|  |
| --- |
|  |
| Windows Server Advanced Power Management |
| User manual |
|  |
|  |
| Version 1.5.4 |

|  |
| --- |
| [decatec.de](https://decatec.de/) |

Table of contents

[Introduction 2](#_Toc459269807)

[System requirements 3](#_Toc459269808)

[Limitations 4](#_Toc459269809)

[Installation 5](#_Toc459269810)

[Uninstall 8](#_Toc459269811)

[Update 9](#_Toc459269812)

[Usage 10](#_Toc459269813)

[Main window 11](#_Toc459269814)

[Settings 14](#_Toc459269815)

[General 14](#_Toc459269816)

[Monitoring (system) 18](#_Toc459269817)

[Monitoring (advanced) 21](#_Toc459269818)

[After policy check 24](#_Toc459269819)

[Wake 26](#_Toc459269820)

[Uptime 30](#_Toc459269821)

[Plugins 33](#_Toc459269822)

[Remote shut down 35](#_Toc459269823)

[Example scenarios 37](#_Toc459269824)

[Troubleshooting 39](#_Toc459269825)

[FAQ 42](#_Toc459269826)

[Windows Server Advanced Power Management for developers 43](#_Toc459269827)

[Plugin development 43](#_Toc459269828)

[Simple and advanced plugins 43](#_Toc459269829)

[Quick start 43](#_Toc459269830)

[Developing simple plugins 43](#_Toc459269831)

[Developing advanced plugins 45](#_Toc459269832)

[Debug plugins 48](#_Toc459269833)

[Make your plugin available to other users 49](#_Toc459269834)

[Remote shut down 50](#_Toc459269835)

# Introduction

Windows Server Advanced Power Management (hereafter often called WSAPM) is a program for Windows operating systems for defining policies to suppress standby mode when the computer is currently in use. This allows saving energy because the hardware is only running when it is currently needed. As WSAPM is running without user logon, it is specifically designed to run on Windows (home) servers. But it can also be installed in desktop computers to avoid unintended standby.

The Windows idle timeout can be specified in the Windows power options. But Windows only recognizes direct user input as action to suppress standby. When a program has to run a long time without user interaction, Windows automatically switches to standby due to idle timeout.

With WSAPM, specific policies can be defined which signal the current use of the computer. By these policies the standby mode can be suppressed.

WSAPM supports following scenarios for standby suppression:

* CPU load
* Memory load
* Network load (download, upload and combined, for single network adapters or all adapters combined)
* Access to network shares (files and/or folders)
* HDDs load (logical volumes)
* Available network devices (computers, smart phones, network-compatible TVs, streaming clients, gaming stations, etc.)
* Running processes

After the policies have been checked, user defined actions can be executed.

Furthermore, WSAPM can define a schedule to wake a computer from standby time-controlled.

It is also possible to define uptimes when the computer should not switch to standby (no matter of other policies defined).

WSAPM also offers a plugin interface and may therefore be extended easily with plugins which can offer own policies to be checked for standby suppression.

In combination with supported apps or programs (e.g. [MagicPacket](https://decatec.de/software/magicpacket_en/)), it is also possible to remotely shut down or restart the computer or send it to standby or hibernate mode.

# System requirements

Basically, WSAPM only needs a computer which supports standby. Following windows versions are supported:

* Windows XP
* Windows Vista
* Windows 7
* Windows 8/Windows 8.1
* Windows 10
* Windows Server 2003 (R2)
* Windows Server 2008 (R2)
* Windows Server 2012 (R2)
* Windows Home Server 2011

WSAPM also needs the [.NET Framework 4.0](http://en.wikipedia.org/wiki/.NET). It is automatically installed by the setup if not already installed on the system.

# Limitations

Currently there are following limitations for WSAPM:

* You need administrator rights to configure WSAPM. Basically WSAPM also runs without administrator rights, but you will not be able to change the configuration.
* The log entries generated by WSAPM are always in the (original) language of the operating system, even if the user’s language will be changed afterwards.

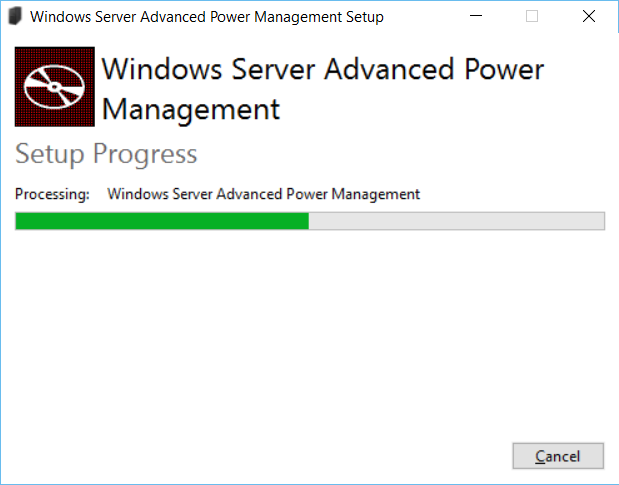
# Installation

The installation is started with the setup file (WaspmSetup.exe). You will need administrator rights to install the software. If the .NET Framework 4.0 is not already installed on the system, it gets installed automatically by the installer. In this case you will need an internet connection.



The path where WSAPM will be installed can be changed by clicking *Options*. By default, it will get installed in *C:\Program files (x86)\Windows Server Advanced Power Management*.

Click *Install* to start the installation.



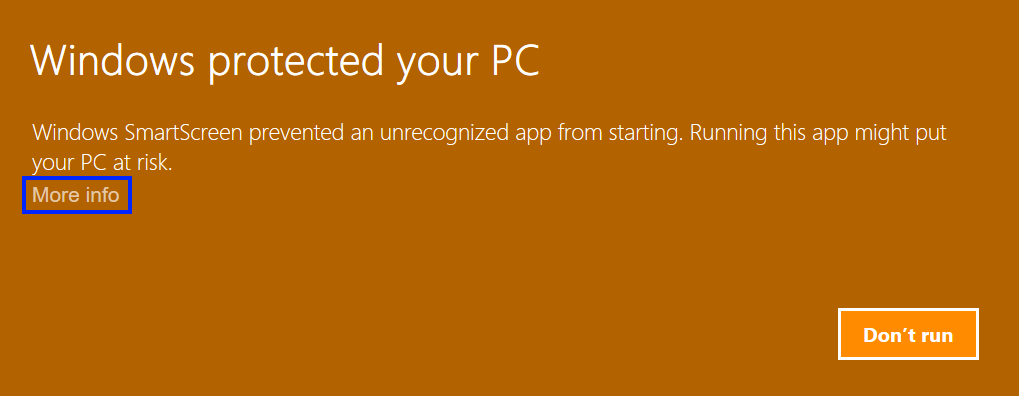
**Firewall exceptions**

During installation, an exception is added to the Windows Firewall, so that incoming remote shut down commands can be processed by WSAPM.

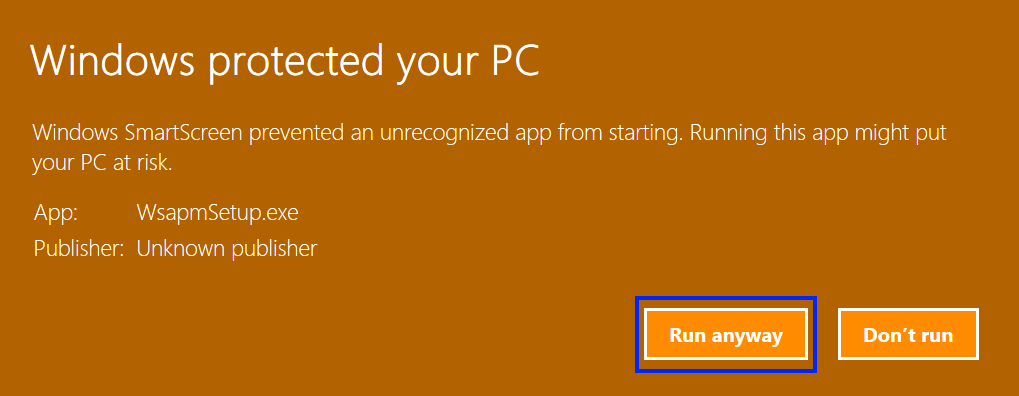
If you use a firewall other than the Windows Firewall, you have to add an exception for the file *Wsapm.Service.exe* (to be found in the install directory of WASPM) to incoming (UDP) traffic manually.

**SmartScreen**

Microsoft introduced a new feature called SmartScreen in Windows 8, which should prevent the execution of malicious software. Unfortunately, SmartScreen also interrupts the installation of software unknown to SmartScreen. If SmartScreen is activated on the computer, a message might appear during the installation of WSAPM.



In this case, this means that the latest version of WSAPM is unknown to SmartScreen. The installer of WSAPM does not contain any malicious software or adware. After a click on *More info* the installation can be continued with the button *Run anyway*.

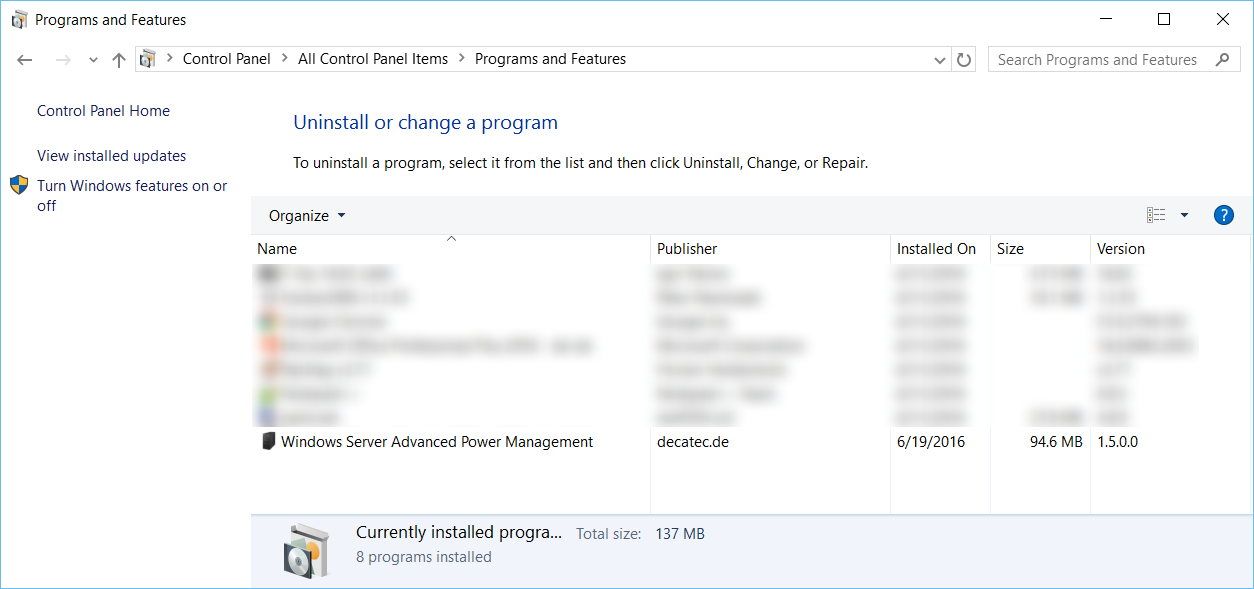


You can find more information on SmartScreen in [Wikipedia](http://en.wikipedia.org/wiki/Microsoft_SmartScreen).

# Uninstall

During uninstall, the program will be removed completely from the system, i.e. the configuration will also be deleted and the exception in the Windows Firewall is removed.

To uninstall WSAPM, start the Windows control panel and chose *Programs and Features*. Double click on the entry *Windows Server Advanced Power Management*.



# Update

When a new version of Windows Server Advanced Power Management is available, it can be installed over the currently installed version (no uninstall needed).

The configuration will not be removed and will stay intact.

