**What is launchd?**

[**Wikipedia**](http://en.wikipedia.org/wiki/Launchd) defines launchd as "a unified, open-source service management framework for starting, stopping and managing daemons, applications, processes, and scripts. Written and designed by Dave Zarzycki at Apple, it was introduced with Mac OS X Tiger and is licensed under the Apache License."

**What is a Daemon**

A daemon is a program running in the background without requiring user input. A typical daemon might for instance perform daily maintenance tasks or scan a device for malware when it is connected.

**Daemons and Agents**

launchd differentiates between agents and daemons. The main difference is that an agent is run on behalf of the logged in user while a daemon runs on behalf of the root user or any user you specify with the UserName key.

**Job Definitions**

The behavior of a daemon/agent is specified in a special XML file called a property list. Depending on where it is stored it will be treated as a daemon or an agent.

Job definitions crucial for the operation of the operating system are stored below /System/Library. You should never need to create a daemon or agent in these directories. Third-Party definitions which are relevant for every user are stored below /Library. Job definitions for a specific user are stored below the respective user's Library directory.

|  |  |  |
| --- | --- | --- |
| **Type** | **Location** | **Run on behalf of** |
| User Agents | ~/Library/LaunchAgents | Currently logged in user |
| Global Agents | /Library/LaunchAgents | Currently logged in user |
| Global Daemons | /Library/LaunchDaemons | root or the user specified with the key UserName |
| System Agents | /System/Library/LaunchAgents | Currently logged in user |
| System Daemons | /System/Library/LaunchDaemons | root or the user specified with the key UserName |

The ~ character stands for the users home directory.

The following example shows a complete job definition with only three keys:

* Label This key is required for every job definition. It identifies the job and has to be unique for the launchd instance. Theoretically it is possible for an agent to have the same label as a daemon, as daemons are loaded by the root launchd whereas agents are loaded by a user launchd, but it is not recommended.
* Program This key defines *what* to start, in this case a shell script /Users/Me/Scripts/cleanup.sh.
* RunAtLoad This is one of several optional keys specifying *when* the job should be run, in this case right after it has been loaded.

**Plain Text**

LaunchControl

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>Label</key> <string>com.example.app</string> <key>Program</key> <string>/Users/Me/Scripts/cleanup.sh</string> <key>RunAtLoad</key> <true/> </dict> </plist>

launchd supports more than 36 different configuration keys. Most of them are explained in the Configuration-Section of this primer.

**Automatic Loading of Job Definitions**

Upon system start the root launchd process will scan the daemon directories /System/Library/LaunchDaemons and /Library/LaunchDaemons for job definitions and load them depending on the existence/value of the Disabled key and the contents of the override database.

When a user logs in a new launchd process will be started for this user. This launchd process will scan the agent directories /System/Library/LaunchAgents, /Library/LaunchAgents and ~/Library/LaunchAgents for job definitions and load them depending on the existence/value of the Disabled key and the contents of the override database.

**Loading vs Starting**

Loading a job definition does not necessarily mean to start the job. When a job is started is determined by the job definition. In fact, only when RunAtLoad or KeepAlive have been specified,launchd will start the job unconditionally when it has been loaded.

Configuring is done using a text editor of your choice or using a 3rd-Party tool like **[LaunchControl](https://www.soma-zone.com/LaunchControl)**. While it is not difficult to create or edit job definitions using a text editor this method is prone to errors and finding a problem in your configuration can be a daunting task. **[LaunchControl](https://www.soma-zone.com/LaunchControl)** will make sure you always create valid job definitions and highlight errors even before you load the job.

All examples provided in this tutorial are shown as plain text along with the corresponding configuration section in **[LaunchControl](https://www.soma-zone.com/LaunchControl)**.

When using a text editor, make sure to save your configuration as plain text.

**Label: Naming your job**

This key is required for every job definition. It identifies the job and has to be unique for the launchd instance. Theoretically it is possible for an agent to have the same label as a daemon, as daemons are loaded by the root launchd whereas agents are loaded by a user launchd, but it is not recommended.

By convention job names are written in [**reverse domain notation**](http://en.wikipedia.org/wiki/Reverse_domain_name_notation). The helper process for our program LaunchControl, for instance, is named com.soma-zone.LaunchControl.Helper. For private agents the domain local is a good choice: local.cleanup

**What to Start**

A valid job definition requires at least one of these keys: Program and ProgramArguments. The first one takes a single string as its value, the latter one requires a list of strings.

**Program**

This is as simple as it gets. Just specify the complete path to your executable:

**Plain Text**

LaunchControl

<key>Program</key> <string>/path/to/program</string>

An OSX application is not an executable, just a directory. Use this recipe to start an application.

