` and `false` were stored and returned by the Loader API function `ioncube_license_properties()` as strings. From version 11, these are stored and returned as booleans. To obtain the previous parsing behaviour when adding properties to a license, add the option `--compat no-bool-props` at the start of the `make_license` command line, before any properties are specified.

5.2.17 Troubleshooting License Problems

If an encoded script that requires a license fails, the particular error message displayed can give a clue as to the cause of the issue. For security reasons the error messages are general rather than specific.

If the license is reported to be *invalid* then a license property set in the license has not been matched in the encoded file. If the license is *corrupt* then either the contents of the license file have been altered, or the passphrase in the encoded file does not match the passphrase used to generate the license file. If the license is *not valid for this server* then a server restriction in the license has not been met.

If it is necessary to determine the contents of a license file the Loader API can be used. Encode a script with the options

```
--with-license license.txt --license-check auto
```

and use the Loader API from the script to output the server restrictions, expiry date, and any properties contained in the license.

6 LOADER API

The ionCube Loader contains an API providing various functions and constants that may be useful in PHP scripts. Most functions return results dependent on whether or not the calling script was encoded.

6.1 File Information and Execution

6.1.1 Checking for an Encoded File [`ioncube_file_is_encoded()`]

This function returns `TRUE` if the file containing the function call is encoded and `FALSE` otherwise.

6.1.2 General Encoded File Information [`ioncube_file_info()`]

This function returns `FALSE` if the file is not encoded. Otherwise it returns an associative array. The contents of the array are as follows:

Key	*Value*
`FILE_EXPIRY`	Either the file expiry time, or the license expiry time if a license file is present. The time is an integer in Unix timestamp format: the number of seconds elapsed since midnight (00:00:00), January 1, 1970.
`ENCODING_TIME`	Unix timestamp representing the time the file was encoded.
`DEMO`	`TRUE` if the file was encoded with an evaluation Encoder, otherwise `FALSE`.

6.1.3 Retrieving Properties Stored in an Encoded File [`ioncube_file_properties()`]

This function returns an associative array consisting of file properties that were added to the encoded file with the Encoder `--property` or `--properties` options. Only properties defined in the calling script are returned.

6.1.4 Retrieving the Loader String Version [`ioncube_loader_version()`]

This function returns the Loader version as a string.

6.1.5 Retrieving the Loader Integer Version [`ioncube_loader_iversion()`]

This function returns the Loader version as an integer format, e.g. `120002` for version 12.0.2.
Note: Prior that prior to PHP 8, the Loader returned this as a string, e.g. `"120002"`.
From PHP 8, an integer type is returned.

6.2 License and Server Information

6.2.1 Retrieving Properties Stored in a License [`ioncube_license_properties()`]

This function returns an associative array consisting of license properties. Properties are added to a license by specifying the `--property` command line option to the `make_license` program. Each value in the associative array retrieved by this API function is itself an array with two values: the license property value itself, and a boolean value to indicate whether the property is enforced.

Recall that an *enforced* property is one that the Loader will attempt to match with an encoded file property if the `--license-check auto` option is passed to the Encoder on the command line.

The return value of this function is `FALSE` if the calling file is not encoded or has no license file.

6.2.2 Retrieving the List of Permissioned Servers [`ioncube_licensed_servers()`]

This function returns an array of server restriction specifications. These are the same strings specified on the command line when the license was created.

6.2.3 Creating a Server Data Block [`ioncube_server_data()`]

When generating a license for an end user it will usually be necessary to retrieve information about the target server. This API function generates a *server data block* containing information about the network adapters installed on the server and the server's domain name. The data block can then be used in conjunction with the `make_license` program to generate a license restricted to the user's domain and server.

This function can be called from either an encoded or non-encoded script.

6.3 License Validation

6.3.1 Validating License Properties [`ioncube_check_license_properties()`]

This API function returns `TRUE` if all enforced license properties are matched in the encoded file. Otherwise an array is returned consisting of all unmatched enforced properties.