# Usage

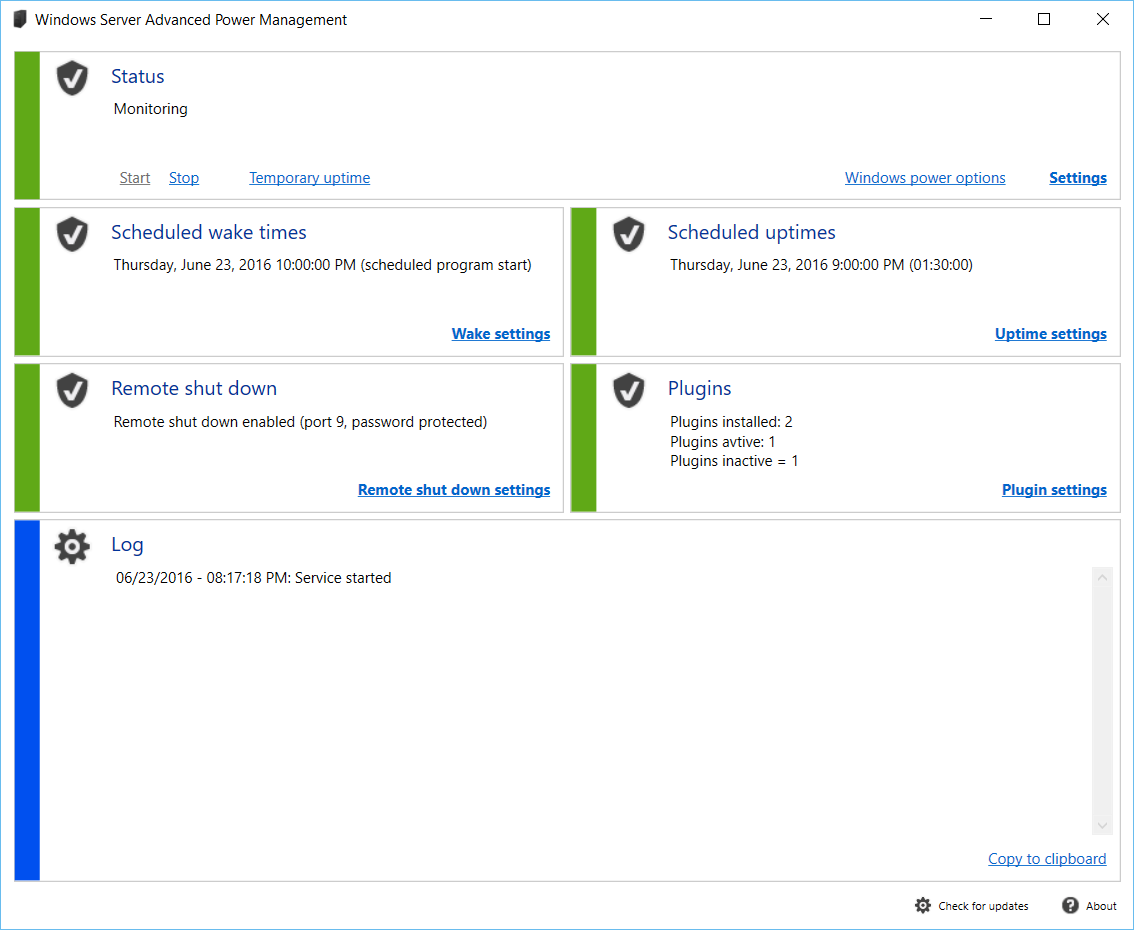
Windows Server Advanced Power Management is separated into two components:

* The program Windows Server Advanced Power Management: With this program the behavior of WSAPM can be changed and adjusted. It can be started by using the shortcut on the desktop or by using the Windows start menu (*All programs* 🡪 *Windows Server Advanced Power Management* 🡪 *Windows Server Advanced Power Management*).  
  Important: This program does not have to run continuously to check the specified policies and suppress standby as appropriate. It only serves the purpose to specify the settings of WSAPM and to provide some additional info (e.g. log entries).
* The Windows Server Advanced Power Management service: This Windows service applies the monitoring and executes all relevant actions. The service is automatically configured during installation and requires no user interaction.

The following description refers to the program Windows Server Advanced Power Management, the Windows service and its settings should not be altered after the installation of the program.

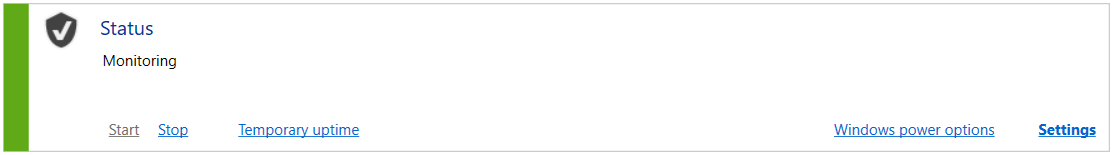
## Main window

After the start of the program you will see the main window:

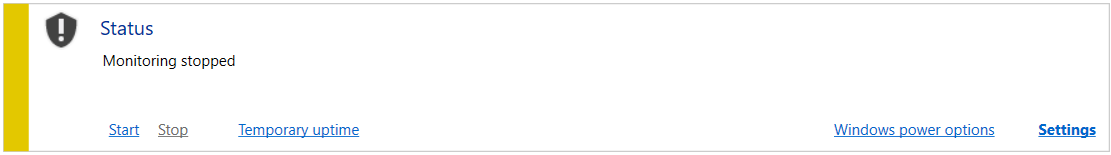


**Status panel *Monitoring***

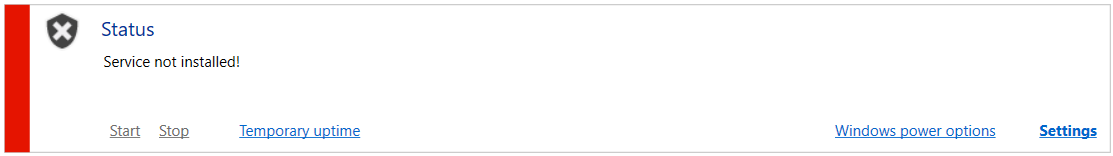
The status panel *Monitoring* displays the status of the monitoring at a glance. This status is defined by the WSAPM service. If the service is running, the status will be *Monitoring*.



If the service is not running or there are other errors, the status will be *Stopped*:

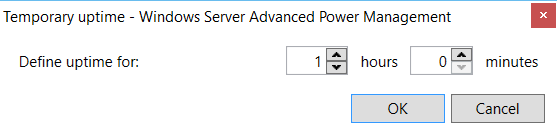


If the service was not installed correctly, this will also be shown in the status panel:



Monitoring can be activated or deactivated manually by the links *Start* and *Stop*. In this case the underlying Windows service is started or stopped.

A temporary uptime can be set by using the corresponding link. During this time span, the computer will not enter standby mode, no matter what other policies are active:



This temporary uptime even persists a restart of the PC and can only be disabled by using the link *Cancel temporary uptime*. The main window will indicate if an uptime is currently defined. Besides a temporary uptime, a planned uptime can be set in the settings (see [*Settings - Uptime*](#_Plugins)).

The link *Settings* opens the configuration dialog (see [*Settings*](#_Settings_1)).

The link *Windows power options* opens the Windows power options. This might be useful in many cases, because WSAPM works tightly together with the Windows power options.

**Status panel *Scheduled wake***

The status panel *Scheduled Wake* shows the next target dates for automatic wake from standby scheduled by WSAPM. If programs should be started when the system is woken, this is indicated by the entry *planned program start*.

The link *Wake settings* opens the configuration dialog for wake settings (see [*Settings - Wake*](#_Wake_1)).

**Status panel *Remote shut down***

The status panel Remote shut down shows if remote shut down is enabled or disabled.

The link *Remote shut down settings* opens the configuration dialog for remote shut down settings (see [*Settings - Remote shut down*](#_Remote_shut_down_1)).

**Status panel *Scheduled uptimes***

This status panel lists time and duration for the next scheduled uptimes.

The link *Uptime settings* opens the configuration dialog for scheduled uptimes (see [*Settings - Uptime*](#_Plugins)).

**Status panel *Plugins***

The status panel Plugins shows the status of currently installed plugins.

Use the link *Plugin settings* in order to open the configuration dialog and to quickly install/uninstall or activate/deactivate plugins (see [*Settings - Plugins*](#_Plugins_1)).

**Status panel *Log***

The log status panel shows all log entries generated by WSAPM, as long the logging option has been enabled in the options (see [*Settings - General*](#_General)). The log entries visualize the state of WSAPM, e.g. when standby was suppressed or when the computer entered standby mode.

You can copy parts of the log by simply selecting some text and copy it to the clipboard. Furthermore, the link *Copy to clipboard* copies the whole log into the clipboard.

**Check for updates**

You can check for a new version of WSAPM by clicking this button. If an update is available, it can be downloaded and installed.

**About**

More information about Windows Server Advanced Power Management (like version information, change log, etc.) can be shown by clicking the button *About*.

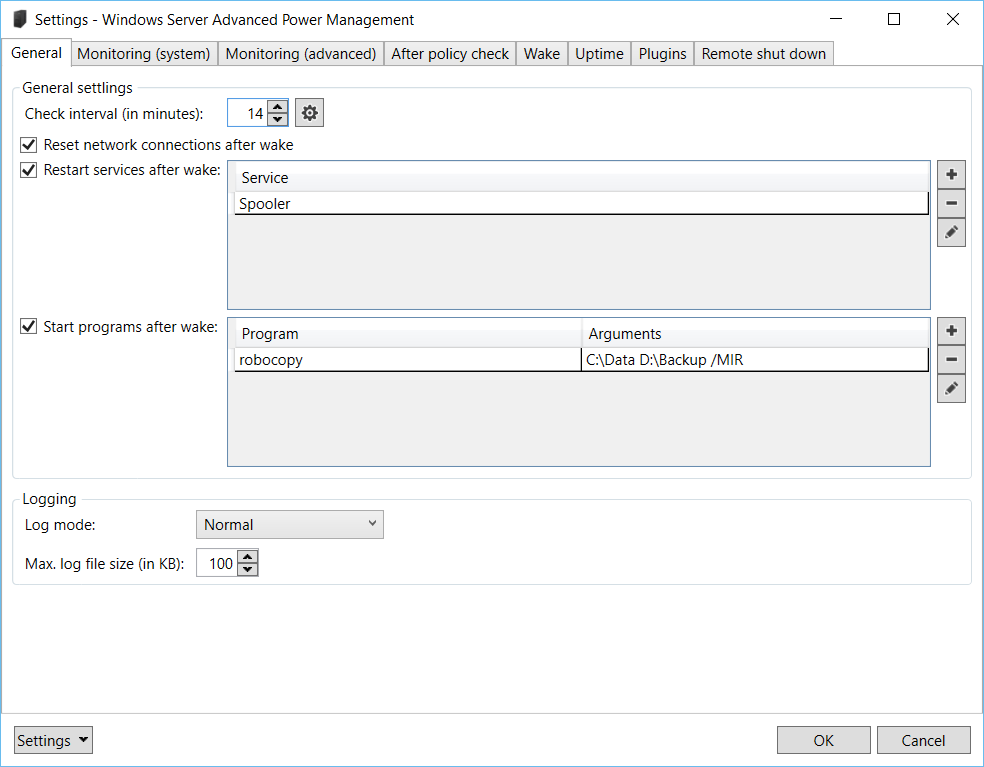
## Settings

With the settings, Windows Server Advanced Power Management can be tailored to the specific needs.

The settings dialog can be accessed by the link *Settings* in the main window.

### General

The tab *General* provides general options for WSAPM.



**Check interval (in minutes)**

Defines the interval of monitoring in minutes, i.e. WSAPM checks all policies periodically every time after this interval elapses. When one or more policies are satisfied that signal that the computer is still in use, standby gets suppressed until the next check.

Important: Because WSAPM works together with the Windows power options, this interval has to be set according to the idle interval in the Windows power options. For example, when the Windows power options are set that the system goes to standby after 15 minutes, the option in WSAPM has to be set to an interval smaller 15 minutes. Only when this option is set this way, WSAPM can check the defined policies before the system goes to standby automatically. Thus when the idle time setting in the power option is changed afterwards, the check interval of WSAPM has to be adjusted accordingly.

With the button next to the input field the optimal value for the check interval can be chosen. This will always be the corresponding Windows option less one minute.

