Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

About NSClient++

NSClient++ (or nscp as I tend to call it nowadays) aims to be a simple yet powerful and secure monitoring daemon for Windows operating systems. It is built for Nagios, but nothing in the daemon is actually Nagios specific and could probably, with little or no change, be integrated into any monitoring software that supports running user tools for polling.

The structure of the daemon is a simple NT service that loads plug-ins to an internal stack. The plug-ins can then request data (poll performance data) from the other plug-ins through the internal stack. As of now there are a few plug-ins for basic performance data collection. For details of supplied modules, see CheckCommands.

NSClient++ can be extended in two ways: you can either write your own plug-in or you can execute an external script (as of now batch/exe/*). Writing your own plug-in is, of course, the most powerful way but requires knowledge of C++ or other languages which can produce DLLs and interface with regular C programs (generally, every other language available, but there is some simple API helpers for C/C++ as well as descriptions).

As for checking with NSClient++, I would recommend NRPE as it is a lot more flexible than check_nt. But NSClient has full support for check_nt, and if there is an interest, I could probably add support for check_nt from nc_net.

Supported OS/Platform

NSClient++ should run on the following operating systems:

- NT4 (SP5?)
- Windows 2000 W2K
- Windows XP
- Windows 2003
- Windows Vista
- Windows 2008 (some issue reported with the installer, run as admin)
- Windows 7 (some issue reported with the installer, run as admin)
- ...

...as well as the following platforms:

- Win32
- x64 (AMT64/EMT64)
- IA64 (Itanium, but no recent builds due to compiler expensiveness, feel free to compile it your self though)

Whats in a name?

Since I have noticed some ppl. use other names for the client I decided to list them here to make it simpler (ie. Goggle might find it) for people to find it.

- NSClient++ (the "real" name)
- NSCP (what I sometimes use)
- NSClientpp (version of NSCLient++)
- NSClient (?)
- Saw a French (Spanish) site use: NSC++

Again I myself as stated before prefer NSClient++ or NSCP.

:-)

My Name is Michael Medin and I am the author of this program (an also handles a lot of the support and such).

You can find me around the web here:

Messengers:

- EMail: michael at medin dot name
- ICQ: 1818494
- MSN: michael at medin dot name
- Jabber: mickem @ jabber dot nakednuns dot org
- ...

Communities:

- LinkedIn?: http://www.linkedin.com/in/mickem
- Last.fm: http://www.last.fm/user/mickem
- LibraryThing?: http://www.librarything.com/profile/mickem
- ...

Become a sponsor

NSClient++ is a free and open source tool not backed by any commercial entity. In fact I don't even work with Nagios so this is a 100% "spare time effort".

So if you like and use NSClient++ and perhaps even make money from using it. Feel free to become an official sponsor of NSClient++. The sponsoring program is pretty loose as of yet but the main ideas are to have three "levels" of sponsoring:

- Gold Sponsor
    - Contribution: Euro 1'000 / year
    - You get you company logo in the projects (this site) website navigation bar. Visible on around 75.000 page impressions per month (12-15.000 unique visitors per month).
    - Free e-mail support[1] (5 premium[2] issues)
    - Custom built[3] (branded) version of NSClient++ (optional)

- Silver Sponsor
    - Contribution: Euro 100 / year
    - You get your company logo on the sponsors page of the projects website (this page).
    - Free e-mail support[1] (1 premium[2] issue).

- NSClient++ Fan
    - Any kind of donation
    - You get your name listed together with your donation on the NSClient++ fan page.
    - "Free support" (when I or the community have the time)
    - Become a Fan today

If you have any questions about becoming a sponsor, please be contact me on info@nsclient.com.

If you have donated before and want to be listed on the fan page contact me and I will add you, since there was no "opt out" I am hesitant to add peoples names without prior consent.

Current sponsors

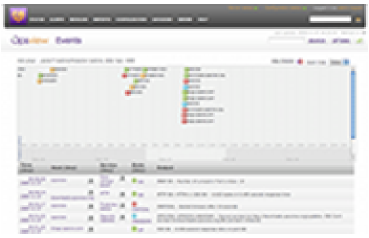
A big thank you to my current sponsors.

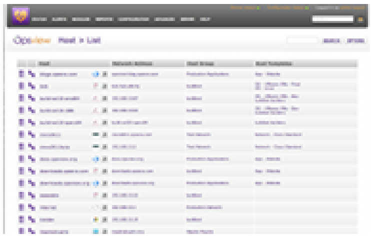
OP5

Opsview

Opsview® is enterprise system monitoring software designed for scalability, flexibility and ease of use. Software was awarded a Product Excellence award for Best System Management Tool at LinuxWorld Expo in San Francisco during August 2008.

Opsview, in development since 2003 is a fully integrated monitoring tool that incorporates popular Open Source software including Nagios® Core, Nagvis, Net-SNMP and RRDtool. The Catalyst web framework provides an extensible monitoring and configuration user interface. Opsview server software runs on Linux and Solaris 10. It will monitor all common operating systems including Linux, AIX, Solaris and Windows. The Opsview engineering group contains members of the Nagios developer team





Opsview provides enterprise monitoring features without the complexity or expense of traditional proprietary software.

- Fully supported. Four levels of Opsview subscription available.
- Incorporates the Nagios® Core monitoring engine allowing use of a wide range of Nagios plugins and add-ons
- Released under the GNU GPL v2 Open Source license
- Scalability and resilience through distributed monitoring architecture
- SNMP polling and trap processing
- Built on extensible web framework
- User interface designed around simplicity and ease of use
- APIs for monitoring, notification and configuration
- Active user and developer communities

Opsview has users in telecommunications, financial services, government, education, media, IT services and e-commerce industries.

---

1. 1. All support is provided on a best-effort basis
2. 2. A premium issue is an issue where you request direct help outside "normal channels"

3. 3. For instance you might want to bundle specific script, and/or configuration to make it simpler for you to roll it out.

Fans of NSClient++

A lit of fans (who have donated to the project):

- ???-??-?? Johnny Wetlaufer nice logo design

- 2009-11-25 Pawe? Sosnowski
- 2009-08-12 Thomas Wallutis
- 2009-08-04 Nicolas Schmitz
- 2009-06-15 Gerhard Laußer
- 2010-04-19 ????? ????????
- 2010-04-25 Anthony Hogan
- 2010-04-29 ????? ???????? (again, thanx :)

Installing NSClient++

This is a grooving process before it was all manual but slowly we are getting a more "automated" installation process so hopefully this will keep improving in the future as well and some of the steps might go away.

1. Installation

NSClient++ comes with an interactive installer (MSI) which should preferably be used. There is also a command line option for registering (and de-registering) the service for details refer to the manual installation guide. If you are using Windows NT4 there is some dependencies you need to manually install for details refer to the NT4 Dependency guide.

Thus to install the Client you simply click the MSI package (for your platform) and follow the wizard through. **BUT** and this is a big but after you have installed it it still needs to be configure (which is done with your favorite text editor).

2. Configuration

Before you start NSClient++ you need to configure it by editing the configuration file (NSC.ini). The configuration file is a simple text file and is explained in detail under Configuration.

The configuration file (NSC.ini) **NEEDS** to be configured as for security reasons all plug-ins are disabled by default. The reason for this is so no one will accidentally install this and get potential security issues, I believe that things should be "off" by default. Also notice that by default allowed_hosts are 127.0.0.1 so you need to modify this as well.

3. System tray

If you plan to use the SystemTray module (that shows a system tray icon on the desktop you need to install the SystemTray module as well as NSClient++ on "old" versions of windows (XP and blow) on modern version of windows (XP and above) you can used the new experimental shared session support. For details on this see the System tray installation guide.

4. Testing and Debugging

After you have installed NSClient++ you need to start it which is done which can be done in several ways as it is a normal service (so either fire up a command line and use the net start/stop command of you can use the computer manager services node.

When you are starting yout and/or configuring your client you can use the "debug" mode which will be very helpfull as you will see the debug log in "real time" whrn you play around with it. To start NSClient++ in test/debug mode use the following command (you can also use the icon on the start menu):

```
NSClient++ /test
```

5. Windows Firewall

I have yet to foigure this one out but hopefully someone can help me write this! I shall for the next version try to make an automated exception thingy for the windows firewall.

6. External Firewall (optional)

Firewall configuration should be pretty straight forward:

- If you use NRPEListener (check_nrpe) you need the NRPE port open (usually 5666) from the nagios server towards the client.
- If you use the NSClientListener (check_nt) you need the (modified) NSClient port open (usually 12489) from the nagios server towards the client.
- If you use the NSCA Module (passive checks) you need the NSCA port open from the client towards the nagios server. client:* -> nagios:5667
- If you use the NRPEClient module to check any remote systems (use NSClient++ as a proxy) you need to have NRPE port (usually 5666) open from NSClient++ (the proxy) to the remote-client in addition to the method you use to submit the results to the server. nsclient-proxy:* -> remote-client:5666

Protocol	Source	Source port	Destination	Destination port	Comment
NRPE	**nagios**	<all>	Client	5666	The nagios server initiates a call to the client on port 5666
NSClient	**nagios**	<all>	Client	12489	The nagios server initiates a call to the client on port 12489
NSCA	client	<all>	**nagios**	5667	The client initiates a call to the nagios server on port 5667
NRPE-proxy	client	<all>	remote-client	5666	The client initiates a call to the remote client on port 5666

- **nagios** Is the ip/host of the main nagios server
- client is the windows computer where you have installed NSClient++
- remote-client is the "other" client you want to check from NSClient++ (using NSClient++ as a proxy)

All these ports can be changed so check your nsc.ini.

Installing NSClient++

This is a grooving process before it was all manual but slowly we are getting a more "automated" installation process so hopefully this will keep improving in the future as well and some of the steps might go away.

1. Installation

NSClient++ comes with an interactive installer (MSI) which should preferably be used. There is also a command line option for registering (and de-registering) the service for details refer to the manual installation guide. If you are using Windows NT4 there is some dependencies you need to manually install for details refer to the NT4 Dependency guide.

Thus to install the Client you simply click the MSI package (for your platform) and follow the wizard through. **BUT** and this is a big but after you have installed it it still needs to be configure (which is done with your favorite text editor).

2. Configuration

Before you start NSClient++ you need to configure it by editing the configuration file (NSC.ini). The configuration file is a simple text file and is explained in detail under Configuration.

The configuration file (NSC.ini) **NEEDS** to be configured as for security reasons all plug-ins are disabled by default. The reason for this is so no one will accidentally install this and get potential security issues, I believe that things should be "off" by default. Also notice that by default allowed_hosts are 127.0.0.1 so you need to modify this as well.

3. System tray

If you plan to use the SystemTray module (that shows a system tray icon on the desktop you need to install the SystemTray module as well as NSClient++ on "old" versions of windows (XP and blow) on modern version of windows (XP and above) you can used the new experimental shared session support. For details on this see the System tray installation guide.

4. Testing and Debugging

After you have installed NSClient++ you need to start it which is done which can be done in several ways as it is a normal service (so either fire up a command line and use the net start/stop command of you can use the computer manager services node.

When you are starting yout and/or configuring your client you can use the "debug" mode which will be very helpfull as you will see the debug log in "real time" whrn you play around with it. To start NSClient++ in test/debug mode use the following command (you can also use the icon on the start menu):

```
NSClient++ /test
```

5. Windows Firewall

I have yet to foigure this one out but hopefully someone can help me write this! I shall for the next version try to make an automated exception thingy for the windows firewall.

6. External Firewall (optional)

Firewall configuration should be pretty straight forward:

- If you use NRPEListener (check_nrpe) you need the NRPE port open (usually 5666) from the nagios server towards the client.
- If you use the NSClientListener (check_nt) you need the (modified) NSClient port open (usually 12489) from the nagios server towards the client.
- If you use the NSCA Module (passive checks) you need the NSCA port open from the client towards the nagios server. client:* -> nagios:5667
- If you use the NRPEClient module to check any remote systems (use NSClient++ as a proxy) you need to have NRPE port (usually 5666) open from NSClient++ (the proxy) to the remote-client in addition to the method you use to submit the results to the server. nsclient-proxy:* -> remote-client:5666

Protocol	Source	Source port	Destination	Destination port	Comment
NRPE	**nagios**	<all>	Client	5666	The nagios server initiates a call to the client on port 5666
NSClient	**nagios**	<all>	Client	12489	The nagios server initiates a call to the client on port 12489
NSCA	client	<all>	**nagios**	5667	The client initiates a call to the nagios server on port 5667
NRPE-proxy	client	<all>	remote-client	5666	The client initiates a call to the remote client on port 5666

- **nagios** Is the ip/host of the main nagios server
- client is the windows computer where you have installed NSClient++
- remote-client is the "other" client you want to check from NSClient++ (using NSClient++ as a proxy)

All these ports can be changed so check your nsc.ini.

Installation

NSClient++ comes simple command line option for registering (and deregistering) the service but it does not have a GUI installer.

Thus to install the Client you only need to copy the files to a directory of your choice and then run ?NSClient++ /install?.

Before you start NSClient++ you need to configure it by editing the configuration file (NSC.ini). The configuration file is a simple text file and is explained in detail under Configuration. The files needed by NSClient++ varies but mainly the exe and DLL's in the NSClient++ root are required as well as all the modules you plan to use from the modules subdirectory (/modules/*).

The configuration file (NSC.ini) **NEEDS** to be configured as for security reasons all plug-ins are disabled by default. The reason for this is so no one will accidentally install this and get potential security issues, I believe that things should be "off" by default. Also notice that by default allowed_hosts are 127.0.0.1 so you need to modify this as well.

If you plan to use the SystemTray module (that shows a system tray icon on the desktop you need to install the SystemTray module as well as NSClient++. To install NSClient++ execute the following command:

```
NSClient++ /install
NSClient++ SysTray install
```

To uninstall NSClient++ execute the following command:

```
NSClient++ SysTray uninstall
NSClient++ /uninstall
```

To start NSClient++ execute the following command:

```
NSClient++ /start
```

To stop NSClient++ execute the following command:

```
NSClient++ /stop
```

If you only wish to test it or debug the client you can use the following without installing it first.

```
NSClient++ /test
```

Firewall

Firewall configuration should be pretty straight forward:

If you use NRPEListener (check_nrpe) you need the NRPE port open (usually 5666) from the nagios server towards the client.

```
nagios:* -> client:5666
```

If you use the NSClientListener (check_nt) you need the (modified) NSClient port open (usually 12489) from the nagios server towards the client.

```
nagios:* -> client:12489
```

If you use the NSCA Module (passive checks) you need the NSCA port open from the client towards the nagios server.

```
client:* -> nagios:5667
```

If you use the NRPEClient module to check any remote systems (use NSClient++ as a proxy) you need to have NRPE port (usually 5666) open from NSClient++ (the proxy) to the remote-client in addition to the method you use to submit the results to the server.

```
nsclient-proxy:* -> remote-client:5666
```

All these ports can be changed so check your nsc.ini.

NT4

NT4 does not come with the PDH library and you need to install that before using NSClient++. PDH can be downloaded from Microsoft: http://support.microsoft.com/default.aspx?scid=kb;en-us;Q284996 and the simplest way to install it is to uncompress it directly into the NSClient++ directory.

NT4 also (sometimes) lack the PSAPI helper which is available in the "Platform SDK Redistributable: PSAPI for Windows NT" from Microsoft. http://www.microsoft.com/downloads/details.aspx?FamilyID=3d1fbaed-d122-45cf-9d46-1cae384097ac as with the PDH either install in system32 or local NSClient++ directory.

System Tray Installation Guide

This is a subject which I think many people have trouble with so I wrote up a simple guide for it.

The first thing you should understand is that there are two ways in windows to do "system trays".

- SERVICE_INTERACTIVE_PROCESS This works splendidly for Windows:es up till and including XP
- Client-server (or shared session) This works splendidly from (including) CP and beyond. This also works (sort of) on windows 2000 (but not NT) but in my case I would recommend using XP since the session handling changed in XP.

SERVICE_INTERACTIVE_PROCESS 'way'

This is the "old" way and has been in use up until including windows XP.

The details on a technical level is that the service has a flag "SERVICE_INTERACTIVE_PROCESS" which allows it to interact with the "desktop" so what we do in NSClient++ is simply add an icon to the desktop and voila system tray support in a few lines of code all neat and tidy inside the same process.

<<<Add nice image here>>>

Client-server 'way'

Now since the "old" way was so simple Microsoft had to go about changing it (of course) so the "new" way which works from (including) XP and above. Technically it works from Windows 2000 but slightly different so I would recommend using it on XP and above.

The reason for this is that in "modern" windows there is no "desktop" there is instead several desktops one of which is there the system tray from NSClient++ will end up (session 0). And unfortunately this (session 0) is not the one where the logged in user ends up (which is session 1 and above). This is all down to the "User switching" which was introduced when terminal server was "sort of integrated" into windows 2000.

So what can we do to circumvent this?

Quite simply we can launch a program in the logged in users session and have them communicate with each other.

This is done by triggering on the "user logged in" (or as it is called in the debug log "Got session change...") and if enabled (shared_session=1) launch a process into that users session and hope that they will communicate with each other. The actual communication is done using a shared session (technically a bunch of shared semaphores and a bunch of shared memory areas).

<<<Add nice image here>>>

BETA WARNING Now this will introduce a lot of challenges and problems and as of now it is more of a technical preview then a stable and mature thing I would enable in production.

Installation guide

THis is split into two section "old" and "modern".

NT4, 2000, XP (old)

NSC.ini

```
[modules]
SysTray.dll
```

Then run:

```
NSClient++ -noboot systray install
```

XP, 2k3, Vista, Windows 7, etc (modern)

```
[settings]
shared_session=1
```

And:

- **Don't** enable SysTray.dll
- **Don't** run the "systray install" thingy above.

Dependencies for Windows NT4

Since windows NT4 is OLD (yes it is really old and you should think about upgrading) it has some dependencies which I have decided not to resolve easily. This means you need to install some Microsoft components to get tings up and running on Windows NT4.

Hopefully this should not be to hard and not cause you any problems.

PDH library (CPU, memory, etc etc)

NT4 does not come with the PDH library and you need to install that before using NSClient++. PDH can be downloaded from Microsoft: http://support.microsoft.com/default.aspx?scid=kb;en-us;Q284996 and the simplest way to install it is to uncompress it directly into the NSClient++ directory.

PSAPI (process checks)

NT4 (sometimes) lack the PSAPI helper which is available in the "Platform SDK Redistributable: PSAPI for Windows NT" from Microsoft.
http://www.microsoft.com/downloads/details.aspx?FamilyID=3d1fbaed-d122-45cf-9d46-1cae384097ac as with the PDH either install in system32 or local NSClient++ directory.

Configuration

Configuration is fairly simple and straight forward. Open the configuration file in notepad (or your favorite editor) "notepad <installation path>\NSC.ini" and edit it accordingly. A longer description of the Configuration file is included in the following page.

The file has sections (denoted with section name in brackets) and key/value pairs (denoted by key=value). Thus it has the same syntax as pretty much any other INI file in windows.