6.3.2 Validating Licensed Servers [`ioncube_license_matches_server()`]

This function returns `FALSE` if the calling file is encoded, requires a license, and if the license has a server restriction that is not met by the current server. In all other cases the function returns `TRUE`.

Note that in the case that an encoded script requires a license but no license could be found, the Loader will prevent execution of the script. The case of a missing license therefore cannot occur when calling the `ioncube_license_matches_server()` API function.

6.3.3 Validating License Expiry [`ioncube_license_has_expired()`]

This function returns `TRUE` if the calling file is encoded and has a license with an expiry time that has passed. In all other cases the function returns `FALSE`.

6.4 Encrypted File Support

6.4.1 Reading Encrypted Files [ioncube_read_file()]

```
mixed ioncube_read_file(string path
           [,bool &was_encrypted [,string passphrase] ] ] )
```

This API function can be used to read files encrypted by the Encoder with the `--encrypt` command-line option. If a file is read successfully the contents are returned as a binary-safe string. An integer is returned in the case of an error condition, which can be tested for by calling the PHP function `is_int()`. The error codes are described in section 6.5 below.

Both plain text and encrypted files can be read by this function, allowing the function to be used in cases where it is not known whether a file will be encrypted. For example, a template engine could be designed that would accept both encrypted and non-encrypted template files. If it is necessary to know whether the file read was encrypted, the second optional argument (passed by reference) can be examined.

If an encrypted file has been written with a custom passphrase (i.e. a non-empty passphrase argument was passed to the `ioncube_write_file()` API function), the same passphrase should be specified as the third argument.

Files encrypted by one Encoder can only be read by PHP scripts encoded by the same Encoder, and encrypted files cannot be read by non-encoded scripts.

6.4.2 Writing Encrypted Files [ioncube_write_file()]

```
integer ioncube_write_file(string path, string content
           [,bool encrypt [,string passphrase] ] ] )
```

Encoded PHP scripts can write encrypted files using this API function. Files written in this way can be read with the `ioncube_read_file()` function.

The first argument is the path of the output file.

The second argument is a binary-safe string containing the content to encrypt.

The optional third argument can be set to `FALSE` to write a plain text file.

The optional fourth argument can be used to specify a custom passphrase to replace the default installation-specific passphrase. If a custom passphrase is used then files encrypted with one installation can be read by a different installation's encoded files, if the correct custom passphrase is passed to the `ioncube_read_file()` function.

6.5 Error codes

The following table lists the error codes that can be returned from the API functions detailed in this section.

Code	*Meaning*
`0`	The file was successfully written.
`1`	The file could not be opened.
`2`	The file is corrupt.
`3`	An updated Loader should be installed to read the file.
`4`	An error occurred while reading the source file.
`5`	An error occurred while writing an encrypted file.
`6`	An error occurred while encrypting the file contents.
`7`	An encrypted file cannot be read by an non-encoded PHP script.
`8`	The wrong passphrase was specified, or the wrong Encoder installation was used to encrypt the file.
`9`	`ioncube_write_file()` must be called from an encoded file to produce an encrypted file without a custom passphrase.

7 ERROR REPORTING

The Encoder reports syntax errors in the format of

```
filename:line number:message
```

This offers possible integration with emacs and direct access to the point of error in source files. For example, given a directory of PHP files called `myproject`, running the emacs `compile` command and specifying the compiler as

```
ioncube_encoder -S myproject
```

would syntax check all PHP files and report errors in a buffer. With the default emacs key bindings, simply hitting `ctrl-X` would visit each file reported as containing an error and place the cursor at the line containing the error.

8 TROUBLESHOOTING

8.1 Unable to Start the Encoder

8.1.1 On Unix (Linux, FreeBSD, MacOS)

Error: Command not found

On Unix, an error similar to

```
$ ioncube_encoder

bash: ioncube_encoder: command not found
```

would most often be because the directory where the Encoder is installed is not listed in the shell `PATH` variable. It is recommended to use either a relative or absolute path to the Encoder rather than adding the Encoder directory to the `PATH` environment variable as this may create product license related problems on some systems.