**Reset network connections after wake**

When this option is enabled, all active network connections are reset after wake (i.e. re-initialized). This option is especially useful if you encounter network problems after the computer wakes from standby.

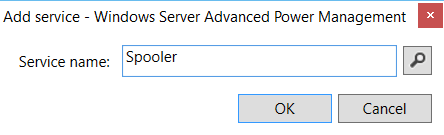
**Restart services after wake**

Here you can specify Windows services which should be restated after the computer wakes from standby. These services are restarted every time the computer wakes from standby, regardless if the computer was woken by a scheduled wake of WSAPM, or woken by any other program or user input.  
This function is particularly useful when some services do not work correctly after the computer is resumes from standby.   
The services are restarted approximately 30 seconds after the computer was woken.  
When more than one service is specified here, the sequence of restarting the services is not set.

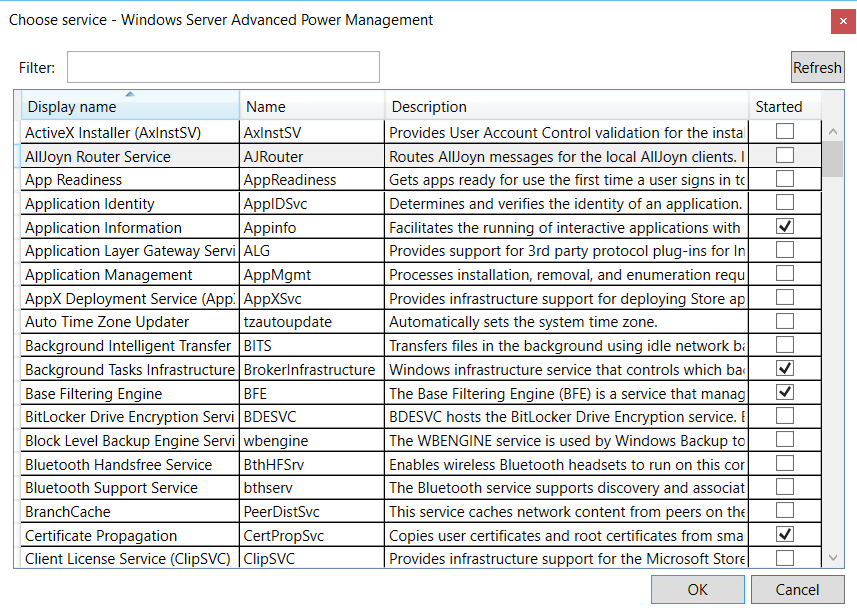
Add a service by clicking the plus button.  
Remove a selected service by clicking the minus button.  
By double clicking or clicking the edit button, the selected service edited.

**Window *Add service***

When a service is added or edited, a window is shown where you can enter the name of the service. This can either be the service name (e.g. “Spooler”) or the display name of the service (in this example this would be “Print Spooler”).



With the button next to the input field you can choose a service from the currently installed services.



**Start programs after wake**

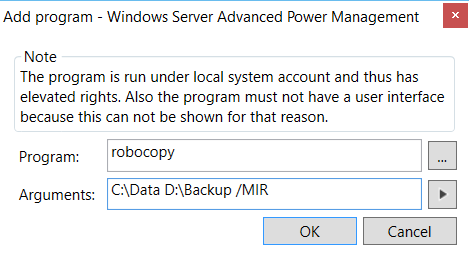
Here you can specify programs (and optional arguments) which should be started when the computer wakes from standby. These programs are started every time the computer wakes from standby, regardless if the computer was woken by a scheduled wake of WSAPM, or woken by any other program or user input.   
If you want to start any program only if the computer was woken by a scheduled wake of WSAPM, use the option *Start programs after wake* in the *Wake* tab (see [*Settings - Wake*](#_Wake_1)).  
The programs are started approximately 30 seconds after the computer was woken.   
When more than one program is specified here, the sequence of starting the programs is not set.

Important: The programs specified are run under the local system account and thus are executed with elevated rights.  
Also these programs must not show a user interface because it is not possible to show a UI in this context. That means that programs with a UI can be started, but are not shown to the currently logged on user.

Add a program by clicking the plus button.  
Remove a selected program by clicking the minus button.  
By double clicking or clicking the edit button, the selected program and its arguments can be edited.

**Window *Add program***

When a program is added or edited, a window is shown where you can enter the path to the program to be executed as well as optional arguments to start the program with. You can test the start of the program and its arguments by clicking the play button.



**Log mode**

Defines the log’s level of detail. Following options can be set:

* *None*: No log entries will be written.
* *Only Errors*: Only errors will appear in the log.
* *Normal*: All relevant information will be logged.
* *Verbose*: Detailed log, all actions by WSAPM will be logged. This option is only recommended in situations where WSAPM does not seem to work properly.

When logging is active (i.e. not set to *None*), a log file will automatically be written and its contents will be shown in the L*og* status panel. When the log mode is set to *None*, no further log entries will be written, but the log file is not deleted automatically.

**Max. log file size (in KB)**

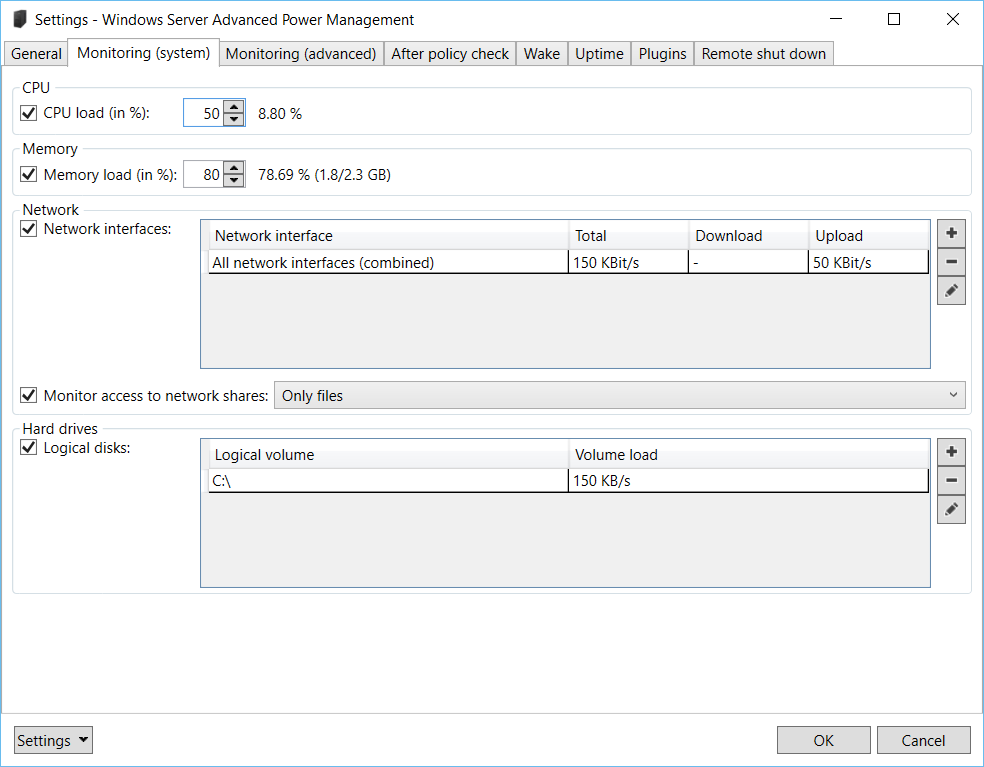
With this option a maximum file size (in KB) of the log file can be specified. When it is set to 0, the log file can grow without limits. When a value greater 0 is specified, the log file gets deleted and new file is created when the set size is exceeded.

**Settings**

With this button, the settings of WSAPM can be reset to the default settings. You can also export or import the current settings.

### Monitoring (system)

In the tab *Monitoring (system)* the policies for standby suppression can be specified, which check specific system parameters. All the defined policies are checked after the check interval elapses (see [*Settings - General*](#_General)). If at least one policy is satisfied, standby will be suppressed.



**CPU load (in %)**

Specifies the CPU load in percent which signals that the system is currently in use.

The current CPU load is shown next to the input field.

**Memory load (in %)**

Defines the memory load in percent which prevents the computer from switching to standby mode.

The current memory load in percent and memory currently in use/total memory (in GB) is also shown.

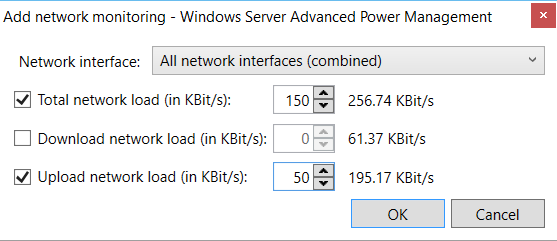
**Network interfaces**

Monitoring of network interfaces can be specified in this area. The parameters can be set for every single network interface or for all network interfaces altogether. This is specifically useful for server systems with more network adapters.

Add a network monitoring by clicking the plus button.  
Remove a monitoring by using the minus button.  
By double clicking or using the edit button, the selected network monitoring can be edited.

**Window *Add network monitoring***

When a monitoring is added, you first have to specify the network interface which should be monitored. When choosing *All network interfaces (combined)*, the load values are added in order to calculate a total load over all interfaces.



The following parameters can be monitored:

* Download network load (in KBit/s)
* Upload network load (in KBit/s)
* Total network load (upload + download) (in KBit/s) – for example, if a download with 100 KBit/s and an upload with 50 KBit/s are currently active, the total network load will be 150 KBit/s.

When the specific load is greater than the load specified, the computer will not enter standby mode.

Next to the input fields, the current load (upload/download/combined) is shown.

**Monitor access to network shares**

If this option is active, the system will not enter standby mode when there is currently access to files on a network share. This is particularly useful when there a network share is infrequently accessed by clients.

The following types of access can be specified here:

* *Only files*: Only access to files are monitored, e.g. when a client opened a Word document on the server’s share.
* *Only directories*: Only access to directories are monitored, e.g. when a client opens a folder on the server’s share.
* *Files and directories*: Both access to files and folders are monitored.

Important: Access to shared directories occurs quite often and in the background on a network. Also keep in mind that closing a shared folder on the client side does not release the access to this folder on the server side immediately. Therefore, monitoring access to shared directories might prevent the server from going to standby as expected. Thus you should take the option to monitor access to shared directories with care.

**Logical disks**

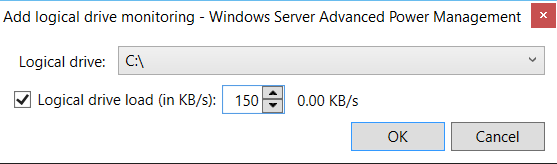
In this area, the monitoring of single logical disks (volumes), or a combined load (for all logical disks) can be specified.

Add a monitoring by clicking the plus button.  
Remove the selected volume by clicking the minus button.  
By double clicking or clicking the edit button, the selected monitoring can be edited.