**ProgramArguments**

Use this one if your executable requires command line options:

**Plain Text**

LaunchControl

<key>ProgramArguments</key> <array> <string>/usr/bin/rsync</string> <string>--archive</string> <string>--compress-level=9</string> <string>/Volumes/Macintosh HD</string> <string>/Volumes/Backup</string> </array>

This is equivalent to the shell command

/usr/bin/rsync --archive --compress-level=9 "/Volumes/Macintosh HD" "/Volumes/Backup"

The first string in the list is *not* the first argument but the path to the executable you want to run.

**Program and ProgramArguments**

When you provide both keys, the value of Program is the executable to be started. The first string in ProgramArguments will show up as argv[0] in the executable. If this makes no sense to you, just think that launchd will ignore it.

**Program environment**

Use the keys in this section to customize the environment the program runs in. Set environment variables, the working directory, standard in- and output, enable core file creation or limit the CPU time the executable gets.

**Setting environment variables: EnvironmentVariables**

**Plain Text**

LaunchControl

<key>EnvironmentVariables</key> <dict> <key>PATH</key> <string>/bin:/usr/bin:/usr/local/bin</string> </dict>

Shell globbing and variable expansion do not work for this key.

You can set environment variables per launchd instance by issuing the launchctl command *setenv*. Logging out (in case of a user launchd) or rebooting (in case of the root launchd) will revert those changes. To make these changes persistent you'll have to placing this command in /etc/launchd-user.conf (in case of a user launchd) or /etc/launchd.conf (for all launchd processes).

**Redirecting input and output: StandardInPath StandardOutPath StandardErrorPath**

All these keys take a string as argument. Provide absolute paths when possible. Relative paths are interpreted relative to RootDirectory or /.

**Plain Text**

LaunchControl

<key>StandardInPath</key> <string>/tmp/test.stdin</string> <key>StandardOutPath</key> <string>/tmp/test.stdout</string> <key>StandardErrorPath</key> <string>/tmp/test.stderr</string>

The keys StandardOutPath and StandardErrorPath are very important when it comes to debugging a job. Always configure these keys should a job not behave as expected. Check the resulting files for errors.

**Setting the working directory: WorkingDirectory**

Use this key to set your programs working directory. Every relative path the executable accesses will be relative to its working directory.

**Plain Text**

LaunchControl

<key>WorkingDirectory</key> <string>/tmp</string>

**Constraining your job: SoftResourceLimit HardResourceLimit**

Sometimes it is desirable to constrain a program to prevent resource starvation or enhance system performance.

A resource limit is specified as a soft limit and a hard limit. When a soft limit is exceeded a process may receive a signal (for example, if the cpu time or file size is exceeded), but it will be allowed to continue execution until it reaches the hard limit (or modifies its resource limit).

Only the super-user may raise the hard limits. Other users may only alter soft limits within the range from 0 to the corresponding hard limit or (irreversibly) lower the hard limit.

The biggest value you can specify is the maximum value of a 64-bit signed integer: 9223372036854775807

***CPU***

The maximum amount of CPU time in seconds the process can use. If it runs for longer than this, it receives signal SIGXCPU.

***FileSize***

The program won't be able to create files larger than this. Trying to create a larger file will result in signal SIGXFSZ being sent.

The following example sets the soft limit to 512kiB, the hard limit to 1MiB.

**Plain Text**

LaunchControl

<key>HardResourceLimits</key> <dict> <key>FileSize</key> <integer>1048576</integer> </dict> <key>SoftResourceLimits</key> <dict> <key>FileSize</key> <integer>524288</integer> </dict>

***NumberOfFiles***

The maximum number of files the program can have opened at the same time. Trying to open more files will result in error EMFILE.

The maximum value for this subkey is OPEN\_MAX as defined in sys/syslimits.h: 10240

***Core***

The maximum size core file that this process can create. If the process terminates and would dump a core file larger than this, then no core file is created, so setting this limit to zero prevents core files from ever being created.

Core files will be written to the directory /cores.

***Data***

The maximum size of data memory in bytes for the process. If the process tries to allocate data memory beyond this amount, the allocation function fails.

***MemoryLock***

The maximum amount of memory in bytes that can be locked into physical memory. Locked memory will never be paged out.

***NumberOfProcesses***

The maximum number of processes that can be created with the same user ID. If you have reached the limit for your user ID, fork will fail with error EAGAIN.

***ResidentSetSize***

The maximum amount of physical memory in bytes that this process should get. This parameter is a guide for the system's scheduler and memory allocator; the system may give the process more memory when there is a surplus.

***Stack***

The maximum stack size for the process. If the process tries to extend its stack past this size, it gets a SIGSEGV signal.

**When to Start**

A job definition is perfectly valid just with a Label and the Program or ProgramArguments key, but it won't do anything. You need to tell launchd *when* to start it.