Examples:

Run from the current directory

```
./ioncube_encoder
```

Run the Encoder installed in `/usr/local/ioncube`

```
/usr/local/ioncube/ioncube_encoder
```

Error: No such file or directory

An error of "No such file or directory", even when the Encoder is launched with a correct absolute or relative path, suggests that the target platform may be 64 bit and that the operating system cannot execute the program. The Encoder will run without any problem on 64 bit systems, but as it is a 32 bit compiled program, 32 bit system libraries must be installed on the machine.

8.1.2 Using a 64 bit system

The ionCube Encoder software is a 32 bit compiled program, and can be used on 64 bit systems provided that 32 bit support is available in the operating system. Most often it is, however if there is a problem launching the software, simply installing the system 32 bit libraries should resolve. For popular Linux distributions, only a single package need be installed.

8.1.3 On Windows

On Windows, you can edit the `PATH` environment variable for the logged in user by going to the control panel, selecting the System item and clicking on the Environment tab.

9 LOADER INSTALLATION

The ionCube Loader is an engine extension to PHP that performs processing and execution of encoded files, as well as other important operations such as license validation. If a Loader is not pre-installed on a target system, a *Loader Wizard* PHP script is available to indicate which Loader is required, where to download it from our website, and also what installation options are available. The Loader Wizard is available at https://www.ioncube.com/lw/

There are two possible installation methods for the Loader, and the following sections give some specific details about this.

9.1 Loader Naming

Loaders are named with the following conventions, dependent on the operating system and whether or not the PHP build has thread safety enabled.

Operating System	*Thread Safety Enabled*	*Loader Name*
Unix (Linux, FreeBSD etc.)	No	`ioncube_loader_<os>_<PHP version>.so` e.g. `ioncube_loader_lin_8.4.so`
Unix (Linux, FreeBSD etc.)	Yes	`ioncube_loader_<os>_<PHP version>_ts.so` e.g. `ioncube_loader_fre_8.4_ts.so`
Windows	Yes / No	`ioncube_loader_win_<PHP version>.dll` e.g. `ioncube_loader_win_8.4.dll`

In the table above, `<os>` is the first 3 letters of the operating system, e.g. `lin` for Linux, `fre` for FreeBSD etc., and `<PHP version>` are the first two numbers from the PHP version. Different Loader packages are available for various architectures, i.e. dependent on whether a platform is 32 or 64 bit and the processor type, but note that Loader naming is independent of architecture.

9.2 Installation in a php.ini File

The recommended method for installation is via a `php.ini` file, requiring the addition of one line to reference the location of the Loader. Example lines to add are shown below, where we will assume that the Loader is located in `/usr/local/ioncube` on Unix and `C:/PHP/ext` on Windows, but this need not be the case. Our example also assumes PHP 8.4 except where noted.

On Linux with thread safety disabled (PHP 8.4)

```
zend_extension = /usr/local/ioncube/ioncube_loader_lin_8.4.so
```

On Linux with thread safety enabled (PHP 8.4)

```
zend_extension = /usr/local/ioncube/ioncube_loader_lin_8.4_ts.so
```

On Linux with thread safety enabled (PHP 5.2)

```
zend_extension_ts = /usr/local/ioncube/ioncube_loader_lin_5.2.so
```

Note: Prior to PHP 5.3, `zend_extension_ts` should be used if PHP has thread safety enabled.

On Windows with thread safety disabled or enabled (PHP 8.4)

```
zend_extension = "c:/PHP/ext/ioncube_loader_win_8.4.dll"
```

Following any change to the php.ini file, the web server software should be restarted unless PHP is being used as CGI, where a fresh PHP process is launched to handle each new request.