# ionCube Performance System 2.0

## User Guide



ionCube Ltd. http://www.ioncube.com

# CONTENTS

# 1 INTRODUCTION

## 1.1 Introduction to PHP bytecode caches

PHP consists of a compiler component and a 'virtual machine' component. The compiler converts PHP source code into a binary format similar to the machine code run on a microprocessor. This PHP bytecode is then run on the PHP virtual machine to produce the desired output from the script.

If PHP is installed with no bytecode cache then both compilation and execution of the bytecodes occur every time a script is run by PHP. The proportion of time taken by these steps varies, but often the compilation of the script takes longer than to execute the bytecodes.

One approach to dramatically reducing the total time taken by a PHP request is to cache the intermediate PHP bytecodes. On the first request the bytecodes are stored after the script has been compiled, then on subsequent requests PHP can omit the compilation step and instead use the cached version.

## 1.2 Types of bytecode cache

The bytecodes can be stored on disk in a binary file, in a similar way to an .exe file on Windows, or they can be stored in *shared memory*. We shall refer to the former approach as a *file cache*, and the latter as a shared memory cache. IPS can use either, or both, of these methods to cache PHP bytecodes.

Usually the memory used by a process on Windows or UNIX is isolated from the memory used by other processes. This is to prevent a failure in one program from causing a different program to crash, and also increases security. Shared memory, however, is a region of memory that a process can create and specify that it should be available to other processes, thus allowing communication between processes. A bytecode cache can use this shared memory to store the compiled bytecodes.

Both file caches and shared memory caches remove the need for PHP to compile scripts on every request, so they both provide a large performance improvement over native PHP. On the other hand, when using a file cache the bytecodes need to be read from the file every request, which takes longer than when restoring from the shared memory. For this reason a shared memory cache has substantially better performance than a file cache.

# 2 GETTING STARTED

## 2.1 Installation

### 2.1.1 Unpacking the archive

Use a tool such as GNU tar to unpack the download into a temporary directory, then move the newly created directory to its permanent location, for example

```
mv /tmp/ips_2.0 /home/ips
```

If you are upgrading from a previous version of IPS you should make sure you stop your web server software before copying over the top of your old IPS installation. Modifying the IPS shared library while the web server software is running can lead to undefined behaviour.

### 2.1.2 Editing the php.ini file

Locate the php.ini file set to be used by PHP. Open the file in a text editor. If the ionCube Loader has already been installed into the php.ini file then the IPS entries should follow the ionCube Loader entries. If the ionCube Loader is not installed in the php.ini then the following IPS lines can be entered at the top of the php.ini file.

A line must be added to tell PHP the location of the IPS shared library. In the case where the web server is not threaded (for example on most UNIX servers), the line takes the form:

```
zend_extension=/home/ips/ips.so
```

where in this example we have taken the IPS install location in the previous example. If the web server is threaded (this is usually the case on Windows), the line must appear as follows:

```
zend_extension_ts=/home/ips/ips.so
```

A second line must be added specifying the folder containing the IPS files, as follows:

```
ips.home=/home/ips/
```

Lastly, on UNIX servers, a pair of entries must be added to specify the user and group that the web server runs as. For example:

```
ips.server_uid=nobody

ips.server_gid=nogroup
```

### 2.1.3 Installing the IPS control panel

IPS has a powerful web-based control panel written in PHP. This application is located in the `control_panel` subdirectory of the IPS home directory. To install this application, either create a link to the htdocs subdirectory in a location accessible from the web, or modify your web server's configuration file to associate a URL with the htdocs subdirectory.

## 2.2 Per-domain settings

IPS can keep track of which domains access which scripts. It can use this information to display shared memory usage for each domain and apply limits to the available shared memory on a per-domain basis.

### 2.2.1 Enabling/disabling per-domain mode

Per-domain settings are enabled by default. To disable this mode open the *Global settings* page in the IPS control panel application. Uncheck the *Enable per-domain settings* box and click *Update*. A restart of the web server software will be necessary before the change takes effect.

To enable per-domain settings choose the Settings page, check the *Enable per-domain settings* box and click *Update*.

### 2.2.2 Default domain settings and global settings

Once per-domain settings are enabled it is possible to modify many settings individually for a particular domain. It is also possible to modify the default value that a setting will have for a particular domain if that setting has not been overridden. To modify these default settings click the *default* item on the first row of the table on the *Domain settings* page.

To illustrate the difference between the *global settings* and the *default domain settings*, consider the example of the shared memory limit. The global shared memory limit determines how much shared memory IPS can use for any purpose. IPS will request shared memory of that size from the operating system. The default domain shared memory, however, determines how much shared memory a single domain can use unless otherwise specified.