The sections are described in short below. The default configuration file has a lot of examples and comments so make sure you change this before you use NSClient++ as some of the examples might be potential security issues.

The configuration can also be stored in the system registry (HKLM\Software\NSClient++) there is currently no UI to configure this so the simplest way is to maintain the configuration in the INI file and "Migrate that" to the registry. This is can be done via the [RemoteConfiguration] module but in short:

```
NSClient++ -noboot RemoteConfiguration ini2reg
```

A sample configuration file is included in the download but can also be found here trunk/NSC.dist

Modules

This is a list of modules to load at startup. All the modules included in this list has to be NSClient++ modules and located in the modules subdirectory. This is in effect the list of plug-ins that will be available as the service is running. For information on the various plug-ins check the Modules section in the navigation box.

A good idea here is to disable all modules you don?t actually use for two reasons. One less code equals less potential security holes and two less modules means less resource drain.

A complete list of all available modules:

- CheckDisk (module)
- CheckEventLog (module)
- CheckExternalScripts (module)
- CheckHelpers (module)
- CheckSystem (module)
- CheckTaskSched (module)
- CheckWMI (module)
- FileLogger (module)
- LUAScript (module)
- NRPEListener (module)
- NSCAAgent (module)
- NSClientListener (module)
- RemoteConfiguration (module)
- SysTray (module)

Settings

This section has generic options for how NSClient++will work, some of these settings (such as allowed_hosts) is inherited in sections below so it is probably a better idea to set them here in the "global" section.

The options you have available here are

Option	Default value	Description
obfuscated_password	...	An obfuscated version of password. For more details refer to the password option below. To create the obfuscated Password use: "NSClient++.exe /encrypt"
password	...	The password used by various (presently only NSClient) daemons. If no password is set everyone will be able to use this service remotely.
allowed_hosts	127.0.0.1	A list (comma separated) with hosts that are allowed to connect and query data. If this is empty all hosts will be allowed to query data. BEWARE: NSClient++ will not resolve the IP address of DNS entries if the service is set to startup automatically. Use an IP address instead.
use_file	0	Has to be set to 1 if you want the file to be read (if set to 0, and the use_reg is set to 1 the registry will be used instead)

Advanced options:

Option	Default value	Description
master_key	...	The secret "key" used when (de)obfuscating passwords.
cache_allowed_hosts	1	Used to cache looked up hosts if you check dynamic/changing hosts set this to 0.

includes

A list of other configuration files to include when reading this file. Might be useful if you have a very complex setup or want to have setting split up in segments.

Module Configuration

NRPE Listener Sections

NRPE Section

This is section is included from the following page NRPEListener/config/nrpe

Overview

This is configuration for the NRPE module that controls how the NRPE listener operates.

Option	**Default**	**Description**
port	5666	The port to listen to
allowed_hosts		A list of hosts allowed to connect via NRPE.
use_ssl	1	Boolean value to toggle SSL encryption on the socket connection
command_timeout	60	The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).
socket_timeout	30	The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Advanced options:

Option	**Default**	**Description**
performance_data	1	Send performance data back to nagios (set this to 0 to remove all performance data)
socket_back_log		

		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
string_length	1024	Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.
script_dir		Load all scripts in a directory and use them as commands. Probably dangerous but usefull if you have loads of scripts :)
bind_to_address		The address to bind to when listening to sockets.

port

The port to listen to

Default
    5666

allowed_hosts

A list (comma separated) with hosts that are allowed to poll information from NRPE. This will replace the one found under Setting for NRPE if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system

Default
    Empty list (falls back to the one defined under [Settings]

use_ssl

Boolean value to toggle SSL (Secure Socket Layer) encryption on the socket connection. This corresponds to the -n flag in check_nrpe

Values

Value	Meaning
0	Don't use SSL
1	Use SSL encryption

Default
    1 (enabled)

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
    IP address of any interface of the server.
Default
    Empty (first (all?) interface will be used)

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones so internal commands may execute forever.

It is usually a good idea to set this to less then the timeout used with check_nrpe

Default
60

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

NOTICE That there are more then one place to set this!

Default
0 (means don't allow arguments)

Values

Value	Meaning
0	Don't allow arguments
1	Allow arguments.

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Default
0 (means don't allow meta characters)

Values

Value	Meaning
0	Don't allow meta characters
1	Allow meta characters

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
30 seconds

script_dir

Load all scripts in a directory and use them as commands. Probably dangerous but useful if you have loads of scripts :)

Default
Empty (don't load any scripts)

performance_data

Send performance data back to Nagios (set this to 0 to remove all performance data)

Default
1

Values

Value	Meaning
0	Don't send performance data
1	Send performance data

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

string_length

Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.

Default
1024

NRPE Handlers Section

This is section is included from the following page NRPEListener/config/nrpe_handlers

Ovreview

DEPRECATED This part of the module is deprecated and should not be used. Refer to the [CheckExternalScripts] module instead. This module can add two types of command handlers.

First there are external command handlers that execute a separate program or script and simply return the output and return status from that. The other possibility is to create an alias for an internal command.

To add an external command you add a command definition under the ?NRPE Handlers? section. A command definition has the following syntax:

```
[NRPE Handlers]
command_name=/some/executable with some arguments
test_batch_file=c:\test.bat foo $ARG1$ bar
command[check_svc]=inject CheckService checkAll
```

The above example will on an incoming ?test_batch_file? execute the c:\test.bat file and return the output as text and the return code as the Nagios status.

Alias (builtin commands)

To add an internal command or alias is perhaps a better word. You add a command definition under the ?NRPE Handlers? section. A command definition with the following syntax:

```
command_name=inject some_other_command with some arguments
check_cpu=inject checkCPU warn=80 crit=90 5 10 15
```

The above example will on an incoming ?check_cpu? execute the internal command ?checkCPU? with predefined arguments give in the command definition.

NRPE_NT Syntax

To leverage existing infrastructure you can copy your old definitions from NRPE_NT as-is. Thus the following:

```
command[check_svc]=inject CheckService checkAll
```

translates into a command called check_svc with the following definition:

```
CheckServcice checkAll
```

File Logging Sections

Log Section

This is section is included from the following page FileLogger/config

Overview

This section has options for how logging is performed with the [FileLogger] module. First off notice that for logging to make sense you need to enable the ?FileLogger.dll? module that logs all log data to a text file in the same directory as the NSClient++ binary if you don?t enable any logging module nothing will be logged.

The options you have available here are

Option	Default	Description
debug	0	A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.
file	nsclient.log	The file to write log data to. If no directory is used this is relative to the NSClient++ binary.

date_mask	%Y-%m-%d %H:%M:%S	The date format used when logging to a file
root_folder	exe	Root folder if not absolute

debug

A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.

Default
: 0

Values

Value	Meaning
0	Don't log debug messages
1	Log debug messages

file

The file to write log data to. If no directory is used this is relative to the NSClient++ binary.

Default
: nsclient.log

date_mask

The date format used when logging to a file

Default
: %Y-%m-%d %H:%M:%S

root_folder

Root folder if not absolute

Default
: exe

Values

local-app-data	The file system directory that contains application data for all users. A typical path is C:\Documents and Settings\All Users\Application Data. This folder is used for application data that is not user specific. For example, an application can store a spell-check dictionary, a database of clip art, or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.
exe	Location of NSClient++ binary

NSClient Sections

NSClient Section

This is section is included from the following page NSClientListener/config

1. 1. 1. 1. Ovreview

Ovreview

This is the [NSClientListener] module configuration options.

Option	Default value	Description
port	12489	The port to listen to
obfuscated_password		An obfuscated version of password.
password		The password that incoming client needs to authorize themselves by.
allowed_hosts		A list (coma separated) with hosts that are allowed to connect to NSClient++ via NSClient protocol.
socket_timeout	30	The timeout when reading packets on incoming sockets.

Advanced options:

Option	Default value	Description
socket_back_log		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
bind_to_address		The address to bind to when listening to sockets, useful if you have more then one NIC/IP address and want the agent to answer on a specific one.
version	auto	The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

port

The port to listen to

Default
: 12489

obfuscated_password

An obfuscated version of password. For more details refer to the password option below.

Default
: Empty string whjich means we will use the value from password instead.

password

The password that incoming client needs to authorize themselves by. This option will replace the one found under Settings for NSClient. If this is blank the option found under Settings will be used. If both are blank everyone will be granted access.

Default
: Empty string whjich means we will use the value from password in the [Settings] section instead.

allowed_hosts

A list (coma separated) with hosts that are allowed to poll information from NSClient++. This will replace the one found under Setting for NSClient if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system.

BEWARE: NSClient++ will not resolve the IP address of DNS entries if the service is set to startup automatically. Use an IP address instead or set cache_allowed_hosts=0 see above.

Default
: Empty list (falls back to the one defined under [Settings]

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
: IP address of any interface of the server.

Default
: Empty (first (all?) interface will be used)

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
: 30 seconds

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

version

The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

Values:

If given any str4ing will be returned unless auto in which case the proper version will be returned

Default
auto

Check System Sections

CheckSystem Section

This is section is included from the following page CheckSystem/config

Overview

The configuration for the CheckSystsem? module should in most cases be automagically detected on most versions of windows (if you have a problem with this let me know so I can update it). Thus you no longer need to configure the advanced options. There is also some other tweaks that can be configured such as check resolution and buffer size.

Option	**Default value**	**Description**
CPUBufferSize	1h	The time to store CPU load data.
CheckResolution?	10	Time between checks in 1/10 of seconds.

Advanced options:

Option	**Default value**	**Description**
auto_detect_pdh	1	Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.
dont_use_pdh_index	0	Set this to 1 if you dont want to use indexes for finding PDH counters.
force_language		Set this to a locale ID if you want to force auto-detection of counters from that locale.
ProcessEnumerationMethod?	auto	Set the method to use when enumerating processes PSAPI, TOOLHELP or auto
check_all_services[<key>]	ignored	

		Set how to handle services set to <key> state when checking all services
MemoryCommitLimit?	\Memory\Commit Limit	Counter to use to check upper memory limit.
MemoryCommitByte?	\Memory\Committed Bytes	Counter to use to check current memory usage.
SystemSystemUpTime?	\System\System Up Time	Counter to use to check the uptime of the system.
SystemTotalProcessorTime?	\Processor(_total)\% Processor Time	Counter to use for CPU load.
debug_skip_data_collection	0	**DEBUG** Used to disable collection of data

CPUBufferSize

The time to store CPU load data. The larger the buffer the more memory is used. This is a time value which takes an optional suffix for which time denominator to use:

Suffix	Meaning
s	second
m	minutes
h	hour
d	day

Default

1h

CheckResolution?

Time between checks in 1/10 of seconds.

Default

10

auto_detect_pdh

Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.

Values

Value	Meaning
0	Don't attempt automagically detect the counter names used.
1	Use various menthods to figure out which counters to use.

Default

1

dont_use_pdh_index

When autodetecting counter names do **NOT** use index to figure out the values.

Values

Value	Meaning
0	Use indexes to automagically detect the counter names used.
1	Do **NOT** use indexes to figure out which counters to use.

Default
: 0

force_language

When index detection fails your local is used. Here you can override the default local to force another one if the detected local is incorrect.

Values
: Any locale string like SE_sv (***not sure here haven't used in years***)

Deafult
: Empty string which means the system local will be used.

ProcessEnumerationMethod?

DEPRECATED Set the method to use when enumerating processes PSAPI, TOOLHELP or auto No longer used (only PSAPI is supported).

check_all_services[<key>]

When using check all in a service check the default behaviour is that service set to auto-start should be started and services set to disabled should be stopped. This can be overridden using this option. Keys avalible:

Key	Default	Meaning
SERVICE_BOOT_START	ignored	TODO
SERVICE_SYSTEM_START	ignored	TODO
SERVICE_AUTO_START	started	TODO
SERVICE_DEMAND_START	ignored	TODO
SERVICE_DISABLED	stopped	TODO

MemoryCommitLimit?

Counter to use to check upper memory limit.

Default
: \Memory\Commit Limit

MemoryCommitByte?

Counter to use to check current memory usage.

Default
: \Memory\Committed Bytes

SystemSystemUpTime?

Counter to use to check the uptime of the system.

Default
\System\System Up Time

SystemTotalProcessorTime?

Counter to use for CPU load.

Default
\Processor(_total)\% Processor Time

debug_skip_data_collection

DEBUG Used to disable collection of data

Default
0

External Script Sections

External Script Section

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_script

Overview

Configure how the External Scripts module works (not to be confused with the "External Scripts" section below that holds scripts that can be run.

Option	Default value	Description
command_timeout	60	The maximum time in seconds that a command can execute.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the command line.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands.
script_dir		When set all files in this directory will be available as scripts. **WARNING**

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.

Values:

Any number (positive integer) representing time in seconds.

Default
60 (seconds).
Example
Set timeout to 120 seconds

```
[External Script]
command_timeout=120
```

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

Values

Value	Meaning
0	Disallow arguments for commands
1	Allow arguments for commands

Default
0 (false).
Example
Allow arguments

```
[External Script]
allow_arguments=1
```

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Values
This list contain all possible values

Value	Meaning
0	Disallow nasty arguments for commands
1	Allow nasty arguments for commands

Default

0 (false)

Example

Allow nasty arguments

```
[External Script]
allow_nasty_meta_chars=1
```

script_dir

When set all files in this directory will be available as scripts. This is pretty dangerous but can be a bit useful if you use many scripts and you are sure no one else can add files there.

Value

Any directory (can be relative to NSClient++)

Default

Empty (meaning no scripts are added)

Example

All scripts ending with bat in the scripts folder (of NSClient++ installation directory) will be added as scripts.

```
[External Script]
script_dir=.\scripts\*.bat
```

External Scripts Section

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_scripts

Overview

A list of scripts and their aliases available to run from the CheckExternalScripts module. Syntax is: **<command>=<script> <arguments>** for instance:

```
check_es_long=scripts\long.bat
check_es_ok=scripts\ok.bat
check_es_nok=scripts\nok.bat
check_vbs_sample=cscript.exe //T:30 //NoLogo scripts\check_vb.vbs
check_es_args=scripts\args.bat static $ARG1$ foo
```

To configure scripts that request arguments, use the following syntax:

```
check_script_with_arguments=scripts\script_with_arguments.bat $ARG1$ $ARG2$ $ARG3$
```

Use ./check_nrpe ... -c check_script_with_arguments -a arg1 arg2 arg3 ... Make sure you type $ARG1$ and not $arg1$ (case sensitive)

NOTICE For the above to work you need to enable allow_arguments in **both** NRPEListener and CheckExternalScripts!

External Alias Section

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_alias

1. 1. 1. 1. Overview

Overview

A simple and nifty way to define aliases in NSClient++. Aliases are good for defining commands locally or just to simply the nagios configuration. There is a series of "useful" aliases defined in the included configuration file which is a good place to start. An alias is an internal command that has been "wrapped" (to add arguments). If you want to create an alias for an external command you can do so but it still needs the normal defnition and the alias will use the internal alias of the external command.

WARNING Be careful so you don't create loops (ie check_loop=check_a, check_a=check_loop)

```
[External Aliases]
alias_cpu=checkCPU warn=80 crit=90 time=5m time=1m time=30s
alias_disk=CheckDriveSize MinWarn=10% MinCrit=5% CheckAll FilterType=FIXED
alias_service=checkServiceState CheckAll
alias_mem=checkMem MaxWarn=80% MaxCrit=90% ShowAll type=physical
```

Event Log Sections

Event Log Section

This is section is included from the following page CheckEventLog/config

EventLog?

The [EventLog?] section is used by the CheckEventLog module.

Advanced options:

Option	Default	Description
debug	0	Log all "hits" and "misses" on the eventlog filter chain, useful for debugging eventlog checks but very very very noisy so you don't want to accidentally set this on a real machine.
buffer_size	65536	Sets the buffer memory size used by NSClient++ when processing event log check commands. For details see below.

debug

Used to log all information regarding hits and misses on the filtering,. This has sever performance impact as well as log file will grow so do not use unless you are debugging.

```
[EventLog]
debug=1
```

buffer_size

This option was added in version 3.4

This parameter is set in the nsc.ini file and needs to be put under a heading of [EventLog?] (this heading may need to be created). The buffer reserves memory each time an eventlog check is being run when so set the size accordingly (or you will be wasting lots of memory).

To change the default setting of 64KB add (or edit) in the nsc.ini file an entry for buffer size (buffer_size=512000) where the value is in bytes. Often times the buffer size will need to be increased when using the %message% variable in return results. Most often you only need to increase this if you get error reported in the log file from NSClient++

```
[EventLog]
buffer_size=512000
```

Complete configuration

This are the default values for the entire EventLog? section

```
[EventLog]
debug=0
buffer_size=64000
```

NSCA Agent Sections

This page describes the configuration options for the NSCA module.

NSCA Agent Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Agent

Ovreview

Options to configure the NSCA module.

Option	Default value	Description
interval	60	Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")
nsca_host	...	The NSCA/Nagios(?) server to report results to.

nsca_port	5667	The NSCA server port
encryption_method	1	Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.
password		The password to use. Again has to be the same as the server or it won't work at all.

Advanced options:

Option	Default value	Description
hostname		The host name of this host if set to blank (default) the windows name of the computer will be used.
debug_threads	1	**DEBUG**Number of threads to run, no reason to change this really (unless you want to stress test something)

interval

Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")

Value
: Any positive integer (time in seconds)

Default
: 60 (seconds)

nsca_host

The NSCA/Nagios(?) server to report results to.

Values
: Hostname or IP address to submit back results to.

Default
: Empty string (will in 3.7 and above mean don't submit results)

nsca_port

The NSCA server port

Values
: Any positive integer (port number ought to be less then 65534)

Default
: 5667

encryption_method

Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.

Values

Supported encryption methods:

#	Algorithm

0	None (**Do NOT use** this option)
1	Simple XOR (**No security, just obfuscation**, but very fast)
2	DES
3	3DES (Triple DES)
4	CAST-128
6	xTEA
8	BLOWFISH
9	TWOFISH
11	RC2
14	RIJNDAEL-128 (AES)
20	SERPENT

Default
: 1 (I am note sure I thought default was 14?)

password

The password to use. Again has to be the same as the server or it won't work at all.

Values
: Any string (should be the same as the one configured in nsca.conf

hostname

The host name of this host if set to blank (default) the windows name of the computer will be used.

Values
: Any string (or auto)

Default
: auto (means windows hostname will be used)

debug_threads

DEBUGNumber of threads to run, no reason to change this really (unless you want to stress test something)

Values
: Any positive integer larger then or equal to 1

Default
: 1

NSCA Commands Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Commands

1.
   1.
      1.
         1. Overview

Overview

A list of commands to run and submit each time we report back to the NSCA server. A command starting with host_ will be submitted as a host command. For an example see below: This will report back one service check (called my_cpu_check) and one host check (host checks have no service name).

```
[NSCA Commands]
my_cpu_check=checkCPU warn=80 crit=90 time=20m time=10s time=4
host_check=check_ok
```

LUA Scripts

A list of LUA script to load at startup. In difference to "external checks" all LUA scripts are loaded at startup. Names have no meaning since the script (on boot) submit which commands are available and tie that to various functions.

```
[LUA Scripts]
scripts\test.lua
```

Problems

1. I am having problems where do I start?

NSCP has a built-in "test and debug" mode that you can activate with the following command

```
nsclient++ /test
```

What this does is two things.