**RunAtLoad: Running the job when it is loaded**

This key is used to start the job as soon as it has been loaded. For daemons this means execution at boot time, for agents execution at login.

**Plain Text**

LaunchControl

<key>RunAtLoad</key> <true/>

**Starting a job every *n* seconds: StartInterval**

If you have a job that you want to execute every *n* seconds, this is for you. To run your job once every hour specify:

**Plain Text**

LaunchControl

<key>StartInterval</key> <integer>3600</integer>

If the system is asleep, the job will be started the next time the computer wakes up. If multiple intervals transpire before the computer is woken, those events will be coalesced into one event upon wake from sleep.

**Starting a job at a specific time/date: StartCalendarInterval**

Use this key to run your job at a specific time, say every day at 3:00 AM:

**Plain Text**

LaunchControl

<key>StartCalendarInterval</key> <dict> <key>Hour</key> <integer>3</integer> <key>Minute</key> <integer>0</integer> </dict>

Available keys are

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Values** |
| *Month* | Integer | Month of year (1..12, 1 being January) |
| *Day* | Integer | Day of month (1..31) |
| *Weekday* | Integer | Day of week (0..7, 0 and 7 being Sunday) |
| *Hour* | Integer | Hour of day (0..23) |
| *Minute* | Integer | Minute of hour (0..59) |

Omitted keys are interpreted as a wildcard. If we omitted the *Minute* key in our previous example the job would run every minute of the third hour of every day.

You may specify several calendar intervals by providing a list of dictionaries. The following job is run every day at 3:00 AM and every full hour on Sundays.

**Plain Text**

LaunchControl

<key>StartCalendarInterval</key> <array> <dict> <key>Hour</key> <integer>3</integer> <key>Minute</key> <integer>0</integer> </dict> <dict> <key>Minute</key> <integer>0</integer> <key>Weekday</key> <integer>0</integer> </dict> </array>

**Cron-style time specifications**

Some intervals are very tedious to specify in launchd. Users familiar with cron know that there is a more elegant way to specify such intervals. A quick example: You want to run a program every five minutes between 20:00 and 23:00. In launchd you have to list all 36 matching timestamps like this:

<key>StartCalendarInterval</key> <array> <dict> <key>Hour</key> <integer>20</integer> <key>Minute</key> <integer>0</integer> </dict> <dict> <key>Hour</key> <integer>20</integer> <key>Minute</key> <integer>5</integer> </dict> . . . <dict> <key>Hour</key> <integer>22</integer> <key>Minute</key> <integer>55</integer> </dict> </array>

LaunchControl allows you to generate this list from the equivalent cron-style specification: \*/5 20-22 \* \* \*

A screenshot of a cell phone

Description automatically generated

The following description of the cron time specification was taken from the cron manual page:

|  |  |
| --- | --- |
| **Field** | **Supported values** |
| Minutes | 0-59 |
| Hours | 0-23 |
| Day of month | 1-31 |
| Month | 1-12 |
| Day of week | 0-7 (0 and 7 is Sunday) |

A field may be an asterisk (\*), which always stands for "first-last".

Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an "hours" entry specifies execution at hours 8, 9, 10 and 11.

Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: "1,2,5,9", "0-4,8-12".

Step values can be used in conjunction with ranges. Following a range with "/<number>" specifies skips of the number's value through the range. For example, "0-23/2" can be used in the hours field to specify command execution every other hour. Steps are also permitted after an asterisk, so if you want to say "every two hours", just use "\*/2".

Unlike cron which skips job invocations when the computer is asleep, launchd will start the job the next time the computer wakes up. If multiple intervals transpire before the computer is woken, those events will be coalesced into one event upon wake from sleep.

**Run when a device has been mounted: StartOnMount**

A job with this key set to true will be started whenever a device is mounted, in other words, when you insert a CD/DVD, connect an external hard disk or mount a virtual filesystem.

**Plain Text**

LaunchControl

<key>StartOnMount</key> <true/>

launchd does not report *which* device has been mounted.

**Run when a path has been modified: WatchPaths**

This key takes a list of strings, each string representing a file or directory.

**Plain Text**

LaunchControl

<key>WatchPaths</key> <array> <string>/path/to/directory\_or\_file</string> </array>

If the path points to a file, creating, removing and writing to this file will be start the job.

If the path points to a directory, creating and removing this directory, as well as creating, removing and writing files in this directory will start the job. Actions performed in subdirectories of this directory will not be detected.

launchd does not report *which* path has been modified.

**Run when files are available for processing: QueueDirectories**

This key takes a list of strings, each string representing a directory.

