NAME

launchd.plist -- System wide and per-user daemon/agent configuration files

DESCRIPTION

This document details the parameters that can be given to an XML property list that can be loaded into **launchd** with **launchctl**.

EXPECTATIONS

Daemons or agents managed by **launchd** are expected to behave certain ways.

A daemon or agent launched by **launchd** MUST NOT do the following in the process directly launched by **launchd**:

+ Call daemon(3).
+ Do the moral equivalent of daemon(3) by calling fork(2) and have the parent process exit(3) or _exit(2).

A **launchd** daemon or agent should not perform the following as part of its initialization, as launchd will always implicitly perform them on behalf of the process.

+ Redirect stdio(3) to /dev/null.

A **launchd** daemon or agent need not perform the following as part of its initialization, since **launchd** can perform them on the process' behalf with the appropriate **launchd.plist** keys specified.

+ Setup the user ID or group ID.
+ Setup the working directory.
+ chroot(2)
+ setsid(2)
+ Close "stray" file descriptors.
+ Setup resource limits with setrlimit(2).
+ Setup priority with setpriority(2).

A daemon or agent launched by **launchd** SHOULD:

+ Launch on demand given criteria specified in the XML property list. More information can be found later in this man page.
+ Handle the SIGTERM signal, preferably with a dispatch(3) source, and respond to this signal by unwinding any outstanding work quickly and then exiting.

A daemon or agent launched by **launchd** <u>MUST</u>:

+ check in for any **MachServices** advertised in its plist, using xpc_connection_create_mach_service(3) (or bootstrap_check_in(3)) if it uses MIG or raw Mach for communi-

cation
+ check in for any **LaunchEvents** advertised in its plist, using xpc_set_event_stream_handler(3)

XML PROPERTY LIST KEYS

The following keys can be used to describe the configuration details of your daemon or agent. Property lists are Apple's standard configuration file format. Please see plist(5) for more information. Please note: property list files are expected to have their name end in ".plist". Also please note that it is the expected convention for launchd property list files to be named <Label>.plist. Thus, if your job label is "com.apple.sshd", your plist file should be named "com.apple.sshd.plist".

Label <string>
This required key uniquely identifies the job to **launchd**.

Disabled <boolean>
This optional key specifies whether the job should be loaded by default. Note that this key may be overridden through the enable subcommand of launchctl(3). Previous Darwin operating systems would modify the configuration file's value for this key, but now this state is kept externally.

UserName <string>
This optional key specifies the user to run the job as. This key is only applicable for services that are loaded into the privileged system domain.

GroupName <string>
This optional key specifies the group to run the job as. This key is only applicable for services that are loaded into the privileged system domain. If UserName is set and GroupName is not, then the group will be set to the primary group of the user.

inetdCompatibility <dictionary>
The presence of this key specifies that the daemon expects to be run as if it were launched from inetd. For new projects, this key should be avoided.

> **Wait <boolean>**
> This flag corresponds to the "wait" or "nowait" option of inetd. If true, then the listening socket is passed via the stdio(3) file descriptors. If false, then accept(2) is called on behalf of the job, and the result is passed via the stdio(3) descriptors.

LimitLoadToHosts <array of strings>
This configuration file only applies to the hosts listed with this key. This key is no longer supported.

LimitLoadFromHosts <array of strings>
This configuration file only applies to hosts NOT listed with this key. This key is no longer supported.

LimitLoadToSessionType <string or array of strings>
This configuration file only applies to sessions of the type(s) specified. This key only applies to jobs which are agents. There are no distinct sessions in the privileged system context.

LimitLoadToHardware <dictionary of arrays>
This configuration file only applies to the hardware listed with this key. Each key in the dictionary defines a subdomain of the "hw" sysctl(3) domain. Each value of the value defines valid values for the job to load. So a key of "model" with an array specifying only "MacBookPro4,2" would only load on a machine whose "hw.model" value was "MacBookPro4,2".

Program <string>
This key maps to the first argument of execv(3) and indicates the absolute path to the executable for the job. If this key is missing, then the first element of the array of strings provided to the **ProgramArguments** will be used instead. This key is required in the absence of the **ProgramArguments** key.

