# winss

*Release dev*

**Dec 13, 2017**

# Fundamentals

A *service directory* may contain the following:

# CHAPTER 1

## run

A file *run* which contains the executable and the arguments to run the *service*. It is read and the process described in the contents are started every time the *service* must be started.

**Note:**

- The startup location for the executable will be the service directory itself.
- Environment variable substitution will be performed on the process described by *run* before executing. Additional environment variables can be created as mentioned below.

CHAPTER 2

# finish

An optional *finish* file which will run after the *run* process has finished. By default, the *finish* process must do its work and exit in less than 5 seconds; if it takes more than that, it is killed. The maximum duration of a *finish* execution can be configured via the *timeout-finish* file mentioned below.

**Note:** Similarly with *run*, environment variables will be substituted with the addition of `SUPERVISE_RUN_EXIT_CODE` which has the exit code of the *run* process. Additional environment variables mentioned below will also be available to the *finish* process.

# CHAPTER 3

## supervise

A directory named *supervise*, which is automatically created by *winss-supervise.exe* to store its information. This directory must be writable.

CHAPTER 4

# down

An optional, empty file *down*, which if exists will make the default state *down* and not *up* such that when *winss-supervise.exe* starts then the *run* process will not be started until signaled using *winss-svc.exe* -u.

# CHAPTER 5

## timeout-finish

An optional file *timeout-finish* which contains an unsigned integer that is the maximum number of milliseconds after which the *finish* process can execute for. It will be terminated after this period has expired. A value of 0 allows the *finish* process to run forever.

# CHAPTER 6

## env

An optional directory named *env* which contains files that represent the environment variable names and their contents are the values of the environment variables. For example a file named **env/USER** with the contents **foo** would be `USER=foo` when running a process. Like the contents of *run* and *finish*, the values can be substituted with current environment variables. Using this you can append to the **PATH** rather than overwriting it.

Multiple *env* dirs are supported if you make *env* a file and put the paths to each *env* dir into the file separated by a new line.

CHAPTER 7

# log

An optional *service directory* named *log*, which if exists and *winss-supervise.exe* is started by *winss-svscan.exe*, then the *winss-svscan.exe* process will start an additional *winss-supervise.exe* on the *log service directory* with the standard input piped from the standard output of the former *winss-supervise.exe* process. For example if the service **foo** has a *log* folder then both **foo** and **foo/log** will be supervised and the output of **foo** will be sent to the input of **foo/log**. Typically *winss-log.exe* can be used to log although not necessarily.

**Warning:** There may be additional files/directories which will be used by *winss-supervise.exe* in the future. The only file/directory which is guaranteed never to be used by *winss-supervise.exe* is **data**. It is therefore recommended that any specific application data in that file/directory.

CHAPTER 8

# Components

**Important:** Some components like *winss-svscan.exe* require the **PATH** environment to be set correctly. To run these commands please append the install directory to the **PATH**.

## 8.1 winss-supervise.exe

*winss-supervise.exe* monitors a *service*, making sure it stays alive, sending notifications to registered processes when it dies, and providing an interface to control its state.

```
Usage: winss-supervise.exe [options] servicedir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level
```

- *winss-supervise.exe* changes directory to `servicedir` *service directory*.
- It exits **100** if another *winss-supervise.exe* process is already monitoring this *service*.
- If the default state is *up* and not *down* then *winss-supervise.exe* starts the *run* process.
- If the *env* dir exists then a new environment block will be constructed and the *run* process will be started with the new environment block.
- If the *run* process fails to start then it will wait 10 seconds before trying to start again. It does not execute *finish* on failure to execute *run*.
- When *run* dies, *winss-supervise.exe* will start the *finish* process if it exists, with the exit code of *run*. The following environment variables will be set:

**SUPERVISE_RUN_EXIT_CODE**
The exit code of the *run* process will be set for the *finish* process.

- By default, *finish* must exit in less than *5-seconds* and will be terminated if still running. This timeout can be customized using the *timeout-finish* file.
- When *finish* dies (or is killed), *winss-supervise.exe* will wait at least *1-second* before starting *run* again to avoid busy-looping if *run* exits too quickly.
- If *finish* exits with 125, then *winss-supervise.exe* will not restart the *run* process. This can be used to signify permanent failure to start the service or you want to control the service coming up manually.