**Window *Add logical drive monitoring***

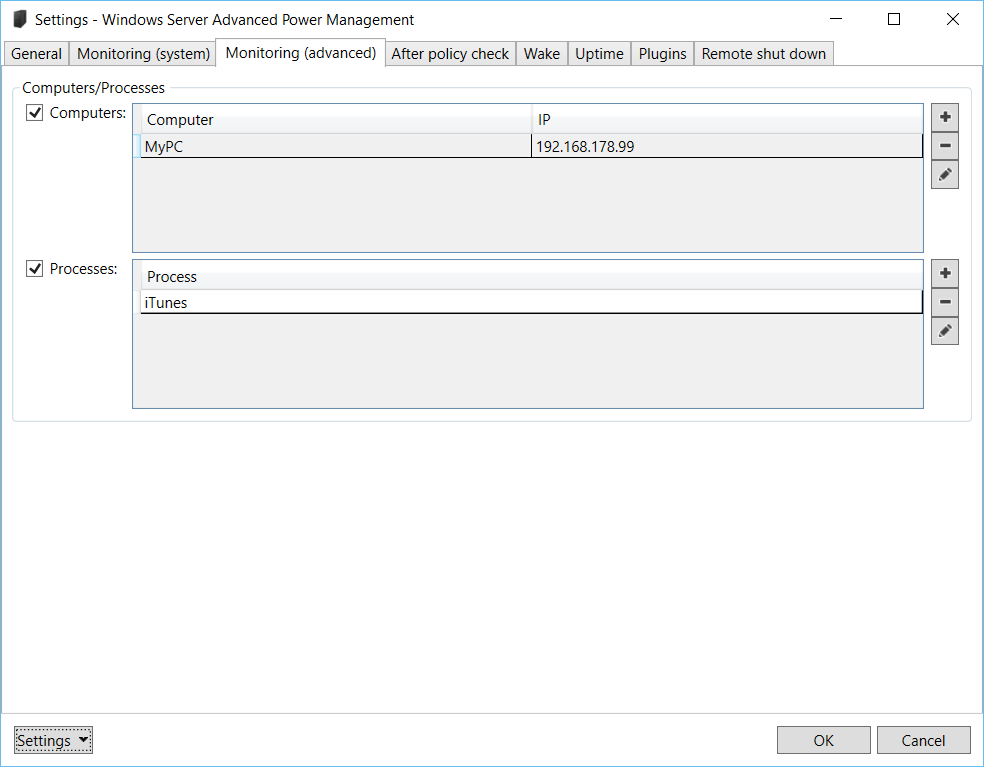
A logical drive and the drive’s load (in KB/s) can be specified in this dialog. When the option *All drives* is chosen, the load values of all logical volumes are summed up when the policies are checked.

The current load of the selected volume is also shown.



### Monitoring (advanced)

In this tab, advanced policies for standby suppression can be specified.



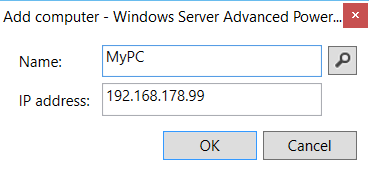
**Computers**

Computers or other network devices can be specified here. When WSAPM is checking its policies and at least one of the network devices is reachable over the network, standby gets suspended. The list is not limited to computers. You can also add all other devices which are connected to your network, e.g. smart phones, network-compatible TVs, streaming clients, gaming stations, etc.

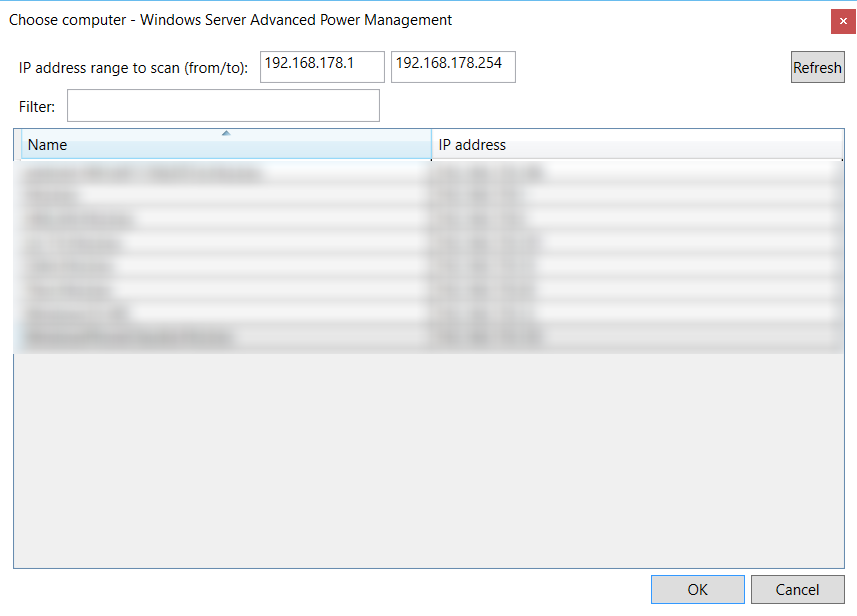
Add computers by clicking the plus button.  
Remove a network device by clicking the minus button.  
By double clicking or clicking the edit button, the selected network device can be edited.

**Window *Add computer***

When a computer is added or edited, a window is shown where you can enter a name by which the device is accessible over the network (e.g. “MyComputer”) and/or an IP address.



With the select button you can search for active devices in the local network. The IP address range which is scanned by default depends on the IP address of the local computer. When the local computer has the IP address 192.168.178.**20**, the range from 192.168.178.**1** to 192.168.178.**254** will be scanned by default. You can change the range which will be scanned in the dialog.  
Attention: Scanning of large IP address ranges might last a long time.



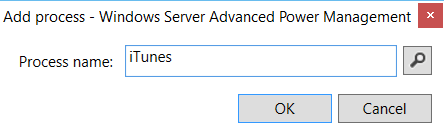
**Processes**

All processes specified here also signal that the system is currently in use.

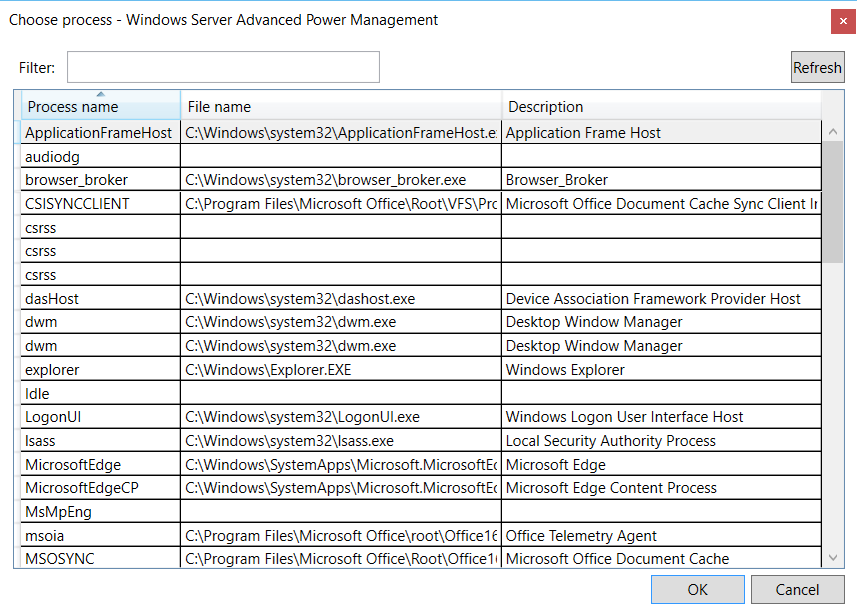
Add a process by clicking the plus button.   
Use the minus button to remove a process from the list.  
By double clicking or clicking the edit button, the selected process can be edited.

**Window *Add process***

When a process is added or edited, a window is shown where you can enter a process name without extension (e.g. “.exe”). You can use the process names as shown in the Windows task manager. For example, if you want to add iTunes to the list, just enter “iTunes” (not “iTunes.exe”).

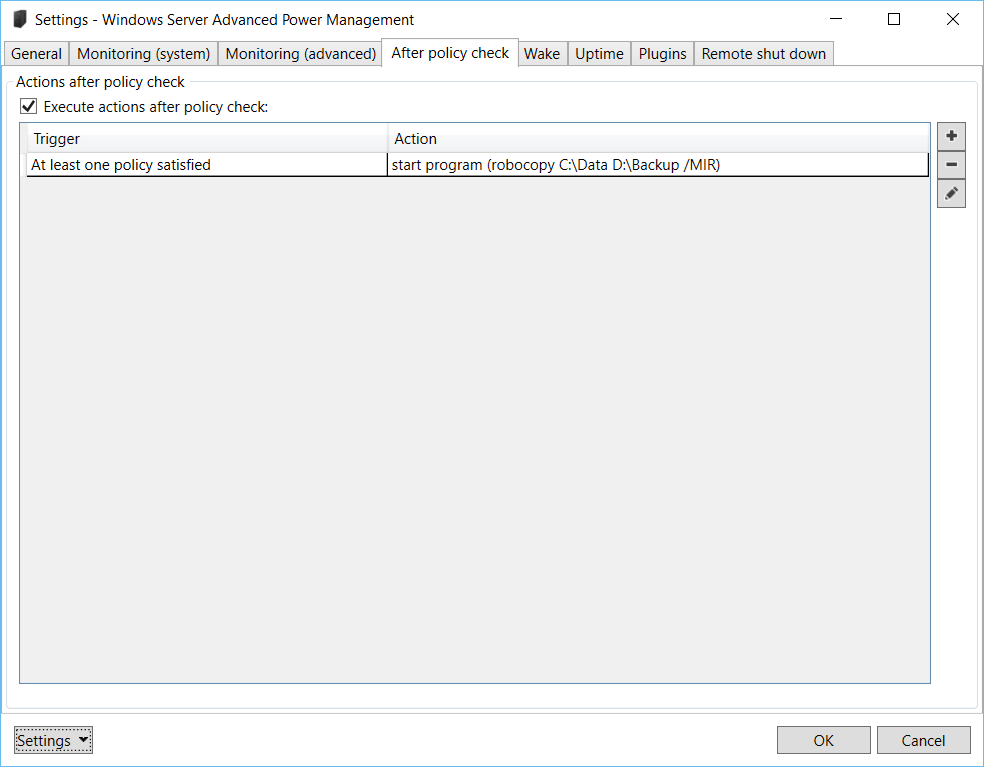


With the button next to the input field you can choose a process from the list of currently running processes.



### After policy check

In the tab *After policy check*, user-defined actions can be specified, which should be executed after the policies of the tab *Monitoring* were checked.



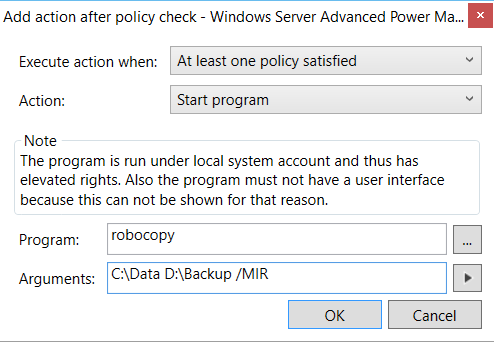
**Execute actions after policy check**

This option specifies if the actions after policy check should be executed. If this option is disabled, the specified actions are never executed.

Add an action by clicking the plus button.  
Remove an action by clicking the minus button.  
To edit an action, use the edit button or double click on an entry.

**Window *Add action after policy check***

When an action is added or edited, a window is shown where the options for the action can be specified.



**Execute action when**

You can find two options here: *No policy satisfied* and *At least one policy satisfied*. The first one specifies that the action should be executed when at least one policy is satisfied, which would enable the computer’s standby mode. When the second option is chosen, the action will be executed when no policy is satisfied.

**Action**

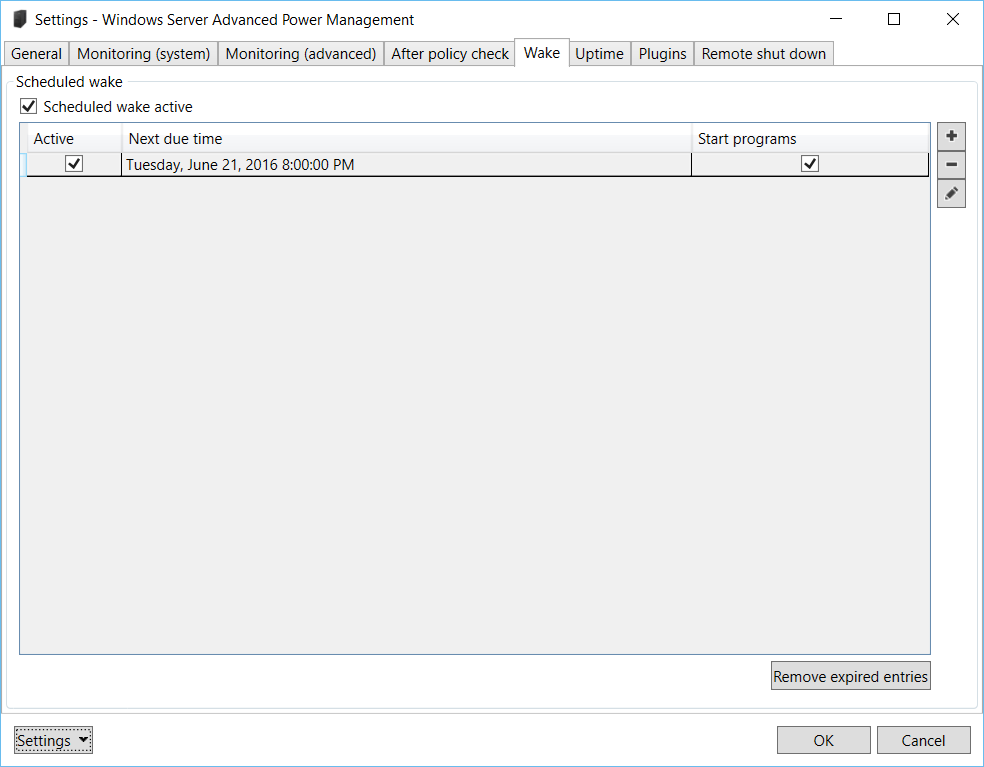
The action to be executed is specified here:

* *Standby*: The computer will enter standby mode.
* *Hibernate*: The computer will enter hibernate mode.
* *Shut down*: The computer will shut down.
* *Start program*: The program with the specified arguments will be started.