1. it starts the deamon as "usual" with the same configuration and such.
2. it enables debug logging and logs to the console.

This makes it quite easy to see what is going on and why things go wrong. .

2. Failed to open performance counters

- The first thing to check is the version if you are using an old version (pre 0.3.x) upgrade!
- Second thing to check are your performance counters working? Sometimes the performance counters end up broken end need to be rebuilt See forum post: here for details or Microsoft KB: http://support.microsoft.com/kb/300956 essentially you need to use the "lodctr /R" command.

3. Bind failed

- Usually this is due to running more then once instance of NSClient++ or possibly running another program that uses the same port.
    - Make sure you don't have any other instance NSCLient++ started.
    - Check if the port is in use (netstat -a look for LISTENING)

4. "EvenlogBuffer?? is too small

- This is because you have one or more entries in your eventlog which are larger then the "default buffer size of 64k". The best way to fix this is to increase the buffer used.

```
[EventLog]
buffer_size=128000
```

NOTE: You should add it to the ini file by yourself.

There are hundreds of options not in the ini file (all covered in the docs though). The default ini is more a "common ones" and not a complete set.

the ini file that comes with the installation does not contain this variable by default.

5. How do I properly escape spaces in strings

When you need to put spaces in a string you do the following:

- nagios:
    - As usual you can do it anyway you like but I prefer: check_nrpe ... 'this is a string'
- NSClient++ (inject, alias, external, etc...)
    - The parser is badly written so the only option is:

      ```
      CHeckSomething "this is a string"
      CheckEventLog "filter-message=substr:Hello World"
      ```
    - **Not** for instance:

      ```
      filter-message="substr:Hello World"
      filter-message=substr:"Hello World"
      ```

6. How do I properly escape $ in strings

From:

- nagios:
    - $$ (you use two $ signs)
- from NSClient++
    - $ (you do not need to escape them at all)

7. System Tray does not work

Older WIndows

If you are using "older windows" (ie. XP and below) you can use the "old" sytray module like so:

```
[modules]
SysTray.dll
```

and then run:

```
NSClient++ -noboot SysTray install
```

"modern" windows

If you are using "modern" windows ie. vista, 2k3, 2k8, etc etc there is no "session 0" (or there is but you do not see it by default) so they sytray (which ends up on session 0) wont be visible by your session 1 (or above). Thus I started work on a new "modern implementation" this comes in the form of a shared session (based on shared memory and mutexes). **But since this is rather new it is very experimental** so use it with care! To enable shared session do the following:

```
[modules]
; SysTray.dll <--- NOTICE THE COMMENT

[Settings]
shared_session=1
```

Modules

NSClient++ comes with a few modules out of the box that perform various checks. A list of the modules and their potential use is listed below. Click each plug-in to see detailed command descriptions and how the various modules can be used.

CheckDisk

Module to do various disk related checks.

- CheckFileSize, Check the size of a file
- CheckDriveSize, Check the size of a fixed drive or mounted volume
- CheckFile, Check various aspects on one or more files or directories.

CheckEventLog

Module to check event log

- CheckEventLog, Check event log for errors

CheckSystem

Module to check system related things

- CheckCPU, Check CPU load averages
- CheckUpTime, Check system uptime
- CheckServiceState, Check State of a service
- CheckProcState, Check state of a process (application)
- CheckMem, Check state of memory (Page file)
- CheckCounter, Check performance counters

CheckHelpers

Various helper function, doesn't check anything in it self but can help make things simpler.

- CheckAlwaysOK, Runs another check and always returns OK regardless of result.
- CheckAlwaysCRITICAL, Runs another check and always returns CRITICAL regardless of result.
- CheckAlwaysWARNING, Runs another check and always returns WARNING regardless of result.
- CheckMultiple, Runs multiple checks and returns them all in one go.
- CheckOK, Always returns OK (useful for NSCA)
- CheckCRITICAL, Always returns CRITICAL (useful for NSCA)
- CheckWARNING, Always returns WARNING (useful for NSCA)
- CheckVersion, Returns the version of NSClient++

FileLogger

Logs all messages (errors, warnings etc) to a file.

NRPEListener

Listens for incoming NRPE calls and handles them by injecting them into the core. It also listens for all NRPE definitions and executes them

NSClientListener

Listens for incoming NSClient calls and handles them accordingly. This only allows a limited subset of functionality and NRPE is recommended.

SysTray

A simple module to show an icon in the tray when the service is running this module does not export any check commands.

CheckWMI

- CheckWMI, Check large resultsets from (for instance are there more then 5-rows matching criteria X, ie. more than 5 internet explorer processes which uses more than 123Mb memory).
- CheckWMIValue, Check the result of a query (ie. are the current memory utilization over X)

CheckTaskSched

- CheckTaskSched, Check if scheduled tasks are working/scheduled/*.

CheckExternalScripts

User defined check commands, allows writing check scripts in external languages (VB, batch, EXE, *).

LUAScript

User defined check commands, allows writing check scripts (and wrap others in) the Lua scripting language.

NSCAAgent

No check commands, has functions to send results from check_commands to a NSCA server.

RemoteConfiguration

DEPRECATED No check commands, has functions to manage the configuration remotely.

All Commands

A list of all commands (alphabetically).

- CheckAlwaysCRITICAL (check)
- CheckAlwaysOK (check)
- CheckAlwaysWARNING (check)
- CheckCPU (check)
- CheckCRITICAL (check)
- CheckCounter (check)
- CheckEventLog/CheckEventLog (check)
- CheckFile (check)
- CheckFileSize (check)
- CheckMem (check)
- CheckMultiple (check)
- CheckOK (check)
- CheckProcState (check)
- CheckServiceState (check)
- CheckTaskSched/CheckTaskSched (check)
- CheckUpTime (check)
- CheckVersion (check)
- CheckWARNING (check)
- CheckWMI/CheckWMI (check)
- CheckWMIValue (check)

CheckDisk.dll

The CheckDisk module has various disk and file related checks. You can either check disk drive and volume sizes as well as files and directories.

- CheckFileSize, Check the size of one or more files or directories.
- CheckDriveSize, Check the size of one or more Drives
- CheckFile, **DEPRECATED** Check various aspects on one or more files or directories.
- CheckFile2, Check various aspects on (one or) more files or directories.

Configuration Sections

This is a wrapper page the actual data is on the following page CheckDisk/config

This module has no configuration.

Configuration for the CheckDisk

This page describes the configuration options for the CheckDisk module.

This is a wrapper page the actual data is on the following page CheckDisk/config

Configuration Sections

This module has no configuration.

CheckFileSize

CheckFileSize is part of the wiki:CheckDisk module

This check does a recursive size calculation of the directory (or file) specified. A request has one or more options described in the table below. The order only matter in that the size has to be specified before the File option this because you can change the size for each drive by specifying multiple Size options.

Option	Values	Description
MaxWarn	size-value	The maximum size the directory is allowed before a warning state is returned.
MaxCrit	size-value	The maximum size the directory is allowed before a critical state is returned.
MinWarn	size-value	The minimum size the directory is allowed before a warning state is returned.
MinCrit	size-value	The minimum size the directory is allowed before a critical state is returned.
ShowAll	None	A Boolean flag to show size of directories that are not in an alarm state. If this is not specified only drives with an alarm state will be listed in the resulting string.
File	File or directory name	The name of the file or directory that should have its size calculated. Notice that large directory structures will take a long time to check.
File:<alias>	File or directory name	Same as the file option but using a short alias in the returned data.
debug		A boolean flag to set debug to true (0.3.8)
ignore-perf-data		A boolean flag to disable returning performance data

The *size-value* is a normal numeric-value with a unit postfix. The available postfixes are B for Byte, K for Kilobyte, M for Megabyte and finally G for Gigabyte.

Examples

Check the size of the windows directory

Check the size of the windows directory and make sure it stays below 1 gigabyte. **Sample Command:**

```
CheckFileSize ShowAll MaxWarn=1024M MaxCrit=4096M File:_WIN=c:\WINDOWS\*.*
```

```
WARNING: WIN: 2G (2325339822B)
```

Nagios Configuration:

```
define command {
  command_name <<CheckFileSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFileSize -a ShowAll MaxWarn=$ARG1$ Max
}
<<CheckFileSize>> 1024M!4096M!_WIN!c:\WINDOWS\*.*
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFileSize -a ShowAll MaxWarn=1024M MaxCrit=4096M File:_WIN=c:\WIN
```

Check the size of the pagefile.sys

Check the size of the pagefile.sys and make sure it stays above 1 gigabyte. **Sample Command:**

```
CheckFileSize ShowAll MinWarn=1G  MinCrit=512M File=c:\pagefile.sys
```

```
OK: c:\pagefile.sys: 1G (1610612736B)
```

Nagios Configuration:

```
define command {
  command_name <<CheckFileSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFileSize -a ShowAll MinWarn=$ARG2$  Mi
}
<<CheckFileSize>> 512M!1G
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFileSize -a ShowAll MinWarn=1G  MinCrit=512M File=c:\pagefile.sy
```

Multiple files

Sample of using individual size for multiple files. **Sample Command:**

```
CheckFileSize MaxWarn=2G MaxCrit=4G File=c:\\pagefile.sys MaxWarn=1K MaxCrit=512 File=c:\\boot.in
```

```
OK: all file sizes are within bounds.
```

Nagios Configuration:

```
define command {
  command_name <<CheckFileSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFileSize -a MaxWarn=2G MaxCrit=4G File
}
<<CheckFileSize>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFileSize -a MaxWarn=2G MaxCrit=4G File=c:\\pagefile.sys MaxWarn=
```

Single file

I have had to set this up like this for our Windows Server. **Sample Command:**

```
CheckFileSize MaxWarn=2G MaxCrit=4G File=c:\\pagefile.sys MaxWarn=1K MaxCrit=512 File=c:\\boot.in
```

```
OK: all file sizes are within bounds.
```

Nagios Configuration:

```
define command {
  command_name <<CheckFileSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFileSize -a MaxWarn=2G MaxCrit=4G File
}
<<CheckFileSize>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFileSize -a MaxWarn=2G MaxCrit=4G File=c:\\pagefile.sys MaxWarn=
```

Some exchange database thing

Sample Command:

```
CheckFileSize MaxWarn=13G MaxCrit=15.5G File=d:\\exchsrvr\\mdbdata\\priv1.edb
```

```
OK: all file sizes are within bounds.
```

Nagios Configuration:

```
define command {
  command_name <<CheckFileSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFileSize -a MaxWarn=$ARG1$ MaxCrit=$AR
}
<<CheckFileSize>> 13G!15.5G!d:\\exchsrvr\\mdbdata\\priv1.edb
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFileSize -a MaxWarn=13G MaxCrit=15.5G File=d:\\exchsrvr\\mdbdata
```

CheckDriveSize

CheckDriveSize is part of the CheckDisk module

This check verifies the size of various drives specified on the command line. A request has one or more options described in the table below. The order only matters in that the size has to be specified before the Drive option because you can change the size for each drive by specifying multiple Size options.

Option	**Values**	**Description**
MaxWarnFree	size-value or %	The maximum allowed free space for the drive(s).
MaxCritFree	size-value or %	The maximum allowed free space for the drive(s).
MinWarnFree	size-value or %	The minimum allowed free space for the drive(s).
MinCritFree	size-value or %	The minimum allowed free space for the drive(s).
MaxWarnUsed	size-value or %	The maximum allowed used space for the drive(s).
MaxCritUsed	size-value or %	The maximum allowed used space for the drive(s).
MinWarnUsed	size-value or %	The minimum allowed used space for the drive(s).
MinCritUsed	size-value or %	The minimum allowed used space for the drive(s).
ShowAll	Empty, long	If present will display information even if an item is not reporting a state. If set to long will display more information.
Drive	A Drive letter or the path of a mounted Volume	The letter of the drive to check.
FilterType	FIXED, CDROM, REMOVABLE, REMOTE	Filter for drive type to prevent checking drives of certain kinds (most useful when using CheckAll?). The default is FIXED
CheckAll	None,volumes	Check all available drives. Volume support is available since (0.3.8)
CheckAllOthers	None	Check all drives (matching Filters) except those specified in the Drive= clause.
ignore-perf-data		A boolean flag to disable returning performance data

The *size-value or %* is a normal numeric-value with an optional unit or percentage postfix to specify large sizes. The available postfixes are B for Byte, K for Kilobyte, M for Megabyte, G for Gigabyte and finally % for percent free space.

Examples

Check C:

Check the size of C:\ and make sure it has 10% free space:

Sample Command:

```
CheckDriveSize ShowAll MinWarnFree=10% MinCritFree=5% Drive=c:\
```

```
CRITICAL: C:: Total: 74.5G - Used: 71.2G (95%) - Free: 3.28G (5%) <critical
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a ShowAll MinWarnFree=$ARG2
}
<<CheckDriveSize>> c:,5%!10%
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a ShowAll MinWarnFree=10% MinCritFree=5% Drive=c:\
```

Volumes

To check the size of mounted volume c:\volumne_test and make sure it has 1M free space

Sample Command:

```
CheckDriveSize ShowAll MaxWarn=1M MaxCrit=2M Drive=c:\volumne_test

CRITICAL: C:: Total: 74.5G - Used: 71.2G (95%) - Free: 3.28G (5%) <critical
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a ShowAll MaxWarn=$ARG2$ Ma
}
<<CheckDriveSize>> c:\volumne_test!1M!2M
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a ShowAll MaxWarn=1M MaxCrit=2M Drive=c:\volumne_test
```

Volumes

To check the size of all volume has 1M free space

Sample Command:

```
CheckDriveSize ShowAll MaxWarn=1M MaxCrit=2M CheckAll=volumes

CRITICAL: C:: Total: 74.5G - Used: 71.2G (95%) - Free: 3.28G (5%) <critical
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a ShowAll MaxWarn=$ARG2$ Ma
}
<<CheckDriveSize>> c:\volumne_test!1M!2M
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a ShowAll MaxWarn=1M MaxCrit=2M CheckAll=volumes
```

All fixed and network disks

To check the size of all fixed and network drives and make sure they have at least 1gig free space

Sample Command:

```
CheckDriveSize MinWarn=50% MinCrit=25% CheckAll FilterType=FIXED FilterType=REMOTE

CRITICAL: C:\;76514398208;1073741824;536870912; D:\;199303897088;1073741824;536870912; X:\;354670
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a MinWarn=$ARG2$ MinCrit=$A
}
<<CheckDriveSize>> 25%!50%
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a MinWarn=50% MinCrit=25% CheckAll FilterType=FIXED F
```

Fixed and Network (ignore some)

Check all fixed and network drives but ignore C and F. Not sure about this (should be simpler ways)

Sample Command:

```
CheckDriveSize CheckAllOthers FilterType=FIXED FilterType=REMOTE MinWarn=25% MinCrit=50% Drive=C

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a CheckAllOthers FilterType
}
<<CheckDriveSize>> 25%!50%
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a CheckAllOthers FilterType=FIXED FilterType=REMOTE M
```

Checking UNC Paths

Important to not forget the trailing \.

Sample Command:

```
CheckDriveSize Drive=x:\ FilterType=REMOTE ShowAll MaxWarnUsed=90% MaxCritUsed=95%

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a Drive=x:\ FilterType=REMO
}
<<CheckDriveSize>> x:,90%!95%
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a Drive=x:\ FilterType=REMOTE ShowAll MaxWarnUsed=90%
```

Simple Config

Another example for a working config.

Sample Command:

```
CheckDriveSize Drive=c:\\volumes\\somevolume\\ ShowAll MaxWarnUsed=90% MaxCritUsed=95%

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckDriveSize>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckDriveSize -a Drive=c:\\volumes\\somevo
}
<<CheckDriveSize>> c:\\volumes\\somevolume\,90%!95%
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckDriveSize -a Drive=c:\\volumes\\somevolume\\ ShowAll MaxWarnUsed
```

CheckFile

A new command to check a bunch of files.

DEPRECATED (use CheckFile2 instead).

FOr details (if you want to use anyway) refer to the deprecated page.

CheckFile2

A command to check aspects on several files it can be used to check one file but that is not the goal. The core scenario is: "do I have more then x files matching this criteria?" but it is flexible enough to be applicable in many other scenarios as well.

The main concept is much like the eventlog checks where you have a data-set which you want to "filter" and then check the resulting number of lines against a criteria.

Option	Values	Description
truncate		Truncate return data (not performance data)
ignore-perf-data		A boolean flag to disable returning performance data
syntax	%filename%	The syntax of how each file is presented, can include the following tokens: %path%, %filename%, %creation%, %access%, %write%, %size%, %version%, %line-count%
master-syntax	%list%	The syntax for the returned message %list% (the list of syntax strings), %matches% (number of matched files), %files% (number of files total)
path		The path to check from (root)
pattern		The file pattern to check
alias		Alias for the check
file		Deprecated (will be split into path and pattern)
debug		Boolean flag to enable debugging
ignore-errors		Boolean flag to ignore any errors
max-dir-depth		Recursion depth
filter	in,out,all,any	Filter mode (see below)

The CheckFile2 command Uses filters to define the "interesting" files.

Syntax

A filter is made up of three things:

- Filter mode Determines what happens when the filter is matched.
- Filter type What the filter will match (ie. which field).
- An Expression What to check for.

The syntax of a filter is: *filter**<mode><type>=<expression>***

Order

Order is important, as soon as a positive (+) or negative (-) rule is matched it is either discarded or included and the entry is "finished" and it will continue with the next entry. The best way here is to have an "idea" either remove all entries first or include all required ones first (depending on what you want to do). You can mix and such but this will probably complicate things for you unless you actually need to.

Filter modes

Capturing files (or discarding them) are done with filters. There are three kinds of filters.

<filter mode>	title	description
+	positive requirements	All these filters must match or the row is discarded.
.	potential matches	If this matches the line is included (unless another lines overrides).
-	negative requirements	None of these filters can match (if any do the row is discarded).

Thus if you want to have: all files from the last month but not the ones smaller then 5kbI would break this down as such: (notice there are other options). - date=older than 2 months + size=larget then 5k This would discard all files older then 2 month and then include all files larger then 5kb.

Filter Types

<filter type>	Values	Description
size	number	The size of the file
creation	[[time expression]]	The date/time the file was created
written	[[time expression]]	The date/time the file was last updated
accessed	[[time expression]]	The date/time the file was last accessed
version	[[string expression]]	The exe file version
line-count	number	Number of lines (in a text file)

time expression

A **time expression** is a date/time interval as a number prefixed by a filter prefix (<, >, =, <>) and followed by a unit postfix (m, s, h, d, w). A few examples of time expression are: filter+generated=>2d means filter will match any records older than 2 days, filter+generated=<2h means match any records newer then 2 hours. Warning, the bash interprets the "<,>,!". Use the "\" to avoid this. e.g. filter+generated=\>2d . On the Client activate the "Nasty Metachars" Option, to allow the \.

string expression

A **string expression** is a key followed by a string that specifies a string expression. Currently substr and regexp are supported. Thus you enter filter.message=regexp:(foo|bar) to enter a regular expression and filter-message=substr:foo to enter a substring patter match.