---

**Note:** The *run* process will be sent a **CTRL-BREAK** signal when it is asked to exit. By default the **CTRL-BREAK** will exit the program but it can be handled and used to exit the program cleanly.

---

**See also:**

***winss-svc.exe*** Can be used to send commands to the *winss-supervise.exe* process; mostly to change the *service* state.

***winss-svok.exe*** Can be used to check whether a *winss-supervise.exe* is successfully running.

***winss-svstat.exe*** Can be used to check the status of a *service*.

## 8.2 winss-svscan.exe

*winss-svscan.exe* starts and monitors a collection of *winss-supervise.exe* processes in a *scan directory*, each of these processes monitoring a single *service*. It is designed to be either the root or a branch of a *supervision tree*.

```
Usage: winss-svscan-g.exe [options] [scandir]

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
  -t<rescan>,  --timeout=<rescan>
                    Sets the rescan timeout.
  -s,          --signals
                    Divert signals.
```

- If given a `scandir` is specified then that is used. Otherwise then the current directory is used.
- It exits **100** if another *winss-svscan.exe* process is already monitoring this *scan directory*.
- If the **./.winss-svscan** control directory does not exist, *winss-svscan.exe* creates it. However, it is recommended to already have a .winss-svscan subdirectory in your *scan directory* directory, because *winss-svscan.exe* may try to launch **.winss-svscan/finish** at some point.
- If the *env* dir exists within **./.winss-svscan** then the current environment will be applied to the scan process.
- *winss-svscan.exe* performs an initial scan of its scan directory.
- *winss-svscan.exe* then occasionally runs scans based on the timeout specified or asked to do so by *winss-svscanctl.exe*.
- *winss-svscan.exe* runs until it is told to stop via *winss-svscanctl.exe*, or a signal. Then it starts the **.winss-svscan/finish** program.

### 8.2.1 Options

**-s, –signals** By default, *winss-svscan.exe* will handle any termination signals that it receives and attempt to propagate these and close. Using divert signals it will instead launch the process defined in **.winss-svscan/SIGTERM**.

*winss-svscan.exe* will not exit its loop on its own when it receives a termination signal and the -s option has been given. To make it exit its loop, invoke a *winss-svscanctl.exe* command from the signal handling process. For instance, a **.winss-svscan/SIGTERM** file could point to a Powershell script like the following:

```
# cleanup here
& winss-svscanctl.exe -q .
```

If an action cannot be taken (the relevant file doesn't exist, or cannot run, or any kind of error happens), *winss-svscan.exe* prints a warning message but does nothing else with the signal.

**-t<rescan>, –timeout=<rescan>** Perform a scan every `rescan` milliseconds. If rescan is **0** (the default), automatic scans are never performed after the first one and *winss-svscan.exe* will only detect new *services* when told to via a *winss-svscanctl.exe* -a command. It is strongly discouraged to set `rescan` to a positive value under **500**.

### 8.2.2 Scan

Every `rescan` milliseconds, or upon receipt of a *winss-svscanctl.exe* -a command, *winss-svscan.exe* runs a scanner routine.

The scanner scans the current directory for subdirectories (or symbolic links to directories), which must be *service directories*. It skips names starting with dots.

For every new subdirectory dir it finds, the scanner spawns a *winss-supervise.exe* process on it. If **dir/log** exists, it spawns a *winss-supervise.exe* process on both **dir** and **dir/log**, and creates a pipe from the service's stdout to the logger's stdin. This is starting the *service*, with or without a corresponding logger. Every *service* the scanner finds is flagged as "active".