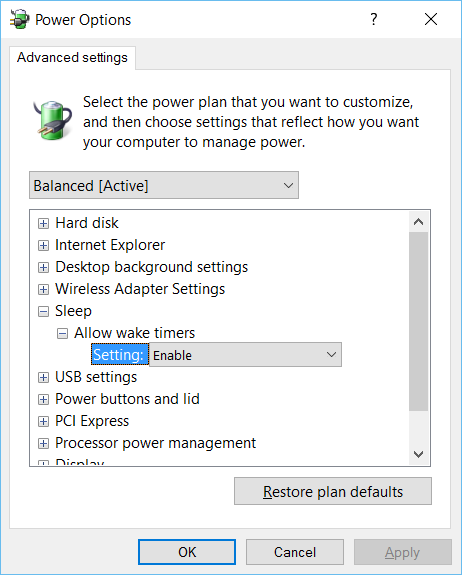
**Hint:** For one trigger (*No policy satisfied*/*At least one policy satisfied*) only one action can be defined which changes the energy state of the computer (*Standby*/*Hibernate*/*Shut down*), because these actions are mutually exclusive for a trigger.

### Wake

In the tab *Wake* the options for time-controlled wake from standby can be specified.



Important: So that the computer can be woken from standby, the option *Allow wake timers* has to be enabled in the Windows power options. You can find it under *Control panel* 🡪 *Power Options* 🡪 *Change plan settings* (for the current plan) 🡪 *Change advanced power settings* 🡪 *Sleep* 🡪 *Allow wake timers*.



**Scheduled wake active**

When this option is enabled, the system will be woken from standby according to the wake schedules specified.

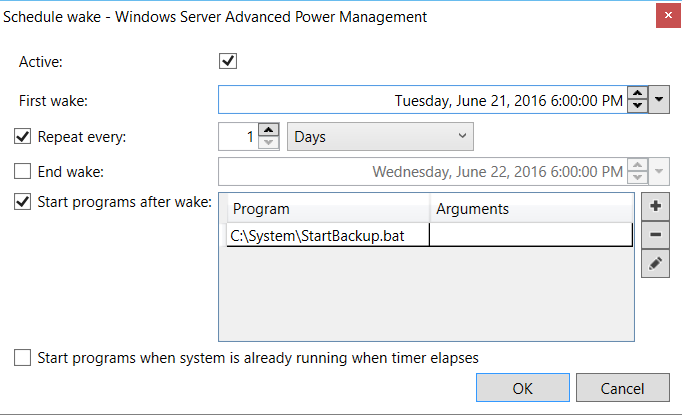
Add a wake schedule by clicking the plus button.  
Remove a wake schedule by clicking the minus button.  
By double clicking or clicking the edit button, the selected wake schedule can be edited.

**Remove expired entries**

Use this button to remove all expired entries, i.e. entries which will never trigger a wake from standby (e.g. because the last date of a scheduled wake is in the past).

**Window *Schedule wake***

When a wake schedule is added or edited, a window is shown where you can enter several options for a time-controlled wake.



**Active**

This option indicates if the wake schedule should be active. If a wake schedule is deactivated, the computer will not be woken according to the schedule.

**First wake**

The first wake time specifies the time the computer should wake automatically from standby.

**Repeat every**

When this option is enabled, an interval can be defined which refers to the start time specified above. The system will be woken from standby every time this interval elapses.

As an example: When the start time is set to 1st January 2014 01:00 PM and should be repeated every day, the system will be woken from standby every day at 01:00 PM, starting on the 1st January 2014.

By disabling this option, the system will be woken only once at the time specified by *First wake*.

**End wake**

You can specify a time when the schedule for automatic wake should end. After this point of time the system will not be woken by WSAPM anymore.

**Start programs after wake**

Here you can specify programs (and optional arguments) which should be started when the computer wakes due to a scheduled wake of WSAPM. These programs are started only when the computer if woken by a scheduled wake of WSAPM. If you want to start any program every time the computer wakes from standby, use the function *Start programs after wake* in the *General* tab (see [*Settings - General*](#_General)).  
The programs are started approximately 30 seconds after the computer was woken.   
When more than one program is specified here, the sequence of starting the programs is not set.

Important: The programs specified are run under the local system account and thus are executed with elevated rights.  
Also these programs must not show a user interface because it is not possible to show a UI in this context. That means that programs with a UI can be started, but are not shown to the currently logged on user.

Add a program by clicking the plus button which will show a window where you can enter the path to the program to be executed as well as optional arguments to start the program with. You can test the start of the program and its arguments by clicking the play button.

Remove a selected program by clicking the minus button.

By double clicking or clicking the edit button, the selected program and its arguments can be edited.

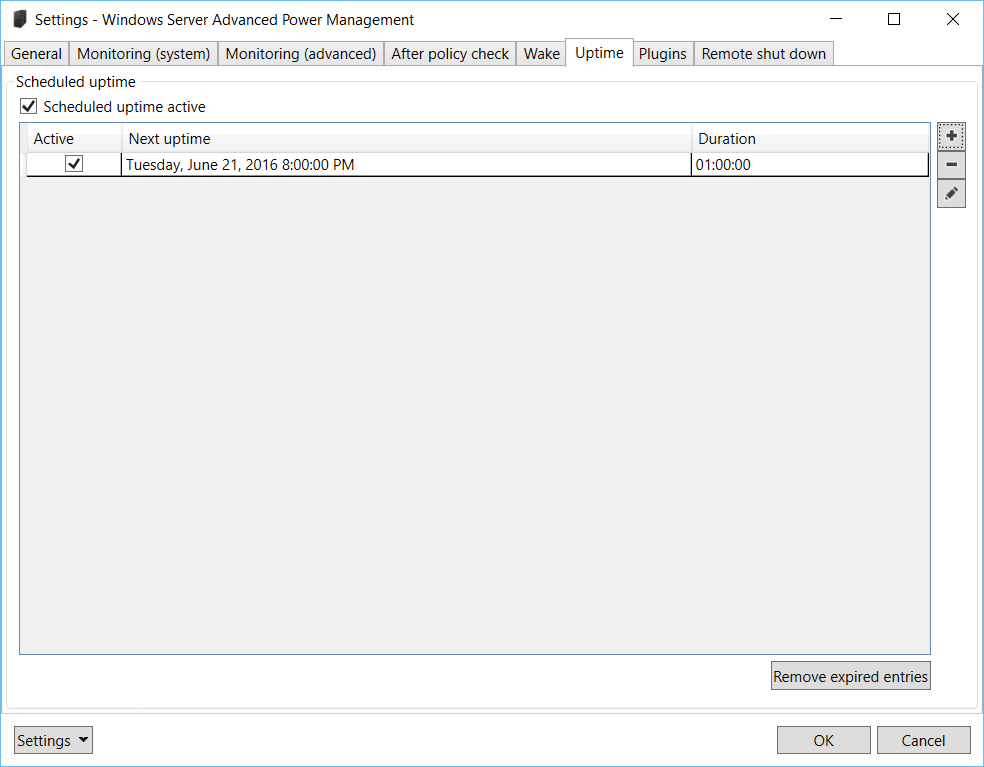
**Start programs when system is already running when timer elapses**

If this option is enabled, the specified programs are also started if the system is already running when the planned wake takes place. Disable this option if the programs should only be started when the system is woken from standby at the specified time.

### Uptime

In this tab, scheduled uptimes can be defined. During the timespan of the uptime(s), the computer will not enter standby mode, no matter of the other policies defined.

**Important:** When an uptime is defined, the computer will not be woken automatically. The uptime only gets active when the computer currently is running. Therefor, when the computer should be woken automatically for an upcoming uptime, this has to be configured in the wake settings (see [*Settings - Wake*](#_Wake_1)).



**Scheduled uptime active**

This option specifies if the uptimes specified are active. If this option is disabled, the specified uptimes are ignored.

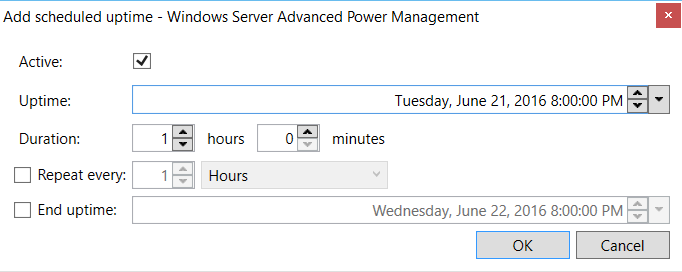
Add an uptime by clicking the plus button.  
Remove an entry by clicking the minus button.  
To edit a scheduled uptime, use the edit button or double click on an entry.

**Remove expired entries**

Use this button to remove all expired entries, i.e. entries which will never trigger a scheduled uptime (e.g. because the last date of a scheduled uptime is in the past).

**Window *Schedule Uptime***

When an uptime schedule is added or edited, a window is shown where you can enter several options for this uptime.



**Active**

Activates or deactivates the scheduled uptime. If an uptime schedule is deactivated, this uptime is ignored.

**Uptime**

Defines the date and time when the uptime should be scheduled.

**Duration**

This is the duration of the scheduled uptime and defines the timespan when the computer should not enter standby mode no matter what other policies are defined.

**Repeat every**

When this option is enabled, an interval can be defined which refers to the uptime’s start time specified above. The system will not enter standby for the duration specified every time this interval elapses.

As an example: When uptime is scheduled for 1st January 2014 01:00 PM for a duration for 1 hour and should be repeated every day, the system will be active every day between 01:00 PM and 02:00 PM, starting on the 1st January 2014.

By disabling this option, the uptime will only appear once at the time specified by *Uptime*.

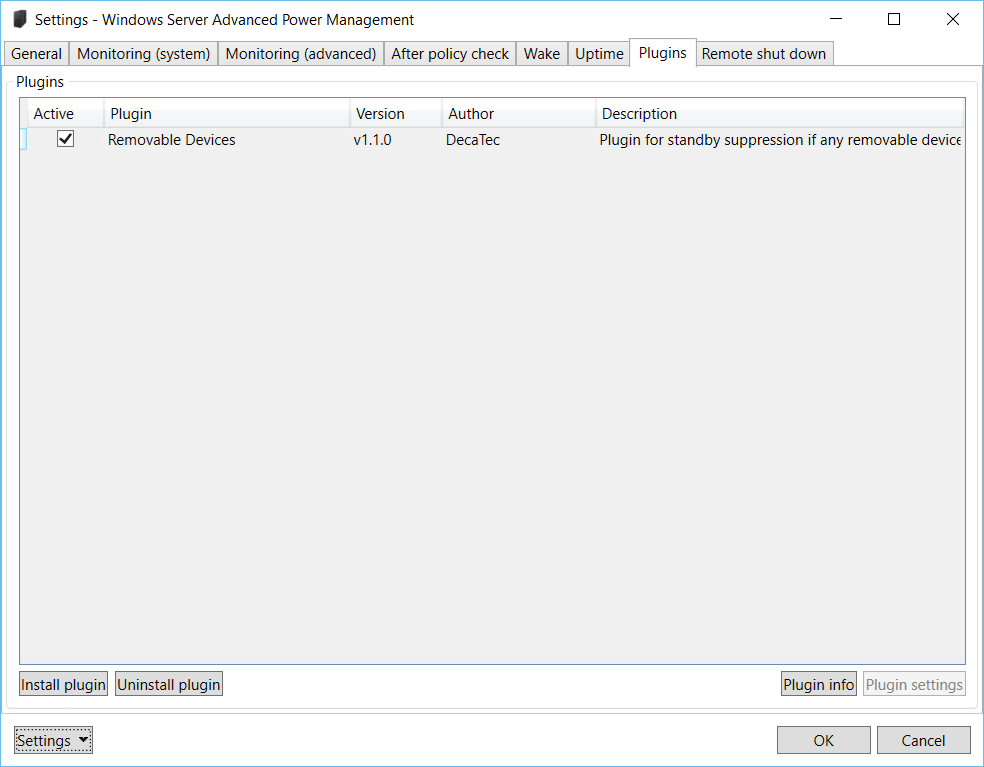
**End uptime**

You can specify a date and time when the uptime schedule should end. After this time, this specific uptime schedule will be ignored.