ProgramArguments <array of strings>
This key maps to the second argument of execvp(3) and specifies the argument vector to be passed to the job when a process is spawned. This key is required in the absence of the Program key. IMPORTANT: Many people are confused by this key. Please read execvp(3) very carefully!

NOTE: The **Program** key must be an absolute path. Previous versions of launchd did not enforce this requirement but failed to run the job. In the absence of the **Program** key, the first element of the **ProgramArguments** array may be either an absolute path, or a relative path which is resolved using _PATH_STDPATH.

EnableGlobbing <boolean>
This flag causes **launchd** to use the glob(3) mechanism to update the program arguments before invocation.

EnableTransactions <boolean>
This key instructs **launchd** that the job uses xpc_transaction_begin(3) and xpc_transaction_end(3) to track outstanding transactions. When a process has an outstanding transaction, it is considered active, otherwise inactive. A transaction is automatically created when an XPC message expecting a reply is received, until the reply is sent or the request message is discarded. When **launchd** stops an active process, it sends SIGTERM first, and then SIGKILL after a reasonable timeout. If the process is inactive, SIGKILL is sent immediately.

EnablePressuredExit <boolean>
This key opts the job into the system's Pressured Exit facility. Use of this key implies **EnableTransactions** , and also lets the system consider process eligible for reclamation under memory pressure when it's inactive. See xpc_main(3) for details. Jobs that opt into Pressured Exit will be automatically relaunched if they exit or crash while holding open transactions.

NOTE: launchd(8) does not respect **EnablePressuredExit** for jobs that have **KeepAlive** set to true.

IMPORTANT: Jobs which opt into Pressured Exit will ignore SIGTERM rather than exiting by default, so a dispatch(3) source must be used when handling this signal.

OnDemand <boolean>
This key does nothing if set to true. If set to false, this key is equivalent to specifying a true value for the **KeepAlive** key. This key should not be used. Please remove this key from your **launchd.plist**.

ServiceIPC <boolean>
Please remove this key from your **launchd.plist**.

KeepAlive <boolean or dictionary of stuff>
This optional key is used to control whether your job is to be kept continuously running or to let demand and conditions control the invocation. The default is false and therefore only demand will start the job. The value may be set to true to unconditionally keep the job alive. Alternatively, a dictionary of conditions may be specified to selectively control whether **launchd** keeps a job alive or not. If multiple keys are provided, launchd ORs them, thus providing maximum flexibility to the job to refine the logic and stall if necessary. If **launchd** finds no reason to restart the job, it falls back on demand based invocation. Jobs that exit quickly and frequently when configured to be kept alive will be throttled to conserve system resources.

SuccessfulExit <boolean>
If true, the job will be restarted as long as the program exits and with an exit status of zero. If false, the job will be restarted in the inverse condition. This key implies that "RunAtLoad" is set to true, since the job needs to run at least once before an exit status can be determined.

NetworkState <boolean>
This key is no longer implemented as it never acted how most users expected.

PathState <dictionary of booleans>
Each key in this dictionary is a file-system path. If the value of the key is true, then the job will be kept alive as long as the path exists. If false, the job will be kept alive in the inverse condition. The intent of this feature is that two or more jobs may create semaphores in the file- system namespace. The following example keeps the job alive as long as the file **/path/to/file** exists.

```
<key>KeepAlive</key>
<dict>
     <key>PathState</key>
```

```
          <dict>
               <key>/path/to/file</key>
               <true/>
          </dict>
     </dict>
```

IMPORTANT: Filesystem monitoring mechanisms are inherently race-prone and lossy. This option should be avoided in favor of demand-based alternatives using IPC.

OtherJobEnabled <dictionary of booleans>
Each key in this dictionary is the name of another job. If the value is true, then the job will be kept alive as long as one of the specified other jobs is loaded in launchd(8).

NOTE: This key only evaluates whether the job is loaded, not whether it is running. Use of this key is highly discouraged. If multiple jobs need to coordinate coordinate their lifecycles, they should establish contracts using IPC.

