



Icinga Version 1.4 Documentation

[Next](#)

Icinga Version 1.4 Documentation

Copyright 2009-2011 Icinga Development Team.

Portions copyright © by Nagios/Icinga community members - see the THANKS file in the Icinga core sources for more information..

Credits to Yoann LAMY for creating the Vautour Style we use for the Icinga Classic UI

Icinga is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation. This gives you legal permission to copy, distribute and/or modify Icinga under certain conditions. Read the 'LICENSE' file in the Icinga distribution or read the online version of the license for more details.

Icinga is provided "AS IS" with "NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE."

Nagios is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation. This gives you legal permission to copy, distribute and/or modify Nagios under certain conditions. Read the 'LICENSE' file in the Nagios distribution or read the online version of the license for more details.

Nagios and the Nagios logo are registered trademarks of Ethan Galstad. All other trademarks, servicemarks, registered trademarks, and registered servicemarks mentioned herein may be the property of their respective owner(s). The information contained herein is provided "AS IS" with "NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE."

2011.05.11

Revision History	
Revision 1.4	2011-05-11
1.4 Icinga Documentation	
Revision 1.x	2010/2011
1.x Icinga Documentation	
Revision 0.1	2009-08-12
First Release	

Table of Contents

- 1. About
 - [About Icinga](#)
 - [What's New in Icinga 1.4](#)
- 2. Getting Started
 - [Advice for beginners](#)
 - [Quickstart Installation Guides](#)
 - [Icinga Quickstart](#)
 - [Icinga Quickstart FreeBSD](#)
 - [Icinga with IDOUtils Quickstart](#)
 - [Icinga and IDOUtils Quickstart on FreeBSD](#)
 - [Links to other published Howtos](#)
 - [Upgrading \(to\) Icinga](#)
 - [Upgrading IDOUtils Database](#)
 - [Monitoring Windows Machines](#)
 - [Monitoring Linux/Unix Machines](#)
 - [Monitoring Netware Servers](#)
 - [Monitoring Network Printers](#)
 - [Monitoring Routers and Switches](#)
 - [Monitoring Publicly Available Services](#)
- 3. Configuring Icinga
 - [Configuration Overview](#)
 - [Main Configuration File Options](#)
 - [Object Configuration Overview](#)
 - [Object Definitions](#)
 - [Host definition](#)
 - [Hostgroup Definition](#)
 - [Service Definition](#)
 - [Servicegroup Definition](#)
 - [Contact Definition](#)
 - [Contactgroup Definition](#)
 - [Timeperiod Definition](#)
 - [Command Definition](#)
 - [Servicedependency Definition](#)
 - [Serviceescalation Definition](#)
 - [Hostdependency Definition](#)
 - [Hostescalation Definition](#)
 - [Hostextinfo Definition](#)
 - [Serviceextinfo Definition](#)

- Module Definition
- Custom Object Variables
- CGI Configuration File Options
- Authentication And Authorization In The CGIs
- 4. Running Icinga
 - Verifying Your Configuration
 - Starting and Stopping Icinga
- 5. The Basics
 - Icinga Plugins
 - Understanding Macros and How They Work
 - Standard Macros in Icinga
 - Host Checks
 - Service Checks
 - Active Checks
 - Passive Checks
 - State Types
 - Time Periods
 - Determining Status and Reachability of Network Hosts
 - Notifications
- 6. User Interfaces
 - Icinga Classic UI: Information On The CGIs
 - Information On CGI parameters
 - Executing CGIs on the command line
 - Installation of the Icinga-Web Frontend
 - Upgrading Icinga-Web and Icinga-Web Database
 - Configuration Overview of Icinga-Web
 - Introduction to Icinga-Web
 - Introduction to Icinga-Web (up to 1.2.x)
 - Introduction to Icinga-Web (>= 1.3.x)
 - Integration of PNP4Nagios into Icinga-Web
- 7. Advanced Topics
 - External Commands
 - Event Handlers
 - Volatile Services
 - Service and Host Freshness Checks
 - Distributed Monitoring
 - Redundant and Failover Network Monitoring
 - Detection and Handling of State Flapping
 - Notification Escalations
 - Escalation Condition
 - On-Call Rotations
 - Monitoring Service and Host Clusters
 - Host and Service Dependencies
 - State Stalking
 - Performance Data
 - Scheduled Downtime
 - Using The Embedded Perl Interpreter
 - Adaptive Monitoring
 - Predictive Dependency Checks
 - Cached Checks
 - Passive Host State Translation
 - Service and Host Check Scheduling
 - Custom CGI Headers and Footers

- [Object Inheritance](#)
- [Time-Saving Tricks For Object Definitions](#)
- [8. Security and Performance Tuning](#)
 - [Security Considerations](#)
 - [Enhanced CGI Security and Authentication](#)
 - [Tuning Icinga For Maximum Performance](#)
 - [Fast Startup Options](#)
 - [Large Installation Tweaks](#)
 - [Using The Icingastats Utility](#)
 - [Graphing Performance Info With PNP4Nagios](#)
 - [Temporary Data](#)
- [9. Integration With Other Software](#)
 - [Integration Overview](#)
 - [SNMP Trap Integration](#)
 - [TCP Wrapper Integration](#)
 - [MKLiveStatus Integration](#)
 - [Installation of the Icinga-Reporting with JasperServer](#)
- [10. Additional software](#)
 - [Icinga Addons](#)
 - [NRPE](#)
 - [NSCA](#)
- [11. Development](#)
 - [Icinga Plugin API](#)
 - [Developing Plugins For Use With Embedded Perl](#)
 - [List of External Commands](#)
 - [Installation and use of the Icinga API](#)
 - [The Icinga-Web REST API](#)
- [12. IDOUtils](#)
 - [Introduction](#)
 - [Purpose](#)
 - [Design Overview](#)
 - [Instances](#)
 - [Installation](#)
 - [Components](#)
 - [Overview](#)
 - [IDOMOD](#)
 - [LOG2IDO](#)
 - [FILE2SOCK](#)
 - [IDO2DB. IDO2DB](#)
 - [Example Configurations](#)
 - [Single Server, Single Instance Setup](#)
 - [Single Server, Multiple Instance Setup](#)
 - [Single Server, Single Instance Log File Import](#)
 - [IDOUtils Database Model](#)
 - [Central Tables](#)
 - [Debugging Tables](#)
 - [Historical Tables](#)
 - [Current Status Tables](#)
 - [Configuration Tables](#)
 - [Database changes/alterations](#)
- [Index](#)