### Plugins

WSAPM offers a plugin interface. With plugins the functionality of WSAPM can be extended with further monitoring policies.

The plugins can be managed in the tab *Plugins*.



A detailed description of the plugin interface for developers can be found in the section [*Plugin development*](#_Plugin_development).

**List of plugins**

The list shows the currently installed plugins. To activate a plugin, i.e. that the policies of the plugin are used for monitoring, the corresponding check box has to be activated.

**Install plugin**

A new plugin can be installed with the button *Install plugin*. Choose the plugin to install in the following dialog. Plugins for WSAPM are shipped as Zip files, so only these files can be selected. During plugin installation WSAPM has to be restarted.  
Don’t forget to enter the settings again after installing a plugin and activate this plugin so that its policies can be checked.

**Uninstall plugin**

You can uninstall a plugin with the button *Uninstall plugin*. Again, WSAPM has to be restarted to uninstall a plugin.

If a plugin cannot be uninstalled with this button, there is as workaround to uninstall a plugin manually: First you have to stop the Windows Server Advanced Power Management service. This can easily be done with the link *Stop* on the main screen of Windows Server Advanced Power Management (see [*Usage - Main window*](#_Main_window)). The application has also to be closed. When the service is stopped, go to the folder where the plugins of WSAPM are installed (usually *C:\Program Files (x86)\Windows Server Advanced Power Management\Plugins*) and delete the folder of the plugin to uninstall. Afterwards, if the plugin had custom settings, these settings can also be deleted. Just go to the folder where WSAPM saves the plugin settings (*C:\ProgramData\Windows Server Advanced Power Management\Plugins*) and delete the folder of the plugin to uninstall. When this is done, the service can be started again.

**Plugin info**

Click this button to show further information for the selected plugin.

**Plugin settings**

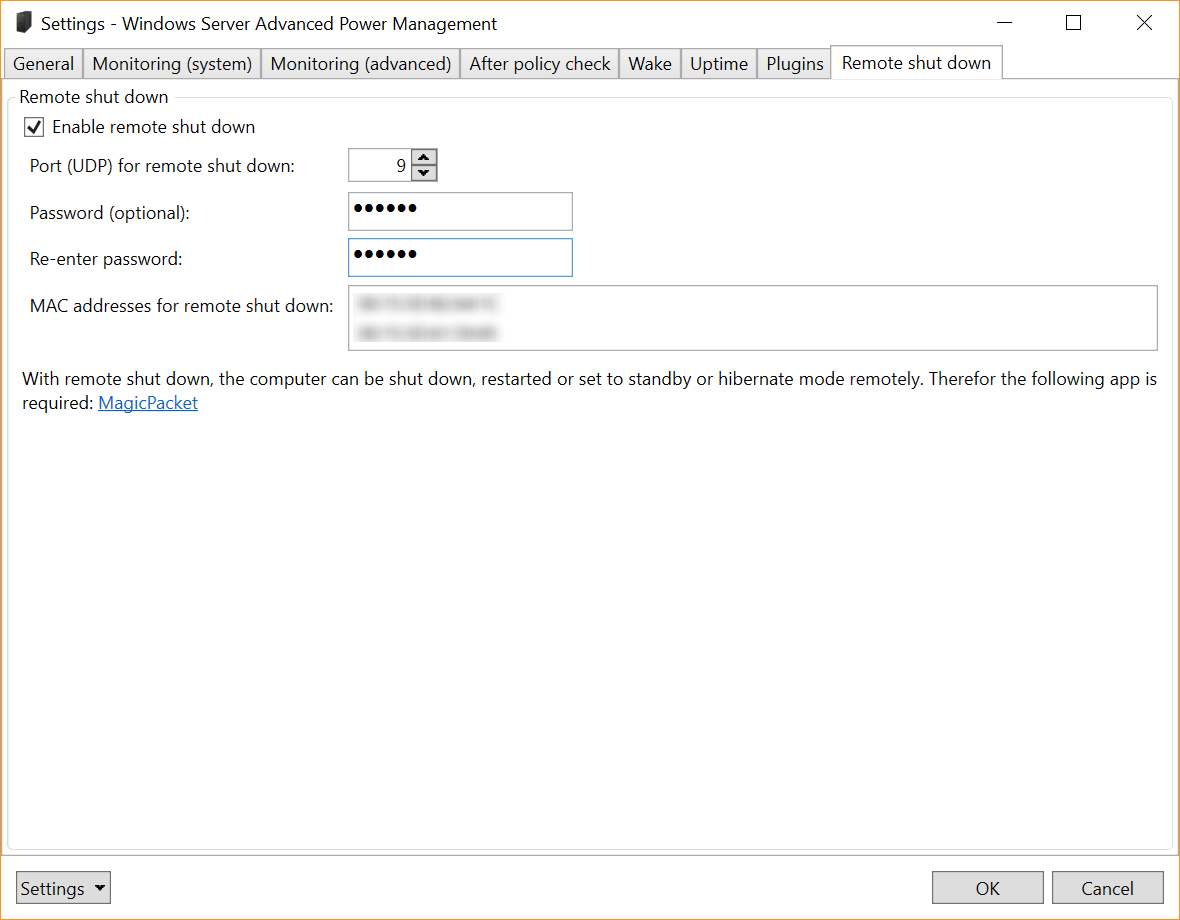
Some plugins for WSAPM offer settings. In that case, these settings can be changed with the button *Plugin settings*.  
If the selected plugin does not offer own settings, this button is disabled.

### Remote shut down

This tab offers the options regarding remote shut down. With remote shut down, the Computer can be shut down, restarted or sent to standby or hibernate mode remotely.

In order to use this feature, the *Remote shut down* option has to be enabled and you have to use a corresponding app or program that can send the remote shutdown commands – e.g. [MagicPaket](https://decatec.de/software/magicpacket_en/) (available for Windows and Windows Phone). A detailed description of how to implement remote shut down with Windows Server Advanced Power Management and MagicPacket can be found in [this article](https://decatec.de/remote-shutdown-waspm-magicpacket_en/).

*Hint for developers:* The protocol used for remote shut down with WSAPM is disclosed (see [*Windows Server Advanced Power Management for developers - Remote shut down*](#_Remote_shut_down)). Therefore, every developer can implement this functionality in own programs or apps.



**Enable remote shutdown**

If this option is enabled, WSAPM listens to remote shut down commands on the network (UDP). Disable this option if you do not want to shut down, restart or sent you computer to standby or hibernate mode remotely.

**Port (UDP) for remote shut down**

Here you can choose the port to be used for remote shut down (default is port 9). The same port has to be specified in the app or program sending the remote shut down commands.

**Password**

Here you can enter a password which then also has to be specified in the app or program sending the remote shut down commands. The password is optional, i.e. just leave the fields empty if you do not want to use a password for remote shut down.

Important: The password is not a real security feature as it is sent over the network as only as hash value (and not encrypted). It is highly recommended that you do not re-use any passwords you are already using for other services.

**MAC addresses for remote shut down**

The list shows the MAC addresses of all active network interfaces. The computer will respond to remote shut down commands sent to these MAC addresses.   
You can copy a MAC address to the clipboard by the context menu of this list.

# Example scenarios

In this section are some example scenarios and the corresponding configuration of Windows Server Advanced Power Management.

**Wake server for planned activities from standby**

Some programs like virus scanners can be configured to plan periodical activities, e.g. scan the system for viruses every day at 2:00 PM. On the other hand, many of these programs are not able to wake the system from standby automatically. Thus these activities are only executed if the system is running exactly at this time, or not at all. WSAPM can help you to cope with situations like these: First, the program which executes planned activities has to be added to the list of processes (see [*Settings - Monitoring (advanced)*](#_Monitoring_(advanced))). In this example it would be “avscan” ([Avira’s](http://www.avira.com/) virus scanner). Next, planned wake from standby has to be configured in WSAPM (see [*Settings - Wake*](#_Wake_1)). This should be planned a few minutes before the planned action takes place in order that the system can completely wake from standby. 1:55 PM would be a good starting point here. Now the system will wake from standby every day at 1:55 PM. At 2:00 PM the virus scan will start. This will be recognized by WSAPM and the standby gets suspended. When the scan process ends, the system will enter standby considering the system policies.

**Suppress standby while server is streaming**

When streaming software like [Twonky](http://twonky.com/) or [AssetUPnP](http://www.dbpoweramp.com/asset-upnp-dlna.htm) is installed on a server, media content can be streamed to streaming clients (e.g. PlayStation). There is no actual option to detect active streaming in WSAPM. Even the option *Monitor access to network share* will not be applicable, because technically streaming is no regular network access to shared folders. Simply said, streaming is just the playback of media contents over a network. Therefore, streaming can be seen as simple network upload at the streaming server. To determine the correct network load value in KBit/s, streaming should be currently active. Then add a new network monitor in the settings. Here you should see the current upload bandwidth. This value can be set in the corresponding input field (see [*Settings - Monitoring (system)*](#_Monitoring)).  
Keep in mind that streaming of HD content (like movies) needs much more bandwidth than streaming of simple (and smaller) MP3 files. Because of this, you should determine the correct value for upload network load while streaming smaller files.

**Manual suppression of standby on demand**

When a manual suppression of standby on demand is required, a program which is currently not used (like Notepad or WSAPM itself) can be added to the list of processes (see [*Settings - Monitoring (advanced)*](#_Monitoring_(advanced))). When the system should not enter standby, you only have to start the specified program and the standby will be suppressed. When the computer should switch back to the normal energy policies, just close this application.

**Manage the state of the computer (standby/hibernate) with WSAPM**

Windows Server Advanced Power Management was primary developed to work with the Windows power options – and not to manage the state of the computer by itself. But there was the request to support this feature: WSAPM should manage the state of the computer by itself.  
Such a configuration is possible with the options *After policy check* in the settings (see [*Settings - After policy check*](#_Wake)). For example, if the target state of a computer is hibernation, the settings can be configured so that the computer will enter hibernation when no policy is satisfied. In this case, the Windows power options can be configured so that the computer will not enter standby.

However, it is recommended not to force the state of the computer in such manner. The better solution is still to work together with the Windows power options and only suppress standby when applicable.

# Troubleshooting

This section provides some clues for troubleshooting if you ran into problems with Windows Server Advanced Power Management.

**WASPM does not seem to work properly**

When WSAPM does not seem to work properly, you should set the log mode to *Verbose* (see [*Settings - General*](#_General)) in order to see all of WSAPM’s activities. Maybe the reason for the problems can be tacked by detailed logs.

On Windows 7 and later, there is a possibility to check the functionality of WSAPM by the tool *powercfg*. Therefore, you have to start the command line with administrator rights and enter following commands:

* *powercfg -requests*: Shows all power requests of programs which currently prevent the computer to enter standby. When a policy of WSAPM is satisfied, an entry of WSAPM should be listed.
* *powercfg -waketimers*: Shows the currently active wake timers on the system. When a wake timer was set by WSAPM, it should be listed.

On Windows Vista or earlier Windows versions, the tool does not support both of these parameters, thus the functionality of WSAPM cannot be inspected this way.