**Plain Text**

LaunchControl

<key>QueueDirectories</key> <array> <string>/path/to/directory</string> </array>

Whenever one of the directories specified is *not* empty, the job will be started. It is the responsibility of the job to remove each processed file, otherwise the job will be restarted after ThrottleInterval seconds.

launchd does not report in *which* directory new files have been found nor what their names are.

**KeepAlive: Keeping a Job Alive...**

launchd may be used to make sure that a job is running depending on certain conditions.

**...no Matter What**

This is the most basic setting. A value of true will run the job as soon as the job definition is loaded and restart it should it ever go down. launchd will wait ThrottleInterval seconds between restarts.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <true/>

If you configure your job to be kept alive 'no matter what', launchd will run the job immediately after loading it.

**...depending on the Last Exit Status: *SuccessfulExit***

When a job terminates it can set a numeric return value, its exit status. By convention an exit status of 0 is used when the program exited successfully. Any other number informs about a problem.

The subkey *SuccessfulExit* can be used to restart your program depending on its previous exit status. When set to true the job will be restarted until it fails. Setting this value to false will restart the job until it succeeds.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>SuccessfulExit</key> <true/> </dict>

If this subkey is specified, launchd will run the job immediately after loading. To avoid this behaviour set the subkey *AfterInitialDemand*.

**...depending on Crashes: *Crashed***

The subkey *Crashed* can be used to restart your program after it has crashed. When set to true the job will be restarted after it crashed. Setting this value to false will restart the job unless it has crashed.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>Crashed</key> <true/> </dict>

If this subkey is specified, launchd will run the job immediately after loading. To avoid this behaviour set the subkey *AfterInitialDemand*.

**...depending on Network Availability: *NetworkState* 10.5 10.10**

Setting this subkey to true will start the job when/while any network is/becomes available. Setting this subkey to false will start the job when/while all network connections are down.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>NetworkState</key> <true/> </dict>

If this subkey is specified, launchd will run the job immediately after loading. To avoid this behaviour set the subkey *AfterInitialDemand*.

**...depending on the Existence of a Path: *PathState***

Use this subkey to keep a job alive as long as a given path exists (true) or does not exist (false).

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>PathState</key> <dict> <key>/tmp/runJob</key> <true/> </dict> </dict>

If this subkey is specified, launchd will run the job immediately after loading. To avoid this behaviour set the subkey *AfterInitialDemand*.

**...depending on Other Jobs: *OtherJobEnabled***

The name of this subkey is somewhat misleading. It does not check if a certain job is enabled, but if a job is *loaded*. The following job will be started (and restarted) when a job with label local.otherJob is not loaded. It will be terminated when a job with label local.otherJob is loaded.

Always specify the Label of the job you are referring to, not the name of the job definition file.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>OtherJobEnabled</key> <dict> <key>local.otherJob</key> <false/> </dict> </dict>

**Delaying the start: *AfterInitialDemand***

This key affects other keys related to run conditions like RunAtLoad, StartCalendarInterval, WatchPaths or KeepAlive. If set, these keys will be ignored until the job has been started manually.

**Plain Text**

LaunchControl

<key>KeepAlive</key> <dict> <key>AfterInitialDemand</key> <dict> <key>local.otherJob</key> <true/> </dict> </dict>

**Permissions and Security**

**UserName GroupName InitGroups**

Using the keys UserName and GroupName you can specify as which user the job should run. Both keys expect a string (the name of the user/group, not the id) as value.

The key InitGroups specifies if launchd should call the function initgroups(3) before starting the job.

**Plain Text**

LaunchControl

<key>UserName</key> <string>nobody</string> <key>GroupName</key> <string>nobody</string> <key>InitGroups</key> <true/>

These keys will be ignored when specified for an agent.

**Umask**

Using this key you can specify which permissions a file or folder will have by default when it is created by this job. The value is expected to be a decimal number.

In the following example we specify that new files should be readable and writable by the user owning the file. In addition new directories should be searchable by the user. Group members should be able to read new files and search created directories. Others should not be able to access new files or directories at all.

**Plain Text**

LaunchControl

First you have to come up with three digits between 0 and 7. The first one defines user access, the second one group access and the third one world access.

|  |  |
| --- | --- |
| **Digit** | **Granted permissions** |
| 0 | read, write and execute/search |
| 1 | read and write |
| 2 | read and execute/search |
| 3 | read only |
| 4 | write and execute |
| 5 | write only |
| 6 | execute/search only |
| 7 | none |

We want read, write and search permissions for the user: 0. We want read and search permissions for the group: 2. We want no access at all for everyone else: 7. Together: 027

The last step is to convert this octal number to its decimal representation. We'll use the bc command line tool to do that:

host:~ user$ **echo "obase=10;ibase=8; *027*" | bc** 23

The number we want is 23:

<key>Umask</key> <integer>23</integer>

This key will be ignored when specified for an agent.