List of Figures

- 3.1. [Example of new header](#)
- 6.1. [Icinga-Web login screen](#)
- 6.2. [Icinga-Web overview](#)
- 6.3. [Icinga-Web main screen](#)
- 6.4. [Icinga-Web status cronk](#)
- 6.5. [Icinga-Web top menu](#)
- 6.6. [Icinga-Web data cronks](#)
- 6.7. [Icinga-Web tactical overview cronks](#)
- 6.8. [Icinga-Web misc cronks](#)
- 6.9. [Icinga-Web live search](#)
- 6.10. [Icinga-Web log](#)
- 6.11. [Icinga-Web cronk bar](#)
- 6.12. [Icinga-Web cronk bar](#)
- 6.13. [Icinga-Web host commands](#)
- 6.14. [Icinga-Web service commands](#)
- 6.15. [Icinga-Web filter restriction](#)
- 6.16. [Icinga-Web filter condition](#)
- 6.17. [Icinga-Web filter active](#)
- 6.18. [Icinga-Web top menu admin](#)
- 6.19. [Icinga-Web user admin](#)
- 6.20. [Icinga-Web edit user](#)
- 6.21. [Icinga-Web group admin](#)
- 6.22. [Icinga-Web groups](#)
- 6.23. [Icinga-Web principals](#)
- 6.24. [Icinga-Web logs](#)
- 6.25. [Icinga-Web login screen](#)
- 6.26. [Icinga-Web overview](#)
- 6.27. [Icinga-Web main screen](#)
- 6.28. [Icinga-Web status cronk](#)
- 6.29. [Icinga-Web top menu](#)
- 6.30. [Icinga-Web data cronks](#)
- 6.31. [Icinga-Web tactical overview cronks](#)
- 6.32. [Icinga-Web misc cronks](#)
- 6.33. [Icinga-Web live search](#)
- 6.34. [Icinga-Web log](#)
- 6.35. [Icinga-Web cronk bar](#)
- 6.36. [Icinga-Web cronk bar](#)
- 6.37. [Icinga-Web host commands](#)
- 6.38. [Icinga-Web service commands](#)
- 6.39. [Icinga-Web filter restriction](#)
- 6.40. [Icinga-Web filter condition](#)
- 6.41. [Icinga-Web filter active](#)
- 6.42. [Icinga-Web top menu admin](#)
- 6.43. [Icinga-Web user admin](#)
- 6.44. [Icinga-Web edit user](#)
- 6.45. [Icinga-Web group admin](#)
- 6.46. [Icinga-Web groups](#)
- 6.47. [Icinga-Web principals](#)
- 6.48. [Icinga-Web logs](#)
- 6.49. [Icinga-Web Tasks](#)
- 6.50. [PNP4Nagios integrated in Icinga-Web](#)
- 7.1. [Cached checks](#)
- 8.1. [Average Host / Service Check Latency](#)

- [8.2. Service Statistics](#)
 - [8.3. Host Statistics](#)
 - [8.4. Average Execution Times](#)
 - [8.5. External Commands](#)
 - [8.6. External Command Buffers](#)
 - [8.7. Cached Host and Service Checks](#)
 - [8.8. Average State Changes](#)
 - [9.1. Icinga-Reporting in Icinga-Web](#)
 - [9.2. Icinga-Reporting TOP10 in Icinga-Web](#)
 - [10.1. NRPE](#)
 - [10.2. NRPE remote](#)
 - [10.3. NSCA](#)
 - [12.1. Multiple instances](#)
 - [12.2. Future development: One Instance, multiple databases](#)
 - [12.3. Instance names based on geographical locations](#)
 - [12.4. Instance names based on their purpose](#)
 - [12.5. Loaded IDOMOD Event broker Module](#)
 - [12.6. IDOMOD Capabilities](#)
 - [12.7. LOG2IDO Utility](#)
 - [12.8. FILE2SOCK Utility](#)
 - [12.9. IDO2DB Daemon](#)
 - [12.10. IDO2DB with multiple Clients](#)
 - [12.11. Single Server, Single Instance Setup](#)
 - [12.12. Single Server, Multiple Instance Setup](#)
 - [12.13. Single Server, Single Instance Log File Import](#)
 - [12.14. Relationship of Central Tables](#)
 - [12.15. Relationship of Debugging Tables](#)
 - [12.16. Relationship of Historical Tables](#)
 - [12.17. Relationship of Current Status Tables](#)
 - [12.18. Relationship of Configuration Tables](#)
-

[Next](#)[Chapter 1. About](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 1. About

[Prev](#)

[Next](#)

Chapter 1. About

Table of Contents

[About Icinga](#)

[What's New in Icinga 1.4](#)

[Prev](#)

[Next](#)

[Icinga Version 1.4 Documentation](#)

[Home](#)

[About Icinga](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



About Icinga

[Prev](#)

Chapter 1. About

[Next](#)

About Icinga

What is Icinga?

