Alternative PHP Cache

Introduction

The Alternative PHP Cache (APC) is a free and open opcode cache for PHP. It was conceived of to provide a free, open, and robust framework for caching and optimizing PHP intermediate code.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

This » PECL extension is not bundled with PHP.

Information for installing this PECL extension may be found in the manual chapter titled Installation of PECL extensions. Additional information such as new releases, downloads, source files, maintainer information, and a CHANGELOG, can be located here: » http://pecl.php.net/package/apc.

The DLL for this PECL extension may be downloaded from either the » PHP Downloads page or from » http://pecl4win.php.net/

Note
On Windows, APC needs a temp path to exist, and be writable by the web server. It checks TMP, TEMP, USERPROFILE environment variables in that order and finally tries the WINDOWS directory if none of those are set.

Note
For more in-depth, highly technical implementation details, see the » developer-supplied TECHNOTES file.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Although the default APC settings are fine for many installations, serious users should consider tuning the following parameters.

There are two main decisions you have to make. First, how much shared memory do you want to set aside for APC, and second, whether you want APC to check if a file has been modified on every request. The two ini directives involved here are *apc.shm_size* and *apc.stat*. Read the sections on these two directives carefully below.

Once you have a running server, you should copy the *apc.php* script that comes with the

extension to somewhere in your docroot and load it up in your browser. It provides you with a detailed look at what is happening in your cache. If you have GD enabled in PHP, it will even have pretty graphs. First thing to check is of course that it is actually caching files. Assuming it is working you should then pay close attention to the *Cache full count* number on the left. That tells you the number of times the cache has filled up and has had to forcefully clean up any entries not accessed within the last *apc.ttl* seconds. You should configure your cache to minimize this number. If you are constantly filling your cache, the resulting cache churn is going to hurt performance. You should either set more memory aside for APC, or use *apc.filters* to cache fewer scripts.

APC configuration options

Name	Default	Changeable	Changelog
apc.enabled	"1"	PHP_INI_SYSTEM	PHP_INI_SYSTEM in APC 2. PHP_INI_ALL in APC <= 3.0.12.
apc.shm_segments	"1"	PHP_INI_SYSTEM	
apc.shm_size	"30"	PHP_INI_SYSTEM	
apc.optimization	"0"	PHP_INI_ALL	PHP_INI_SYSTEM in APC 2. Removed in APC 3.0.13.
apc.num_files_hint	"1000"	PHP_INI_SYSTEM	
apc.user_entries_hint	"4096"	PHP_INI_SYSTEM	Available since APC 3.0.0.
apc.ttl	"0"	PHP_INI_SYSTEM	Available since APC 3.0.0.
apc.user_ttl	"0"	PHP_INI_SYSTEM	Available since APC 3.0.0.
apc.gc_ttl	"3600"	PHP_INI_SYSTEM	
apc.cache_by_default	"1"	PHP_INI_ALL	PHP_INI_SYSTEM in APC <= 3.0.12. Available since APC 3.0.0.
apc.filters	NULL	PHP_INI_SYSTEM	
apc.mmap_file_mask	NULL	PHP_INI_SYSTEM	
apc.slam_defense	"0"	PHP_INI_SYSTEM	Available since APC 3.0.0.

apc.file_update_protection	"2"	PHP_INI_SYSTEM	Available since APC 3.0.6.
apc.enable_cli	"0"	PHP_INI_SYSTEM	Available since APC 3.0.7.
apc.max_file_size	"1M"	PHP_INI_SYSTEM	Available since APC 3.0.7.
apc.stat	"1"	PHP_INI_SYSTEM	Available since APC 3.0.10.
apc.write_lock	"1"	PHP_INI_SYSTEM	Available since APC 3.0.11.
apc.report_autofilter	"0"	PHP_INI_SYSTEM	Available since APC 3.0.11.
apc.include_once_override	"0"	PHP_INI_SYSTEM	Available since APC 3.0.12.
apc.rfc1867	"0"	PHP_INI_SYSTEM	Available since APC 3.0.13.
apc.rfc1867_prefix	"upload_"	PHP_INI_SYSTEM	
apc.rfc1867_name	"APC_UPLOAD_PROGRESS"	PHP_INI_SYSTEM	
apc.rfc1867_freq	"0"	PHP_INI_SYSTEM	
apc.localcache	"0"	PHP_INI_SYSTEM	Available since APC 3.0.14.
apc.localcache.size	"512"	PHP_INI_SYSTEM	Available since APC 3.0.14.
apc.coredump_unmap	"0"	PHP_INI_SYSTEM	Available since APC 3.0.16.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

`apc.enabled` boolean
apc.enabled can be set to 0 to disable APC. This is primarily useful when APC is statically compiled into PHP, since there is no other way to disable it (when compiled as a DSO, the *extension* line in *php.ini* can just be commented-out).