**WSAPM shows error messages (“Category does not exist”) when checking network or CPU/memory/network/logical disk load or cannot find available network interfaces/logical disks**

WSAPM often uses WMI ([Windows Management Instrumentation](http://en.wikipedia.org/wiki/Windows_Management_Instrumentation)) to determine values important for checking the policies (e.g. CPU or memory load, etc.). If errors occur in this context (“Category does not exist”) or the available network interfaces/logical disks cannot be identified, the Windows performance counters are possibly broken. This can be due to other programs changing the performance counters in an unusual way.

These performance counters can be repaired by Windows automatically. This is done with the command prompt (started as administrator). You can try following possibilities:

1. This command restores the performance counters from a backup:   
   *cd c:\windows\system32  
   lodctr /R:PerfStringBackup.ini*The computer should be restarted afterwards.
2. If the problem still exists, the following command restores the performance counters completely:   
   *cd c:\windows\system32  
   lodctr /R*Again, the computer should be restarted after executing the command.
3. If that does not solve the problem, follow the steps in this [Microsoft Knowledge Base article](http://support.microsoft.com/kb/300956).

After following these steps, WSAPM should work as expected again.

**The monitoring cannot be activated**

When WSAPM or its monitoring does not work as expected, maybe the underlying Windows service is not running correctly. This can be checked by the MMC snap in “services.msc”. It can be started by *Control panel* 🡪 *Administrative Tools* 🡪 *Services* or Windows+R (“Run“) 🡪 “services.msc”. You should find an entry *Windows Server Advanced Power Management*. If not, WSAPM was not installed correctly and the installation should be repeated.

Next thing to check are following options for the service:

* *Service status*: *Running* – when the service is not running, no monitoring will take place.
* *Startup type*: *Automatic* – makes sure that the service is started automatically during system start.
* *Log On: Local System account* – the service has to run under local system account in order to have the appropriate rights for monitoring and controlling of the standby mode.

Basically, there should be no reason to change options for the service manually. All relevant options are set during installation and should not be altered afterwards.

**The time-controlled wake from standby does not work properly**

Make sure that the option *Allow wake timers* is set to *Active* in the Windows power options (see [*Settings - Wake*](#_Wake_1)). If this function is disabled, Windows will not be able for time-controlled wake from standby at all.

**Starting a program after wake from standby does not seem to work. WSAPM indicates in the log that the program was started, but the program is not shown on the desktop**

The programs which should be started after wake are run under the local system account. In this context, it is not possible to show a user interface. That means that programs with a user interface are started correctly, but will not be shown to the currently logged on user. It is best practice to only start programs after wake which do not have any user interface at all.

**The log entries appear in a different language than the other parts of the program**

This is due to the fact that the WSAPM service is running under the *Local System account*. This is a build in account which is created during the initial installation of Windows. Thus, the language and region options entered during Windows installation will be applied to this account. These language options are then used by the WSAPM service which will write the log entries.  
It is no problem to change the user language after the initial installation of Windows (you simply have to install a so called MUI package for a different language), but the *Local System account* will not be affected by these changes. In this case, the user interface of WSAPM will appear in the user’s language, but the service will still write log entries in the language which was specified during the initial windows installation.

**A plugin cannot be uninstalled with the button *Uninstall plugin*, only an error message appears**

When a plugin cannot be uninstalled, this can be due to an erroneous plugin. In this case there is a workaround to uninstall a plugin manually: First, the Windows Server Advanced Power Management service has to be stopped with the link on the main screen (see [*Usage - Main window*](#_Main_window)) and the application has to be closed. Next, go to the folder where the plugins of WSAPM are installed (usually *C:\Program Files (x86)\Windows Server Advanced Power Management\Plugins*) and delete the folder of the plugin to uninstall. Afterwards, if the plugin had custom settings, these settings can also be deleted. Just go to the folder where WSAPM saves the plugin settings (*C:\ProgramData\Windows Server Advanced Power Management\Plugins*) and delete the folder of the plugin to uninstall. After this, the WSAPM service can be started again.

**Remote shut down does not work**

In order to shut down the computer remotely, the corresponding option in WSAPM’s settings has to be enabled and a UDP port has to be specified. You also have to use an application or app which sends the commands for remote shut down WSAPM can understand (e.g. [MagicPacket](https://decatec.de/software/magicpacket_en/)).

When remote shut down does still not work, check your firewall settings if it blocks the communication between Windows Server Advanced Power Management and the app. During the installation of WSAPM an exception for the Windows Firewall is created. When another firewall is installed in your machine, you have to add this exception (for incoming UDP traffic) manually.

A detailed description of how to implement remote shut down with Windows Server Advanced Power Management and MagicPacket can be found in [this article](https://decatec.de/remote-shutdown-waspm-magicpacket_en/).

# FAQ

This section answers the most common questions about Windows Server Advanced Power Management.

**What is a Windows service and why WSAPM needs such a service?**

A Windows service is a program which runs without user interface and often provides low level functionality on Windows operating systems. An important difference to desktop applications is that a service can run without user logon. Due to this, the main functionality of WSAPM is implemented as Windows service. You can find more information to Windows services on [Wikipedia](http://en.wikipedia.org/wiki/Windows_service).

**Where the program settings of WSAPM are saved?**

The program settings for WSAPM and also the log file are stored *in C:\ProgramData\Windows Server Advanced Power Management*. By default, this folder is hidden by Windows. You can change this by going to *Control Panel* 🡪 *Folder Options* 🡪 *View* and choosing the option *Show hidden files, folders and drives*.

**Why WSAPM needs over 90 MB of hard disk space according to the Windows control panel (*Programs and Features*)?**

WSAPM only needs about 5 MB of hard disk space. The value shown in the Windows control panel includes the space needed by the .NET Framework. As the .NET Framework is already installed in many cases, the installation of WSAPM will usually only cost about 5 MB of hard disk space.

# Windows Server Advanced Power Management for developers

The following part of the user manual is specifically for developers, who want to extend Windows Server Advanced Power Management with own plugins or plan to support the remote shutdown functionality of this program in their own applications or apps.

You might skip this section if you just want to use WSAPM and are not a developer.

## Plugin development

Windows Server Advanced Power Management offers an interface for plugins. With plugins, WSAPM can be extended with more monitoring policies in an easy way.

WSAPM was implemented with C# and the .NET Framework, so plugins can be also developed in a .NET compatible programming language. All samples are shown with Visual Studio 2013, but can also be implemented with any other IDE.

The plugin interface and the procedure to implement own plugins for WSAPM is covered in the following.

### Simple and advanced plugins

WSAPM can be extended by two types of plugins:

* Simple plugins: These are plugins which basically offer a routine to check specific policies. These policies are integrated in the check routines of WSAPM.
* Advanced plugins: These plugins offer the same as the simple plugins, but also support settings and a corresponding UI.

When you want to implement a plugin for WSAPM, you should always start with a simple plugin, because this makes development easier and faster. Only in the case that your plugin will need settings (independent from the WSAPM settings) and a UI, you should develop an advanced plugin.

### Quick start

To make the development of a WSAPM plugin easy and straightforward, you can download pre-built Visual Studio solutions, which you can use as a good starting point. It is recommended to take a look at these solutions, because they also contain more detailed information.

* Pre-built Visual Studio solution which implements a simple plugin: [Download](https://decatec.de/downloads/wsapm/plugintemplates/WsapmPluginTemplate.zip)
* Pre-built Visual Studio solution which implements an advanced plugin: [Download](https://decatec.de/downloads/wsapm/plugintemplates/WsapmPluginAdvancedTemplate.zip)

### Developing simple plugins

You can find a step by step instruction of how to implement a simple plugin for WSAPM in the following.  
If you are using the pre-built Visual Studio solution (see [*Plugin development - Quick start*](#_Quick_start)), many steps are already done for you.

1. Create a new Visual Studio project: As WSAPM plugins are DLL file(s), you should start by creating a Visual Studio project as class library.
2. Add a reference for WSAPM: Add a reference to *Wsapm.Extensions.dll* to your project. You can find this file in the install folder of WSAPM (usually *C:\Program Files (x86)\Windows Server Advanced Power Management*).
3. Derive from base class *WsapmPluginBase*: The class implementing the logic of your plugin has to be derived from *WsapmPluginBase*.
4. Implement the abstract base methods: As *WsapmPluginBase* is an abstract class, you have to implement the following methods in your plugin class:
   * protected override bool Initialize()  
     This is the initialization method of your plugin and is called at least once after the plugin was loaded. Here you should implement all functions, which needs to be called only once.  
     The return value indicates if the initialization was successful. If *false* is returned here, the plugin was not initialized correctly (e.g. threw an exception) and will not be integrated into the policy check of WSAPM. If your plugin does not need any initialization, just return *true*.
   * protected override bool Prepare()  
     This method is used to prepare your plugin for the following check method and is called every time just before the check method is called.  
     The return value indicates if the preparation was successful. If *false* is returned here, the plugin was not prepared correctly (e.g. threw an exception) and will not be integrated into the policy check of WSAPM. If your plugin does not need any preparation, just return *true*.
   * protected override PluginCheckSuspendResult CheckPluginPolicy()  
     The is the main check method where you plugin might check its policies. The return value is a *PluginCheckSuspendResult*. The constructor of this class expects two parameters: The first one indicates if standby should be suppressed by WSAPM (*true*: standby should be suppressed; *false*: standby should not be suppressed). The second parameter is a string giving a reason for standby suppression. This parameter is only evaluated by WSAPM if the standby should be suppressed.
   * protected override bool TearDown()  
     This method is called directly after the check method and should be used to implement any tidy up functions.  
     The return value indicates if the tear down was successful. If *false* is returned here, the plugin was not torn down correctly (e.g. threw an exception). If your plugin does not need any tearing down, just return *true*.
5. Add attributes to the plugin class: Your plugin class needs two attributes so that WSAPM recognizes it as plugin:
   * [Export(typeof(WsapmPluginBase))]  
     This attribute has to be specified always as shown. You will need to add a reference to the *System.ComponentModel.Composition* .NET assembly so that the type of the attribute can be found.
   * [WsapmPlugin("Internal name of plugin", "v1.0.0", "{YOUR-GUID-HERE}")]  
     This attribute provides three important pieces to information to your plugin: The first parameter is you plugin’s name. This is only an internal name for the plugin and controls only the name of the folder where your plugin gets installed. The second parameter is your plugin’s version. The last parameter has to be a GUID which identifies your plugin and makes it distinctive from other plugins.  
     Important: This GUID should never be changed. Even if a new version of the plugin is created, it has to have the same GUID (but another version number) so that WSAPM can identify the relation to earlier versions of the plugin.  
     A new GUID can easily be created by the Create GUID utility in Visual Studio (*Tools* 🡪 *Create GUID*).
6. The plugin manifest: Every plugin also needs a plugin manifest. This is an XML file with further descriptions of the plugin. This manifest has a certain structure:

<?xml version="1.0" encoding="utf-8" ?>  
<WsapmPlugin>  
 <DescriptionSet lang="en">  
 <PluginName>My plugin</PluginName>  
 <Description>My plugin’s name</Description>  
 <AuthorName>decatec.de</AuthorName>  
 </DescriptionSet>

<DescriptionSet lang="de">  
 <PluginName>Mein Plugin</PluginName>  
 <Description>Beschreibung meines Plugins </Description>  
 <AuthorName>decatec.de</AuthorName>  
 </DescriptionSet>   
</WsapmPlugin>

A *DescriptionSet* describes the plugin for a certain language. The language is specified with the *lang* attribute and is a two-letter language code according to [ISO 639-1](http://en.wikipedia.org/wiki/ISO_639) (e.g. “en” for English or “de” for German).  
The plugin name, description and author name will be shown on the *Plugin* tab in the settings (see [*Settings - Plugins*](#_Plugins_1)).

