title: Logagent filter functions description: Logagent features modular logging architecture framework where each input or output module is implemented as a plugin, and loaded on demand as declared in the configuration file. Input and output filters are available, and they drop, transform or aggregate log events and hook into the processing chain.

Filters

Filters can drop, transform or aggregate log events and hook into the processing chain.

There are two types of filters:

1. Input filters - process raw input from input plugins before log events get parsed
2. Output filters - process parsed log events before they are passed to output plugins

   Input Plugins -> Input Filters -> Parser -> Output Filter -> Output Plugins

Example:

```
1. Input: Tail Web Server Log -g '/var/log/httpd/access.log'
2. Input Filter: Grep URLs of interest 'login|register|upgrade'
3. Parser: Parse Log and generate fields like URL, status code, size, referrer, country etc.
4. Output Filter: Drop irelevant log events like redirects (status=302)
5. Output Plugin: Store filtered log-events in Elasticsearch
```

Filters can be declared inline as JavaScript in function or as a reference to npm modules in a Logagent config file.

Input filter

Function parameters for input filters:

- sourceName - the name of the log source e.g. '/var/log/httpd/access.log'
- config - the configuration options from the config file
- data - the raw (input filter) or parsed data (output filter)
- callback - MUST be called.
  - callback() without parameters drops the event.
  - callback (null, data) will pass the log event to the next filter or output plugin.
  - callback(error) will report an error and drops the event

Node.js modules can be loaded as filter function with the `module` keyword. A module can be declared inline as a JavaScript function using `!!js/function >>` in the module property. Properties in the config section are passed to the filter function as "config" object.

Example, using npm modules:

```
inputFilter:
  - module: logagent-filter-input-grep
    config:
      matchSource: !!js/regexp /myapp.log/
      include: !!js/regexp /info|error/i
      exclude: !!js/regexp /test/i
```

Example, inline JavaScript function:

```
inputFilter:
  - module: logagent-filter-input-grep
    config:
      matchSource: !!js/regexp /myapp.log/
      include: !!js/regexp /info|error/i
      exclude: !!js/regexp /test/i
    module: !!js/function >>
        function (sourceName, config, data, callback) {
              try {
                var drop = false
                if (config.matchSource) {
                  if (!config.matchSource.test(sourceName)) {
                    // pass data for unmatched source names
                    return callback(null, data)
                  }
                }
                // filter data for matched source names
                if (config.include) {
                  drop = !config.include.test(data)
                }
                if (config.exclude) {
                  drop = config.exclude.test(data) || drop
                }
                drop ? callback() : callback(null, data)
              } catch (err) {
                return callback(null, data)
              }
            }
```

Output filter

Function parameters for output filters:

- context - an object providing information about the log source, e.g. context.source
- config - the configuration options from the config file

- eventEmitter - the eventEmitter sends new events to Logagent plugins emit('data.parsed', context, data). Required for aggregation plugins, which typically drop all events and generate new events with aggregated stats.
- data - the raw (input filter) or parsed data (output filter)
- callback - MUST be called.
  - callback() without parameters drops the event.
  - callback (null, data) will pass the log event to the next filter or output plugin.
  - callback(error) will report an error and drops the event.

Node.js modules can be loaded as filter function with the `module` keyword. A module can be declared inline as a JavaScript function using `!!js/function >>` in the module property. Properties in the config section are passed to the filter function as "config" object.

Example, an inline declaration to implement the grep filter from above applied to data.message field.

```
outputFilter:
  - config:
      matchSource: !!js/regexp /myapp.log/
      include: !!js/regexp /info|error/i
      exclude: !!js/regexp /test/i
    module: !!js/function >>
        function (context, config, eventEmitter, data, callback)  {
              try {
                var sourceName = context.source
                var drop = false
                if (config.matchSource) {
                  if (!config.matchSource.test(sourceName)) {
                    // pass data for unmatched source names
                    return callback(null, data)
                  }
                }
                // filter data for matched source names
                if (config.include) {
                  drop = !config.include.test(data.message)
                }
                if (config.exclude) {
                  drop = config.exclude.test(data) || drop
                }
                drop ? callback() : callback(null, data)
              } catch (err) {
                // pass all events to next filter
                return callback(null, data)
              }
```

```
}
```