`apc.shm_segments` integer
The number of shared memory segments to allocate for the compiler cache. If APC is

running out of shared memory but you have already set *apc.shm_size* as high as your system allows, you can try raising this value.

apc.shm_size integer
The size of each shared memory segment in MB. By default, some systems (including most BSD variants) have very low limits on the size of a shared memory segment.

apc.optimization integer
The optimization level. Zero disables the optimizer, and higher values use more aggressive optimizations. Expect very modest speed improvements. This is experimental.

apc.num_files_hint integer
A "hint" about the number of distinct source files that will be included or requested on your web server. Set to zero or omit if you're not sure; this setting is mainly useful for sites that have many thousands of source files.

apc.user_entries_hint integer
Just like apc.num_files_hint, a "hint" about the number of distinct user cache variables to store. Set to zero or omit if not sure.

apc.ttl integer
The number of seconds a cache entry is allowed to idle in a slot in case this cache entry slot is needed by another entry. Leaving this at zero means that your cache could potentially fill up with stale entries while newer entries won't be cached.

apc.user_ttl integer
The number of seconds a user cache entry is allowed to idle in a slot in case this cache entry slot is needed by another entry. Leaving this at zero means that your cache could potentially fill up with stale entries while newer entries won't be cached.

apc.gc_ttl integer
The number of seconds that a cache entry may remain on the garbage-collection list. This value provides a fail-safe in the event that a server process dies while executing a cached source file; if that source file is modified, the memory allocated for the old version will not be reclaimed until this TTL reached. Set to zero to disable this feature.

apc.cache_by_default boolean
On by default, but can be set to off and used in conjunction with positive *apc.filters* so that files are only cached if matched by a positive filter.

apc.filters string
A comma-separated list of POSIX extended regular expressions. If any pattern matches the source filename, the file will not be cached. Note that the filename used for matching is the one passed to include/require, not the absolute path. If the first character of the expression is a + then the expression will be additive in the sense that any files matched by the expression will be cached, and if the first character is a - then anything matched will not be cached. The - case is the default, so it can be left off.

apc.mmap_file_mask string
If compiled with MMAP support by using *--enable-mmap* this is the mktemp-style file_mask to pass to the mmap module for determing whether your mmap'ed memory

region is going to be file-backed or shared memory backed. For straight file-backed mmap, set it to something like */tmp/apc.XXXXXX* (exactly 6 *X* s). To use POSIX-style shm_open/mmap put a*.shm* somewhere in your mask. e.g. */apc.shm.XXXXXX* You can also set it to */dev/zero* to use your kernel's */dev/zero* interface to anonymous mmap'ed memory. Leaving it undefined will force an anonymous mmap.

`apc.slam_defense` integer
: On very busy servers whenever you start the server or modify files you can create a race of many processes all trying to cache the same file at the same time. This option sets the percentage of processes that will skip trying to cache an uncached file. Or think of it as the probability of a single process to skip caching. For example, setting *apc.slam_defense* to *75* would mean that there is a 75% chance that the process will not cache an uncached file. So, the higher the setting the greater the defense against cache slams. Setting this to *0* disables this feature. Deprecated by apc.write_lock.

`apc.file_update_protection` integer
: When you modify a file on a live web server you really should do so in an atomic manner. That is, write to a temporary file and rename (*mv*) the file into its permanent position when it is ready. Many text editors, cp, tar and other such programs don't do this. This means that there is a chance that a file is accessed (and cached) while it is still being written to. This *apc.file_update_protection* setting puts a delay on caching brand new files. The default is 2 seconds which means that if the modification timestamp (*mtime*) on a file shows that it is less than 2 seconds old when it is accessed, it will not be cached. The unfortunate person who accessed this half-written file will still see weirdness, but at least it won't persist. If you are certain you always atomically update your files by using something like rsync which does this correctly, you can turn this protection off by setting it to 0. If you have a system that is flooded with io causing some update procedure to take longer than 2 seconds, you may want to increase this a bit.