Crashed <boolean>
If true, the the job will be restarted as long as it exited due to a signal which is typically associated with a crash (SIGILL, SIGSEGV, etc.). If false, the job will be restarted in the inverse condition.

RunAtLoad <boolean>
This optional key is used to control whether your job is launched once at the time the job is loaded. The default is false. This key should be avoided, as speculative job launches have an adverse effect on system-boot and user-login scenarios.

RootDirectory <string>
This optional key is used to specify a directory to chroot(2) to before running the job.

IMPORTANT: iOS and OS X both make significant use of IPC to implement features. The details of the communication between a client and server are typically implemented in dynamic library code that is abstracted away from the caller beneath the API boundary so that the client of a daemon is not aware of any IPC that is happening.

So unless the library stack which exists in the jail specified by this key or a call to chroot(2) is identical to the one shipping on the system, there is no guarantee that a process running in that jail will know how to communicate with the daemons on the system. Mismatches in the library stack between the jail and the system can manifest as random failures, hangs and crashes.

For these reasons, it is highly recommended that you avoid making use of this key unless you have taken special precautions to ensure that the job in question never attempts any IPC by setting the XPC_NULL_BOOTSTRAP environment variable to a value of "1". Note that even if you have done this, you must also take special care to propagate this environment variable to any child processes your job may spawn through fork(2) or posix_spawn(2). And even if you have done that, there is no guarantee that any subprocesses spawned by your child processes will take care to do the same thing unless you completely control all possible chains of execution, which is unlikely.

WorkingDirectory <string>
This optional key is used to specify a directory to chdir(2) to before running the job.

EnvironmentVariables <dictionary of strings>
This optional key is used to specify additional environmental variables to be set before running the job. Each key in the dictionary is the name of an environment variable, with the corresponding value being a string representing the desired value. NOTE: Values other than strings will be ignored.

Umask <integer or string>
This optional key specifies what value should be passed to umask(2) before running the job. If the value specified is an integer, it must be a decimal representation of the desired umask, as property lists do not support encoding integers in octal. If a string is given, the string will be converted into an integer as per the rules described in strtoul(3), and an octal value may be specified by prefixing the string with a '0'. If a string that does not cleanly convert to an integer is specified, the behavior will be to set a umask(2) according to the strtoul(3) parsing rules.

TimeOut <integer>
The recommended idle time out (in seconds) to pass to the job. This key never did anything interesting and is no longer implemented. Jobs seeking to exit when idle should use the EnablePressuredExit key to opt into the system mechanism for reclaiming killable jobs under memory pressure.

ExitTimeOut <integer>
The amount of time **launchd** waits between sending the SIGTERM signal and before sending a SIGKILL signal when the job is to be stopped. The default value is system-defined. The value zero is interpreted as infinity and should not be used, as it can stall system shutdown forever.

ThrottleInterval <integer>
This key lets one override the default throttling policy imposed on jobs by **launchd**. The value is in seconds, and by default, jobs will not be spawned more than once every 10 seconds. The principle behind this is that jobs should linger around just in case they are needed again in the near future. This not only reduces the latency of responses, but it encourages developers to amortize the cost of program invocation.

InitGroups <boolean>
This optional key specifies whether initgroups(3) to initialize the group

list for the job. The default is true. This key will be ignored if the **UserName** key is not set. Note that for agents, the **UserName** key is ignored.

WatchPaths <array of strings>
This optional key causes the job to be started if any one of the listed paths are modified.

<u>IMPORTANT</u>: Use of this key is highly discouraged, as filesystem event monitoring is highly race-prone, and it is entirely possible for modifications to be missed. When modifications are caught, there is no guarantee that the file will be in a consistent state when the job is launched.

QueueDirectories <array of strings>
This optional key keeps the job alive as long as the directory or directories specified are not empty.

StartOnMount <boolean>
This optional key causes the job to be started every time a filesystem is mounted.

StartInterval <integer>
This optional key causes the job to be started every N seconds. If the system is asleep during the time of the next scheduled interval firing, that interval will be missed due to shortcomings in kqueue(3). If the job is running during an interval firing, that interval firing will likewise be missed.

