

# Logger Configuration

## Log Writer Configuration

Most of the logger configuration can be done by configuring the log writers.

The configuration of log writers can be either done in the application.ini / installation.ini or hardcoded in PHP.

System-Admins should add their custom writer configuration to the installation.ini.

Please see [ErrorHandling and Application Logger](#) for information about the log levels.

The configuration in application.ini is reserved to translate5 developers - since this file is overwritten on each update.

### Current writer configuration in application.ini

```
resources.ZfExtended_Resource_Logger.writer.default.type = 'ErrorLog'
resources.ZfExtended_Resource_Logger.writer.default.filter[] = "level <=
warn"                                     ; warn logs from fatal to warn into the error log

; Test config:
resources.ZfExtended_Resource_Logger.writer.mail.type = 'DirectMail'
resources.ZfExtended_Resource_Logger.writer.mail.filter[] = "level <=
error"                                   ; warn logs from fatal till error via E-Mail
resources.ZfExtended_Resource_Logger.writer.mail.sender = 'sysadmin@example.
com'                                     ; should be set with useful values in installation.ini,

; defaults to legacy value from resources.mail.defaultFrom.email
resources.ZfExtended_Resource_Logger.writer.mail.receiver[] = 'sysadmin@example.
com'                                     ; should be set with useful values in installation.ini,

; if not set defaults to legacy resources.mail.defaultFrom.email
resources.ZfExtended_Resource_Logger.writer.mail.receiver[] = 'other-important-person@example.com' ;
should be set with useful values in installation.ini

resources.ZfExtended_Resource_Logger.writer.database.type = 'Database'
; via the formatter define what the user gets: full debug content, or only notice and so on.
resources.ZfExtended_Resource_Logger.writer.database.filter[] = "level <=
debug"                                   ; logs from fatal to debug into the database
```



When overwriting resources.ZfExtended\_Resource\_Logger.writer.mail.receiver in installation.ini the default value from application.ini is overwritten too. That means the default receiver should be added to installation.ini too:

```
resources.ZfExtended_Resource_Logger.writer.mail.receiver[] = 'your receiver'
resources.ZfExtended_Resource_Logger.writer.mail.receiver[] = 'our default receiver from application.ini'
```

## DirectMail different users for different errors

With the filters it is possible to setup multiple DirectMail Loggers with different receivers for differently filtered events:

### Example for different mail writers

```
resources.ZfExtended_Resource_Logger.writer.mail1.type = 'DirectMail'
resources.ZfExtended_Resource_Logger.writer.mail1.filter[] = "level <= error; domain != foo.
bar"                                     ; exclude events from technical domain foo.bar
resources.ZfExtended_Resource_Logger.writer.mail1.receiver[] = 'sysadmin@example.
com'                                     ; sysadmin gets all error events except from domain foo.bar

resources.ZfExtended_Resource_Logger.writer.mail2.type = 'DirectMail'
resources.ZfExtended_Resource_Logger.writer.mail2.filter[] = "level <= error; domain == foo.bar"
resources.ZfExtended_Resource_Logger.writer.mail2.receiver[] = 'otheruser@example.
com'                                     ; otheruser gets only error events from domain foo.bar
```

# Task Writer Configuration

The TaskWriter is a editor specific writer, it logs only events containing information about a task.

The events are logged in a dedicated table, accessible via the frontend.

Only warnings or events more severe as warnings are written by default to the task log. So no notices about tasks are additionally logged.

task log writer config in application/modules/editor/configs/module.ini

```
resources.ZfExtended_Resource_Logger.writer.tasklog.type = 'editor_Logger_TaskWriter'
resources.ZfExtended_Resource_Logger.writer.tasklog.filter[] = "level <= warn" ; logs
only from fatal to warning

; example for an additional filter to log from fatal to debug for events with an domain starting with "core.foo.
bar"
resources.ZfExtended_Resource_Logger.writer.tasklog.filter[] = "level <= debug; domain ^= core.foo.bar"
```

Also on code level new writers can be added on the fly to a logger instance.

In PHP just add new writers to the wanted logger instance

```
// use the default logger instance:
$logger = Zend_Registry::get('logger');
$logger->addWriter('name', $writerInstance);
```

## Filtering

Each writer can have an list of filter rules. See the above module.ini example. The filter rules are connected via "or", that means one of the filter rules must be valid in order that an event is logged to that writer.


task log writer config in application/modules/editor/configs/module.ini

```
resources.ZfExtended_Resource_Logger.writer.tasklog.filter[] = "level <=
warn" ; first filter rule
resources.ZfExtended_Resource_Logger.writer.tasklog.filter[] = "level <= debug; domain ^= core.foo.bar"
; second filter rule

;ecodes can be filtered too (they are added internally in the filter to the domain):
resources.ZfExtended_Resource_Logger.writer.tasklog.filter[] = "level <= warn; domain $= E1234" ;
filtering a specific ecode
```

Each filter rule is it self, a semicolon separated list of expressions. All expressions of one rule are connected via "and", so all expressions must be evaluated to true in order that the rule is valid.

An expression consists of a keyword, an operator and a value, for example: "level <= warn". Each keyword can be used multiple times, so "level >= warn; level <= trace" would also be possible.

 They keyword must be always the first operand! So "warn <= level" would give an error since it can not be parsed.

In the following table a list of valid keywords and operands is listed:

Keyword	Valid Operands	Description
level	=	<p>Is used to compare with the log levels as defined at the top of this page.</p> <p>The second operand must be a valid log level: "fatal", "error", "warn" etc. Since internally the levels are integers, only numeric operators are usable here.</p> <p>Example: given FATAL(1), configured "level &lt;= WARN(4)" the event is filtered, since level 1 (fatal) is lesser then level 4 (warn)</p>

	<=	Is true if the current level is equal or has a higher severity as the configured one.
	>=	Is true if the current level is equal or has a lower severity as the configured one.
domain	=	The domain (origin) of the event must be matched completely.
	!=	The domain (origin) must not be equal to the configured one.
	^=	The domain (origin) must start with the configured string.
	\$=	The domain (origin) must end with the configured string.
	*=	The domain (origin) must contain the configured string.
exception	=	The exception equals the given exception class name
	!=	The exception is different to the given exception class name
	*=	The current exception is a subclass of the given exception name