`apc.enable_cli` integer
: Mostly for testing and debugging. Setting this enables APC for the CLI version of PHP. Normally you wouldn't want to create, populate and tear down the APC cache on every CLI request, but for various test scenarios it is handy to be able to enable APC for the CLI version of APC easily.

`apc.max_file_size` integer
: Prevent files larger than this value from getting cached. Defaults to 1M.

`apc.stat` integer
: Be careful if you change this setting. The default is for this to be On which means that APC will stat (check) the script on each request to see if it has been modified. If it has been modified it will recompile and cache the new version. If you turn this setting off, it will not check. That means that in order to have changes become active you need to restart your web server. On a production server where you rarely change the code, turning stats off can produce a significant performance boost. For included/required files this option applies as well, but note that if you are using relative path includes (any path that doesn't start with / on Unix) APC has to check in order to uniquely identify the file. If you use absolute path includes APC can skip the stat and use that absolute path as the unique identifier for the file.

`apc.write_lock` boolean

On busy servers when you first start up the server, or when many files are modified, you can end up with all your processes trying to compile and cache the same files. With write_lock enabled, only one process at a time will try to compile an uncached script while the other processes will run uncached instead of sitting around waiting on a lock.

apc.report_autofilter boolean
Logs any scripts that were automatically excluded from being cached due to early/late binding issues.

apc.include_once_override boolean
Optimize **include_once()** and **require_once()** calls and avoid the expensive system calls used.

apc.rfc1867 boolean
RFC1867 File Upload Progress hook handler is only available if you compiled APC against PHP 5.2.0 or later. When enabled, any file uploads which includes a field called *APC_UPLOAD_PROGRESS* before the file field in an upload form will cause APC to automatically create an upload_ *key* user cache entry where *key* is the value of the *APC_UPLOAD_PROGRESS* form entry. Note that the file upload tracking is not threadsafe at this point, so new uploads that happen while a previous one is still going will disable the tracking for the previous.

Example #1 - An apc.rfc1867 example

```
<?php
print_r(apc_fetch("upload_$_POST[APC_UPLOAD_PROGRESS]"));
?>
```

The above example will output something similar to:

```
Array
(
   [total] => 1142543
   [current] => 1142543
   [rate] => 1828068.8
   [filename] => test
   [name] => file
   [temp_filename] => /tmp/php8F
   [cancel_upload] => 0
   [done] => 1
)
```

apc.rfc1867_prefix string
Key prefix to use for the user cache entry generated by rfc1867 upload progress functionality.

apc.rfc1867_name string
Specify the hidden form entry name that activates APC upload progress and specifies the user cache key suffix.

apc.rfc1867_freq string

The frequency that updates should be made to the user cache entry for upload progress. This can take the form of a percentage of the total file size or a size in bytes optionally suffixed with 'k', 'm', or 'g' for kilobytes, megabytes, or gigabytes respectively (case insensitive). A setting of 0 updates as often as possible, which may cause slower uploads.

apc.localcache boolean
: This enables a lock-free local process shadow-cache which reduces lock contention when the cache is being written to.

apc.localcache.size integer
: The size of the local process shadow-cache, should be set to a sufficently large value, approximately half of apc.num_files_hint.

apc.coredump_unmap boolean
: Enables APC handling of signals, such as SIGSEGV, that write core files when signaled. When these signals are received, APC will attempt to unmap the shared memory segment in order to exclude it from the core file. This setting may improve system stability when fatal signals are received and a large APC shared memory segment is configured.

Warning
This feature is potentially dangerous. Unmapping the shared memory segment in a fatal signal handler may cause undefined behaviour if a fatal error occurs.

Note
Although some kernels may provide a facility to ignore various types of shared memory when generating a core dump file, these implementations may also ignore important shared memory segments such as the Apache scoreboard.

Resource Types

This extension has no resource types defined.

Predefined Constants

This extension has no constants defined.

APC Functions

apc_add

apc_add -- Cache a variable in the data store

Description

bool **apc_add** (string `$key`, mixed `$var` [, int `$ttl`])

Caches a variable in the data store, only if it's not already stored.

Note
Unlike many other mechanisms in PHP, variables stored using apc_add() will persist between requests (until the value is removed from the cache).