Icinga is a monitoring system checking hosts and services you specify and notifying you when things go wrong and when they recover. It runs on multiple Linux distributions (including [Fedora](#), [Ubuntu](#), and [openSuSE](#)) as well as several Unix platforms (including [Solaris](#) and [HP](#)). The systems to be monitored can be nearly anything connected to a network.

Some of the many features of Icinga include:

- Monitoring of network services (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Monitoring of host resources (CPU load, disk usage, etc.)
- Simple plugin design that allows users to easily develop their own service checks
- Parallelized service checks
- Ability to define network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable
- Contact notifications when service or host problems occur and get resolved (via email, pager, or user-defined method)
- Ability to define event handlers to be run during service or host events for proactive problem resolution
- Automatic log file rotation
- Support for implementing redundant monitoring hosts
- Optional classic web interface for viewing current network status, notification and problem history, log file, etc.
- Optional new Icinga web interface based on Icinga Core, IDOUtils, API using a modern and refreshed web 2.0 GUI showing current states, historical information, using cronks and filters, creating reports with multilanguage support

System requirements

As mentioned above you'll need a machine running Linux or a Unix variant. If there is no precompiled version or if you want to compile from source you need a C compiler like gcc.



Note

Some compilers may not be suitable. That includes the C compiler on HP-UX which is used to compile a new kernel.

You may also want to have TCP/IP configured as most checks will require access via the network.

You are not required to use one of the web interfaces included with Icinga. However, if you do decide to use them, you will need additional software:

1. A Web-Server (preferably [Apache](#))
2. Thomas Boutell's [gd library](#) Version 1.6.3 or higher (required by the [statusmap](#)- and [trends](#)-CGIs)
3. PHP

Licensing

Icinga is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation. This gives you legal permission to copy, distribute and/or modify Icinga under certain conditions. Read the 'LICENSE' file in the Icinga distribution or read the online version of the license for more details. Icinga is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgements

Several people have contributed to Icinga by either reporting bugs, suggesting improvements, writing plugins, etc. A list of some of the many contributors to the development of Icinga can be found at <http://www.icinga.org/>.

Downloading The Latest Version

You can check for new versions of Icinga at <http://www.icinga.org/>.

Compatibility

Icinga is a fork of the well-known monitoring system [Nagios](#). Being 100% compatible with the internal structures of the latter Icinga enables you to use all plugins and add-ons which were/are developed by several companies and the large community.

[Prev](#)

[Up](#)

[Next](#)

[Chapter 1. About](#)

[Home](#)

[What's New in Icinga 1.4](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



What's New in Icinga 1.4

[Prev](#)

[Chapter 1. About](#)

[Next](#)

What's New in Icinga 1.4

1.4.0

Icinga-Core:

- re-allow perfdata with empty results being put on perfdata channel, disable via opt-in cfg option
- add downtime delete commands made "distributable" by deleting by host group name, host name or start time/comment (Opsview team)
- add 'module' as object config, allowing cfg_dir usage loading multiple modules without touching broker_module in icinga.cfg
- fix: flexible downtime on service hard state change doesn't get triggered/activated
- fix: timeperiods daylight saving time problem (Luca Di Stefano)

Icinga IDOUtils:

- add db socket as config option in ido2db.cfg for mysql and postgresql
- reduce housekeeping cycle to 3600s, set housekeeping thread startup delay to 300s
- introduce schema version and check against that instead of program version
- install sample (commented) config in modules/idoutils.cfg using new 'module' object config
- fix: update oracle hints in ido2db.cfg with tnsnames.ora and port cfg
- fix: idomod: larger buffer size (by Opsview)
- fix: rdbms disconnect after connection error
- fix: race condition when issuing multiple reloads results in hanging IDO2DB processes
- fix: postgresql: integer not big enough for bytes_processed (Stig Sandbeck)

Icinga Classic UI:

- merged reading of logfiles into one function. It's easier now to add enhancements.
- adding some more icons to showlog.cgi
- adding entry time of comments in tooltip's in status.cgi
- searching in the Icinga Logfile
- changed print_generic_error() function to support csv output
- added parameter to get_log_entries() function to use beginning and end timestamp
- show downtime in host detail and service detail view
- store cmd.cgi submissions in log
- enforce a need for comment for action taken in cmd.cgi
- add config option to set start of week (sunday/monday)
- allow display of Network Outages for authorized hosts (thx mjbrooks)
- remove useless memory allocation when reading logfiles reverse (lifo)
- speed up data processing in summary.cgi
- add an optional alternative CGI driven view for the top frame (Matthew Brooks)
- added json output "&jsonoutput" to nearly all pages in classic ui
- allow searching for host display_name normal and via regexp
- display host/service dependencies in host/service details in extinfo.cgi
- fix: tooltip's in status.cgi, not showing messages with carriage return
- fix: csv export link to make it XSS save (IE)
- fix: cmd.cgi: acknowledgement multiline comment -> command not being processed
- fix: statusmap.cgi: fixed XSS vulnerability
- fix: display_name survive reconfiguration and is use instead of host_name in classic ui
- fix: don't show pause/continue urls on non-refreshable pages
- fix: segfaults if no default_user_name= given in cgi.cfg
- fix: Prevent statusmap.cgi markup from drawing when host should not be drawn (Matthew Brooks)