Filter in/out

There are two basic ways to filter:

- in When you filter in it means all records matching your filter will be returned (the "simplest way")
- out When you filter out it means all records matching your filter will be discarded.

So:

```
filter=in filter+size==5k
...
filter=out filter-size=ne:5k
```

Will both have the same effect as the first one filters "in" and matches all siles with 5kb and the second one filters out and discards all files not 5kb.

Sample Command:

```
CheckFile2 path=c:\test pattern=*.txt MaxCrit=1 filter+written=gt:2h

ok: CheckFile ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=$ARG1$ pattern=*.txt Max
}
<<CheckFile2>> c:\test!2h
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=c:\test pattern=*.txt MaxCrit=1 filter+written=gt:
```

Some more Examples (for 0.3.8)

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.exe filter+version=!=1.0 "syntax=%filename%: %version%" MaxWarn=

warning: WARNING:explorer.exe: 6.0.2900.5512
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.exe fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.exe filter+version=!=1.0 "syntax=
```

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.exe filter+version=!=6.0.2900.5512 "syntax=%filename%: %version%

ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.exe fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.exe filter+version=!=6.0.2900.551
```

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.txt filter+line-count=!=2 "syntax=%filename%: %line-count%" MaxW
```

```
warning: WARNING:test.txt: 3
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt filter+line-count=!=2 "syntax
```

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.txt filter+line-count=ne:3 "syntax=%filename%: %line-count%" Max
```

```
ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt filter+line-count=ne:3 "synta
```

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.txt filter+size=gt:20 "syntax=%filename%: %size%" MaxWarn=1
```

```
warning: WARNING:test.txt: 26B
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt filter+size=gt:20 "syntax=%fi
```

Sample Command:

```
CheckFile2 path=D:\tmp\empty pattern=*.tx filter+size=gt:20 "syntax=%filename%: %size%" MaxWarn=1
```

```
ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp\empty pattern=*.t
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp\empty pattern=*.tx filter+size=gt:20 "synta
```

Sample Command:

```
CheckFile2 path=D:\tmp pattern=*.txt filter+size=gt:30 "syntax=%filename%: %size%" MaxWarn=1
```

```
ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt fil
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp pattern=*.txt filter+size=gt:30 "syntax=%fi
```

Sample Command:

```
CheckFile2 path=D:\tmp\empty pattern=*.tx filter+size=gt:30 "syntax=%filename%: %size%" MaxWarn=1
```

```
ok
```

Nagios Configuration:

```
define command {
  command_name <<CheckFile2>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckFile2 -a path=D:\tmp\empty pattern=*.t
}
<<CheckFile2>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckFile2 -a path=D:\tmp\empty pattern=*.tx filter+size=gt:30 "synta
```

CheckEventLog.dll

The CheckEventLog module checks for problems reported to the windows event log.

- CheckEventLog, Check to find errors in the event log.

Configuration Sections

This is a wrapper page the actual data is on the following page CheckEventLog/config

EventLog?

The [EventLog?] section is used by the CheckEventLog module.

Advanced options:

Option	Default	Description
debug	0	Log all "hits" and "misses" on the eventlog filter chain, useful for debugging eventlog checks but very very very noisy so you don't want to accidentally set this on a real machine.
buffer_size	65536	Sets the buffer memory size used by NSClient++ when processing event log check commands. For details see below.

debug

Used to log all information regarding hits and misses on the filtering,. This has sever performance impact as well as log file will grow so do not use unless you are debugging.

```
[EventLog]
debug=1
```

buffer_size

This option was added in version 3.4

This parameter is set in the nsc.ini file and needs to be put under a heading of [EventLog?] (this heading may need to be created). The buffer reserves memory each time an eventlog check is being run when so set the size accordingly (or you will be wasting lots of memory).

To change the default setting of 64KB add (or edit) in the nsc.ini file an entry for buffer size (buffer_size=512000) where the value is in bytes. Often times the buffer size will need to be increased when using the %message% variable in return results. Most often you only need to increase this if you get error reported in the log file from NSClient++

```
[EventLog]
buffer_size=512000
```

Complete configuration

This are the default values for the entire EventLog? section

```
[EventLog]
debug=0
buffer_size=64000
```

Configuration for the CheckEventLog

This page describes the configuration options for the CheckEventLog module.

This is a wrapper page the actual data is on the following page CheckEventLog/config

Configuration Sections

EventLog?

The [EventLog?] section is used by the CheckEventLog module.

Advanced options:

Option	Default	Description
debug	0	Log all "hits" and "misses" on the eventlog filter chain, useful for debugging eventlog checks but very very very noisy so you don't want to accidentally set this on a real machine.
buffer_size	65536	Sets the buffer memory size used by NSClient++ when processing event log check commands. For details see below.

debug

Used to log all information regarding hits and misses on the filtering,. This has sever performance impact as well as log file will grow so do not use unless you are debugging.

```
[EventLog]
debug=1
```

buffer_size

This option was added in version 3.4

This parameter is set in the nsc.ini file and needs to be put under a heading of [EventLog?] (this heading may need to be created). The buffer reserves memory each time an eventlog check is being run when so set the size accordingly (or you will be wasting lots of memory).

To change the default setting of 64KB add (or edit) in the nsc.ini file an entry for buffer size (buffer_size=512000) where the value is in bytes. Often times the buffer size will need to be increased when using the %message% variable in return results. Most often you only need to increase this if you get error reported in the log file from NSClient++

```
[EventLog]
buffer_size=512000
```

Complete configuration

This are the default values for the entire EventLog? section

```
[EventLog]
```

```
debug=0
buffer_size=64000
```

CheckEventLog

Check eventlog has many "versions" this page reflects the brand new (still **Experimental**) version. For the previous version see [this page].

CheckEventLog is part of the wiki:CheckEventLog module.

This check enumerates all event in the event log and filters out (or in) events and then the resulting list is used to determine state.

Option	Values	Description
file	An event log file name(application, security, system, etc.)	The name of an eventlog file the default ones are Application, Security and System. If the specified eventlog was not found due to some idiotic reason windows opens the "application" log instead.
filter	filter expression	The filter expression
descriptions	None	Flag to specify if you want to include string representation of the error messages.
truncate	length of the returned set	This will truncate the output after the specified length. As NRPE can only handle 1024 chars you need to truncate the output.
MaxWarn	number of records	The maximum records to allow before reporting a warning state.
MaxCrit	number of records	The maximum records to allow before reporting a critical state.
syntax	String	A string to use to represent each matched eventlog entry the following keywords will be replaced with corresponding values: %source%, %generated%, %written%, %type%, %severity%, %strings%, %id% and %message% (%message% requires you to set the *description* flag.) %count% (requires the unique flag) can be used to display a count of the records returned.
unique		Flag to indicate unique filtering is used.

Filter Keywords

Keyword	Values	Description
type	*event type expression*	An event type to filter out: error, warning, info, auditSuccess or auditFailure.
source	string-expression	The name of the source of the event.
generated	time-expression	Time ago the message was generated
written	time-expression	Time ago the message was written to the log
message	string-expression	Filter strings in the message. Can be a substring or regularexpression
id	numeric-expression	Filter based on the event id of the log message.
severity	*event severity expression*	Filter based on event severity: error, warning, informational

Operators

Operator	Safe	Description
=	eq	Equality (**a = b**) or (**a eq b**)
!=	ne	Not equal (**a <= b**) or (**a le b**)

>	gt	Greater then (**a > b**) or (**a gt b**)
<	lt	Less then (**a < b**) or (**a lt b**)
>=	ge	Greater then or equal (**a >= b**) or (**a ge b**)
<=	le	Less then or equal (**a <= b**) or (**a le b**)
LIKE		Compare to strings using substring matching (**'a' like 'apple'**)
AND		Both statements has to be true (**a AND b**)
OR		Either statement has to be true (**a OR b**)
IN		See if a value matches a given list (**'a' IN (123, 456, 789)**)
NOT IN		See if a value does not matches a given list (**'a' IN (123, 456, 789)**)

Writing Filters

Filters are based upon SQL Where clauses. For instance the following is a valid filter:

```
severity = 'error'
```

This is interpreted as "severity" has to be an "error".

Apart from the various keywords (and their valid values) there is a set of operators which can be used to construct expressions. Thus you can extend the previous to

```
severity = 'error' OR severity = 'informational' OR severity = 'warning'
```

You can also use parenthesis to group the order

```
severity = 'error' AND (severity = 'informational' OR severity = 'warning')
```

Using Keywords

Since they keywords are handled a bit differently I will write up a short section on each one of them.

id (event id)

The event id is a unique identifier identifying a message "Number" inside an application. Thus it is uniq together with an event source. Event id is a number which means it unsurprisingly can be compared with other numbers like 1 or 2 or even 3 or perhaps even 4 (not sure about 5 though) and lets not forget the ohh so hyped 42. Comparing numbers are done using the equality or no equality operators (= and !=) you can also use the in and not in grouping operators.

The simplest usage is: **id = 1008**

```
og file=application debug=true MaxWarn=1 MaxCrit=1 "filter=id = 1008" truncate=800 unique descriptions "syntax
08: (94), 1008: (1), eventlog: 95 > critical|'eventlog'=95;1;1;
```

The reason we get "More then one" 1008 result back is because different sources have generated the same error number (remember id AND source) is unique not by them selves. So adding source to the syntax shows us this: **id = 1008**

```
ile=application debug=true MaxWarn=1 MaxCrit=1 "filter=id = 1008" truncate=800 unique descriptions "syntax=%sou
er Experience Improvement Program 1008: (94), Windows Search Service 1008: (1), eventlog: 95 > critical|'eventl
```

The next thing we can do to extend this is to make a slightly more complicated filter: **id = 1008 OR id = 1005 OR id = 123 OR id = 8224**

```
n debug=true MaxWarn=1 MaxCrit=1 "filter=id = 1008 OR id = 1005 OR id = 123 OR id = 8224" truncate=800 unique d
3), 1008: (94), 1008: (1), 8224: (164), eventlog: 379 > critical|'eventlog'=379;1;1;
```

This can of cource be simplified (in this case) using the IN operator: **id IN (1008, 1005, 123, 8224)**

```
pplication debug=true MaxWarn=1 MaxCrit=1 "filter=id IN (1008, 1005, 123, 8224)" truncate=800 unique descriptio
1005: (118), 1008: (94), 1008: (1), 8224: (164), eventlog: 379 > critical|'eventlog'=379;1;1;
```

The opposite can also be used like so: **id NOT IN (1008, 1005, 123, 8224)**

```
ication debug=true MaxWarn=1 MaxCrit=1 "filter=id NOT IN (1008, 1005, 123, 8224)" truncate=800 unique descript
': (12), 3408...|'eventlog'=5435;1;1;
```

source (program generating the event)

Next up is the event source it is set to a string value representing the program generating the event. A string can be compared to other strings and in addition to the equality and non equality (= and !=) like makes a lot of sense to match sub strings. You can also use the IN and NOT IN if you want to specify more then one application but this, I guess, will make less sense. Notice that there is **NO** regular expression yet this will be added in the next version.

The simplest example here is: **source = 'MsiInstaller?'**

```
application debug=true MaxWarn=1 MaxCrit=1 "filter=source = 'MsiInstaller'" truncate=800 unique descriptions "s
er: (78), MsiInstaller: (2), MsiInstaller: (178), ... MsiInstaller: (1), eventlog: 1347 > critical|'eventlog'=1
```

Then we can try using the like sub string matching like so: **source like 's'**

```
e=application debug=true MaxWarn=1 MaxCrit=1 "filter=source like 's'" truncate=800 unique descriptions "syntax=
51), Microsoft-Windows-ApplicationExperienceInfrastructure: (4), SQLBrowser: (26),... MsiInstaller: (254), Mic
```

An important not about the like operator is that it is the same from both sides ie: 'hello' like 'h' and 'h' like 'hello' are both true.

Using the in list operator like so: **source IN ('idsvc', 'MsiInstaller?')**

```
ation debug=true MaxWarn=1 MaxCrit=1 "filter=source IN ('idsvc', 'MsiInstaller')" truncate=800 unique descripti
nstaller: (78), MsiInstaller: (2), ... MsiInstaller: (1), eventlog: 1348 > critical|'eventlog'=1348;1;1;
```

generated (when the event was generated)

This is the "hardest" in terms of using I guess. Dates are treated as number internally and thus works much like a number. The added value that numbers expose are the ability to translate numbers to dates. This happends by adding a suffix ti a number. For instance **2d** is read as **2 days**. And **5h** is read as **5 hours**. The other important aspect of using dates are how negative numbers are used. Normal dates **5d** will be interpreted as "<now> + 5 days". But when you add a negation sign (-) before it will be negated (ish). And this is the crux. What actually happens is that **-5d** is treated as **neg(5 days from now)** and neg negates the date around "now" so in effect it becomes **5 days ago**. This might sound complicated but it not really and you probably don't need to understand it but I think explains why < and > will work.

Thus the effect of this is that you can do: **generated > -5d'**

```
e=application debug=true MaxWarn=1 MaxCrit=1 "filter=generated > -5d" truncate=800 unique descriptions "syntax=
, May 13, 2010 03:18:46: (11), ...|'eventlog'=276;1;1;
```

Written

For details see generated as they work the same. This is the date when the event was written to the log.

Severity

Type

Message

Strings

Examples

```
 MaxCrit=1 "filter=generated gt -2d AND severity NOT IN ('success', 'informational')" truncate=800 unique desc
```

CheckEventLog

CheckEventLog is part of the wiki:CheckEventLog module. This page describes the new syntax, for the old syntax refer to the old page: CheckEventLogOld The new syntax is a bit sketchy in the docs as of yet... I shall try to fix some better examples.. but the best idea would be for someone that uses this to help me with that :)

Before you start using CheckEventLog use this command (it is long but a good place to start):

```
CheckEventLog file=application file=system filter=new filter=out
      MaxWarn=1 MaxCrit=1
      filter-generated=>2d filter-severity==success filter-severity==informational
      truncate=1023 unique descriptions "syntax=%severity%: %source%: %message% (%count%)"
```

This check enumerates all event in the event log and filters out (or in) events and then the resulting list is used to determine state.

Option	Values	Description
file	An event log file name(application, security, system, etc.)	The name of an eventlog file the default ones are Application, Security and System. If the specified eventlog was not found due to some idiotic reason windows opens the "application" log instead.
filter	in, out, any, all	Specify the way you want to filter things. (See section below)
filter	new	Has to be set to use this syntax
descriptions	None	Flag to specify if you want to include string representation of the error messages.
truncate	length of the returned set	This will truncate the output after the specified length. As NRPE can only handle 1024 chars you need to truncate the output.
MaxWarn	number of records	The maximum records to allow before reporting a warning state.
MaxCrit	number of records	The maximum records to allow before reporting a critical state.
syntax	String	A string to use to represent each matched eventlog entry the following keywords will be replaced with corresponding values: %source%, %generated%, %written%, %type%, %severity%, %strings%, %id% and %message% (%message% requires you to set the *description* flag.) %count% (requires the unique flag) can be used to display a count of the records returned.
filter<mode><type>	<filter value>	A number of strings to use for filtering the event log
unique		Flag to indicate unique filtering is used.

The CheckEventLog uses filters to define the "interesting" records from the eventlog.

Syntax

A filter is made up of three things:

- Filter mode Determines what happens when the filter is matched.

- Filter type What the filter will match (ie. which field).
- An Expression What to check for.

The syntax of a filter is: *filter**<mode><type>=<expression>***

Order

Order is important, as soon as a positive (+) or negative (-) rule is matched it is either discarded or included and the entry is "finished" and it will continue with the next entry. The best way here is to have an "idea" either remove all entries first or include all required ones first (depending on what you want to do). You can mix and such but this will probably complicate things for you unless you actually need to.

Filter modes

Capturing eventlog entries (or discarding them) are done with filters. There are three kinds of filters.

<filter mode>	**title**	**description**
+	positive requirements	All these filters must match or the row is discarded.
.	potential matches	If this matches the line is included (unless another lines overrides).
-	negative requirements	None of these filters can match (if any do the row is discarded).

Thus if you want to have: all errors and entries from the last month but not the ones from the cdrom, but if the source is MyModule? get everything. I would break this down as such: (notice there are other options). + type=error - date=older than 2 months . source=MyModule? This would pick up all errors, and drop all old records and then pickup all remaining "MyModule?" records (in this case you could have used + on the source filter since there are no more rules).

other example to simplify it: if for example you want to monitor all errors and to ignore warning and success in the eventlog you can write the following: filter+severity==error filter-severity==success filter-severity==informational

and the command with those parameters with others can be like the following: CheckEventLog file=application file=system filter=new filter=out MaxWarn=1 MaxCrit=1 filter-generated=>2d filter+severity==error filter-severity==success filter-severity==informational truncate=1023 unique descriptions "syntax=%severity%: %source%: %message% (%count%)"

Filter Types

<filter type>	**Values**	**Description**
eventType	*event type expression*	An event type to filter out: error, warning, info, auditSuccess or auditFailure. Note that unlike other commands, this requires '==', for example filter-eventType==info. The info,error, etc are all case sensitive.
eventSource	string-expression	The name of the source of the event. Can be a substring or regularexpression
generated	time-expression	Time ago the message was generated
written	time-expression	Time ago the message was written to the log
message	string-expression	Filter strings in the message. Can be a substring or regularexpression

eventID	numeric-expression	Filter based on the event id of the log message.
severity	*event severity expression*	Filter based on event severity. (filter-severity==warning)

event type expression

An **event type expression** is similar to a numeric-expression but instead of a number a "keyword" is taken: error, warning, info, auditSuccess, auditFailure. So *filter.eventType==warning* or *filter.eventType=<>warning* are examples of event type expressions. Yes this is correct the syntax is: *filter*___<mode><type>___*=*___<expression>___ in this case <mode> is ".", <type> is "eventType" and <expression> is "<>warning". This **IS** confusing but it is "simpler to parse" some day maybe I shall improve this.

filter<key><

event severity expression

An **event severity expression** is similar to a numeric-expression but instead of a number a "keyword" is taken: success, informational, warning or error

time expression

A **time expression** is a date/time interval as a number prefixed by a filter prefix (<, >, =, <>) and followed by a unit postfix (m, s, h, d, w). A few examples of time expression are: filter+generated=>2d means filter will match any records older than 2 days, filter+generated=<2h means match any records newer then 2 hours. Warning, the bash interprets the "<,>,!". Use the "\" to avoid this. e.g. filter+generated=\>2d . On the Client activate the "Nasty Metachars" Option, to allow the \.

string expression

A **string expression** is a key followed by a string that specifies a string expression. Currently substr and regexp are supported. Thus you enter filter.message=regexp:(foo|bar) to enter a regular expression and filter-message=substr:foo to enter a substring patter match.

Filter in/out

There are two basic ways to filter:

- in When you filter in it means all records matching your filter will be returned (the "simplest way")
- out When you filter out it means all records matching your filter will be discarded.

So:

```
filter=in filter+eventType==warning
...
filter=out filter-eventType==warning
```

Will both have the same effect as the first one filters "in" and matches all warnings and the second one filters out and discards all warnings. There is one very fundamental difference though the first one will **only return the warnings** where as the second one will return **all entries and all warnings**.

Unique

When *unique* is present any duplicate entries matching the filter will be discarded so you will only get back one of each "kind" of error. Uniqueness is determined by log-file, event-id, event-type and event-category.

Examples

Sample Eventlog Command

Check by EventID for target errors that may have transpired over the past 2 hours.
$ARG1$ = file to check ie. Application, Security, System
$ARG2$ = Max Warn amount
$ARG3$ = Max Critical amount
$ARG4$ = eventID Number

Sample Command:

```
CheckEventLog filter=new file="application" MaxWarn=10 MaxCrit=20 filter-generated=

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckEventLog>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckEventLog -a filter=new file="$ARG1$" M
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckEventLog -a filter=new file="application" MaxWarn=10 MaxCrit=20
```

Another sample

Check the Application event log for errors over the past 48 hours. Filter out any Cdrom and NSClient Errors as well as all Warnings. Allow 3 target Errors before firing a Warning, and 7 Errors before firing a Critical State.

This is the corresponding command: **Sample Command:**

```
CheckEventLog filter=new file=system file=application MaxWarn=1 MaxCrit=1 filter-generated=>2d fi

CRITICAL: 27 > critical: ESENT
```

Nagios Configuration:

```
define command {
  command_name <<CheckEventLog>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckEventLog -a filter=new file=system fil
}
<<CheckEventLog>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckEventLog -a filter=new file=system file=application MaxWarn=1 Ma
```

Please note: You need to allow_nasty_meta_chars=1 in the NSC.ini to use time filters like "<2d" (last 48 hours).

Check if a script is running as it should

Just to show a 'hidden' parameter ... I check that a script has successfully finished by writing into the eventlog. If after 1 day there is no new log entry, I get the message from Nagios.

Sample Command:

```
CheckEventLog filter=new file=application MinWarn=0 MinCrit=0 filter-generated=\>1d filter+eventS

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckEventLog>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckEventLog -a filter=new file=applicatio
}
<<CheckEventLog>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckEventLog -a filter=new file=application MinWarn=0 MinCrit=0 filt
```

Don't understand filtering ?

Yes I know, there are a lot of options regarding filtering, and they are a bit hard understand. This section tries to give a more formal definition of what the various options do (form a programming perspective).

Options

There are three different option pairs all used with the same key filter

1. filter=new / filter=old This one decides the code which is used there are two completely different concepts and the "Old" one is preferably not to be used as it is slightly less sane.
2. filter=all / filter=any This is **NOT** used with filter=new so this option is deprecated and should not be used unless you are using filter=old.
3. filter=in / filter=out This option decides what happens if "nothing matches" if you have filter=in that means if nothing matches you will still get the option where as if you have filter=out you wont. So this is the last thing that happens in the code.

So what you end up with is:

- ~~filter=new~~ this is default in newer versions, so you don't really need this anymore.
- ~~filter=old~~ Should not be used any more.
- ~~filter=any~~ is not used any more.
- ~~filter=all~~ is not used any more.
- filter=in Means you want everything **except** "something"
- filter=out Means you **only** want "something"

Rules

There are three different filter rule types all used in the same way except swapping the - for + for .

1. filter+severity==error This means that any entries matching this rule is automatically included in the result. After this we instantly stop matching more rules for this entry.
2. filter-severity==error This means that any entries matching this rule is automatically excluded in the result. After this we instantly stop matching more rules for this entry.
3. filter.severity==error This means that any entries matching this rule is neither automatically excluded nor automatically included in the result. **But** after this rule we **continue** matching more more rules for this entry. Since this is the default behavior with filter=in there is no reason to use filter=. in this mode.

How it works

Pseudo code (filter=new)

This is how the filtering is decided:

- bFilterIn=true for filter=in and false for filter=out

```
bool bMatch = !bFilterIn;
```

```
for(<each rule>) {
        bTmpMatched=<result of rule evaluation>
        if ((mode == filter_minus)&&(bTmpMatched)) {
                // a -<filter> hit so thrash item and bail out!
                bMatch = false;
                break;
        } else if ((mode == filter_plus)&&(!bTmpMatched)) {
                // a +<filter> missed hit so thrash item and bail out!
                bMatch = false;
                break;
        } else if (bTmpMatched) {
                bMatch = true;
        }
}
if (bMatch) {
        <deciding factor> = true;
}
```

CheckSystem.dll

A module to check various system related things. A list of the modules and there potential use is listed below here.

- wiki:CheckCPU, Check CPU load
- wiki:CheckUpTime, Check system uptime
- wiki:CheckServiceState, Check state of a service
- wiki:CheckProcState, Check state of a process
- wiki:CheckMem, Check memory usage (page)

Command Line

To simplify debug and setup there is two commandline options that list and test all avalible PDH counters.

- NSClient++ -noboot CheckSystem debugpdh
- NSClient++ -noboot CheckSystem listpdh
- NSClient++ -noboot CheckSystem pdhlookup <part of counter name>
- NSClient++ -noboot CheckSystem pdhmatch <pattern>
- NSClient++ -noboot CheckSystem pdhobject <counter root>

Configuration Sections

CheckSystem Section

This is a wrapper page the actual data is on the following page CheckSystem/config

Overview

The configuration for the CheckSystsem? module should in most cases be automagically detected on most versions of windows (if you have a problem with this let me know so I can update it). Thus you no longer need to configure the advanced options. There is also some other tweaks that can be configured such as check resolution and buffer size.

Option	**Default value**	**Description**
CPUBufferSize	1h	The time to store CPU load data.
CheckResolution?	10	Time between checks in 1/10 of seconds.

Advanced options:

Option	**Default value**	**Description**
auto_detect_pdh	1	Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.
dont_use_pdh_index	0	Set this to 1 if you dont want to use indexes for finding PDH counters.
force_language		Set this to a locale ID if you want to force auto-detection of counters from that locale.
ProcessEnumerationMethod?	auto	Set the method to use when enumerating processes PSAPI, TOOLHELP or auto
check_all_services[<key>]	ignored	Set how to handle services set to <key> state when checking all services
MemoryCommitLimit?	\Memory\Commit Limit	Counter to use to check upper memory limit.
MemoryCommitByte?	\Memory\Committed Bytes	Counter to use to check current memory usage.
SystemSystemUpTime?	\System\System Up Time	Counter to use to check the uptime of the system.
SystemTotalProcessorTime?	\Processor(_total)\% Processor Time	Counter to use for CPU load.
debug_skip_data_collection	0	**DEBUG** Used to disable collection of data

CPUBufferSize

The time to store CPU load data. The larger the buffer the more memory is used. This is a time value which takes an optional suffix for which time denominator to use:

Suffix	Meaning
s	second
m	minutes
h	hour
d	day

Default
1h

CheckResolution?

Time between checks in 1/10 of seconds.

Default
10

auto_detect_pdh

Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.

Values

Value	Meaning
0	Don't attempt automagically detect the counter names used.
1	Use various menthods to figure out which counters to use.

Default
1

dont_use_pdh_index

When autodetecting counter names do **NOT** use index to figure out the values.

Values

Value	Meaning
0	Use indexes to automagically detect the counter names used.
1	Do **NOT** use indexes to figure out which counters to use.

Default
0

force_language

When index detection fails your local is used. Here you can override the default local to force another one if the detected local is incorrect.

Values
Any locale string like SE_sv (***not sure here haven't used in years***)
Deafult
Empty string which means the system local will be used.

ProcessEnumerationMethod?

DEPRECATED Set the method to use when enumerating processes PSAPI, TOOLHELP or auto No longer used (only PSAPI is supported).

check_all_services[<key>]

When using check all in a service check the default behaviour is that service set to auto-start should be started and services set to disabled should be stopped. This can be overridden using this option. Keys avalible:

Key	Default	Meaning
SERVICE_BOOT_START	ignored	TODO
SERVICE_SYSTEM_START	ignored	TODO
SERVICE_AUTO_START	started	TODO
SERVICE_DEMAND_START	ignored	TODO
SERVICE_DISABLED	stopped	TODO

MemoryCommitLimit?

Counter to use to check upper memory limit.

Default

\Memory\Commit Limit

MemoryCommitByte?

Counter to use to check current memory usage.

Default

\Memory\Committed Bytes

SystemSystemUpTime?

Counter to use to check the uptime of the system.

Default

\System\System Up Time

SystemTotalProcessorTime?

Counter to use for CPU load.

Default

\Processor(_total)\% Processor Time

debug_skip_data_collection

DEBUG Used to disable collection of data

Default

0

Configuration for the CheckSystem

This page describes the configuration options for the CheckSystem module.

CheckSystem Section

This is a wrapper page the actual data is on the following page CheckSystem/config

Overview

The configuration for the CheckSystsem? module should in most cases be automagically detected on most versions of windows (if you have a problem with this let me know so I can update it). Thus you no longer need to configure the advanced options. There is also some other tweaks that can be configured such as check resolution and buffer size.

Option	Default value	Description
CPUBufferSize	1h	The time to store CPU load data.
CheckResolution?	10	Time between checks in 1/10 of seconds.

Advanced options:

Option	Default value	Description
auto_detect_pdh	1	Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.
dont_use_pdh_index	0	Set this to 1 if you dont want to use indexes for finding PDH counters.
force_language		Set this to a locale ID if you want to force auto-detection of counters from that locale.
ProcessEnumerationMethod?	auto	Set the method to use when enumerating processes PSAPI, TOOLHELP or auto
check_all_services[<key>]	ignored	Set how to handle services set to <key> state when checking all services
MemoryCommitLimit?	\Memory\Commit Limit	Counter to use to check upper memory limit.
MemoryCommitByte?	\Memory\Committed Bytes	Counter to use to check current memory usage.

SystemSystemUpTime?	\System\System Up Time	Counter to use to check the uptime of the system.
SystemTotalProcessorTime?	\Processor(_total)\% Processor Time	Counter to use for CPU load.
debug_skip_data_collection	0	**DEBUG** Used to disable collection of data

CPUBufferSize

The time to store CPU load data. The larger the buffer the more memory is used. This is a time value which takes an optional suffix for which time denominator to use:

Suffix	Meaning
s	second
m	minutes
h	hour
d	day

Default
1h

CheckResolution?

Time between checks in 1/10 of seconds.

Default
10

auto_detect_pdh

Set this to 0 to disable auto detect (counters.defs) PDH language and OS version.

Values

Value	Meaning
0	Don't attempt automagically detect the counter names used.
1	Use various menthods to figure out which counters to use.

Default
1

dont_use_pdh_index

When autodetecting counter names do **NOT** use index to figure out the values.

Values

Value	Meaning
0	Use indexes to automagically detect the counter names used.
1	Do **NOT** use indexes to figure out which counters to use.

Default

0

force_language

When index detection fails your local is used. Here you can override the default local to force another one if the detected local is incorrect.

Values

Any locale string like SE_sv (***not sure here haven't used in years***)

Deafult

Empty string which means the system local will be used.

ProcessEnumerationMethod?

DEPRECATED Set the method to use when enumerating processes PSAPI, TOOLHELP or auto No longer used (only PSAPI is supported).

check_all_services[<key>]

When using check all in a service check the default behaviour is that service set to auto-start should be started and services set to disabled should be stopped. This can be overridden using this option. Keys avalible:

Key	Default	Meaning
SERVICE_BOOT_START	ignored	TODO
SERVICE_SYSTEM_START	ignored	TODO
SERVICE_AUTO_START	started	TODO
SERVICE_DEMAND_START	ignored	TODO
SERVICE_DISABLED	stopped	TODO

MemoryCommitLimit?

Counter to use to check upper memory limit.

Default

\Memory\Commit Limit

MemoryCommitByte?

Counter to use to check current memory usage.

Default

\Memory\Committed Bytes

SystemSystemUpTime?

Counter to use to check the uptime of the system.

Default

\System\System Up Time

SystemTotalProcessorTime?

Counter to use for CPU load.

Default

\Processor(_total)\% Processor Time

debug_skip_data_collection

DEBUG Used to disable collection of data

Default

0

CheckCPU

CheckCPU is part of the wiki:CheckSystem module.

This check calculates an average of CPU usage for a specified period of time. The data is always collected in the background and the size and interval is configured from the CPUBufferSize and CheckResolution? options. A request has one or more options described in the table below.

Option	Values	Description
warn	load in %	Load to go above to generate a warning.
crit	load in %	Load to go above to generate a critical state.
Time	time with optional prefix	The time to calculate average over. Multiple time= entries can be given - generating multiple CPU usage summaries and multiple warn/crits.
nsclient		Flag to make the plug in run in NSClient compatibility mode
ShowAll	none, long	Add this option to show info even if no errors are detected. Set it to long to show detailed information.

Time can use any of the following postfixes. w=week, d=day, h=hour, m=minute and s=second.

Configuration

The size and frequency of sampled CPU data can be configured and for details refer to the configuration section for the CheckSystem module

FAQ

- **Q:** "NSClient - ERROR: Could not get data for 60 perhaps we don"t collect data this far back?"
- **A:** See the configuration section on how to configure the "CPUBufferSize" it has to be LARGER then your collection time here.

- **Q:** How does it handle multi CPU machines?
- **A:** The returned value is the average value of the CPU load of all the processors.

Examples

Sample Command

Check that the CPU load for various times is below 80%:

Sample Command:

```
CheckCPU warn=80 crit=90 time=20m time=10s time=4
```

```
OK: CPU Load ok.
```

Nagios Configuration:

```
define command {
  command_name <<CheckCPU>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCPU -a warn=$ARG1$ crit=$ARG2$ time=20
}
<<CheckCPU>> 80!90
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCPU -a warn=80 crit=90 time=20m time=10s time=4
```

Multiple Time entry

Showing multiple time entry usage and returned data

Sample Command:

```
CheckCPU warn=2 crit=80 time=5m time=1m time=10s
```

```
WARNING: WARNING: 5m: average load 8% > warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckCPU>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCPU -a warn=2 crit=$ARG1$ time=5m time
}
<<CheckCPU>> 80
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCPU -a warn=2 crit=80 time=5m time=1m time=10s
```

check_load

Check CPU load with intervals like known from Linux/Unix (with example thresholds):

Sample Command:

```
CheckCPU warn=100 crit=100 time=1 warn=95 crit=99 time=5 warn=90 crit=95 time=15
```

```
OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckCPU>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCPU -a warn=100 crit=100 time=1 warn=9
}
<<CheckCPU>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCPU -a warn=100 crit=100 time=1 warn=95 crit=99 time=5 warn=90 c
```

CheckUpTime

This check checks the uptime of a server and if the time is less then the times given as arguments a state is returned.

Option	Values	Description
MaxCrit	time	Maximum time the system is allowed to be up
MinCrit	time	Minimum time the system is allowed to be up
MaxWarn	time	Maximum time the system is allowed to be up
MinWarn	time	Minimum time the system is allowed to be up
nsclient		Flag to make the plug in run in NSClient compatibility mode
ShowAll		Add this option to show details even if an error is not encountered.
Alias	string	A string to use as alias for the values (default is uptime)

Examples

Check that the system has been running for at least a day:

Sample Command:

```
CheckUpTime MinWarn=1d MinCrit=12h
```

```
WARNING: Client has uptime (19h) <warning (24h)
```

Nagios Configuration:

```
define command {
  command_name <<CheckUpTime>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckUpTime -a MinWarn=$ARG1$ MinCrit=$ARG2
}
<<CheckUpTime>> 1d!12h
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckUpTime -a MinWarn=1d MinCrit=12h
```

CheckServiceState

This check checks the state of one or more service on the system and generates a critical state if any service is not in the required state.

Option	Values	Description
ShowAll		A flag to toggle if all service states should be listed.
ShowFail?	(default)	A flag to indicate if only failed service states should be listed.
service=state		A service name or service display name and a state the service should have. The state can be either started or stopped. If no state is given started is assumed.
CheckAll?		Check to see that all services set to auto-start are started and all set to disabled are not started.
exclude	service name	Exclude this service from CheckAll?
truncate	off	Truncate the returned data.

Configuration

ow to interpret "CheckAll?" can be configred and for details refer to the configuration section for the CheckSystem module

Examples

Sample check

Check that myService is running and that MyStoppedService? is not running:

Sample Command:

```
CheckServiceState ShowAll myService MyStoppedService=stopped

OK: myService : Running - MyStoppedService : Stopped
```

Nagios Configuration:

```
define command {
  command_name <<CheckServiceState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckServiceState -a ShowAll $ARG1$ $ARG2$=
}
<<CheckServiceState>> myService!MyStoppedService
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckServiceState -a ShowAll myService MyStoppedService=stopped
```

Auto started

Check that all auto-start services are running but exclude some that are intentionaly not in the correct state:

Sample Command:

```
CheckServiceState CheckAll exclude=wampmysqld exclude=MpfService

OK: OK: All services are running.
```

Nagios Configuration:

```
define command {
  command_name <<CheckServiceState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckServiceState -a CheckAll exclude=$ARG1
}
<<CheckServiceState>> wampmysqld!MpfService
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckServiceState -a CheckAll exclude=wampmysqld exclude=MpfService
```

Service name with spaces

If the service you are excluding has spaces in, you need to enclose the entire option in quotes, not just the string.

Sample Command:

```
CheckServiceState CheckAll exclude=wampmysqld "exclude=NetBackup SAN Client Fibre Transport Servi