Parameters

key
: Store the variable using this name. *key* s are cache-unique, so attempting to use apc_add() to store data with a key that already exists will not overwrite the existing data, and will instead return **FALSE**. (This is the only difference between apc_add() and apc_store().)

var
: The variable to store

ttl
: Time To Live; store *var* in the cache for *ttl* seconds. After the *ttl* has passed, the stored variable will be expunged from the cache (on the next request). If no *ttl* is supplied (or if the *ttl* is *0*), the value will persist until it is removed from the cache manually, or otherwise fails to exist in the cache (clear, restart, etc.).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #2 - A apc_add() example

```
<?php
$bar = 'BAR';
apc_add('foo', $bar);
```

```
var_dump(apc_fetch('foo'));
echo "\n";
$bar = 'NEVER GETS SET';
apc_add('foo', $bar);
var_dump(apc_fetch('foo'));
echo "\n";
?>
```

The above example will output:

```
string(3) "BAR"
string(3) "BAR"
```

See Also

- apc_store()
- apc_fetch()
- apc_delete()

apc_cache_info

apc_cache_info -- Retrieves cached information from APC's data store

Description

array **apc_cache_info** ([string `$cache_type` [, bool `$limited`]])

Retrieves cached information and meta-data from APC's data store.

Return Values

Array of cached data (and meta-data), or **FALSE** on failure

Note
apc_cache_info() will raise a warning if it is unable to retrieve APC cache data. This typically occurs when APC is not enabled.

Parameters

`cache_type`
: If *`cache_type`* is " *user* ", information about the user cache will be returned. If *`cache_type`* is " *filehits* ", information about which files have been served from the bytecode cache for the current request will be returned. This feature must be enabled at compile time using *--enable-filehits*. If an invalid or no *`cache_type`* is specified, information about the system cache (cached files) will be returned.

`limited`
: If *`limited`* is **TRUE**, the return value will exclude the individual list of cache entries. This is usefull when trying to optimize calls for statistics gathering.

ChangeLog

Version	Description
3.0.11	The *`limited`* parameter was introduced.
3.0.16	The " *filehits* " option for the *`cache_type`* parameter was introduced.

Examples

Example #3 - A apc_cache_info() example

```
<?php
print_r(apc_cache_info());
?>
```

The above example will output something similar to:

```
Array
(
   [num_slots] => 2000
   [ttl] => 0
   [num_hits] => 9
   [num_misses] => 3
   [start_time] => 1123958803
   [cache_list] => Array
       (
           [0] => Array
               (
                   [filename] => /path/to/apc_test.php
                   [device] => 29954
                   [inode] => 1130511
                   [type] => file
                   [num_hits] => 1
                   [mtime] => 1123960686
                   [creation_time] => 1123960696
                   [deletion_time] => 0
                   [access_time] => 1123962864
                   [ref_count] => 1
                   [mem_size] => 677
               )
           [1] => Array (...iterates for each cached file)
)
```

See Also

- APC configuration directives

apc_clear_cache

apc_clear_cache -- Clears the APC cache

Description

bool **apc_clear_cache** ([string `$cache_type`])

Clears the user/system cache.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Parameters

`cache_type`
: If *`cache_type`* is " *user* ", the user cache will be cleared; otherwise, the system cache (cached files) will be cleared.

See Also

- apc_cache_info()

apc_compile_file

apc_compile_file -- Stores a file in the bytecode cache, bypassing all filters.

Description

bool **apc_compile_file** (string `$filename`)

Stores a file in the bytecode cache, bypassing all filters.

Parameters

`filename`
: Full or relative path to a PHP file that will be compiled and stored in the bytecode cache.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

apc_define_constants

apc_define_constants -- Defines a set of constants for retrieval and mass-definition

Description

bool **apc_define_constants** (string `$key`, array `$constants` [, bool `$case_sensitive`])

define() is notoriously slow. Since the main benefit of APC is to increase the performance of scripts/applications, this mechanism is provided to streamline the process of mass constant definition. However, this function does not perform as well as anticipated.

For a better-performing solution, try the » hidef extension from PECL.

Note
To remove a set of stored constants (without clearing the entire cache), an empty array may be passed as the *`constants`* parameter, effectively clearing the stored value(s).

Parameters

`key`
: The *`key`* serves as the name of the constant set being stored. This *`key`* is used to retrieve the stored constants in apc_load_constants().

`constants`
: An associative array of *constant_name => value* pairs. The *constant_name* must follow the normal constant naming rules. *value* must evaluate to a scalar value.