New files will never be created with the execute bit set.

**RootDirectory**

Using this key you can execute the job in a chroot(2) jail. As far as the job is concerned, the directory provided with this key is the root of the filesystem. This implies that you need copies of all files required for the job inside this directory. This includes libraries, frameworks, command line tools and configuration files.

**Plain Text**

LaunchControl

<key>RootDirectory</key> <string>/var/jail</string>

This key will be ignored when specified for an agent.

**Miscellaneous**

**AbandonProcessGroup**

When launchd wants to terminate a job it sends a SIGTERM signal which will be propagated to all child processes of the job as well. Setting the value of this key to true will stop this propagation, allowing the child processes to survive their parents.

**Plain Text**

LaunchControl

<key>AbandonProcessGroup</key> <true/>

**ExitTimeOut**

launchd stops a job by sending the signal SIGTERM to the process. Should the job not terminate within ExitTimeOut seconds (20 seconds by default), launchd will send signal SIGKILL to force-quit it.

**Plain Text**

LaunchControl

<key>ExitTimeOut</key> <integer>30</integer>

**TimeOut 10.10**

The suggested idle time in seconds before the job should quit.

**Plain Text**

LaunchControl

<key>TimeOut</key> <integer>30</integer>

launchd will not stop the job. This is the responsibility of the job itself.

**ThrottleInterval**

Time in seconds to wait between program invocations.

This key may me used in combination with KeepAlive to run a job every n seconds while a certain condition applies.

**Plain Text**

LaunchControl

<key>ThrottleInterval</key> <integer>30</integer>

**Performance**

**LegacyTimers 10.9.2**

This key controls the behavior of timers created by the job. By default on OS X Mavericks version 10.9 and later, timers created by launchd jobs are coalesced. Batching the firing of timers with similar deadlines improves the overall energy efficiency of the system. If this key is set to true, timers created by the job will opt into less efficient but more precise behavior and not be coalesced with other timers.

**Plain Text**

LaunchControl

<key>LegacyTimers</key> <true/>

**Nice**

Run a job at an altered scheduling priority. Possible values range from -20 to 20. The default value is 0. Lower nice values cause more favorable scheduling.

**Plain Text**

LaunchControl

<key>Nice</key> <integer>-5</integer>

Nice values lower than zero are honored by daemons only.

This section explains how to acquire information about launchd jobs and how to load, unload, start and stop them.

All of this can be accomplished using the command line utility launchctl, but you will see that it is far easier using **[LaunchControl](https://www.soma-zone.com/LaunchControl)**. To [**quote**](http://nathangrigg.net/2014/03/launchcontrol-for-launchd/) Nathan Grigg:

*"****[LaunchControl](https://www.soma-zone.com/LaunchControl)****is a Mac app by soma-zone that helps manage launchd lobs. It aims to do “one thing well” and succeeds spectacularly. Whether you are new to writing launchd agents or you already have some system in place, go buy LaunchControl now."*

All examples assume that you are logged in as a regular (non-privileged) user.

All examples refer to agents. Should you want to work with daemons you have to use sudo launchctl instead of launchctl. You will be prompted for the admin password when working with daemons.

**Bold** text is input, regular text is output.

**Getting information about a job**

We will get some information about a job with Label com.example.app.

**launchctl**

LaunchControl

We issue the command launchctl list, which returns a long list of currently loaded jobs. To search for our specific job with the label com.example.app we send the output to the command grep using the pipe character |.

host:~ user$ **launchctl list | grep com.example.app** - 2 com.example.app

* We received one line of output, which means that our job is currently **loaded**.
* The first colum contains the current process id. We received a - instead of a number, which means that while the job is loaded it is currently not **running**. Any numeric value in this column would indicate a running job.
* The second column displays the last **exit code**. A value of 0 indicates that the job finished successfully, a positive number that the job has reported an error, a negative number that the process was terminated because it received a signal.

launchctl also accepts the Label of a job as an optional argument. This results in a more detailed output in a JSON-like format.

To find out if a job's Disabled key has been **overridden** we query the override database using the xpath command line utility. Its first argument is the name of the file to query, its second argument the query itself. You have to replace the job label in the query.

host:~ user$ **xpath /var/db/launchd.db/com.apple.launchd.peruser.`id -u`/overrides.plist \ "/plist/dict/key[text()='*com.example.app*']/following-sibling::\*[1]/key/following-sibling::\*[1]"** Found 1 nodes: -- NODE -- <true />

The input has been split across two lines, hence the \ at the end of the first line. You can enter the command as a single line by ommitting this character and the newline.

We get a single hit, which means that the Disabled key is overridden. Its value is true. We would receive No nodes found if the key had not been overriden.