Config / Install:

- config: increase default debug file size to 100M
- install: add --with-ext-cmd-file-dir= to configure, allowing icinga.cmd dir to be altered

- fix: install: use *.so instead of *.o for solaris, patch in contrib/solaris/

Icinga-API:

- SID support for Oracle
- Added missing commands
- UTF-8 output support (database independent)
- Enhanced query data support (fields, joins)
- Consistent pending status boolean values

Icinga-Web:

- Better check_multi integration
- Seamless pending status integration
- Seamless integration for alias and display names (Search and display)
- LDAP Auth hardening
- Privileges for logging data (view based)
- Usability changes for buttons, controls, single clicks, etc
- Separated refresh settings
- check for Icinga being loaded completely
- fix: session cookie lifetime
- fix: missing commands: remove acknowledgements
- Portal view fixes and cronk integration
- Typos and translations

Icinga-Docs:

- provide basic NRPE documentation
- add cgiparams / filter properties
- reformat several listings (too wide in PDF)
- add colours to HTML pages (xsl stylesheet)

[More at Icinga Wiki](#)

Should you find any issues, please report them to one of the following links:

[Report Issue Icinga-Core](#)

[Report Issue Icinga-Web](#)

[Report Issue Icinga-API](#)

[Report Issue Icinga-Reporting](#)

[Report Issue Icinga-Docs](#)

[Prev](#)

[Up](#)

[Next](#)

[About Icinga](#)

[Home](#)

[Chapter 2. Getting Started](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 2. Getting Started

[Prev](#)

[Next](#)

Chapter 2. Getting Started

Table of Contents

- [Advice for beginners](#)
 - [Quickstart Installation Guides](#)
 - [Icinga Quickstart](#)
 - [Icinga Quickstart FreeBSD](#)
 - [Icinga with IDOUtils Quickstart](#)
 - [Icinga and IDOUtils Quickstart on FreeBSD](#)
 - [Links to other published Howtos](#)
 - [Upgrading \(to\) Icinga](#)
 - [Upgrading IDOUtils Database](#)
 - [Monitoring Windows Machines](#)
 - [Monitoring Linux/Unix Machines](#)
 - [Monitoring Netware Servers](#)
 - [Monitoring Network Printers](#)
 - [Monitoring Routers and Switches](#)
 - [Monitoring Publicly Available Services](#)
-

[Prev](#)

[Next](#)

[What's New in Icinga 1.4](#)

[Home](#)

[Advice for beginners](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Advice for beginners**[Prev](#)**Chapter 2. Getting Started**[Next](#)

Advice for beginners

Thanks for choosing Icinga! Icinga is quite powerful and flexible and although you can see the first results in less than 30 minutes, this is just the beginning of your journey into system monitoring. Configuring it to your needs can take a lot of work depending on the number of things to be monitored and the complexity of your environment. Once you become familiar with how it works and what it can do for you, you'll never want to be without it. :-) Here are some important things to keep in mind for first-time Icinga users:

1. Relax - it's going to take some time. Don't expect to be able to get things working exactly the way you want them right off the bat. It's not that easy. Setting up Icinga can involve a bit of work - partly because of the options that Icinga offers, partly because you need to know what to monitor on your network (and how best to do it)
2. Please use the [quickstart](#) instructions. The quickstart installation guide is designed to get most new users up and running with a basic Icinga setup fairly quickly. Within 20 minutes you can have Icinga installed and monitoring your local system. Once that's complete, you can move on to learning how to configure Icinga to do more.
3. Read the documentation. Icinga can be tricky to configure when you've got a good grasp of what's going on, and nearly impossible if you don't. Make sure you read the documentation (particularly the sections on "[Configuring Icinga](#)" and "[The Basics](#)"). Save the [advanced topics](#) for when you've got a good understanding of the basics.
4. Seek the help of others. If you've read the documentation, reviewed the sample config files, and are still having problems, send an email message describing your problems to the [icinga-users mailing list](#). If you've done some background reading and you provide a good problem description, odds are that someone will give you some pointers on getting things working properly. "good description" means to include details on your operating system, the version of Icinga, as well as what you tried and the results of it (preferably copy&paste). More information on subscribing to the mailing lists or searching the list archives can be found at <http://www.icinga.org/community/>.

[Prev](#)[Up](#)[Next](#)[Chapter 2. Getting Started](#)[Home](#)[Quickstart Installation Guides](#)



Quickstart Installation Guides

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Quickstart Installation Guides

Introduction

These quickstart guides will provide you with simple instructions on how to install Icinga from source and have it monitoring your local machine within 20 to 30 minutes. Advanced installation options and configuration are discussed elsewhere.

Links to configuration tools and other addons can be found [here](#).

Besides, a new quick install guide for using the IDOUtils of Icinga is also available enabling you to use an RDBMS based on MySQL, Oracle, or PostgreSQL, respectively.

Guides

Quickstart installation guides are currently available for various Linux distributions (most common Fedora, Ubuntu, openSuSE). Because some distributions are similar you may succeed to use these instructions for RedHat, CentOS, Debian and SLES as well. Marcel Hecko provided a quickstart quide for FreeBSD.



Note

Please note that sometimes names of packages change between different versions of the same OS so if you don't find the packages mentioned it might be a good idea to use your favourite search engine to find out the correct name.

The [Quickstart guide for Linux](#) and [Quickstart guide for FreeBSD](#), resp., provide you with instructions on how to install Icinga, basic plugins to check several things and the classical GUI accessible through your web browser. The look and feel is quite similar to Nagios although the GUI offers some enhancements (extended CSV export, submit commands for a group of objects, etc.). There is no database involved and all information is stored in flat files. Icinga-Web is *not* available in this setup.