`case_sensitive`
: The default behaviour for constants is to be declared case-sensitive; i.e. *CONSTANT* and *Constant* represent different values. If this parameter evaluates to **FALSE** the constants will be declared as case-insensitive symbols.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #4 - apc_define_constants() example

```
<?php
$constants = array(
   'ONE'   => 1,
   'TWO'   => 2,
   'THREE' => 3,
);
apc_define_constants('numbers', $constants);
echo ONE, TWO, THREE;
?>
```

The above example will output:

```
123
```

See Also

- apc_load_constants()
- define()
- constant()
- Or the PHP constants reference

apc_delete

apc_delete -- Removes a stored variable from the cache

Description

bool **apc_delete** (string `$key`)

Removes a stored variable from the cache.

Parameters

key
The *key* used to store the value (with apc_store()).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #5 - A apc_delete() example

```
<?php
$bar = 'BAR';
apc_store('foo', $bar);
apc_delete('foo');
// this is obviously useless in this form
?>
```

See Also

- apc_store()
- apc_fetch()

apc_fetch

apc_fetch -- Fetch a stored variable from the cache

Description

mixed **apc_fetch** (string `$key`)

Fetchs a stored variable from the cache.

Parameters

key
The *key* used to store the value (with apc_store()).

Return Values

The stored variable on success; **FALSE** on failure

Examples

Example #6 - A apc_fetch() example

```
<?php
$bar = 'BAR';
apc_store('foo', $bar);
var_dump(apc_fetch('foo'));
?>
```

The above example will output:

```
string(3) "BAR"
```

See Also

- apc_store()
- apc_delete()

apc_load_constants

apc_load_constants -- Loads a set of constants from the cache

Description

bool **apc_load_constants** (string `$key` [, bool `$case_sensitive`])

Loads a set of constants from the cache.

Parameters

`key`
The name of the constant set (that was stored with apc_define_constants()) to be retrieved.

`case_sensitive`
The default behaviour for constants is to be declared case-sensitive; i.e. *CONSTANT* and *Constant* represent different values. If this parameter evaluates to **FALSE** the constants will be declared as case-insensitive symbols.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #7 - apc_load_constants() example

```
<?php
$constants = array(
   'ONE'   => 1,
   'TWO'   => 2,
   'THREE' => 3,
);
apc_define_constants('numbers', $constants);
apc_load_constants('numbers');
echo ONE, TWO, THREE;
?>
```

The above example will output:

```
123
```

See Also

- apc_define_constants()
- define()
- constant()
- Or the PHP constants reference

apc_sma_info

apc_sma_info -- Retrieves APC's Shared Memory Allocation information

Description

array **apc_sma_info** ([bool `$limited`])

Retrieves APC's Shared Memory Allocation information.

Parameters

`limited`
When set to **FALSE** (default) apc_sma_info() will return a detailed information about each segment.

Return Values

Array of Shared Memory Allocation data; **FALSE** on failure.

Examples

Example #8 - A apc_sma_info() example

```
<?php
print_r(apc_sma_info());
?>
```

The above example will output something similar to:

```
Array
(
   [num_seg] => 1
   [seg_size] => 31457280
   [avail_mem] => 31448408
   [block_lists] => Array
       (
           [0] => Array
               (
                   [0] => Array
                       (
                           [size] => 31448408
                           [offset] => 8864
                       )

               )

       )
```

```
)
```

See Also

- APC configuration directives

apc_store

apc_store -- Cache a variable in the data store

Description

bool **apc_store** (string `$key`, mixed `$var` [, int `$ttl`])

Cache a variable in the data store.

Note
Unlike many other mechanisms in PHP, variables stored using apc_store() will persist between requests (until the value is removed from the cache).

Parameters

key
: Store the variable using this name. *key* s are cache-unique, so storing a second value with the same *key* will overwrite the original value.

var
: The variable to store

ttl
: Time To Live; store *var* in the cache for *ttl* seconds. After the *ttl* has passed, the stored variable will be expunged from the cache (on the next request). If no *ttl* is supplied (or if the *ttl* is *0*), the value will persist until it is removed from the cache manually, or otherwise fails to exist in the cache (clear, restart, etc.).

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #9 - A apc_store() example

```
<?php
$bar = 'BAR';
apc_store('foo', $bar);
var_dump(apc_fetch('foo'));
?>
```

The above example will output:

```
string(3) "BAR"
```

See Also

- apc_add()
- apc_fetch()
- apc_delete()