The database for daemons is /var/db/launchd.db/com.apple.launchd/overrides.plist (OSX 10.5 – OSX 10.9) or /var/db/com.apple.xpc.launchd/disabled.plist (OSX 10.10 and later).

**Loading a Job**

**launchctl**

LaunchControl

We have already learned that launchd loads jobs automatically, daemons at boot time, agents when the user logs in. launchctl allows you to load jobs manually. The following example loads the agent com.example.error. It is an user agent, so we know its definition will be in ~/Library/LaunchAgents.

host:~ user$ **launchctl load ~/Library/LaunchAgents/com.example.app.plist**

If everything went ok you will see no output. But you may receive this line:

host:~ user$ **launchctl load ~/Library/LaunchAgents/com.example.app.plist** nothing found to load

If this happens the job has either set the Disabled key to true or the job has been disabled in the override database. In either case you can force launchctl to load the job like this:

host:~ user$ **launchctl load -F ~/Library/LaunchAgents/com.example.app.plist**

**Unloading a Job**

**launchctl**

LaunchControl

launchctl allows you to unload jobs manually. The following example unloads the agent com.example.app. It is an user agent, so we know its definition will be in ~/Library/LaunchAgents.

host:~ user$ **launchctl unload ~/Library/LaunchAgents/com.example.app.plist**

If everything went ok you will see no output.

**Starting a Job**

Sometimes you may want to start a job regardless of its run conditions. This section explains how to do that.

**launchctl**

LaunchControl

This command expects the job Label as its argument.

host:~ user$ **launchctl start com.example.app**

If everything went ok you will see no output. But you may get something like this:

host:~ user$ **launchctl start com.example.app** launchctl start error: No such process

You either misspelled the Label of the job or forgot to load it in the first place.

A job has to be loaded before it can be started.

**Stopping a Job**

**launchctl**

LaunchControl

This command expects the job Label as its argument.

host:~ user$ **launchctl stop com.example.app**

If everything went ok you will see no output. But you may get something like this:

host:~ user$ **launchctl stop com.example.app** launchctl stop error: No such process

You either misspelled the Label of the job or forgot to load it in the first place.

Stopping a job will send the signal SIGTERM to the process. Should this not stop the process launchd will wait ExitTimeOut seconds (20 seconds by default) before sending SIGKILL.

The job may be intantly restarted when its run conditions are met.

**The override database**

When launchd is about to load a job it will check if it has the Disabled key set. Disabled jobs will not be loaded. But the value of this key can be overridden.

**Permanently disabling a job**

**Terminal**

LaunchControl

host:~ user$ **launchctl unload -w ~/Library/LaunchAgents/com.example.app.plist**

OSX 10.10 Yosemite and later: While you will always be able to switch a job's permanent status between enabled and disabled, you cannot (trivially) remove it from the override database again.

**Permanently enabling a job**

**Terminal**

LaunchControl

host:~ user$ **launchctl load -w ~/Library/LaunchAgents/com.example.app.plist**

OSX 10.10 Yosemite and later: While you will always be able to switch a job's permanent status between enabled and disabled, you cannot (trivially) remove it from the override database again.

**Removing a job from the override database (OSX 10.5 Leopard - OSX 10.9 Mavericks)**

**Terminal**

LaunchControl

The procedure is different for agents and daemons. You'll need the label of the job you want to remove. In this example: local.job. For agents type:

host:~ user$ **sudo /usr/libexec/Plistbuddy /var/db/launchd.db/com.apple.launchd.peruser.`echo $UID`/overrides.plist -c Delete:*local.job***

For daemons the command reads

host:~ user$ **sudo /usr/libexec/Plistbuddy /var/db/launchd.db/com.apple.launchd/overrides.plist -c Delete:*local.job***

**Removing a job from the override database (OSX 10.10 Yosemite and later)**

Starting with OSX 10.10 Yosemite the override database was moved to a different location. Additionally the database is read only when launchd starts. Modifications to the database will be overwritten when launchd quits. Because of this modifications have to be done in Recovery Mode. On OSX 10.10 Yosemite and later LaunchControl can not be used to remove jobs from the override database.

**Note current UID (user id) if you want to remove an agent from the override database.**

Open Terminal.app and enter

id -u

The number returned is the user id of the current user. You will need this number later. In this example we assume that the returned user id is 502.

**Boot into Recovery Mode**

* Restart your Mac
* Hold down ⌘R before OSX starts up. Hold these keys until the Apple logo appears. After your computer finishes starting up, you should see a desktop with an OS X menu bar and an OS X Utilities window
* Select Terminal from the Utilities menu

**Remove the job from the override database**

The following command lines assume that your system volume is called “Macintosh HD”. Please adjust the commands below in case your system volume name differs.