### 2.2.3 UseCanonicalNames: Configuring Apache for per-domain mode

IPS deduces the active domain for a request from the `$_SERVER['SERVER_NAME']` PHP variable. The value of this variable depends on the `UseCanonicalNames` Apache configuration file directive.

If this directive is on then domains which are specified with the `ServerAlias` directive will use their associated `ServerName`, whereas if `UseCanonicalNames` is off they will use the alias itself.

For example, if `UseCanonicalNames` is off then IPS may treat mydomain.com and www.mydomain.com as different domains, so it is usually desirable to switch this directive on.

## 2.3 CGI mode and the persistent cache

### 2.3.1 Rationale

When a web server is running PHP in CGI mode it is possible that at any given time there will be no PHP process running on the web server. Since all PHP processes will have ended, any shared memory used by IPS will have been destroyed and the next PHP process which starts will then have to recreate the cache. The result could be performance worse than if no cache was installed.

### 2.3.2 Enabling the persistent cache

To specify that IPS should not delete the shared memory when a PHP process exits, it is necessary to enable the IPS *persistent cache* mode. To do this, select the *Global settings* page on the control panel, check the *Enable persistent cache* checkbox, and click *Update*.

## 2.4 Encoded files – IPS and the ionCube Loader

### 2.4.1 Installing an appropriate Loader version

When using IPS it is important to use the most up to date Loader available from the ionCube web site. If an old Loader is installed then IPS will still function correctly, but ionCube encoded files will receive no performance boost.

### 2.4.2 Caching encoded files

Using encoded files without an accelerator is similar to using a file cache; encoded files consist of encrypted PHP bytecodes. For this reason, and also to provide extra security, encoded files will not be stored in the IPS file cache. Encoded files can be cached in shared memory, however, providing a good performance improvement.

# 3 PERFORMANCE TUNING

IPS has features designed to automatically achieve the best performance with minimal user interaction. In some cases it may be necessary to override the default behaviour. In this chapter we explain how to use the IPS control panel to change the default settings.

## 3.1 Ignoring scripts

It may be necessary to make IPS ignore certain PHP scripts. An ignored script will not be added to the cache and will be compiled and executed by PHP as if IPS was not installed.

For example, if a script is modified several times every second then the total time taken to cache the file may exceed the time saved by the file being in the cache. It would then make sense to make IPS ignore this script.

### 3.1.1 Global scripts

To make IPS ignore a script choose the scripts page on the control panel, select *All domains* from the combo box, and select the script you want to ignore, and click the *Ignore* button. The full path to the script will be added to the list on the Filter tab of the *Global settings* page.

The path to a script may be added manually to the *Global settings* page, or a pattern with optional wildcards. The pattern may be an absolute or relative path. For example, adding

```
mydir/myfile.*
```

will make IPS ignore files of all extensions in directories called `mydir`.

### 3.1.2 Scripts for a particular domain

IPS can be configured to ignore certain scripts when accessed via a particular domain. The *per-domain settings* option must be enabled for this to work.

To make a domain ignore a script, select the scripts page on the control panel, select the domain you wish to consider, select the script to be ignored, and click *Ignore*. The script will be added to the list on the domain's settings page.

Patterns may be added manually as in the case of global scripts.

## 3.2 Shared memory size

If shared memory is enabled then the default amount of memory reserved for use by IPS is 32MB. Depending on the resources available on your server, and the total size of all scripts that should be cached, it may be desirable to increase or decrease the shared memory allocation.

### 3.2.1 Decreasing the shared memory

For example, if the summary page of the control panel records that only a small fraction of the shared memory has been used, then the total shared memory may be safely reduces. It is important to wait until all the most commonly used scripts have been accessed over the web before making this decision.

The shared memory should also be decreased if the limit has been set so high that other applications do not have sufficient memory to function efficiently.

#### 3.2.2 Increasing the shared memory

On the other hand, if your server has plenty of memory that is not being used by PHP or other applications, and IPS has used most of its allocated shared memory, then it may be a good idea to increase the IPS shared memory limit.

#### 3.2.3 Applying the changes

To modify the amount of shared memory available to IPS, select the *Global settings* page on the control panel, select the *SHM cache* tab, and enter the desired shared memory size.

After clicking *Update* it is necessary to restart the web server before the settings will take effect.

### 3.3 Shared memory restart interval

IPS will periodically clean and restart the shared memory cache. This is an effective way to cut down the memory fragmentation which can occur if scripts are modified. During the period when the shared memory is being restarted the shared memory cache cannot be used, but the file cache will be used instead providing a good level of acceleration.

#### 3.3.1 Choosing an appropriate interval

By default IPS will restart the shared memory cache every 6 hours. If many scripts on your site are modified frequently it may be a good idea to reduce this interval.