1. More information about the plugin for the user: You can also add a file *ReadMe.txt* to the plugin with more information about this plugin (e.g. a detailed description what the plugin does). This information will be shown to the user when he clicks the button *Plugin info* on the tab *Plugins* of the settings (see [*Settings - Plugins*](#_Plugins_1)). You can also specify more files with language codes (e.g. *ReadMe\_de.txt*) for a German description of your plugin.  
   These files are optional. If you do not provide further information, a general plugin description will be shown when the user clicks the button *Plugin info*.
2. By this point you have finished the development of your WSAPM plugin. To ship your plugin, you have to create a Zip file containing your plugin. Just zip all of the plugin’s files in the project output directory.  
   Important: You will find the following files in the project output directory, which were added as reference to your project. These files should not be shipped with your plugin and therefore not be included in your plugin’s Zip file:
   * *Wsapm.Extensions.dll*
   * *de\Wsapm.Extensions.resources.dll*

### Developing advanced plugins

You can find a step by step instruction of how to implement am advanced plugin for WSAPM in the following. As mentioned earlier, you should only develop an advanced plugin if you need plugin specific settings and a UI where these settings can be changed. Otherwise you should use a simple plugin because this will make plugin development a lot easier.  
If you are using the pre-built Visual Studio solution (see [*Plugin development - Quick start*](#_Quick_start)), many steps are already done for you.

1. Create a new Visual Studio project: As WSAPM plugins are DLL file(s), you should start by creating a Visual Studio project as class library.
2. Add a reference for WSAPM: Add a reference to *Wsapm.Extensions.dll* to your project. You can find this file in the install folder of WSAPM (usually *C:\Program Files (x86)\Windows Server Advanced Power Management*).
3. Implement a settings class: The settings of the plugin should be implemented in an own settings class. The class has to be *public* and serializable (use the *Serializable* attribute over the class declaration).
4. Implement the UI for the plugin: The UI of your plugin can be implemented using WPF ([Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation)) or [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms). So add a new UserControl to your project. The UI should only contain controls which represent the settings of your plugin, i.e. you will not need buttons like *OK* or *Cancel* the give back the control to WSAPM.  
   The UI class has to implement the interface *IWsapmPluginSettingsControl*, which contains two methods:
   * public void SetSettings(object settings)  
     With this method the current settings are loaded into your UI class. Here you should fill your UI elements (TextBoxes, CheckBoxes, etc) with the data from the settings. Before you can do that, you have to cast the settings to the type of the settings class.
   * public object GetSettingsBeforeSave()  
     This method will be called when the plugin settings should be saved (by WSAPM, not by your plugin). Here you should return an instance of your settings class which is filled with data from the UI elements. These settings are saved afterwards by WSAPM.
5. Implement the actual plugin class: Now you will need another class representing your plugin in your project. All following activities apply to this plugin class.
6. Derive from base class *WsapmPluginAdvancedBase*: The class implementing the logic of your plugin has to be derived from *WsapmPluginAdvancedBase*.
7. Add a *public* constructor to the plugin class: The plugin class will need a constructor which calls the constructor of the base class specifying the type of your settings class.
8. Implement the abstract base methods: As *WsapmPluginAdvancedBase* is an abstract class, you have to implement the following methods in your plugin class:
   * protected override bool Initialize()  
     This is the initialization method of your plugin and is called at least once after the plugin was loaded. Here you should implement all functions, which needs to be called only once.  
     The return value indicates if the initialization was successful. If *false* is returned here, the plugin was not initialized correctly (e.g. threw an exception) and will not be integrated into the policy check of WSAPM. If your plugin does not need any initialization, just return *true*.
   * protected override bool Prepare()  
     This method is used to prepare your plugin for the following check method and is called every time just before the check method is called.  
     The return value indicates if the preparation was successful. If *false* is returned here, the plugin was not prepared correctly (e.g. threw an exception) and will not be integrated into the policy check of WSAPM. If your plugin does not need any preparation, just return *true*.
   * protected override PluginCheckSuspendResult CheckPluginPolicy()  
     The is the main check method where you plugin might check its policies. The return value is a *PluginCheckSuspendResult*. The constructor of this class expects two parameters: The first one indicates if standby should be suppressed by WSAPM (*true*: standby should be suppressed; *false*: standby should not be suppressed). The second parameter is a string giving a reason for standby suppression. This parameter is only evaluated by WSAPM if the standby should be suppressed.
   * protected override bool TearDown()  
     This method is called directly after the check method and should be used to implement any tidy up functions.  
     The return value indicates if the tear down was successful. If *false* is returned here, the plugin was not torn down correctly (e.g. threw an exception). If your plugin does not need any tearing down, just return *true*.
   * protected override object LoadDefaultSettings()  
     This method should return the default settings of your plugin. Therefore, you should create a new instance of the plugin’s settings class (see paragraph 3 *Implement a settings class*) and provide the default settings.  
     The default settings will always be used when no settings are available, e.g. when the plugin’s UI is started for the first time.
   * public override object SettingsControl  
     Here you have to implement the getter to return an instance of the plugin’s UI class. When this instance is created the first time, you should save it in a private variable and return always this instance in all subsequent calls to this property.
9. Add attributes to the plugin class: Your plugin class needs two attributes so that WSAPM recognizes it as plugin:
   * [Export(typeof(WsapmPluginBase))]  
     This attribute has to be specified always as shown. You will need to add a reference to the *System.ComponentModel.Composition* .NET assembly so that the type of the attribute can be found.  
     The type is *WsapmPluginBase* even if you are implementing an advanced plugin with another base class (*WsapmPluginAdvancedBase*).
   * [WsapmPlugin("Internal name of plugin", "v1.0.0", "{YOUR-GUID-HERE}")]  
     This attribute provides three important pieces to information to your plugin: The first parameter is you plugin’s name. This is only an internal name for the plugin and controls only the name of the folder where your plugin gets installed. The second parameter is your plugin’s version. The last parameter has to be a GUID which identifies your plugin and makes it distinctive from other plugins.  
     Important: This GUID should never be changed. Even if a new version of the plugin is created, it has to have the same GUID (but another version number) so that WSAPM can identify the relation to earlier versions of the plugin.  
     A new GUID can easily be created by the Create GUID utility in Visual Studio (*Tools* 🡪 *Create GUID*).
10. Access settings in the plugin’s code: To access current settings in your plugin’s code, use the property *CurrentSettings* of the base class. You will need a cast to your specific settings class.
11. Access the UI in the plugin’s code: Access to the UI class (or any other UI related code) may only be done in the getter of the property *SettingsControl*. You must not call UI related code in any other place in your code!
12. Save and load settings: Saving and loading the plugin’s settings is done by WSAPM and is not handled in your plugin’s code. Therefore, you do not have to implement any code which handles saving or loading of settings.  
    The plugin’s settings can be found in the following directory: *C:\ProgramData\Windows Server Advanced Power Management\Plugins\<name of the plugin> (<GUID of the plugin>)\PluginSettings.xml*
13. The plugin manifest: Every plugin also needs a plugin manifest. This is an XML file with further descriptions of the plugin. This manifest has a certain structure:

<?xml version="1.0" encoding="utf-8" ?>  
<WsapmPlugin>  
 <DescriptionSet lang="en">  
 <PluginName>My plugin</PluginName>  
 <Description>My plugin’s name</Description>  
 <AuthorName>decatec.de</AuthorName>  
 </DescriptionSet>

<DescriptionSet lang="de">  
 <PluginName>Mein Plugin</PluginName>  
 <Description>Beschreibung meines Plugins </Description>  
 <AuthorName>decatec.de</AuthorName>  
 </DescriptionSet>   
</WsapmPlugin>

A *DescriptionSet* describes the plugin for a certain language. The language is specified with the *lang* attribute and is a two-letter language code according to [ISO 639-1](http://en.wikipedia.org/wiki/ISO_639) (e.g. “en” for English or “de” for German).  
The plugin name, description and author name will be shown on the *Plugin* tab in the settings (see [*Settings - Plugins*](#_Plugins_1)).

1. By this point you have finished the development of your WSAPM plugin. To ship your plugin, you have to create a Zip file containing your plugin. Just zip all of the plugin’s files in the project output directory.  
   Important: You will find the following files in the project output directory, which were added as reference to your project. These files should not be shipped with your plugin and therefore not be included in your plugin’s Zip file:
   * *Wsapm.Extensions.dll*
   * *de\Wsapm.Extensions.resources.dll*

### Debug plugins

Because plugins are no executable files and are run in the context of WSAPM, you cannot debug plugins with F5 (Visual Studio). However, you can debug your plugin by adding following code in the desired place:

if (!System.Diagnostics.Debugger.IsAttached)

System.Diagnostics.Debugger.Launch();

System.Diagnostics.Debugger.Break();

You also have to build the plugin in debug mode and add the PDB files to the plugin’s directory (usually *C:\Program Files (x86)\Windows Server Advanced Power Management\Plugins\<name of the plugin>*). Now you can attach the debugger from within Visual Studio (*Debug* 🡪 *Attach to Process…*). In the list of processes to attach, you have to choose one of the following processes:

* *Wsapm.exe* if you want to debug parts of your plugin’s UI.
* *Wsapm.Service.exe* if you want to debug the specific plugin logic.

Now, when the plugin is loaded/executed or the plugin’s UI is shown, a window will appear where you can start Visual Studio for debugging. The debugger should break on the code part shown above.

### Make your plugin available to other users

When you have developed your own plugin for WSAPM and you think that it might also be useful for other users of WSAPM, please send me an [e-mail](mailto:jr@decatec.de). Then I might publish your plugin on WSAPM’s website ([decatec.de](https://decatec.de/)).

## Remote shut down

Windows Server Advanced Power Management has a feature called remote shut down allowing users to shut down or restart the computer or put it in standby or hibernate mode remotely.  
Therefore a program or app is needed on the client side in order to send these remote shutdown commands (e.g. [MagicPacket](https://decatec.de/software/magicpacket_en/)). The following section describes the protocol for remote shutdown so that developers can implement this functionality in own applications or apps.

If the corresponding option is set to active in Windows Server Advanced Power Management, the program listens at the specified port for incoming [UDP packets](https://en.wikipedia.org/wiki/User_Datagram_Protocol). The packets used for remote shut down are very similar to [Wake On LAN](https://en.wikipedia.org/wiki/Wake-on-LAN) commands.

**Wake On LAN**

Wake On LAN commands are simple UDP packets (so called Magic Packets) containing the following information:

* 6x the hexadecimal value of FF
* 16x the MAC address of the target system

These commands are then sent to the network’s [broadcast address](https://en.wikipedia.org/wiki/Broadcast_address) (255.255.255.255).

**Commands for remote shut down**

The commands for remote shutdown are just like Magic Packets used for Wake On LAN, but these do not start with the value FF, but use one the following values:

* Standby: AA
* Hibernate: BB
* Restart: CC
* Shut down: DD

An optional password is then attached at the end of the UDP packet. If a password for remote shut down is specified in WSAPM’s settings, a remote shut down packet is only processed if the received password is the same as the password entered in the program’s settings.

**Important:** The password for remote shut down is not a real security feature as it is sent over the network as plain text, i.e. unencrypted.

*Example:* The target machine has the MAC address 23:D0:43:F1:2C:0A and should be shut down remotely. Therefore, a UDP packet needs to be build and sent to the broadcast address (255.255.255.255):

* 6x hexadecimal value of DD (for shut down)
* 16x the MAC address (in this case 23:D0:43:F1:2C:0A)

A simple implementation in C# to build such a UDP packet would be:

private static byte[] GetShutdownPacket(byte[] macAddress, string password)

{

byte[] passwordBytes = System.Text.Encoding.UTF8.GetBytes(password);

byte[] packet = new byte[102 + passwordBytes.Length];

// 6x AA -> standby

// 6x BB -> hibernate

// 6x CC -> restart

// 6x DD -> shut down

byte headerByte = 0xDD;

// 6x DD (for shut down)

for (int i = 0; i < 6; i++)

{

packet[i] = headerByte;

}

// 16x the MAC address of the target machine

for (int i = 1; i < 17; i++)

{

macAddress.CopyTo(packet, i \* 6);

}

// The password is attached at the end of the packet

for (int i = 0; i < passwordBytes.Length; i++)

{

packet[102 + i] = passwordBytes[i];

}

return packet;

}

I’m looking forward for other programs or apps supporting the remote shut down functionality of Windows Server Advanced Power Management. I also would welcome supporting other developers with the implementation (just write me an [e-mail](mailto:jr@decatec.de)).