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckServiceState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckServiceState -a CheckAll exclude=$ARG1
}
<<CheckServiceState>> wampmysqld!NetBackup SAN Client Fibre Transport Service
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckServiceState -a CheckAll exclude=wampmysqld "exclude=NetBackup S
```

CheckProcState

This check checks the state of one or more processes on the system and generates a critical state if any process is not in the required state.

Option	Values	Description
match=(strings|substr|regexp)		Specifies if the supplied value must match the actual process name or command line exactly (strings, the default), must match a part of the actual value (substr), or if it should be matched as a regular expression.
cmdLine		If present, the process name specified is compared to the entire command line. By default it is matched to the process name only.
ShowAll		A flag to toggle if all process states should be listed.
ShowFail?	(default)	A flag to indicate if only failed process states should be listed.
process=state		A process name and a state the process should have. The state can be either started or stopped. If no state is given started is assumed. The name is the name of the executable.
Alias	alias	Give a process an alias
ignore-perf-data		If present performance data will be stripped out
Proc:<alias>=<state>		A process name and a state the process should have.
(Max|Min)(Warn|Crit)Count	number	Process count bounds For instance: MaxCritCount?=4 means if a process has more then 4 instances it will be a critical condition.

The commands given in the examples below should be edited (for your own needs) and copied to the nsc.ini file (comes with installation and can be found in the monitored machine, aka the client) under section [External Alias].

Please remember that for each external alias there's a command declared (under NSCA command or NRPE command sections ? depends what you're working with) that uses the external alias declared.

The command also need to have identical name to the value defined for that service check in the Nagios linux server (usually windows.cfg file and service_description field in define Service block).

Let's start with a simple one ? check that a file named NameOfMonitoredFileReplaceWithYours?.exe is running (aka in started state in the windows machine).

In the [External Alias] section in the nsc.ini file there's an alias that looks like this:

alias_process=checkProcState $ARG1$=started

so we will leave it and we just have to supply it with our file name as an argument.

We will need to add the following line to the command section (NRPE command section if that's what you are working with).

NEED_TO_COMPLETE_EXAMPLE_HERE_BY_MICKEM_STAY_TUNED

NSCA is all client side, so if you're working with NSCA you cant use arguments and probably the following line (without the alias section) will be good for you: Check Process=CheckProcState

NameOfMonitoredFileReplaceWithYours?.exe=started

again, Check Process is the name declared in Nagios server and it has to be the same.

Examples

Process running/not running

Check that quake.exe is not running and NSClient++.exe is running:

Sample Command:

```
CheckProcState ShowAll quake.exe=stopped NSClient++.exe=started
```

```
OK: quake.exe : Stopped - NSClient++.exe : Running
```

Nagios Configuration:

```
define command {
  command_name <<CheckProcState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckProcState -a ShowAll $ARG1$=stopped $A
}
<<CheckProcState>> quake.exe!NSClient++.exe
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckProcState -a ShowAll quake.exe=stopped NSClient++.exe=started
```

Process running/not running

Check that quake.exe is not running and my.exe and NSClient++.exe is running and only show problems:

Sample Command:

```
CheckProcState quake.exe=stopped NSClient++.exe=started
```

```
CRITICAL: NSClient++.exe : Stopped
```

Nagios Configuration:

```
define command {
  command_name <<CheckProcState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckProcState -a $ARG1$=stopped $ARG2$=sta
}
<<CheckProcState>> quake.exe!NSClient++.exe
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckProcState -a quake.exe=stopped NSClient++.exe=started
```

Check number of processes running

make sure that atleast 50 instance of svchost.xe is running.

Sample Command:

```
CheckProcState MinCritCount=50 svchost.exe=started
```

```
OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckProcState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckProcState -a MinCritCount=50 $ARG1$=st
}
<<CheckProcState>> svchost.exe
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckProcState -a MinCritCount=50 svchost.exe=started
```

Substrings and commandline

Check that cmd.exe with substring printloop in commandline is running:

Sample Command:

```
CheckProcState match=regexp cmdLine ShowAll .*cmd.*printloop.*=started
```

```
OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckProcState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckProcState -a match=regexp cmdLine Show
}
<<CheckProcState>> printloop
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckProcState -a match=regexp cmdLine ShowAll .*cmd.*printloop.*=sta
```

More process counts

Check if there's any notepad.exe running. OK if there's none, warn if there's 4, critical when there's 10:

Sample Command:

```
CheckProcState MaxWarnCount=4 MaxCritCount=10 ShowAll notepad.exe=started
```

```
OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckProcState>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckProcState -a MaxWarnCount=$ARG2$ MaxCr
}
<<CheckProcState>> notepad.exe!4!10
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckProcState -a MaxWarnCount=4 MaxCritCount=10 ShowAll notepad.exe=
```

CheckMem

This check checks the memory (page) usage and generates a state if the memory is above or below give parameters.

Option	Values	Description
MaxWarn	size-value or %	The maximum size allowed before a warning state is returned.
MaxCrit	size-value or %	The maximum size allowed before a critical state is returned.
MinWarn	size-value or %	The minimum size allowed before a warning state is returned.
MinCrit	size-value or %	The minimum size allowed before a critical state is returned.
ShowAll	None, long	Add this option to show info even if no errors are detected. Set it to long to show detailed information.
type	page, paged, virtual, physical	What kind of memory to check (does not yet support stacking to check multiple kinds)

The *size-value or %* is a normal numeric-value with an optional unit or percentage postfix to specify large sizes. The available postfixes are B for Byte, K for Kilobyte, M for Megabyte, G for Gigabyte and finally % for percent free space.

What the different types really mean

Type	Meaning
page	The maximum amount of memory the current process can commit, in bytes. This value is equal to or smaller than the system-wide available commit value.
paged	System-wide committed memory limit (same as used in NSCLient, ie. via PDH, available on NT4)
virtual	Number of pages of swap currently in use (note - it does ***NOT*** = (physical + swap) as on *nix boxes) According to M$ this is: Size of unreserved and uncommitted memory in the user mode portion of the virtual address space of the calling process, in bytes.
physical	The amount of physical memory currently available, in bytes. This is the amount of physical memory that can be immediately reused without having to write its contents to disk first. It is the sum of the size of the standby, free, and zero lists.

Examples

Page

Check that the page is below 80%:

Sample Command:

```
CheckMEM MaxWarn=80% MaxCrit=90% ShowAll type=page
```

```
OK: OK: page: 758M (795205632B)
```

Nagios Configuration:

```
define command {
  command_name <<CheckMEM>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckMEM -a MaxWarn=$ARG1$% MaxCrit=$ARG2$%
}
```

```
<<CheckMEM>> 80!90
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckMEM -a MaxWarn=80% MaxCrit=90% ShowAll type=page
```

Physical

Check that the physical is below 80%:

Sample Command:

```
CheckMEM MaxWarn=80% MaxCrit=90% ShowAll type=physical

OK: OK: physical: 758M (795205632B)
```

Nagios Configuration:

```
define command {
  command_name <<CheckMEM>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckMEM -a MaxWarn=$ARG1$% MaxCrit=$ARG2$%
}
<<CheckMEM>> 80!90
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckMEM -a MaxWarn=80% MaxCrit=90% ShowAll type=physical
```

Multiple

Check that the physical is below 80%:

Sample Command:

```
CheckMEM MaxWarn=80% MaxCrit=90% ShowAll type=physical type=page type=virtual

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckMEM>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckMEM -a MaxWarn=$ARG1$% MaxCrit=$ARG2$%
}
<<CheckMEM>> 80!90
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckMEM -a MaxWarn=80% MaxCrit=90% ShowAll type=physical type=page t
```

CheckCounter

Used to check performance counters. This is probably how most things can be checked as there are a lot of performance counters. To find performance counters, use the program perfmon that is shipped with windows. You can list all available performance counters using the listpdh option as shown in the examples below.

An important note is that performance counters are language and version specific.

Option	Values	Description
MaxWarn	Number	Maximum allowed number
MaxCrit	Number	Maximum allowed number
MinWarn	Number	Minimum allowed number
MinCrit	Number	Minimum allowed number
ShowAll	None	A Boolean flag to show value even if no state is returned.
Counter	Performance Counter	Add a performance counter to this check
Counter:<name>	Performance Counter	Add a named performance counter. The <name> will be used as an alias.
Averages	true, false	Set this to false to make performance checking faster of counters that doesn't represent average values.
InvalidStatus?	UNKNOWN	The status to return if an invalid counter was requested.

FAQ

- **Q:** When using CheckCounter, does the NSClient++ client need to allow nasty chars? E.g. do you have to change the default setting: allow_nasty_chars=0 to.... allow_nasty_chars=1 ?
- **A:** No, you can configure it via aliases as well using the CHeckExternalCommands module. but to configure it from "nagios" you need it.

- **Q:** How do you define local commands?
- **A:** Use the alias section from external commands

- **Q:** How do you list all instances of a counter?
- **A:** Use the **listCounterInstances** command e.g. check_nrpe -c listCounterInstances -a "MSExchange Database"

Command line

List all available performance counters, and debug them (means, open, try to read, close, etc)

```
nsclient++ CheckSystem listpdh
...
"NSClient++.exe" CheckSystem debugpdh
...
```

check_nt vs. check_nrpe

```
define command {
        command_name    check_counter
#       command_line    $USER1$/check_nt $HOSTADDRESS$ -p 12489
                -v COUNTER -l $ARG1$ -d SHOWALL -w $ARG2$ -c $ARG3$
        command_line    $USER1$/check_nrpe -H $HOSTADDRESS$
                -c CheckCounter -a $ARG1$ ShowAll MaxWarn=$ARG2$ MaxCrit=$ARG3$
}
```

Examples

Sample Command

Check that mutex count (on WinXP) is below 500:

Sample Command:

```
CheckCounter "Counter:mutex=\\Objects\\Mutexes" ShowAll MaxWarn=500 MaxCrit=1000
```

```
WARNING: WARNING: mutex: 673 > warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckCounter>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCounter -a "Counter:$ARG1$=$ARG2$" Sho
}
<<CheckCounter>> mutex!\\Objects\\Mutexes!500!1000
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCounter -a "Counter:mutex=\\Objects\\Mutexes" ShowAll MaxWarn=50
```

Using Instances

Using instances in counters

Sample Command:

```
CheckCounter "Counter:proc=\\Processor(_total)\\% Processor Time" ShowAll MaxWarn=50 MaxCrit=80
```

```
OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckCounter>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCounter -a "Counter:$ARG1$=$ARG2$" Sho
}
<<CheckCounter>> proc!\\Processor(_total)\\% Processor Time!50!80
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCounter -a "Counter:proc=\\Processor(_total)\\% Processor Time"
```

Microsoft Exchange 5.5 IS RPC Operations / Sec

Sample Command:

```
CheckCounter "Counter=\MSExchangeIS\RPC Operations/sec" ShowAll MaxWarn=300 MaxCrit=400

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckCounter>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCounter -a "Counter=$ARG1$" ShowAll Ma
}
<<CheckCounter>> \MSExchangeIS\RPC Operations/sec!300!400
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCounter -a "Counter=\MSExchangeIS\RPC Operations/sec" ShowAll Ma
```

Windows 2000/2003 Physical Disk Time

Sample Command:

```
CheckCounter "Counter=\PhysicalDisk(_Total)\% Disk Time" ShowAll MaxWarn=60 MaxCrit=90

OK: ...
```

Nagios Configuration:

```
define command {
  command_name <<CheckCounter>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckCounter -a "Counter=$ARG1$" ShowAll Ma
}
<<CheckCounter>> \PhysicalDisk(_Total)\% Disk Time!60!90
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckCounter -a "Counter=\PhysicalDisk(_Total)\% Disk Time" ShowAll M
```

CheckHelpers.dll

The CheckHelpers module has various helper function to alter other checks in various ways. This module does not check anything by it self.

- wiki:CheckAlwaysOK, Alter the return code of another check to always return OK.
- wiki:CheckAlwaysCRITICAL, Alter the return code of another check to always return CRITICAL.
- wiki:CheckAlwaysWARNING, Alter the return code of another check to always return WARNING.
- wiki:CheckMultiple, Runs multiple checks and returns the worst state. Useful for minimizing network traffic and command definitions.

Configuration

This module has no configuration directives.

CheckAlwaysOK

Runs another check and alters the return state to always return OK. This command has no options instead everything is inject as a new command. Notice that this command does not alter the return string so if a check (as most do) prints WARNING that will still remain, only the return state is altered. The return state is what Nagios will use to determine state.

Examples

Run the sample check from CheckDisk but alter the return state. Notice that WARNING is still printed, yet Nagios will interpret this as an OK state.

```
CheckAlwaysOK CheckFileSize ShowAll MaxWarn=1024M MaxCrit=4096M File:WIN=c:\WINDOWS\*.*
WARNING: WIN: 2G (2325339822B)|WIN=2325339822;1073741824;4294967296
(but this will be returned with an "OK" state so Nagios will show this as working even though it is infac not w
```

CheckAlwaysCRITICAL

Runs another check and alters the return state to always return CRITICAL. This command has no options instead everything is inject as a new command. Notice that this command does not alter the return string so if a check (as most do) prints WARNING that will still remain, only the return state is altered. The return state is what Nagios will use to determine state.

Examples

Run the sample check from CheckDisk but alter the return state. Notice that WARNING is still printed, yet Nagios will interpret this as a critical state.

```
CheckAlwaysCRITICAL CheckFileSize ShowAll MaxWarn=1024M MaxCrit=4096M File:WIN=c:\WINDOWS\*.*
WARNING: WIN: 2G (2325339822B)|WIN=2325339822;1073741824;4294967296
```

CheckAlwaysWARNING

Runs another check and alter the return state to always return WARNING. This command has no options instead everything is inject as a new command. Notice that this command does not alter the return string so if a check (as most do) prints WARNING that will still remain, only the return state is altered. The return state is what Nagios will use to determine state.

Examples

Run the sample check from CheckDisk but alter the return state. Notice that WARNING is still printed, yet Nagios will interpret this as an warning state.

```
CheckAlwaysOK CheckFileSize ShowAll MaxWarn=1024M MaxCrit=4096M File:WIN=c:\WINDOWS\*.*
WARNING: WIN: 2G (2325339822B)|WIN=2325339822;1073741824;4294967296
```

CheckOK

This command always returns "OK".

Examples

```
CheckOK Hello
OK: Hello
```

CheckCRITICAL

This command always returns "CRITICAL".

Examples

```
CheckCRITICAL Hello
CRITICAL: Hello
```

CheckWARNING

This command always returns "WARNING".

Examples

```
CheckWARNINGHello
WARNING: Hello
```

CheckMultiple

Runs multiple checks and returns the worst state. It allows you to check an entire system in one go.

Option	Values	Description
command	The command to run	A command to execute, you can have any number of this option.

Be careful when combining multiple checks/commands which may return UNKNOWN status. If the first command returns OK, but the other returns UNKNOWN, the final status result in OK.

Examples

Run two checks (CheckDriveSize and ChEckMeM) and return the worst state. Performance data and message is collected and concatenated.

```
and=CheckDriveSize MaxWarn=1M MaxCrit=2M Drive=c:\\volumne_test\\ command=ChEckMeM MaxWarn=80% MaxCrit=90%
ne_test\: 3M (4193280B) > critical, OK memory within bounds.|c:\volumne_test\=4193280;1024K (1048576B);2M (209
```

CheckVersion

This command returns the current version (as a string)

Examples

```
CheckVersion
OK '0.3.3.20 2008-07-02'
```

CheckTaskSched.dll

The CheckTaskSched module check check various aspects of the task scheduler. Feel free to request checks that you need.

- CheckTaskSched, Check if tasks are working.

Configuration

This module has no configuration directives.

CheckTaskSched

CheckTaskSchedis? part of the wiki:CheckTaskSched module.

Arguments:

Option	Values	Description
filter	in, out, any, all	Specify the way you want to filter things. (See section below)
truncate	length of the returned set	This will truncate the output after the specified length. As NRPE can only handle 1024 chars you need to truncate the output.
Alias	string-expression	The alias to use.
debug	N/A	Print the filter chain.
MaxWarn	number of records	The maximum hits to allow before reporting a warning state.
MaxCrit	number of records	The maximum hits to allow before reporting a critical state.
MinWarn	number of records	The minimum hits to allow before reporting a warning state.
MinCrit	number of records	The minimum hits to allow before reporting a critical state.
+filter-accountName	string-expression	TODO.
+filter-applicationName	string-expression	TODO.
+filter-comment	string-expression	TODO.
+filter-creator	string-expression	TODO.
+filter-exitCode	numeric-expression	TODO.
+filter-flags	numeric-expression	TODO.
+filter-mostRecentRunTime	date-expression?	TODO.
+filter-nextRunTime	date-expression?	TODO.
+filter-parameters	string-expression	TODO.
+filter-workingDirectory	string-expression	TODO.
-filter-accountName	string-expression	TODO.
-filter-applicationName	string-expression	TODO.
-filter-comment	string-expression	TODO.
-filter-creator	string-expression	TODO.
-filter-exitCode	numeric-expression	TODO.
-filter-flags	numeric-expression	TODO.
-filter-mostRecentRunTime	date-expression?	TODO.
-filter-nextRunTime	date-expression?	TODO.
-filter-parameters	string-expression	TODO.
-filter-workingDirectory	string-expression	TODO.

Make sure we dont have any tasks wit last-exit code equals to 1