Using the [Quickstart guide including IDOUtils](#) and [Quickstart guide including IDOUtils on FreeBSD](#), resp., you will get the things mentioned above plus a database (using MySQL, Oracle, or PostgreSQL) to store current and historical information. The database is *not* used to enable configuration via a web interface. There are several [addons](#) specialised on this and there are no plans for a builtin tool. The new web interface can be installed when you use this setup. There are [separate instructions](#) on how to do this after you installed Icinga and IDOUtils.

- See: [Quickstart guide for Linux](#)
- See: [Quickstart guide for FreeBSD](#)
- See: [Quickstart guide including IDOUtils](#)
- See: [Quickstart guide including IDOUtils on FreeBSD](#)

If you are installing Icinga on an operating system or Linux distribution that isn't listed above, read [Quickstart guide for Linux](#) for an overview on what you'll need to do.

Command names, paths, etc. vary widely across different Operating Systems/Distributions, so you'll likely need to tweak the installation docs a bit to work for your particular case.

[Prev](#)[Up](#)[Next](#)[Advice for beginners](#)[Home](#)[Icinga Quickstart](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Quickstart

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Icinga Quickstart

Introduction

This guide is intended to provide you with simple instructions on how to install Icinga from source (code) and have it monitoring your local machine within 20 minutes.

No advanced installation options are discussed here - just the basics that will work for most of the users who want to get started.

This guide will give you examples for currently three different Linux distributions: [Fedora](#), [Ubuntu](#) and [openSuSE](#). Similar distributions may work as well. That should include [RedHat](#), [CentOS](#), [Debian](#) and [SLES](#).

For instructions on how to install Icinga on FreeBSD please read [Icinga on FreeBSD](#) instead.

Other distributions may inherit from these examples.

If you are planning to use a database with IDOUtils, or wish to use the new web interface then please read [Icinga with IDOUtils](#) instead!

What You'll End Up With

If you follow these instructions, here's what you'll end up with:

- Icinga and the plugins will be installed underneath /usr/local/icinga
- Icinga will be configured to monitor a few aspects of your local system (CPU load, disk usage, etc.)
- The Icinga classic web interface will be accessible at <http://localhost/icinga/> or <http://yourdomain.com/icinga>

Prerequisites

During portions of the installation you'll need to have **root** access to your machine.

Make sure you've installed the following packages on your system before continuing.

- [Apache](#)

- GCC compiler
- C/C++ development libraries
- [GD](#) development libraries

Optional

At one time or another you may need to use SNMP-based checks so it is a good idea to install the required packages now. Otherwise some plugins are not compiled i.e. not available when you need them and it would require a recompile of the plugins.

Install packages

You can install these packages by running the following commands (as root or sudo):

Fedora/RHEL/CentOS

```
#> yum install httpd gcc glibc glibc-common gd gd-devel
#> yum install libjpeg libjpeg-devel libpng libpng-devel
#> yum install net-snmp net-snmp-devel net-snmp-utils
```

Debian/Ubuntu

```
#> apt-get install apache2 build-essential libgd2-xpm-dev
#> apt-get install libjpeg62 libjpeg62-dev libpng12 libpng12-dev
#> apt-get install snmp libsnmp5-dev
```



Note

The numbers <62/12> might differ, depending on your distribution

openSuSE/SLES

Please use YaST to install at least the packages gd, gd-devel, libjpeg, libjpeg-devel, libpng, libpng-devel and, optionally, net-snmp, net-snmp-devel and perl-Net-SNMP.



Note

Depending on the software selection during the installation of the OS you may need to install additional packages (i.e. apache2, gcc). The devel packages might be placed on the SDK DVDs.

Create Account Information

Become the root user.

```
$> su -l
```

Create a new *icinga* user account and give it a password:

```
#> /usr/sbin/useradd -m icinga
#> passwd icinga
```

On some distributions you'll need to add the group in a single step:

```
#> /usr/sbin/groupadd icinga
```

For sending commands from the classic web interface to Icinga, you'll need to create a new group icinga-cmd. Add the webuser and the Icinga user to this group:

```
#> /usr/sbin/groupadd icinga-cmd
#> /usr/sbin/usermod -a -G icinga-cmd icinga
#> /usr/sbin/usermod -a -G icinga-cmd www-data
```

(or www, wwwrun, apache, depending on the distribution)



Note

Some usermod-versions (e.g. in OpenSuSE 11 and SLES 11, resp.) are lacking the option -a. In this case please omit the option -a.



Note

Solaris only supports groupnames with max. 8 characters, please use icingcmd instead of icinga-cmd.

Download Icinga and the Plugins

Change to your local source directory i.e. /usr/src

```
#> cd /usr/src
```

Either fetch the current icinga-core snapshot from Icinga GIT

```
#> git clone git://git.icinga.org/icinga-core.git
```

or from the [Icinga Website](#).

Don't forget to download the [Nagios Plugins](#).

Compile and install Icinga

Extract the Icinga source code tarball (or change directory to the GIT snapshot)

```
#> cd /usr/src/
#> tar xvzf icinga-1.4.tar.gz
#> cd icinga-1.4
```

Run the Icinga configure script. You will get help by using the --help flag.

```
#> ./configure --with-command-group=icinga-cmd
```

Compile the Icinga source code. To see available options, only use "make".

```
#> make all
```

Install binaries, init script, sample config files and set permissions on the external command directory.

```
#> make install
#> make install-init
#> make install-config
#> make install-commandmode
```

or shorter

```
#> make fullinstall
```

Don't start Icinga yet - there's still more that needs to be done...