The scanner remembers the *services* it found. If a *service* has been started in an earlier scan, but the current scan can't find the corresponding directory, the *service* is then flagged as inactive. No command is sent to stop inactive *winss-supervise.exe* processes (unless the administrator uses *winss-svscanctl.exe* -n), but inactive *winss-supervise.exe* processes will not be restarted if they die.

---

**Note:** *winss-supervise.exe* is used by *winss-svscan.exe* and must be in the **PATH**.

---

**See also:**

***winss-svscanctl.exe*** Can be used to send commands to the *winss-svscan.exe* process; mostly to signal a rescan.

## 8.3 winss-log.exe

*winss-log.exe* is a reliable logging program with automated log rotation.

```
Usage: winss-log.exe [options] script

Options:
  --help       Print usage and exit.
```

```
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
```

*winss-log.exe* reads and compiles logging script to an internal form. Then it reads its standard input, line by line, and performs actions on it, following the script it is given. It does its best to ensure there is never any log loss. It exits cleanly when stdin closes.

**Note:** The current logging script is limited to a single set of settings which can rotate files which exceed size `s`, keep `n` backups and output to a single *logdir*.

### 8.3.1 Logdirs

A *logdir (logging directory)* is a place where logs are stored. Currently *winss-log.exe* can only be configured to output to a single directory.

A *logdir* may contain the following files:

- **current**: the file where the current log stream is appended to.
- **@timestamp.u**: old log files which have been rotated.

#### Rotation

When the **current** file gets too big then a *rotation* occurs. The *archived* log file will be in the form *@timestamp.u* where *timestamp* is the number of seconds since the epoch. If there are too many archived log files in the *logdir*, the older ones are then removed. The logging stream will continue to log to a brand new **current** file.

### 8.3.2 Script

When starting up, *winss-log.exe* reads its arguments one by one; this argument sequence, or *directive sequence*, forms a *logging script* which tells *winss-log.exe* what to log, where, and how.

Every directive can be a *control directive* or an *action directive*. A valid logging script always contains at least one *action directive*; every *action directive* can be preceded by zero or more *control directives*. *winss-log.exe* will exit 100 if the script is invalid.

#### Control

These directives tune *winss-log.exe*'s behavior for the next actions.

- **n** *number*: next logdirs will contain up to *number* archived log files. If there are more, the oldest archived log files will be suppressed, only the latest *number* will be kept. By default, *number* is 10.
- **s** *filesize*: next rotations will occur when current log files approach *filesize* bytes. By default, *filesize* is 99999; it cannot be set lower than 4096 or higher than 16777215.
- **T**: the selected line will be prepended with a ISO 8601 timestamp.

### Action

These directives determine what *winss-log.exe* actually does with the logs.

- **dir** (must start with '.' or '[A-Z]:'): logdir. *winss-log.exe* will log the line into the log *dir*. *winss-log.exe* must have the right to write to the log *dir*.

  The drive letter needs to be different from a control directive otherwise it will not be interpreted as a log *dir*. Unfortunately UNC paths are not supported at this time but this will solve this issue.

### Examples

*winss-log.exe* n20 s1000000 .

## 8.4 winss-svc.exe

*winss-svc.exe* sends commands to a running *winss-supervise.exe* process. In other words, it's used to control a supervised process.

```
Usage: winss-svc.exe [options] servicedir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
  -k,          --kill
                    Terminate the process.
  -t,          --term
                    Send a CTRL+BREAK to the process
  -o,          --once
                    Equivalent to '-uO'.
  -d,          --down
                    Stop the supervised process.
  -u,          --up
                    Starts the supervised process.
  -x,          --exit
                    Stop the process and supervisor.
  -O,          --onceatmost
                    Only run supervised process once.
  -T<ms>,      --timeout=<ms>
                    Wait timeout in milliseconds if -w is specified.
  -w<dDur>,    --wait=<dDur>
                    Wait on (d)own/finishe(D)/(u)p/(r)estart.
```

*winss-svc.exe* sends the given series of commands in the order given to the *winss-supervise.exe* process monitoring the *service directory*, then exits 0. It exists 111 if it cannot send a command, or 100 if no *winss-supervise.exe* process is running on *service directory*

### 8.4.1 Options

**-k, –kill** Instruct the supervisor to kill the supervised process.

**-t, –term** Instruct the supervisor to send a `Control-Break` to the supervised process.

**-o, –once** Equivalent to "-uO".

**-d, –down** If the supervised process is up, send it a `Control-Break`. Do not restart it.

**-u, –up** If the supervised process is down, start it. Automatically restart it when it dies.

**-x, –exit** When the service is asked to be down and the supervised process dies, *winss-supervise.exe* will exit too. This command should normally never be used on a working system.