StartCalendarInterval <dictionary of integers or array of dictionaries of integers>
This optional key causes the job to be started every calendar interval as specified. Missing arguments are considered to be wildcard. The semantics are similar to crontab(5) in how firing dates are specified. Multiple dictionaries may be specified in an array to schedule multiple calendar intervals.

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.

Note that **StartInterval** and **StartCalendarInterval** are not aware of each other. They are evaluated completely independently by the system.

Minute <integer>
The minute (0-59) on which this job will be run.

Hour <integer>
The hour (0-23) on which this job will be run.

Day <integer>
The day of the month (1-31) on which this job will be run.

Weekday <integer>
The weekday on which this job will be run (0 and 7 are Sunday). If both **Day** and **Weekday** are specificed, then the job will be started if either one matches the current date.

Month <integer>
The month (1-12) on which this job will be run.

StandardInPath <string>
This optional key specifies that the given path should be mapped to the job's stdin(4), and that the contents of that file will be readable from the job's stdin(4). If the file does not exist, no data will be delivered to the process' stdin(4).

StandardOutPath <string>
This optional key specifies that the given path should be mapped to the job's stdout(4), and that any writes to the job's stdout(4) will go to the given file. If the file does not exist, it will be created with writable permissions and ownership reflecting the user and/or group specified as the **UserName** and/or **GroupName**, respectively (if set) and permissions reflecting the umask(2) specified by the **Umask** key, if set.

StandardErrorPath <string>
This optional key specifies that the given path should be mapped to the job's stderr(4), and that any writes to the job's stderr(4) will go to the given file. Note that this file is opened as readable and writable as mandated by the POSIX specification for unclear reasons. If the file does not exist, it will be created with ownership reflecting the user and/or group specified as the **UserName** and/or **GroupName**, respectively (if set) and permissions reflecting the umask(2) specified by the **Umask** key, if set.

Debug <boolean>
This optional key specifies that **launchd** should adjust its log mask temporarily to LOG_DEBUG while dealing with this job.

WaitForDebugger <boolean>
This optional key specifies that **launchd** should launch the job in a suspended state so that a debugger can be attached to the process as early as possible (at the first instruction).

SoftResourceLimits <dictionary of integers>

HardResourceLimits <dictionary of integers>
Resource limits to be imposed on the job. These adjust variables set with setrlimit(2). The following keys apply:

Core <integer>
The largest size (in bytes) core file that may be created.

CPU <integer>

The maximum amount of cpu time (in seconds) to be used by each process.

Data <integer>
The maximum size (in bytes) of the data segment for a process; this defines how far a program may extend its break with the sbrk(2) system call.

FileSize <integer>
The largest size (in bytes) file that may be created.

MemoryLock <integer>
The maximum size (in bytes) which a process may lock into memory using the mlock(2) function.

NumberOfFiles <integer>
The maximum number of open files for this process. Setting this value in a system wide daemon will set the sysctl(3) kern.maxfiles (SoftResourceLimits) or kern.maxfilesperproc (HardResourceLimits) value in addition to the setrlimit(2) values.

NumberOfProcesses <integer>
The maximum number of simultaneous processes for this UID. Setting this value in a system wide daemon will set the sysctl(3) kern.max-proc (SoftResourceLimits) or kern.maxprocperuid (HardResourceLim-its) value in addition to the setrlimit(2) values.

ResidentSetSize <integer>
The maximum size (in bytes) to which a process's resident set size may grow. This imposes a limit on the amount of physical memory to be given to a process; if memory is tight, the system will prefer to take memory from processes that are exceeding their declared resident set size.

Stack <integer>
The maximum size (in bytes) of the stack segment for a process; this defines how far a program's stack segment may be extended. Stack extension is performed automatically by the system.

Nice <integer>
This optional key specifies what nice(3) value should be applied to the daemon.