```
CheckTaskSched +filter-exit-code==1 ShowAll MaxWarn=1 MaxCrit=1
```

TODO On some case, NSClient++.exe crash while "CheckTaskSched ShowAll" CheckTaskSched.dll fault at address 0x00014324 (NSClient++.exe version 20081113-1003)

FileLogger.dll

A module that logs all messages to file if no logging module is loaded no error messages will be logged thus it is hard to find problems.

Configuration Sections

This is a wrapper page the actual data is on the following page FileLogger/config

Overview

This section has options for how logging is performed with the [FileLogger] module. First off notice that for logging to make sense you need to enable the ?FileLogger.dll? module that logs all log data to a text file in the same directory as the NSClient++ binary if you don?t enable any logging module nothing will be logged.

The options you have available here are

Option	**Default**	**Description**
debug	0	A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.
file	nsclient.log	The file to write log data to. If no directory is used this is relative to the NSClient++ binary.
date_mask	%Y-%m-%d %H:%M:%S	The date format used when logging to a file
root_folder	exe	Root folder if not absolute

debug

A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.

Default
: 0

Values

Value	Meaning
0	Don't log debug messages
1	Log debug messages

file

The file to write log data to. If no directory is used this is relative to the NSClient++ binary.

Default
: nsclient.log

date_mask

The date format used when logging to a file

Default
%Y-%m-%d %H:%M:%S

root_folder

Root folder if not absolute

Default
exe

Values

local-app-data	The file system directory that contains application data for all users. A typical path is C:\Documents and Settings\All Users\Application Data. This folder is used for application data that is not user specific. For example, an application can store a spell-check dictionary, a database of clip art, or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.
exe	Location of NSClient++ binary

Configuration for the FileLogger

This page describes the configuration options for the File logging module.

This is a wrapper page the actual data is on the following page FileLogger/config

Configuration Sections

Overview

This section has options for how logging is performed with the [FileLogger] module. First off notice that for logging to make sense you need to enable the ?FileLogger.dll? module that logs all log data to a text file in the same directory as the NSClient++ binary if you don?t enable any logging module nothing will be logged.

The options you have available here are

Option	**Default**	**Description**
debug	0	A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.
file	nsclient.log	The file to write log data to. If no directory is used this is relative to the NSClient++ binary.
date_mask	%Y-%m-%d %H:%M:%S	The date format used when logging to a file
root_folder	exe	Root folder if not absolute

debug

A Boolean value that toggles if debug information should be logged or not. This can be either 1 or 0.

Default
: 0

Values

Value	Meaning
0	Don't log debug messages
1	Log debug messages

file

The file to write log data to. If no directory is used this is relative to the NSClient++ binary.

Default
: nsclient.log

date_mask

The date format used when logging to a file

Default

%Y-%m-%d %H:%M:%S

root_folder

Root folder if not absolute

Default

exe

Values

local-app-data	The file system directory that contains application data for all users. A typical path is C:\Documents and Settings\All Users\Application Data. This folder is used for application data that is not user specific. For example, an application can store a spell-check dictionary, a database of clip art, or a log file in the CSIDL_COMMON_APPDATA folder. This information will not roam and is available to anyone using the computer.
exe	Location of NSClient++ binary

NRPEListener.dll

This module accepts incoming NRPE connections and responds by executing various checks and returns their result. To use this you need to have check_nrpe or another NRPE client. This is similar to check_nt (NSClient) but much more flexible and supports encryption. This only drawback is that it lacks any authorization.

As this module has the ability to generate command handlers by configuration there are command handlers but nothing built in. This is present for compatiblity only it is suggested to use the [CheckExternalScripts] instead.

Configuration Sections

NRPE Section

This is a wrapper page the actual data is on the following page NRPEListener/config/nrpe

Overview

This is configuration for the NRPE module that controls how the NRPE listener operates.

Option	Default	Description
port	5666	The port to listen to
allowed_hosts		A list of hosts allowed to connect via NRPE.
use_ssl	1	Boolean value to toggle SSL encryption on the socket connection
command_timeout	60	The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

socket_timeout	30	The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Advanced options:

Option	Default	Description
performance_data	1	Send performance data back to nagios (set this to 0 to remove all performance data)
socket_back_log		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
string_length	1024	Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.
script_dir		Load all scripts in a directory and use them as commands. Probably dangerous but usefull if you have loads of scripts :)
bind_to_address		The address to bind to when listening to sockets.

port

The port to listen to

Default

5666

allowed_hosts

A list (comma separated) with hosts that are allowed to poll information from NRPE. This will replace the one found under Setting for NRPE if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system

Default

Empty list (falls back to the one defined under [Settings]

use_ssl

Boolean value to toggle SSL (Secure Socket Layer) encryption on the socket connection. This corresponds to the -n flag in check_nrpe

Values

Value	Meaning
0	Don't use SSL
1	Use SSL encryption

Default

1 (enabled)

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
: IP address of any interface of the server.

Default
: Empty (first (all?) interface will be used)

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones so internal commands may execute forever.

It is usually a good idea to set this to less then the timeout used with check_nrpe

Default
: 60

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

NOTICE That there are more then one place to set this!

Default
: 0 (means don't allow arguments)

Values

Value	Meaning
0	Don't allow arguments
1	Allow arguments.

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Default
: 0 (means don't allow meta characters)

Values

Value	Meaning
0	Don't allow meta characters
1	Allow meta characters

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
: 30 seconds

script_dir

Load all scripts in a directory and use them as commands. Probably dangerous but useful if you have loads of scripts :)

Default
: Empty (don't load any scripts)

performance_data

Send performance data back to Nagios (set this to 0 to remove all performance data)

Default
: 1

Values

Value	Meaning
0	Don't send performance data
1	Send performance data

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

string_length

Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.

Default
: 1024

NRPE Handler Section

This is a wrapper page the actual data is on the following page NRPEListener/config/nrpe_handlers

Ovreview

DEPRECATED This part of the module is deprecated and should not be used. Refer to the [CheckExternalScripts] module instead. This module can add two types of command handlers.

First there are external command handlers that execute a separate program or script and simply return the output and return status from that. The other possibility is to create an alias for an internal command.

To add an external command you add a command definition under the ?NRPE Handlers? section. A command definition has the following syntax:

```
[NRPE Handlers]
command_name=/some/executable with some arguments
test_batch_file=c:\test.bat foo $ARG1$ bar
command[check_svc]=inject CheckService checkAll
```

The above example will on an incoming ?test_batch_file? execute the c:\test.bat file and return the output as text and the return code as the Nagios status.

Alias (builtin commands)

To add an internal command or alias is perhaps a better word. You add a command definition under the ?NRPE Handlers? section. A command definition with the following syntax:

```
command_name=inject some_other_command with some arguments
check_cpu=inject checkCPU warn=80 crit=90 5 10 15
```

The above example will on an incoming ?check_cpu? execute the internal command ?checkCPU? with predefined arguments give in the command definition.

NRPE_NT Syntax

To leverage existing infrastructure you can copy your old definitions from NRPE_NT as-is. Thus the following:

```
command[check_svc]=inject CheckService checkAll
```

translates into a command called check_svc with the following definition:

```
CheckServcice checkAll
```

Configuration for the NRPEListener

This page describes the configuration options for the NRPE module.

NRPE Section

This is a wrapper page the actual data is on the following page NRPEListener/config/nrpe

Overview

This is configuration for the NRPE module that controls how the NRPE listener operates.

Option	Default	Description
port	5666	The port to listen to
allowed_hosts		A list of hosts allowed to connect via NRPE.
use_ssl	1	Boolean value to toggle SSL encryption on the socket connection
command_timeout	60	The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).
socket_timeout	30	The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Advanced options:

Option	Default	Description
performance_data	1	Send performance data back to nagios (set this to 0 to remove all performance data)
socket_back_log		

		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
string_length	1024	Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.
script_dir		Load all scripts in a directory and use them as commands. Probably dangerous but usefull if you have loads of scripts :)
bind_to_address		The address to bind to when listening to sockets.

port

The port to listen to

Default
5666

allowed_hosts

A list (comma separated) with hosts that are allowed to poll information from NRPE. This will replace the one found under Setting for NRPE if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system

Default
Empty list (falls back to the one defined under [Settings]

use_ssl

Boolean value to toggle SSL (Secure Socket Layer) encryption on the socket connection. This corresponds to the -n flag in check_nrpe

Values

Value	Meaning
0	Don't use SSL
1	Use SSL encryption

Default
1 (enabled)

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
IP address of any interface of the server.

Default
Empty (first (all?) interface will be used)

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones so internal commands may execute forever.

It is usually a good idea to set this to less then the timeout used with check_nrpe

Default
: 60

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

NOTICE That there are more then one place to set this!

Default
: 0 (means don't allow arguments)

Values

Value	Meaning
0	Don't allow arguments
1	Allow arguments.

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Default
: 0 (means don't allow meta characters)

Values

Value	Meaning
0	Don't allow meta characters
1	Allow meta characters

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
: 30 seconds

script_dir

Load all scripts in a directory and use them as commands. Probably dangerous but useful if you have loads of scripts :)

Default

Empty (don't load any scripts)

performance_data

Send performance data back to Nagios (set this to 0 to remove all performance data)

Default

1

Values

Value	Meaning
0	Don't send performance data
1	Send performance data

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

string_length

Length of payload to/from the NRPE agent. This is a hard specific value so you have to "configure" (read recompile) your NRPE agent to use the same value for it to work.

Default

1024

NRPE Handler Section

This is a wrapper page the actual data is on the following page NRPEListener/config/nrpe_handlers

Ovreview

DEPRECATED This part of the module is deprecated and should not be used. Refer to the [CheckExternalScripts] module instead. This module can add two types of command handlers.

First there are external command handlers that execute a separate program or script and simply return the output and return status from that. The other possibility is to create an alias for an internal command.

To add an external command you add a command definition under the ?NRPE Handlers? section. A command definition has the following syntax:

```
[NRPE Handlers]
command_name=/some/executable with some arguments
test_batch_file=c:\test.bat foo $ARG1$ bar
command[check_svc]=inject CheckService checkAll
```

The above example will on an incoming ?test_batch_file? execute the c:\test.bat file and return the output as text and the return code as the Nagios status.

Alias (builtin commands)

To add an internal command or alias is perhaps a better word. You add a command definition under the ?NRPE Handlers? section. A command definition with the following syntax:

```
command_name=inject some_other_command with some arguments
check_cpu=inject checkCPU warn=80 crit=90 5 10 15
```

The above example will on an incoming ?check_cpu? execute the internal command ?checkCPU? with predefined arguments give in the command definition.

NRPE_NT Syntax

To leverage existing infrastructure you can copy your old definitions from NRPE_NT as-is. Thus the following:

```
command[check_svc]=inject CheckService checkAll
```

translates into a command called check_svc with the following definition:

```
CheckServcice checkAll
```

NSClientListener.dll

The NSClientListener module is written to allow backwards compatibility with the old NSClient and check_nt. It has a listener (server) that accepts checks from the check_nt command and responds accordingly. Due to the nature of the protocol and the limitation in the client NRPE is recommended but if you like this works just fine for "simple things".

The following check_nt checks are supported.

- CLIENTVERSION
- CPULOAD
- UPTIME
- USEDDISKSPACE
- MEMUSE
- SERVICESTATE
- PROCSTATE
- COUNTER

Configuration Sections

NSClient Section

This is a wrapper page the actual data is on the following page NSClientListener/config

Ovreview

This is the [NSClientListener] module configuration options.

Option	Default value	Description
port	12489	The port to listen to
obfuscated_password		An obfuscated version of password.
password		The password that incoming client needs to authorize themselves by.
allowed_hosts		A list (coma separated) with hosts that are allowed to connect to NSClient++ via NSClient protocol.
socket_timeout	30	The timeout when reading packets on incoming sockets.

Advanced options:

Option	Default value	Description
socket_back_log		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
bind_to_address		The address to bind to when listening to sockets, useful if you have more then one NIC/IP address and want the agent to answer on a specific one.
version	auto	The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

port

The port to listen to

Default
12489

obfuscated_password

An obfuscated version of password. For more details refer to the password option below.

Default
Empty string whjich means we will use the value from password instead.

password

The password that incoming client needs to authorize themselves by. This option will replace the one found under Settings for NSClient. If this is blank the option found under Settings will be used. If both are blank everyone will be granted access.

Default
Empty string whjich means we will use the value from password in the [Settings] section instead.

allowed_hosts

A list (coma separated) with hosts that are allowed to poll information from NSClient++. This will replace the one found under Setting for NSClient if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system.

BEWARE: NSClient++ will not resolve the IP address of DNS entries if the service is set to startup automatically. Use an IP address instead or set cache_allowed_hosts=0 see above.

Default
Empty list (falls back to the one defined under [Settings]

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
: IP address of any interface of the server.

Default
: Empty (first (all?) interface will be used)

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
: 30 seconds

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

version

The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

> **Values**:
>
> > If given any str4ing will be returned unless auto in which case the proper version will be returned

Default
: auto

Examples

Check the size of C:\ and make sure it warns at 80% used and a critical warning at 95% used:

```
define command {
  command_name check_nt_disk
  command_line $USER1$/check_nt -s passphrase -H $HOSTADDRESS$ -p 12489
     -v USEDDISKSPACE -l $ARG1$ -w $ARG2$ -c $ARG3$
}
check_command check_nt_disk!c!80!95
```

Check Perfmon value for File IO reads

```
define command {
  command_name check_io_read
  command_line /usr/local/nagios/libexec/check_nt -H $HOSTADDRESS$
     -v COUNTER -l "\\System\\File Read Bytes/sec"
}
check_command check_io_read
```

Configuration for the NSClientListener

This page describes the configuration options for the NSClient module.

NSClient Section

This is a wrapper page the actual data is on the following page NSClientListener/config

Ovreview

This is the [NSClientListener] module configuration options.

Option	Default value	Description
port	12489	The port to listen to
obfuscated_password		An obfuscated version of password.
password		The password that incoming client needs to authorize themselves by.
allowed_hosts		A list (coma separated) with hosts that are allowed to connect to NSClient++ via NSClient protocol.
socket_timeout	30	The timeout when reading packets on incoming sockets.

Advanced options:

Option	Default value	Description
socket_back_log		Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.
bind_to_address		The address to bind to when listening to sockets, useful if you have more then one NIC/IP address and want the agent to answer on a specific one.
version	auto	The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

port

The port to listen to

Default
: 12489

obfuscated_password

An obfuscated version of password. For more details refer to the password option below.

Default
: Empty string whjich means we will use the value from password instead.

password

The password that incoming client needs to authorize themselves by. This option will replace the one found under Settings for NSClient. If this is blank the option found under Settings will be used. If both are blank everyone will be granted access.

Default
: Empty string whjich means we will use the value from password in the [Settings] section instead.

allowed_hosts

A list (coma separated) with hosts that are allowed to poll information from NSClient++. This will replace the one found under Setting for NSClient if present. If not present the same option found under Settings will be used. If both are blank all hosts will be allowed to access the system.

BEWARE: NSClient++ will not resolve the IP address of DNS entries if the service is set to startup automatically. Use an IP address instead or set cache_allowed_hosts=0 see above.

Default
: Empty list (falls back to the one defined under [Settings]

bind_to_address

The address to bind to when listening to sockets. If not specified the "first" (all?) one will be used (often the correct one).

Values
: IP address of any interface of the server.

Default
: Empty (first (all?) interface will be used)

socket_timeout

The timeout when reading packets on incoming sockets. If the data has not arrived within this time we will bail out. and discard the connection.

Default
: 30 seconds

socket_back_log

Number of sockets to queue before starting to refuse new incoming connections. This can be used to tweak the amount of simultaneous sockets that the server accepts. This is an advanced option and should not be used.

version

The version number to return for the CLIENTVERSION check (useful to "simulate" an old/different version of the client, auto will be generated from the compiled version string inside NSClient++

Values:

> If given any str4ing will be returned unless auto in which case the proper version will be returned

Default
auto

SysTray.dll

A simple module to show an icon in the tray when the service is running this module does not export any check commands.

NOTICE This is not used on windows vista and above!

On Vista you enable the shared session like so:

```
[Settings]
...
shared_Session=1
```

For a service to be able to "interact with the desktop" it has to be set to do so. By default (for "security reasons") it is not enabled so if you want to use this module you need to enable that. The simplest way to do is to run the following command:

```
# Make sure the service is installed
nsclient++ -install
# Change the service so it can interact with the desktop
nsclient++ -noboot SysTray install
```

For this to work the service has to have been installed first (ie. "nsclient++ -install").

CheckWMI.dll

The CheckWMI module has various WMI related functions used to query and check the WMI (Windows Managment Instrumenation). Feel free to request checks that you need.

- CheckWMI, Check large resultsets from (for instance are there more than 5-rows matching criteria X, ie. more than 5 Internet Explorer processes witch uses more then 123Mb memory).
- CheckWMIValue, Check the result of a query (ie. are the current memory utilization over X)

Configuration

This module has no configuration directives.

CheckWMI

CheckWMI is part of the wiki:CheckWMI module.

New version that is *a lot* more usefull (i hope). It is still alpha need to do more testing but I would like to get some initialö feedback on the syntax and such. Also feel free to try it out and report bugs to me (might wanna keep track of memory and such as I have not done so myself yet) To debug and help you setup your queries there is also a command line interface to run queries and see the result.

```
nsclient++ CheckWMI <query>
```

The syntax for this is Similar to CheckEventLog but simpliefied in regards to syntax so I hope it shall be easier to use and udnerstand.

This check enumerates all rows returned from your query filtering results and check the count against a set war and crit threshold. If you want to check a value there will soon be a separet check for that. This is designed to find "anomalies" in result-sets.

Option	Values	Description
filter	any, all	Specify the way you want to filter things. (See section below) Not yet implemented (default is all)
truncate	length of the returned set	This will truncate the output after the specified length. As NRPE can only handle 1024 chars you need to truncate the output.
MaxWarn	number of records	The maximum records to allow before reporting a warning state.
MaxCrit	number of records	The maximum records to allow before reporting a critical state.
<mode>filter-<type>:<Column>	<filter value>	A number of strings to use for filtering the event log
namespace	root\\cimv2	Namespace to use when querying
Alias		Alias to use for returned data
columnSeparator	", "	Field separator in the returned string.
columnSyntax		Syntax for the returned message.

Filters

Capturing result entries (or discarding them) are done with filters. There are three kinds of filters.

* positive requirments (+)

All these filters must match or the row is discarded.

* negative requriments (-)

None of these filters can match (if any do the row is discarded).

* normal matches (.)

If this matches the line is included.

The syntax of the filter is: <mode>filter-<type>:<Column>=<expression>

Filter <Mode>s

<mode>	title	description
+	required filter	If you miss this filter the line is discarded
.	normal filter	If a hit the line is included
-	negative filter	If a line hits this it is discarded

Filter <Type>s

<type>	Value	Description
string	[[string expression]]	Match the column againast a string expression
numeric	[[numeric expression]]	Match the column againast a numeric expression

Filter <Columns>s

A Column (if specified) will make the filter woirk againast a specific column in the result set.

string expression

A **string expression** is a key followed by a string that specifies a string expression. Currently substr and regexp are supported. Thus you enter filter.message=regexp:(foo|bar) to enter a regular expression and filter-message=substr:foo to enter a substring patter match.

columnSyntax

The column syntax field can be used to alter the renderd output. It has the following keys (everything else will be a string):

Key	Description
%column%	The name of the current column
%value%	The value
%<column>%	The value of a named column

Examples

A sample query

A not very useful check which serves to illustrate how to use the command. Check to see if there is 2 CPUs present (or cores)

Sample Command:

```
CheckWMI MaxCrit=3 MinWarn=1 "Query=Select * from win32_Processor"
```

```
WARNING: WARNING:: 1 <warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMI>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query=Sele
}
<<CheckWMI>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query=Select * from win32_Processor"
```

Using Query Alias

Adding query alias to the not very useful check above (Alias is **cpu**)

Sample Command:

```
CheckWMI MaxCrit=3 MinWarn=1 "Query:cpu=Select * from win32_Processor"
```

```
WARNING: WARNING:: cpu: 1 <warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMI>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query:cpu=
}
<<CheckWMI>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query:cpu=Select * from win32_Proces
```

Overriding Query Alias

Overriding the previous query alias with **foobar**

Sample Command:

```
CheckWMI MaxCrit=3 MinWarn=1 "Query:cpu=Select * from win32_Processor" Alias=foobar
```

```
WARNING: WARNING:: foobar: 1 <warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMI>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query:cpu=
}
<<CheckWMI>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMI -a MaxCrit=3 MinWarn=1 "Query:cpu=Select * from win32_Proces
```

Checking With filters

This uses the UserAccount? object to query if any enabled users have password expires set to false.

Sample Command:

```
CheckWMI CheckWMI MaxCrit=3 MaxWarn=1 "Query:badUsers=Select Name, PasswordExpires, Disabled from

WARNING: WARNING:mickem & Xiqun Liao
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMI>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMI -a CheckWMI MaxCrit=3 MaxWarn=1 "Q
}
<<CheckWMI>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMI -a CheckWMI MaxCrit=3 MaxWarn=1 "Query:badUsers=Select Name,
```

Debbuging queries

To try a query use the following syntax:

```
nsclient++ CheckWMI Select * from win32_Processor
```

CheckWMIValue

CheckWMIValue is part of the wiki:CheckWMI module.

It is still alpha need to do more testing but I would like to get some initial feedback on the syntax and such. Also feel free to try it out and report bugs to me (might wanna keep track of memory and such as I have not done so myself yet) To debug and help you setup your queries there is also a command line interface to run queries and see the result.