Customise Configuration

Sample configuration files have been installed by using

```
#> make install-config
```

into `/usr/local/icinga/etc/`. You'll need to make just one change before you proceed...

Edit the `/usr/local/icinga/etc/objects/contacts.cfg` config file with your favourite editor and change the email address associated with the `icingaadmin` contact definition to the address you'd like to use for receiving alerts.

```
#> vi /usr/local/icinga/etc/objects/contacts.cfg
```

Configure the Classic Web Interface

Icinga ships with the Classic Web Interface ("the CGIs") which can be installed via

```
#> make cgis
#> make install-cgis
#> make install-html
```

If you are interested in the new Icinga Web, please refer to [Install Icinga Web Interface](#).

Install the Icinga Classic web config file in the Apache conf.d directory.

```
#> make install-webconf
```

Create an `icingaadmin` account for logging into the Icinga classic web interface. If you want to change it later, use the same command. Remember the password you assign to this account - you'll need it later.

```
#> htpasswd -c /usr/local/icinga/etc/htpasswd.users icingaadmin
```

If you want to change it later or want to add another user, use the following command:

```
#> htpasswd /usr/local/icinga/etc/htpasswd.users <USERNAME>
```



Note

Depending on your distribution/Apache-version you may have to use `htpasswd2` instead.

Reload/Restart Apache to make the new settings take effect.

Fedora/RHEL/CentOS

```
#> service httpd restart
```

Debian/Ubuntu/openSUSE

```
#> /etc/init.d/apache2 reload
```

Compile and Install the Nagios Plugins

Extract the Nagios plugins source code tarball.

```
#> cd /usr/src
#> tar xvzf nagios-plugins-1.4.15.tar.gz
#> cd nagios-plugins-1.4.15
```

Compile and install the plugins by changing install directory to /usr/local/icinga

```
#> ./configure \
--prefix=/usr/local/icinga --with-cgiurl=/icinga/cgi-bin --with-htmurl=/icinga \
--with-nagios-user=icinga --with-nagios-group=icinga
#> make
#> make install
```

Adjusting the SELinux settings

RHEL and derived distributions like Fedora and CentOS are shipped with activated SELinux (Security Enhanced Linux) running in "enforcing" mode. This may lead to "Internal Server Error" messages when you try to invoke the Icinga-CGI's.

Check if SELinux runs in enforcing mode

```
#> getenforce
```

Set SELinux in "permissive" mode

```
#> setenforce 0
```

To make this change permanent you have to adjust this setting in */etc/selinux/config* and restart the system.

Instead of deactivating SELinux or setting it into permissive mode you can use the following commands to run the CGI's in enforcing/targeted mode:

```
#> chcon -R -t httpd_sys_script_exec_t /usr/local/icinga/sbin/
#> chcon -R -t httpd_sys_content_t /usr/local/icinga/share/
#> chcon -t httpd_sys_script_rw_t /usr/local/icinga/var/rw/icinga.cmd
```

Visit the NagiosCommunity.org-Wiki at <http://www.nagioscommunity.org/wiki> to get information how to run the Icinga-CGI's in enforcing mode with a targeted policy.

Start Icinga

Add Icinga to the list of system services and have it automatically start when the system boots (make sure you have installed the init script before).

Fedora/RHEL/CentOS/openSuSE

```
#> chkconfig --add icinga
#> chkconfig icinga on
```

Debian/Ubuntu

```
#> update-rc.d icinga defaults
```

Verify the sample Icinga configuration files.

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
```

Instead of specifying the paths to binary and config file you can issue

```
#> /etc/init.d/icinga show-errors
```

which results in an OK message if everything is fine or several lines which show the location of the error(s).

If there are no errors, start Icinga.

Fedora/RHEL/CentOS/Ubuntu

```
#> service icinga start
```

Debian/openSuSE

```
#> /etc/init.d/icinga start
```

Login to the Classic Web Interface

You should now be able to access the Icinga classic web interface at the URL below. You'll be prompted for the username (*icingaadmin*) and password you specified earlier.

```
http://localhost/icinga/
```

or

```
http://yourdomain.com/icinga/
```

Click on the "Service Detail" navbar link to see details of what's being monitored on your local machine. It will take a few minutes for Icinga to check all the services associated with your machine.

Other Modifications

Make sure your system's firewall rules are configured to allow access to the web server if you want to access the Icinga classic interface remotely.

```
#> iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

Setting up your mail transfer agent (MTA) like exim, sendmail or postfix to allow Icinga sending notification emails won't be explained here.

Please refer to the [Nagios-Wiki](#) for more resources.

You're Done

Congratulations! You successfully installed Icinga. Your journey into monitoring has just begun.

You'll no doubt want to monitor more than just your local machine, so check out the chapter on "[Getting Started](#)" about "Monitoring ..."

[Prev](#)

[Up](#)

[Next](#)

[Quickstart Installation Guides](#)

[Home](#)

[Icinga Quickstart FreeBSD](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Quickstart FreeBSD

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Icinga Quickstart FreeBSD

Introduction

This guide is intended to provide you with simple instructions on how to install Icinga from source (code) and have it monitoring your local machine within 20 minutes.

No advanced installation options are discussed here - just the basics that will work for most of the users who want to get started.

This guide will give you examples for installation on [FreeBSD 7.2](#).

Later distributions of FreeBSD may inherit from these examples.