ProcessType <string>
This optional key describes, at a high level, the intended purpose of the job. The system will apply resource limits based on what kind of job it is. If left unspecified, the system will apply light resource limits to the job, throttling its CPU usage and I/O bandwidth. This classification is preferable to using the HardResourceLimits, SoftResourceLimits and Nice keys. The following are valid values:

Background

Background jobs are generally processes that do work that was not directly requested by the user. The resource limits applied to Background jobs are intended to prevent them from disrupting the user experience.

Standard
Standard jobs are equivalent to no ProcessType being set.

Adaptive
Adaptive jobs move between the Background and Interactive classifications based on activity over XPC connections. See xpc_transaction_begin(3) for details.

Interactive
Interactive jobs run with the same resource limitations as apps, that is to say, none. Interactive jobs are critical to maintaining a responsive user experience, and this key should only be used if an app's ability to be responsive depends on it, and cannot be made Adaptive.

AbandonProcessGroup <boolean>
When a job dies, **launchd** kills any remaining processes with the same process group ID as the job. Setting this key to true disables that behavior.

LowPriorityIO <boolean>
This optional key specifies whether the kernel should consider this daemon to be low priority when doing filesystem I/O.

LowPriorityBackgroundIO <boolean>
This optional key specifies whether the kernel should consider this daemon to be low priority when doing filesystem I/O when the process is throttled with the Darwin-background classification.

MaterializeDatalessFiles
This optional key specifies the dataless file materialization policy. Setting this key to true causes dataless files to be materialized. Setting this key to false causes dataless files to not be materialized. If this key is not set, the default system policy for dataless files will be used. See setiopolicy_np(3)

LaunchOnlyOnce <boolean>
This optional key specifies whether the job can only be run once and only once. In other words, if the job cannot be safely respawned without a full machine reboot, then set this key to be true.

MachServices <dictionary of booleans or a dictionary of dictionaries>
This optional key is used to specify Mach services to be registered with the Mach bootstrap namespace. Each key in this dictionary should be the name of a service to be advertised. The value of the key must be a boolean and set to true or a dictionary in order for the service to be advertised. Valid keys in this dictionary are:

ResetAtClose <boolean>
The default value for this key is false, and so the port is recycled, thus leaving clients to remain oblivious to the demand nature of the job. If the value is set to true, clients receive port death notifications when the job lets go of the receive right. The port will be recreated atomically with respect to bootstrap_look_up() calls, so that clients can trust that after receiving a port-death notification, the new port will have already been recreated. Setting the value to true should be done with care. Not all clients may be able to handle this behavior. The default value is false.

Note that this option is not compatible with xpc(3), which automatically handles notifying clients of interrupted connections and server death.

HideUntilCheckIn <boolean>
Reserve the name in the namespace, but cause bootstrap_look_up() to fail until the job has checked in with **launchd**.

This option is incompatible with xpc(3), which relies on the constant availability of services. This option also encourages polling for service availability and is therefore generally discouraged. Future implementations will penalize use of this option in subtle and creative ways.

Jobs can dequeue messages from the MachServices they advertised with xpc_connection_create_mach_service(3) or bootstrap_check_in() API (to obtain the underlying port's receive right) and the Mach APIs to dequeue messages from that port.

Sockets <dictionary of dictionaries... OR dictionary of array of dictionaries...>
This optional key is used to specify launch on demand sockets that can be used to let **launchd** know when to run the job. The job must check-in to get a copy of the file descriptors using the launch_activate_socket(3) API. The keys of the top level Sockets dictionary can be anything. These keys are meant for the application developer to associate which socket descriptors correspond to which application level protocols (e.g. http vs. ftp vs. DNS...).

The parameters below are used as inputs to call getaddrinfo(3).

SockType <string>
This optional key tells **launchd** what type of socket to create. The default is "stream" and other valid values for this key are "dgram" and "seqpacket" respectively.

SockPassive <boolean>
This optional key specifies whether listen(2) or connect(2) should be called on the created file descriptor. The default is true, to listen for new connections.

SockNodeName <string>
This optional key specifies the node to connect(2) or bind(2) to.

SockServiceName <string or integer>
This optional key specifies the service on the node to connect(2) or bind(2) to. It may be a port number represented as an integer or a service name represented as a string ("ssh", "telnet", etc.)