```
nsclient++ CheckWMI <query>
```

The syntax for this is Similar to other check commands so it should be pretty straight forward to set it up. The plugin will run a WMI query and check the returned columns against bounds provided by the checker (nagios) and report the result.

Option	Values	Description
MaxWarn	Numeric value	The maximum allowed value for the column(s).
MaxCrit	Numeric value	The maximum allowed value for the column(s).
MinWarn	Numeric value	The minimum allowed value for the column(s).
MinCrit	Numeric value	The minimum allowed value for the column(s).
ShowAll	Empty, long	If present will display information even if an item is not reporting a state. If set to long will display more information.
Query	WMI Query	The WMI query to ask (not stackable, only one query at a time)
Check	A column name	A column name to check (if * all columns will be checked) (this is stackable, so you can compare any number of columns)
truncate	numeric value	The maximum length of the query-result.
AliasCol?	Column name	A column to be included (prefixed) in the alias for matching columns.

Examples

Check the CPU load on all CPUs and warn if above 50 and critical if above 80

Sample Command:

```
CheckWMIValue "Query=Select * from win32_Processor" MaxWarn=50 MaxCrit=80 Check:CPU=LoadPercentag
```

```
OK: OK: Everything seems fine.
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMIValue>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMIValue -a "Query=Select * from win32
}
<<CheckWMIValue>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMIValue -a "Query=Select * from win32_Processor" MaxWarn=50 Max
```

CHeck Threads in a process

Check threads in processes and make sure a process does not have more then 50 threads (critical at 100)

Sample Command:

```
CheckWMIValue "Query=select Caption, ThreadCount from Win32_Process" MaxWarn=50 MaxCrit=100 Check

WARNING: System threads: 98 > warning
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMIValue>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMIValue -a "Query=select Caption, Thr
}
<<CheckWMIValue>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMIValue -a "Query=select Caption, ThreadCount from Win32_Proces
```

Ping status

Little example on W32_PingStatus, I use this to check my VPN tunnels.

Sample Command:

```
CheckWMIValue 'Query=SELECT StatusCode FROM Win32_PingStatus WHERE Address="$ARG1$"' MaxCrit=1 Ch
```

Nagios Configuration:

```
define command {
  command_name <<CheckWMIValue>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c CheckWMIValue -a 'Query=SELECT StatusCode F
}
<<CheckWMIValue>>
```

From Commandline (with NRPE):

```
check_nrpe -H IP -p 5666 -c CheckWMIValue -a 'Query=SELECT StatusCode FROM Win32_PingStatus WHERE
```

Status-Code = 0 means good. Everything above 0 is BAD

For more information on Win32_PingStatus See:
http://msdn.microsoft.com/en-us/library/aa394350(VS.85).aspx

Using from command line

To try a query use the following syntax:

```
nsclient++ CheckWMI Select * from win32_Processor
```

CheckExternalScripts.dll

This module allows you to check external scripts and/or programs.

Configuration for the CheckExternalScripts

This page describes the configuration options for the CheckExternalScriptsmodule.

External Script

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_script

Overview

Configure how the External Scripts module works (not to be confused with the "External Scripts" section below that holds scripts that can be run.

Option	Default value	Description
command_timeout	60	The maximum time in seconds that a command can execute.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the command line.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands.
script_dir		When set all files in this directory will be available as scripts. **WARNING**

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.

Values:

Any number (positive integer) representing time in seconds.

Default
60 (seconds).

Example
Set timeout to 120 seconds

```
[External Script]
command_timeout=120
```

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

Values

Value	Meaning
0	Disallow arguments for commands
1	Allow arguments for commands

Default
: 0 (false).

Example
: Allow arguments

```
[External Script]
allow_arguments=1
```

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Values
: This list contain all possible values

Value	Meaning
0	Disallow nasty arguments for commands
1	Allow nasty arguments for commands

Default
: 0 (false)

Example
: Allow nasty arguments

```
[External Script]
allow_nasty_meta_chars=1
```

script_dir

When set all files in this directory will be available as scripts. This is pretty dangerous but can be a bit useful if you use many scripts and you are sure no one else can add files there.

Value
: Any directory (can be relative to NSClient++)

Default
: Empty (meaning no scripts are added)

Example

All scripts ending with bat in the scripts folder (of NSClient++ installation directory) will be added as scripts.

```
[External Script]
script_dir=.\scripts\*.bat
```

External Scripts

This is a wrapper page the actual data is on the following page *CheckExternalScripts/config/external_scripts*

1.
   1.
      1.
         1. Overview

Overview

A list of scripts and their aliases available to run from the CheckExternalScripts module. Syntax is: **<command>=<script> <arguments>** for instance:

```
check_es_long=scripts\long.bat
check_es_ok=scripts\ok.bat
check_es_nok=scripts\nok.bat
check_vbs_sample=cscript.exe //T:30 //NoLogo scripts\check_vb.vbs
check_es_args=scripts\args.bat static $ARG1$ foo
```

To configure scripts that request arguments, use the following syntax:

```
check_script_with_arguments=scripts\script_with_arguments.bat $ARG1$ $ARG2$ $ARG3$
```

Use ./check_nrpe ... -c check_script_with_arguments -a arg1 arg2 arg3 ... Make sure you type $ARG1$ and not $arg1$ (case sensitive)

NOTICE For the above to work you need to enable allow_arguments in **both** NRPEListener and CheckExternalScripts!

External Alias

This is a wrapper page the actual data is on the following page *CheckExternalScripts/config/external_alias*

1.
   1.
      1.
         1. Overview

Overview

A simple and nifty way to define aliases in NSClient++. Aliases are good for defining commands locally or just to simply the nagios configuration. There is a series of "useful" aliases defined in the included configuration file which is a good place to start. An alias is an internal command that has been "wrapped" (to add arguments). If you want to create an alias for an external command you can do so but it still needs the normal defnition and the alias will use the internal alias of the external command.

WARNING Be careful so you don't create loops (ie check_loop=check_a, check_a=check_loop)

```
[External Aliases]
alias_cpu=checkCPU warn=80 crit=90 time=5m time=1m time=30s
```

```
alias_disk=CheckDriveSize MinWarn=10% MinCrit=5% CheckAll FilterType=FIXED
alias_service=checkServiceState CheckAll
alias_mem=checkMem MaxWarn=80% MaxCrit=90% ShowAll type=physical
```

Examples

Check Users Logged In

This script enables you to check how many users are active (logged in via RDP) on any given server. This is especially useful for Terminal Servers to check the user load. The script also supplies performance data so you can graph the user load.

Check_users_loggedin accept none OR two arguments. Specifying only one or more then two arguments will return an error.
If no arguments are given, only the current user load is returned as an 'OK' state. Specifying a warning and critical number will return a warning or a critical state if the user load exceeds the threshold.

Sample command

```
check_users_loggedin=scripts\check_users_loggedin $ARG1$ $ARG2$
```

```
check_nrpe_users_loggedin,40 50
OK: Users logged in: 1|'users'=1;40;50;
```

Nagios configuration

```
define command {
  command_name <<CheckUsersLoggedIn>>
  command_line check_nrpe -H $HOSTADDRESS$ -p 5666 -c check_users_loggedin -a 40 50
}
<<CheckUsersLoggedIn>>
```

Configuration for the CheckExternalScripts

This page describes the configuration options for the CheckExternalScriptsmodule.

External Script

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_script

Overview

Configure how the External Scripts module works (not to be confused with the "External Scripts" section below that holds scripts that can be run.

Option	Default value	Description
command_timeout	60	The maximum time in seconds that a command can execute.
allow_arguments	0	A Boolean flag to determine if arguments are accepted on the command line.
allow_nasty_meta_chars	0	Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands.
script_dir		When set all files in this directory will be available as scripts. **WARNING**

command_timeout

The maximum time in seconds that a command can execute. (if more then this execution will be aborted). NOTICE this only affects external commands not internal ones.

Values:

Any number (positive integer) representing time in seconds.

Default
60 (seconds).
Example
Set timeout to 120 seconds

```
[External Script]
command_timeout=120
```

allow_arguments

A Boolean flag to determine if arguments are accepted on the incoming socket. If arguments are not accepted you can still use external commands that need arguments but you have to define them in the NRPE handlers below. This is similar to the NRPE "dont_blame_nrpe" option.

Values

Value	Meaning
0	Disallow arguments for commands
1	Allow arguments for commands

Default

0 (false).

Example

Allow arguments

```
[External Script]
allow_arguments=1
```

allow_nasty_meta_chars

Allow NRPE execution to have ?nasty? meta characters that might affect execution of external commands (things like > ? etc).

Values

This list contain all possible values

Value	Meaning
0	Disallow nasty arguments for commands
1	Allow nasty arguments for commands

Default

0 (false)

Example

Allow nasty arguments

```
[External Script]
allow_nasty_meta_chars=1
```

script_dir

When set all files in this directory will be available as scripts. This is pretty dangerous but can be a bit useful if you use many scripts and you are sure no one else can add files there.

Value

Any directory (can be relative to NSClient++)

Default

Empty (meaning no scripts are added)

Example

All scripts ending with bat in the scripts folder (of NSClient++ installation directory) will be added as scripts.

```
[External Script]
```

```
script_dir=.\scripts\*.bat
```

External Scripts

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_scripts

1.
   1.
      1.
         1. Overview

Overview

A list of scripts and their aliases available to run from the CheckExternalScripts module. Syntax is: **<command>=<script> <arguments>** for instance:

```
check_es_long=scripts\long.bat
check_es_ok=scripts\ok.bat
check_es_nok=scripts\nok.bat
check_vbs_sample=cscript.exe //T:30 //NoLogo scripts\check_vb.vbs
check_es_args=scripts\args.bat static $ARG1$ foo
```

To configure scripts that request arguments, use the following syntax:

```
check_script_with_arguments=scripts\script_with_arguments.bat $ARG1$ $ARG2$ $ARG3$
```

Use ./check_nrpe ... -c check_script_with_arguments -a arg1 arg2 arg3 ... Make sure you type $ARG1$ and not $arg1$ (case sensitive)

NOTICE For the above to work you need to enable allow_arguments in **both** NRPEListener and CheckExternalScripts!

External Alias

This is a wrapper page the actual data is on the following page CheckExternalScripts/config/external_alias

1.
   1.
      1.
         1. Overview

Overview

A simple and nifty way to define aliases in NSClient++. Aliases are good for defining commands locally or just to simply the nagios configuration. There is a series of "useful" aliases defined in the included configuration file which is a good place to start. An alias is an internal command that has been "wrapped" (to add arguments). If you want to create an alias for an external command you can do so but it still needs the normal defnition and the alias will use the internal alias of the external command.

WARNING Be careful so you don't create loops (ie check_loop=check_a, check_a=check_loop)

```
[External Aliases]
alias_cpu=checkCPU warn=80 crit=90 time=5m time=1m time=30s
alias_disk=CheckDriveSize MinWarn=10% MinCrit=5% CheckAll FilterType=FIXED
alias_service=checkServiceState CheckAll
alias_mem=checkMem MaxWarn=80% MaxCrit=90% ShowAll type=physical
```

LUAScript.dll

This module allows you to write and change checks in the Lua scripting language. For a quite "guide" on how to write scripts for NSClient++ see the LUAScript/guide page. For information on the Lua scripting language and built-in modules and commands refer to the official Lua pages at: http://www.lua.org/ and http://lua-users.org/wiki/SampleCode

Configuration

[LUA Scripts]

A list of LUA script to load at startup. In difference to "external checks" all LUA scripts are loaded at startup. Names have no meaning since the script (on boot) submit which commands are available and tie that to various functions.

```
[LUA Scripts]
scripts\test.lua
```

This is just a quick intro, I will try to add more info here and also try to add more system related functions (like WMI and performance counter access) in the future.

Debugging Lua

Use the print statement to print to the console (can bee sen from nsclient++ /test).

With a Lua script like this loaded:

```
nscp.register('lua_debug', 'debug')

function debug (command)
    print ('Hello world: ' .. command)
end
```

Then you run nsclient++ /test:

```
nsclient++ /test
...
lua_debug Greetings
...
d \nsclient++.cpp(540) Injecting: lua_debug: Greetings
Hello world: lua_debug
e \script_wrapper.hpp(280) No arguments returned from script.
l \nsclient++.cpp(575) No handler for command: 'lua_debug'
```

A simple script

```
print('Loading test script...') -- Just print some debug info

nscp.register('check_something', 'something') -- Register a check-command to a function

function something (command)
  -- Check command function (notice arguments are not supported yet)

  -- Inject and run another check command
  code, msg, perf = nscp.execute('CheckCPU','time=5','MaxCrit=5')
  -- Print the resulting code
  print(code .. ': ' .. msg .. ', ' .. perf)
  -- Return the information (slightly modified)
  return code, 'hello from LUA: ' .. msg, perf
end
```

Structure of a script

First all script register all commands they will use (it is possible to register commands at a later time) So you could have a command that "turn on" other commands, but since there is no "turn off" (ie. remove) it does not make much sense as of yet.

To register command you call the **nscp.register** function like so:

```
nscp.register('command_alias', 'function_in_lua_to_use');
```

This will when the command command_alias is run execute the function_in_lua_to_use in your script. You can have as many commands as you like so the following is possible:

```
nscp.register('lua_1', 'lua_function_1');
nscp.register('lua_2', 'lua_function_2');
nscp.register('lua_3', 'lua_function_3');
nscp.register('lua_4', 'lua_function_4');
nscp.register('lua_5', 'lua_function_5');
```

The functions have the following syntax:

```
function lua_check_function (command)
  print ('Hello world: ' .. command)
  return 'ok', 'Everything is fine!', 'fine=10%;80;90;'
```

As of now there are no support for arguments but in the future they will be added. Printing from a check_function is useless (apart from debug) so generally don't do that. The return is a variable list If;

- 3 options are returned they are assumed to be in order: code, message and performance data
- 2 options are returned they are assumed to be in order: code, message
- 1 options are returned they are assumed to be in order: code

The code can be:

- crit (critical)
- warn (warning)
- ok (ok)
- error (critical)

A 'useful' script

```
-- Register the command
nscp.register('has', 'check_file_exists')

-- Return true if file exists and is readable.
function file_exists(path)
  local file = io.open(path, "rb")
  if file then file:close() end
  return file ~= nil
end

function check_file_exists (command)
  if file_exists('c:\\foo.bar') then
    return 'ok', 'File exists'
  else
    return 'crit', 'File does not exist'
  end
end
```

NSCAAgent.dll

This module periodically runs a set of check_commands and submits the results to an NSCA server.

Configuration

This page describes the configuration options for the NSCA module.

NSCA Agent Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Agent

Ovreview

Options to configure the NSCA module.

Option	Default value	Description
interval	60	Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")
nsca_host	...	The NSCA/Nagios(?) server to report results to.
nsca_port	5667	The NSCA server port
encryption_method	1	Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.
password		The password to use. Again has to be the same as the server or it won't work at all.

Advanced options:

Option	Default value	Description
hostname		The host name of this host if set to blank (default) the windows name of the computer will be used.
debug_threads	1	**DEBUG**Number of threads to run, no reason to change this really (unless you want to stress test something)

interval

Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")

Value
Any positive integer (time in seconds)
Default
60 (seconds)

nsca_host

The NSCA/Nagios(?) server to report results to.

Values
Hostname or IP address to submit back results to.
Default
Empty string (will in 3.7 and above mean don't submit results)

nsca_port

The NSCA server port

Values
Any positive integer (port number ought to be less then 65534)
Default
5667

encryption_method

Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.

Values

Supported encryption methods:

#	Algorithm
0	None (**Do NOT use** this option)
1	Simple XOR (**No security, just obfuscation**, but very fast)
2	DES
3	3DES (Triple DES)
4	CAST-128
6	xTEA
8	BLOWFISH
9	TWOFISH
11	RC2
14	RIJNDAEL-128 (AES)
20	SERPENT

Default
1 (I am note sure I thought default was 14?)

password

The password to use. Again has to be the same as the server or it won't work at all.

Values
Any string (should be the same as the one configured in nsca.conf

hostname

The host name of this host if set to blank (default) the windows name of the computer will be used.

Values
Any string (or auto)
Default
auto (means windows hostname will be used)

debug_threads

DEBUGNumber of threads to run, no reason to change this really (unless you want to stress test something)

Values
Any positive integer larger then or equal to 1
Default
1

NSCA Commands Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Commands

1. 1. 1. 1. Overview

Overview

A list of commands to run and submit each time we report back to the NSCA server. A command starting with host_ will be submitted as a host command. For an example see below: This will report back one service check (called my_cpu_check) and one host check (host checks have no service name).

```
[NSCA Commands]
my_cpu_check=checkCPU warn=80 crit=90 time=20m time=10s time=4
host_check=check_ok
```

Configuration for the NSCAAgent

This page describes the configuration options for the NSCA module.

NSCA Agent Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Agent

Ovreview

Options to configure the NSCA module.

Option	Default value	Description
interval	60	Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")
nsca_host	...	The NSCA/Nagios(?) server to report results to.
nsca_port	5667	The NSCA server port
encryption_method	1	Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.
password		The password to use. Again has to be the same as the server or it won't work at all.

Advanced options:

Option	Default value	Description
hostname		The host name of this host if set to blank (default) the windows name of the computer will be used.
debug_threads	1	**DEBUG**Number of threads to run, no reason to change this really (unless you want to stress test something)

interval

Time in seconds between each report back to the server (cant as of yet be set individually so this is for all "checks")

Value
: Any positive integer (time in seconds)

Default
: 60 (seconds)

nsca_host

The NSCA/Nagios(?) server to report results to.

Values
: Hostname or IP address to submit back results to.

Default
: Empty string (will in 3.7 and above mean don't submit results)

nsca_port

The NSCA server port

Values
: Any positive integer (port number ought to be less then 65534)

Default
: 5667

encryption_method

Number corresponding to the various encryption algorithms (see below). Has to be the same as the server or it wont work at all.

Values

Supported encryption methods:

#	Algorithm
0	None (**Do NOT use** this option)
1	Simple XOR (**No security, just obfuscation**, but very fast)
2	DES
3	3DES (Triple DES)
4	CAST-128
6	xTEA
8	BLOWFISH
9	TWOFISH
11	RC2
14	RIJNDAEL-128 (AES)
20	SERPENT

Default
: 1 (I am note sure I thought default was 14?)

password

The password to use. Again has to be the same as the server or it won't work at all.

Values
: Any string (should be the same as the one configured in nsca.conf

hostname

The host name of this host if set to blank (default) the windows name of the computer will be used.

Values
    Any string (or auto)
Default
    auto (means windows hostname will be used)

debug_threads

DEBUGNumber of threads to run, no reason to change this really (unless you want to stress test something)

Values
    Any positive integer larger then or equal to 1
Default
    1

NSCA Commands Section

This is a wrapper page the actual data is on the following page NSCAAgent/config/NSCA_Commands

Overview

A list of commands to run and submit each time we report back to the NSCA server. A command starting with host_ will be submitted as a host command. For an example see below: This will report back one service check (called my_cpu_check) and one host check (host checks have no service name).

```
[NSCA Commands]
my_cpu_check=checkCPU warn=80 crit=90 time=20m time=10s time=4
host_check=check_ok
```