What You'll End Up With

If you follow these instructions, here's what you'll end up with:

- Icinga and the plugins will be installed underneath /usr/local/icinga
- Icinga will be configured to monitor a few aspects of your local system (CPU load, disk usage, etc.)
- The Icinga classic web interface will be accessible at <http://localhost/icinga/> or <http://yourdomain.com/icinga>

Prerequisites

During portions of the installation you'll need to have **root** access to your machine.

Make sure you've installed the following packages on your system before continuing.

- [Apache](#)
- GCC compiler
- C/C++ development libraries
- [GD](#) development libraries

Install ports

You can install these ports by running the following commands (as root):

Please update your ports before doing so.

```
#> cd /usr/ports/devel/libtool22/ && make deinstall && make clean && make install
#> cd /usr/ports/graphics/jpeg && make deinstall && make clean && make && make install
#> cd /usr/ports/graphics/png && make deinstall && make clean && make && make install
#> cd /usr/ports/graphics/gd && make deinstall && make clean && make && make install
```



Note

Please make sure you have Apache installed - the process will not be discussed here, however the lead is `# cd /usr/ports/www/apache22 && make clean && make`.

Create Account Information

Become the root user.

```
$> su -l
```

Create a new *icinga* user account without a password and without the ability to log-in (set no password when asked):

```
#> adduser -D -w no -s nologin
```

For sending commands from the classic web interface to Icinga, you'll need to create a new group *icinga-cmd* and add the webuser (*www*) and the *Icingauser* to this group:

```
#> pw groupadd -n icinga-cmd -M icinga,www
```

Download Icinga and the Plugins

Change to your local source directory i.e. `~/src`

```
#> mkdir ~src
#> cd ~src
```

Either fetch the current *icinga-core* snapshot from Icinga GIT

```
#> git clone git://git.icinga.org/icinga-core.git
```

or from the [Icinga Website](#).

Don't forget to download the [Nagios Plugins](#).

Compile and install Icinga

Extract the Icinga source code tarball (or change directory to the GIT snapshot)

```
#> cd ~/src/
#> tar xvzf icinga-1.4.tar.gz
#> cd icinga-1.4
```

Run the Icinga configure script. You will get help by using the `--help` flag.

```
#> ./configure --with-httd-conf=/usr/local/etc/apache22/Includes/ \
--with-gd-lib=/usr/local/lib/ \
--with-gd-inc=/usr/local/include/ \
--with-command-group=icinga-cmd
```

Compile the Icinga source code. To see available options, only use "make".

```
#> make all
```

Install binaries, init script, sample config files and set permissions on the external command directory.

```
#> make install
#> make install-init
#> make install-config
#> make install-commandmode
```

or shorter

```
#> make fullinstall
```

The Icinga-API will be installed during "make install" so if you are required to install it manually please try:

```
#> make install-api
```

This will be mandatory for Icinga Web.

Don't start Icinga yet - there's still more that needs to be done...

Customise Configuration

Sample configuration files have been installed using

```
#> make install-config
```

into /usr/local/icinga/etc/. You'll need to make just one change before you proceed...

Edit the */usr/local/icinga/etc/objects/contacts.cfg* config file with your favourite editor and change the email address associated with the *icingaadmin* contact definition to the address you'd like to use for receiving alerts.

```
#> vi /usr/local/icinga/etc/objects/contacts.cfg
```

Install and configure the Classic Web Interface

Icinga ships with the Classic Web Interface ("the CGIs") which can be installed via

```
#> make cgis
#> make install-cgis
#> make install-html
```

If you are interested in the new Icinga Web, please refer to [Install Icinga Web Interface](#).

Install the Icinga Classic web config file in the Apache conf.d directory.



Note

There is currently a bug in Icinga Makefile which directly prevents this *make* command, please edit Makefile file in Icinga source directory and change the line

```
$(INSTALL) -D -m 644 sample-config/httpd.conf $(DESTDIR)$(HTTPD_CONF)/icinga.conf
```

to

```
$(INSTALL) -m 644 sample-config/httpd.conf $(DESTDIR)$(HTTPD_CONF)/icinga.conf
```

```
#> make install-webconf
```

Create an *icingaadmin* account for logging into the Icinga classic web interface. If you want to change it later, use the same command. Remember the password you assign to this account - you'll need it later.

```
#> htpasswd -c /usr/local/icinga/etc/htpasswd.users icingaadmin
```

If you want to change it later or want to add another user, use the following command:

```
#> htpasswd /usr/local/icinga/etc/htpasswd.users <USERNAME>
```

Reload/Restart Apache to make the new settings take effect.

```
#> /usr/local/etc/rc.d/apache22 reload
```

Compile and Install the Nagios Plugins

Extract the Nagios plugins source code tarball.

```
#> cd ~/src
#> tar xvzf nagios-plugins-1.4.15.tar.gz
#> cd nagios-plugins-1.4.15
```

Compile and install the plugins by changing install directory to /usr/local/icinga

```
#> ./configure --prefix=/usr/local/icinga \
--with-cgiurl=/icinga/cgi-bin --with-htmurl=/icinga \
--with-nagios-user=icinga --with-nagios-group=icinga
#> make
#> make install
```

Start Icinga

Add Icinga to the list of system services and have it automatically start when the system boots (make sure you have installed the init script before).

```
#> echo icinga_enable=\"YES\" >> /etc/rc.conf
```

Verify the sample Icinga configuration files.

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
```

If there are no errors, start Icinga.