SockFamily <string>
This optional key can be used to specifically request that "IPv4" or "IPv6" socket(s) be created. An additional option, "IPv4v6" indicates that a single socket that listens for both IPv4 and IPv6 connections should be created.

SockProtocol <string>
This optional key specifies the protocol to be passed to socket(2). The only values understood by this key at the moment are "TCP" and "UDP".

SockPathName <string>
This optional key implies SockFamily is set to "Unix". It specifies the path to connect(2) or bind(2) to.

SecureSocketWithKey <string>
This optional key is a variant of SockPathName. Instead of binding to a known path, a securely generated socket is created and the path is assigned to the environment variable that is inherited by all jobs spawned in the job's context.

SockPathOwner <integer>
This optional key specifies the user ID that should be the domain socket's owner.

SockPathGroup <integer>
This optional key specifies the group ID that should be set as the domain socket's group.

SockPathMode <integer>
This optional key specifies the mode of the socket. Known bug: Property lists don't support octal, so please convert the value to decimal.

Bonjour <boolean or string or array of strings>
This optional key can be used to request that the service be registered with the the Bonjour subsystem. If the value is boolean, the service name is inferred from the SockServiceName.

MulticastGroup <string>
This optional key can be used to request that the datagram socket join a multicast group. If the value is a hostname, then getaddrinfo(3) will be used to join the correct multicast address

for a given socket family. If an explicit IPv4 or IPv6 address is given, it is required that the SockFamily family also be set, otherwise the results are undefined.

LaunchEvents <dictionary of dictionaries of dictionaries>
Specifies higher-level event types to be used as launch-on-demand event sources. Each sub-dictionary defines events for a particular event subsystem, such as "com.apple.iokit.matching", which can be used to launch jobs based on the appearance of nodes in the IORegistry. Each dictionary within the sub-dictionary specifies an event descriptor that is specified to each event subsystem. With this key, the job promises to use the xpc_set_event_stream_handler(3) API to consume events. See xpc_events(3) for more details on event sources.

HopefullyExitsLast <string>
This key was a hack for jobs which could not properly keep track of their clients and is no longer implemented.

HopefullyExitsFirst <string>
This key was a hack for jobs which could not properly keep track of their clients and is no longer implemented.

SessionCreate <boolean>
This key specifies that the job should be spawned into a new security audit session rather than the default session for the context is belongs to. See auditon(2) for details.

LegacyTimers <boolean>
This optional 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. This key may have no effect if the job's ProcessType is not set to Interactive.

CAVEATS

Daemons and agents managed by launchd are subject to macOS user privacy protections. Specifying privacy sensitive files and folders in a launchd plist may not have the desired effect, and may prevent the job from running.

DEPENDENCIES

Unlike many bootstrapping daemons, launchd has no explicit dependency model. Interdependencies are expected to be solved through the use of IPC. It is therefore in the best interest of a job developer who expects dependents to define all of the sockets in the configuration file. This has the added benefit of making it possible to start the job based on demand instead of immediately. **launchd** will continue to place as many restrictions on jobs that do not conform to this model as possible.

EXAMPLE XML PROPERTY LISTS

The following XML Property List describes an on-demand daemon that will only launch when a message arrives on the "com.example.exampled" MachService.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//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.exampled</string>
     <key>Program</key>
     <string>/path/tp/exampled</string>
     <key>ProgramArguments</key>
     <array>
          <string>exampled</string>
          <string>argv1</string>
          <string>argv2</string>
     </array>
     <key>MachServices</key>
     <dict>
          <key>com.example.exampled</key>
          <true/>
     </dict>
</dict>
</plist>
```

FILES

~/Library/LaunchAgents	Per-user agents provided by the user.
/Library/LaunchAgents	Per-user agents provided by the administrator.
/Library/LaunchDaemons	System-wide daemons provided by the administrator.
/System/Library/LaunchAgents	Per-user agents provided by OS X.
/System/Library/LaunchDaemons	System-wide daemons provided by OS X.

SEE ALSO

launchctl(1), sysctl(3), launchd(8), plist(5)

[Process completed]