The aim of restarting the cache is to reduce memory fragmentation thus allowing more scripts to be cached in the shared memory. If the number of scripts in shared memory seems to decrease over time then decreasing the restart interval may be beneficial.

On the other hand, if scripts are rarely modified or removed from your site then a longer restart interval may be appropriate. Restarting the shared memory impacts the performance of the system, so it is not a good idea to restart the cache unless necessary.

#### 3.3.2 Applying the changes

To modify the shared memory restart interval select the *SHM cache* tab on the *Global settings* page, then edit the *Shared memory restart interval* field and click update. The change will take effect immediately without requiring a restart of the web server software.

# 4 Put / Get Persistence API

## 4.1 Overview

IPS allows the persistence of arbitrary PHP data as well as PHP scripts. This data can be stored in shared memory or in files and can be used to cache data or content used by scripts. The data is shared between all requests. In this chapter we shall document the PHP functions used to access and store this data.

### 4.1.1 Enabling the put/get API

The put/get feature of IPS is enabled by default. If per-domain settings are off then this means the put/get API will work. However, if per-domain settings are on then the put/get feature needs to be enable for each required domain. This is to avoid domains on shared servers from using more shared memory than an administrator wants.

To enable the put/get feature for a particular domain, choose the *Domain settings* page, click the link for the appropriate domain, choose *On* from the *Enable put-get API* combo box, then click *Update*.

### 4.1.2 API function return codes

Many of the put/get API functions described in this chapter return a code indicating either success or a particular error condition. The codes and their meanings are given in the following table.

| *Code* | *Meaning* |
|---|---|
| 0 | Success. |
| 1 | Invalid key. |
| 2 | The shared memory store is full. |
| 3 | The file store is full. |
| 4 | The item has expired. |
| 5 | The item could not be found. |
| 6 | The item was found in the file store but the file is corrupt. |
| 7 | The stored item is corrupt. |
| 8 | The wrong number of arguments was supplied. |
| 9 | The location parameter is invalid. |
| 10 | The directory could not be created. |
| 11 | Access denied. |
| 12 | An error occurred while writing the item. |
| 13 | Not enough shared memory for the put/get manager to function. |
| 14 | The path was invalid. |
| 15 | The put/get API is disabled globally or for this domain. |

#### 4.1.3 Usage limits

Shared memory used by the put/get store is not reserved ahead of time, but is used as needed. For this reason it is not usually necessary to provide limits on the shared memory available to the put/get feature.

On the other hand, if the put/get store is being used to cache a large amount of data, for example to cache the HTML output by a script, then this data could significantly reduce the shared memory available to cache PHP script bytecodes. In this situation it may be advisable to set a limit on the amount of shared memory that can by used by the put/get feature. To do this, select the *Put/get API* tab on the *Global settings page*, enter the desired limit in the *Shared memory limit for put/get API* field, and click *Update*. A value of 0 indicates that there is no limit.

If per-domain settings are enabled it is possible to set limits for each domain individually by modifying the appropriate field on the domain's settings page.

## 4.2 Data storage

#### 4.2.1 Storing a single value [`ips_put_value`]

```
integer ips_put_value(string key
            [,integer location [,integer expiry_period] ] ] )
```

This function is used to store or update a single value associated with a given key. On Windows servers the key must only contain characters which may form a valid file name. In particular the characters \/:*?\"<>| are illegal. On UNIX servers the characters \/\0 are not allowed.

The optional location argument may be any of the following PHP constants:

| *Constant* | *Meaning* |
|---|---|
| IPS_SHM_OR_FILE | The item will be stored in shared memory if possible, otherwise it will be stored in the file cache. |
| IPS_SHM_ONLY | The item will be stored in shared memory if possible, otherwise the function will fail. |
| IPS_FILE_ONLY | The item will be stored in a file if possible, otherwise the function will fail. |
| IPS_FILE_AND_SHM | The item will be stored both in shared memory and the file store. If either is not possible then the function will fail. |

The default value for this parameter is the constant IPS_SHM_OR_FILE.

The optional expiry period argument determines the period in seconds after which an item will effectively be removed from the store. If an item which has expired is retrieved from the store the retrieval will fail with return code 4. The expiry period is measured in seconds, and the default value of -1, or IPS_NO_EXPIRY, means that the item will never expire.

## 4.3 Data retrieval

### 4.3.1 Retrieving a single value [ips_get_value]

```
mixed ips_get_value(string key
          [,integer &result_code] )
```

This function returns a value stored using the ips_put_value API function. If no value is found with the specified key or an error occurs then the return value will be null and the optional second argument can be used to retrieve the error code.