**-O, –onceatmost** Do not restart the supervised process when it dies. If it is down when the command is received, do not even start it.

**-t<ms>, –timeout=<ms>** If the -wstate option has been given, -T specifies a timeout (in milliseconds) after which *winss-svc.exe* will exit 1 with an error message if the service still hasn't reached the desired state. By default, the timeout is 0, which means that *winss-svc.exe* will block indefinitely.

**-wd, –wait=d** *winss-svc.exe* will not exit until the *service* is down, i.e. until the *run* process has died.

**-wD, –wait=D** *winss-svc.exe* will not exit until the *service* is *down* and *ready* to be brought *up*, i.e. a possible *finish* script has exited.

**-wu, –wait=u** *winss-svc.exe* will not exit until the *service* is *up*, i.e. there is a process running the *run* executable.

**-wr, –wait=r** *winss-svc.exe* will not exit until the *service* has been started or restarted.

**See also:**

***winss-svwait.exe*** Can be used to wait on the *winss-supervise.exe* process without sending any commands.

## 8.5 winss-svok.exe

*winss-svok.exe* checks whether a *service directory* is currently supervised.

```
Usage: winss-svok.exe [options] servicedir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
```

*winss-svok.exe* exits 0 if there is a *winss-supervise.exe* process monitoring the *servicedir* *service directory*, or 1 if there is not.

## 8.6 winss-svstat.exe

*winss-svstat.exe* prints a short, human-readable summary of the state of a process monitored by *winss-supervise.exe*.

```
Usage: winss-svstat.exe [options] servicedir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
```

*winss-svstat.exe* gives information about the process being monitored at the *servicedir* *service directory*, then exits 0. The information includes the following:

- whether the process is up or down, and if it's up, the number of seconds that it has been up.
- the process' pid, if it is up, or its last exit code or terminating signal, if it is down.
- what its default state is, if it is different from its current state.
- the number of seconds since it last changed states.
- whether the A *service* is ready and if it is, the number of seconds that it has been. A A *service* reported as down and ready simply means that it is ready to be brought up. A *service* is down and not ready when it is in the cleanup phase, i.e. the *finish* script is still being executed.

### 8.6.1 Exit Codes

- 0: success
- 1: *winss-supervise.exe* not running on *servicedir* *service directory*
- 100: wrong usage
- 111: system call failed

## 8.7 winss-svwait.exe

*winss-svwait.exe* blocks until a collection of supervised services goes up, or down.

*winss-svwait.exe* only waits for notifications; it never polls.

```
Usage: winss-svwait.exe [options] servicedir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
  -u,          --up
                    Wait until the services are up.
  -d,          --down
                    Wait until the services are down.
  -D,          --finished
                    Wait until the services are really down.
  -o,          --or
                    Wait until one of the services comes up or down.
  -a,          --and
                    Wait until all of the services comes up or down.
  -t<ms>,      --timeout=<ms>
                    Wait timeout in milliseconds.
```

*winss-svwait.exe* monitors one or more *service directories* given as its arguments, waiting for a state (ready, up or down) to happen. It exits 0 when the wanted condition becomes true.

### 8.7.1 Options

**-u, –up** *winss-svwait.exe* will wait until the *services* are up, as reported by *winss-supervise.exe*. This is the default; it is not reliable, but it does not depend on specific support in the service programs.

**-d, –down** *winss-svwait.exe* will wait until the *services* are down.

**-D, –finished** *winss-svwait.exe* will wait until the *services* are down and the cleanup scripts in *finish* for every *servicedir* have finished executing (or have timed out and been killed).