The procedure is different for agents and daemons. You'll need the label of the job you want to remove. In this example: local.job. For agents type:

host:~ user$ **/usr/libexec/Plistbuddy "/Volumes/Macintosh HD/var/db/com.apple.xpc.launchd/disabled.502.plist" -c Delete:*local.job***

Don't forget to replace the 502 with your actual user id. For daemons the command reads

host:~ user$ **/usr/libexec/Plistbuddy "/Volumes/Macintosh HD/var/db/com.apple.xpc.launchd/disabled.plist" -c Delete:*local.job***

**Reboot**

* Type reboot and press the ⏎ key.

Finding out why a launchd job does not perform as expected can be a time consuming task if you have to do it manually. Using **[LaunchControl](https://www.soma-zone.com/LaunchControl)** you can troubleshoot a job in a matter of seconds. Most of the time just selecting the job will highlight the problem and give advice on how to fix it.

Remember to always (re)load a job definition after changing it. Just saving it will leave launchd oblivious to the change.

**Common mistakes**

**launchd(8) is not init(8)**

It is important to remember that launchd is responsible for turning a process into a background process. When referencing a shell script in Program or ProgramArguments make sure the script does not put the actual program into the background. Some programs require special command line arguments (something like --foreground) to prevent them from daemonizing themselves. If in doubt try to run the program/script from the terminal. The program/script should only return after the process is done.

The same goes for resource limits, redirection of standard input/output/error, re-nicing and so on. launchd supports a wealth of configuration options. Use them and don't try to emulate their effects.

**launchd(8) is no shell**

Pipes (|), redirection (<,>) and variable expansion ($VAR) are only three examples for shell constructs that won't be interpreted by launchd(8). If you want to write shell code, give it to the shell. Instead of

<key>ProgramArguments</key> <array> <string>/bin/cat</string> <string>/etc/passwd</string> <string>|</string> <string>sort</string> </array>

write

<key>ProgramArguments</key> <array> <string>/bin/sh</string> <string>-c</string> <string>cat /etc/passwd | sort</string> </array>

**Don't use sudo in agents**

The Unix command sudo is used to run unix commands as a different user. If executed by an unprivileged (non-root) user it requires the user to enter a password. Obviously, when used by a LaunchAgent the password cannot be entered. Solution: Convert the agent to a daemon.

**My job is not loaded on login/boot**

**Terminal**

LaunchControl

Load the job manually and see what launchd has to say. The following example did not load because the job definition had the wrong permissions.

host:~ user$ **launchctl load ~/Library/LaunchAgents/com.example.app.plist** launchctl: Dubious permissions on file (skipping): /Users/rpaehlig/Library/LaunchAgents/com.example.app.plist nothing found to load

Make sure to load your job definition into the correct launchd instance. The example above loads an agent definition into the user launchd instance. Execute this command with root privileges (sudo) to load a daemon definition into the system launchd instance.

Checklist:

* Check the job definition has the proper file extension (.plist (Error: Dubious file. Not of type .plist)
* Check the job definition for wellformedness (Error: no plist was returned for)

host:~ user$ **plutil -lint ~/Library/LaunchAgents/com.example.app.plist** /Users/Guest/Library/LaunchAgents/com.example.app.plist: OK

* Check the Disabled key. Disabled jobs won't be automatically loaded.
* Check if the job has been disabled in the override database
* Check the permissions of the job definition. Job definitions have to be readable by the user. Job definitions must not be writable by group or other. Suggested privileges: 0644. (Error: Dubious permissions on file)
* Check the permissions of the directory containing the job definition. It has to be searchable by the user and must not be writable by group or other. Suggested privileges: 0755
* Check who owns the job definition. Agents must be owned by their respective user, daemons by root. (Error: Dubious ownership on file)
* Make sure the Label you provided is unique for this particular launchd instance.

**My job does not start automatically after loading**

Make sure you defined a run condition. LaunchControl will display a warning when no run condition has been specified.

**My job does not start**

**Terminal**

LaunchControl

Open two windows in Terminal.app. In the first one we will monitor the system log file for errors of our launchd instance. Replace the argument of the second grep with the Label of the job refusing to start. If you want to debug an agent enter:

host:~ user$ **tail -F /var/log/system.log | grep --line-buffered "com.apple.launchd.peruser.`id -u`\[" | grep "com.example.app"**

If you want to debug a daemon enter:

host:~ user$ **sudo tail -F /var/log/system.log | grep --line-buffered "com.apple.launchd\[" | grep "com.example.app"**

The last command asks for your super user password. Now start the job manually in the second window and see what launchd has to say. The following example did not start because the script did not define an executing interpreter:

Sep 19 12:01:52 host com.apple.launchd.peruser.201[235] (com.example.app[38643]): Job failed to exec(3) for weird reason: 8

launchd tells us that it was unable to execute the program specified in the job definition. To find out what "weird reason: 8" is, type this command:

host:~ user$ **perl -E 'say $!=shift' 8** Exec format error

But you may also get a line like this:

Sep 19 12:01:52 host com.apple.launchd.peruser.201[235] (com.example.app[38643]): Exited with code: 8

While the code is the same, the problem is a different one. launchd has executed the program specified in our job definition just fine, but the program exited with exit code 8. Unfortunately there is no way of knowing what this code means, as a programmer can choose arbitrary exit codes if he wants. Maybe the program uses standard exit codes, in which case you can use the Perl one-liner to get a description. If you are lucky the user manual of the program contains a list of all exit codes with an explanation. Otherwise ask the author about the meaning of the error code.

Fortunately exit codes are not the only way programs can communicate with the user. Add the keys StandardOutPath and StandardErrorPath to your job definition. Reload it and try to start it again. Now examine the files you specified for those keys. It is very likely that you will find the problem description in one of those files.

Some programs accept special arguments to set the location of log files. Make sure to examine those files too.

**I cannot save changes to System Agents and Daemons**

**OS X 10.11 El Capitan and newer**

OS X 10.11 El Capitan introduced System Integrity Protection (SIP). This feature provides an additional layer of security by protecting certain sytem files from modification even by root. In order to make changes to those protected files you have to disable SIP:

* Restart your Mac
* Hold down ⌘R before macOS starts up. Hold these keys until the Apple logo appears. After your computer finishes starting up, you should see a Desktop with a menu bar and a window offering several utility functions.
* Select Terminal from the Utilities menu (the menu, not said window)
* Type csrutil disable and press the ⏎ key.
* You should see a message that System Integrity Protection has been disabled
* Reboot by entering reboot and press the ⏎ key.

SIP is now disabled.

It is recommended to enable SIP again. Follow the instructions above. The command to enable SIP is csrutil enable

**macOS 10.15 Catalina**

In addition to SIP macOS 10.15 Catalina separates the Operating System (OS) from the user data. The OS is mounted read-only, so even with SIP disabled you cannot make changes to system files. In order to mount the OS partition in read/write mode disable SIP (see above), open Termina.app and enter:

sudo mount -uw /

This is a collection of recipes serving as examples and inspiration.

**Automatically restart application**

The trick here is that an OSX application is a folder *containing* (among other things) the executable file. The name of the executable is stored in a file called Info.plist inside this directory. Fortunately the command line utility open knows how to deal with application bundles.

**Plain Text**

LaunchControl

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>KeepAlive</key> <true/> <key>Label</key> <string>local.Safari.keepAlive</string> <key>ProgramArguments</key> <array> <string>/usr/bin/open</string> <string>-W</string> <string>/Applications/Safari.app</string> </array> </dict> </plist>

**Run job every hour while a certain file exists**

We want to perform a backup of a database but this backup should only be performed while the database is running. We will use the PID file to find out if it is running or not.

Using the subkey *PathState* of the key KeepAlive we can make sure that our job is run (and kept alive) while our PID file exists. We will delay restarts using the ThrottleInterval key.

**Plain Text**

LaunchControl

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>Label</key> <string>com.example.app</string> <key>Program</key> <string>/Users/Me/Scripts/backup.sh</string> <key>KeepAlive</key> <dict> <key>PathState</key> <dict> <key>/var/log/mysql.pid</key> <true/> </dict> </dict> <key>ThrottleInterval</key> <integer>3600</integer> </dict> </plist>

**Creating a network server**

launchd makes it very easy to create servers. In fact, launchd can turn any program reading from standard input into a server.

In this recipe we will turn Unix cat(1), a program which copies STDIN to STDOUT, into an network server listening on port 8888 on our machine, copying everything it receives to the file /tmp/test.stdout.

**Plain Text**

LaunchControl

<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"> <plist version="1.0"> <dict> <key>Label</key> <string>com.example.srv</string> <key>Program</key> <string>/bin/cat</string> <key>Sockets</key> <dict> <key>sock1</key> <dict> <key>SockNodeName</key> <string>localhost</string> <key>SockPassive</key> <true/> <key>SockServiceName</key> <string>8888</string> </dict> </dict> <key>StandardOutPath</key> <string>/tmp/test.stdout</string> <key>inetdCompatibility</key> <dict> <key>Wait</key> <false/> </dict> </dict> </plist>

After loading our job definition we will do a quick test.

1. Open Terminal.app and establish a network connection to port 8888: telnet localhost 8888
2. Open another Terminal.app window and have a look at our job's STDOUT: tail -f /tmp/test.out
3. Type some text in the first window, press enter.
4. The text you entered in the first window appears in the second one.