### 4.3.2 Retrieving all keys [ips_get_all_keys]

This function returns an array consisting of all keys used to store data with ips_put_value. In the case where per-domain settings are enabled only the keys stored by the currently active domain will be returned, whereas if per-domain settings are disabled all keys will be returned.

If an error occurs null is returned and the second argument can be used to discover the cause of the error.

### 4.3.3 Retrieving all data [ips_get_all_values]

```
array ips_get_value( [integer &result_code] )
```

This function will return an array consisting of all key/value pairs stored with ips_put_value. In the case where per-domain settings are enabled only the data stored by the currently active domain will be returned, whereas if per-domain settings are disabled all data will be returned.

If an error occurs null is returned and the second argument can be used to discover the cause of the error.

## 4.4 Data removal

### 4.4.1 Removing a single key [ips_remove_key]

```
integer ips_remove_key(string key)
```

This function will remove the data associated with the given key. The function returns a code as listed in the table at the beginning of this chapter.

### 4.4.2 Removing all keys [ips_remove_all_keys]

```
integer ips_remove_all_keys()
```

This function will remove all keys stored with the ips_put_value function. If per-domain settings are enabled then only those keys stored by the active domain will be deleted, otherwise all keys will be removed.

### 4.4.3 Removing expired keys [ips_remove_expired_keys]

```
integer ips_remove_expired_keys()
```

When an expiry time is set on a key the associated data will be removed next time the key is accessed. In some situations it may be desirable to remove all expired data even if the assoicated keys are not known. This API function will remove all expired keys from both the shared memory and file store.

## 4.5 Metadata retrieval

The results from the functions in this section must be used with care as 'race conditions' may occur. For example, if ips_key_exists returns false, it is possible a different request may add the key during the time between ips_key_exists being called and its result being used. The functions ips_lock and ips_unlock can be used to ensure this behaviour does not occur.

### 4.5.1 Retrieving the last update time [ips_get_last_update_time]

```
integer ips_get_last_update_time(string key
          [,integer &result_code] )
```

This function returns the time at which the item was last updated. The time is represented as the number of seconds since 1st January 1970.

### 4.5.2 Retrieving the expiry time [ips_get_last_update_time]

```
integer ips_get_expiry_time(string key
          [,integer &result_code] )
```

This function returns the time at which the item will expire. The time is represented as the number of seconds since 1st January 1970.

### 4.5.3 Testing for existence of a key [ips_key_exists]

```
boolean ips_key_exists(string key)
```

This function returns true or false according to whether there is data stored with the specified key.

## 4.6 Locking

### 4.6.1 Obtaining a lock [ips_lock]

This function will block until the lock is available and then will proceed. The lock will be released either when `ips_unlock` is called or the request ends.

The lock is reference counted, so if `ips_lock` is called twice by the same request without `ips_unlock` then the second call will never block but `ips_unlock` will need to be called twice before the lock is released.

If per-domain settings are enabled then each domain has its own lock, otherwise there is a single lock.

### 4.6.2 Releasing a lock [ips_unlock]

This function will release a lock obtained with the `ips_lock` function.

# 5 TROUBLESHOOTING

## 5.1 IPS is unable to start

### 5.1.1 Check the IPS log file

If IPS cannot start up there will usually be an entry in the IPS log explaining why it could not start. By default the log file is located at `/tmp/ips_log.txt`.

### 5.1.2 Error reopening the shared memory

If the shared memory has not be deleted after the web server software has been shut down and IPS is unable to reopen the shared memory then it may be necessary to manually delete the IPS shared memory.

To locate the shared memory open a shell window and type `ipcs`. In the *Shared Memory Segments* section there will be an entry with key `0x18453141`. Locate the `shmid` for this entry. As root type

```
ipcrm shm <shmid>,
```

substituting the correct value for the shared memory ID.

## 5.2 Restarting the shared memory cache

### 5.2.1 How long will a restart take?

When the shared memory cache is scheduled for a restart IPS must wait until all active requests finish using the shared memory. During this time new requests will use the file cache rather than shared memory.

Once all requests have finished with the shared memory the cache will restart. Usually this will happen a few seconds after the restart is scheduled but on some Apache installations it may take up to a few minutes.

### 5.2.2 Can I force a restart?

If IPS is waiting for all processes to finish with the shared memory it is possible to force IPS to restart by clicking the *Force Restart* button at the top of the *Summary* page. It is usually a good idea to wait until all requests have finished using the shared memory and the cache restarts automatically as manually forcing a restart may make some processes crash.

## 5.3 IPS control panel problems

### 5.3.1 Forgotten admin password

The admin password can be reset to empty by opening the IPS global configuration file and removing the line starting `admin_password_hash`.