**-o, –or** *winss-svwait.exe* will wait until *one* of the given *services* comes up or down.

**-a, –and** *winss-svwait.exe* will wait until *all* of the given *services* comes up or down. This is the default.

**-t<ms>, –timeout=<ms>** If the requested events have not happened after *timeout* milliseconds, *winss-svwait.exe* will print a message to stderr and exit 1. By default, *timeout* is 0, which means no time limit.

---

**Note:**

- *winss-svwait.exe* should be given one or more *service directories* as arguments, not a *scan directory*. If you need to wait for a whole *scan directory*, give all its contents as arguments to *winss-svwait.exe*.
- *winss-svwait.exe* will only work on *service directories* that are already active, i.e. have a *winss-supervise.exe* process running on them. It will not work on a *service directory* where *winss-supervise.exe* has not been started yet.

---

**See also:**

***winss-svc.exe*** Can be used to send commands to the *winss-supervise.exe* process.

## 8.8 winss-svscanctl.exe

*winss-svscanctl.exe* sends commands to a running *winss-svscan.exe* process.

```
Usage: winss-svscanctl.exe [options] scandir

Options:
  --help       Print usage and exit.
  --version    Print the current version of winss and exit.
  -v[<level>], --verbose[=<level>]
                    Sets the verbose level.
  -a,          --alarm
                    Perform a scan of scandir.
  -b,          --abort
                    Close svscan only.
  -n,          --nuke
                    Prune supervision tree.
  -q,          --quit
                    Stop supervised process and svscan.
```

*winss-svscanctl.exe* sends the given series of commands to the *winss-svscan.exe* process monitoring the *scandir* *scan directory*, then exits 0. It exits 111 if it cannot send a command, or 100 if no *winss-svscan.exe* process is running on *scandir*.

### 8.8.1 Options

**-a, –alarm** *winss-svscan.exe* will immediately perform a scan of *scandir* to check for *services*.

**-b, –abort** *winss-svscan.exe* will run into its finishing procedure. It will not kill any of the maintained *winss-supervise.exe* processes.

**-n, –nuke** *winss-svscan.exe* will kill all the *winss-supervise.exe* processes it has launched but that did not match a *service directory* last time *scandir* was scanned, i.e. it prunes the supervision tree so that it matches exactly what was in *scandir* at the time of the last scan. A `Control-Break` is sent to the *winss-supervise.exe* processes supervising *services* and also the *winss-supervise.exe* processes supervising loggers.

CHAPTER 9

# Windows Supervision Suite

## 9.1 About

**winss** is a Morgan Stanley port of s6. Details of how to get started on **winss** either from source or pre-build binaries is available in the readme.

**Email** Contact the development team.

## 9.2 Definitions

**Important:**

**Service** In Windows a *service* typically involves developing for the Windows Service API which would be started using Service Control Manager. Here a *service* means a long running console application.

**Service Directory** A *service directory* is a directory containing all the information related to a *service* and forms the base of the *components* below.

See *here* for more information.

**Scan Directory** A *scan directory* is a directory containing a list of *service directories*, or symbolic links pointing to *service directories*.

## 9.3 Components

- *winss-supervise.exe*: monitors a long-lived process/service.
- *winss-svscan.exe*: starts and monitors a collection of *winss-supervise.exe* processes.
- *winss-log.exe*: reliable logging program with automated log rotation.

- *winss-svc.exe*: sends commands to a running *winss-supervise.exe* process.
- *winss-svok.exe*: checks whether a service directory is currently supervised
- *winss-svstat.exe*: prints a short, human-readable summary of the state of the process monitored by *winss-supervise.exe*.
- *winss-svwait.exe*: blocks until a collection of *winss-supervise.exe* processes goes up, or down.
- *winss-svscanctl.exe*: sends commands to a running *winss-svscan.exe* process.

## 9.4 Library

Details of the *winss* lib can be found here.

# Index