```
#> /usr/local/etc/rc.d/icinga start
```

Login to the Classic Web Interface

You should now be able to access the Icinga classic web interface at the URL below. You'll be prompted for the username (*icingaadmin*) and password you specified earlier.

<http://localhost/icinga/>

or

<http://yourdomain.com/icinga/>

Click on the "Service Detail" navbar link to see details of what's being monitored on your local machine. It will take a few minutes for Icinga to check all the services associated with your machine.

Other Modifications

Make sure your system's firewall rules are configured to allow access to the web server if you want to access the Icinga classic interface remotely.

```
#> TCP port 80
```

Setting up your mail transfer agent (MTA) like exim, sendmail or postfix to allow Icinga sending notification emails won't be explained here.

Please refer to the [Nagios-Wiki](#) for more resources.

You're Done

Congratulations! You successfully installed Icinga. Your journey into monitoring has just begun.

You'll no doubt want to monitor more than just your local machine, so check out the chapter on "[Getting Started](#)" about "Monitoring ..."

[Prev](#)

[Up](#)

[Next](#)

[Icinga Quickstart](#)

[Home](#)

[Icinga with IDOUtils Quickstart](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga with IDOUtils Quickstart

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Icinga with IDOUtils Quickstart

Introduction

This guide is intended to provide you with simple instructions on how to install Icinga from source (code) and have it monitoring your local machine within 30 minutes.

No advanced installation options are discussed here - just the basics that will work for most of the users who want to get started.

This guide will give you examples for currently three different Linux distributions: [Fedora](#), [Ubuntu](#) and [openSuSE](#). Similar distributions may work as well. That should include [RedHat](#), [CentOS](#), [Debian](#) and [SLES](#).

Other distributions may inherit from these examples.

If you are planning to use Icinga without IDOUtils please read "[Icinga Quickstart](#)" instead!

What You'll End Up With

If you follow these instructions, here's what you'll end up with:

- Icinga and the plugins will be installed underneath /usr/local/icinga
- Icinga will be configured to monitor a few aspects of your local system (CPU load, disk usage, etc.)
- The Icinga classic web interface will be accessible at <http://localhost/icinga/> or <http://yourdomain.com/icinga>
- A database which is filled by Icinga using IDOUtils

Prerequisites

During portions of the installation you'll need to have **root** access to your machine.

Make sure you've installed the following packages on your system before continuing. IDOUtils use the [libdbi](#) and the libdbi-drivers for several databases. The development libraries are also required. The following examples will show how to install the IDOUtils with the libdbi using MySQL or PostgreSQL.

- Apache
- GCC compiler
- C/C++ development libraries
- [GD](#) development libraries
- libdbi/libdbi-drivers, database like MySQL or PostgreSQL

Optional

At one time or another you may need to use SNMP-based checks so it is a good idea to install the required packages now. Otherwise some plugins are not compiled i.e. not available when you need them and it would require a recompile of the plugins.

New features for the IDOUtils:

SSL-Encryption between idomod and ido2db

If you want to use SSL-encryption you'll need openssl and openssl-devel/libssl-dev (Ubuntu) to be installed !

Oracle Database Support

If you want Oracle as RDBM you'll need to install ocilib instead of libdbi.

- Oracle libraries and SDK (e.g. Instant Client)

If you install from package, make sure the libraries are within PATH. Otherwise you need to set the Oracle libs during Icinga IDOUtils install using e.g.
`--with-oracle-lib=/path/to/instantclient`

- ocilib instead of libdbi

Get it from <http://ocilib.sourceforge.net/> and point configure to your Oracle libraries and header files e.g. from the Oracle instant client:

```
#> ./configure --with-oracle-headers-path=/path/to/instantclient/sdk/include \
  --with-oracle-lib-path=/path/to/instantclient/
#> make
#> make install
```

Icinga 1.4

As of Icinga 1.4 the minimum expected Oracle version is Oracle 10gR2. Older versions may work, but are not supported. Oracle scripts are designed to split data, index and lobs into different tablespaces. For this reason there is a new prerequisite to define the tablespace names you want to use. If you are working in a small environment, you can set all defines to the same real tablespace. You will find a new script `icinga_defines.sql` which you have to adapt for your needs before applying `oracle.sql`. For your convenience there is a new script `create_oracle_sys.sql` included, which should help you to create the necessary tablespaces and an Icinga application user and must run as SYS. It will make use of `icinga_defines.sql` as well. Object creation has been moved from `oracle.sql` to the script `create_icinga_objects_oracle.sql`. The former `oracle.sql` has been redesigned as a master script to record all other scripts as includes and expects these includes within the current directory. For this reason you should start `sqlplus` executing `oracle.sql` within this directory. This way the creation of user and tablespaces and the creation of Icinga

tables runs in one step. As an all-in-one sample you will find a new script db/scripts/create_oracledb.sh. Edit variables to suit your needs and enjoy. If you prefer to do SYS steps for yourself, please uncomment create_oracle_sys.sql and make sure that your database Icinga user and tables exist and are defined with the same (or more) rights and that the correct settings have been applied to icinga_defines.sql.

Install Packages

You can install your packages by running the following commands (as root or sudo):

Fedora/RHEL/CentOS:

```
#> yum install httpd gcc glibc glibc-common gd gd-devel
#> yum install libjpeg libjpeg-devel libpng libpng-devel
```

MySQL:

```
#> yum install mysql mysql-server libdbi libdbi-devel libdbi-drivers libdbi-dbd-mysql
```

PostgreSQL:

```
#> yum install postgresql postgresql-server libdbi libdbi-devel libdbi-drivers libdbi-dbd-pgsql
```

Debian/Ubuntu:

```
#> apt-get install apache2 build-essential libgd2-xpm-dev
#> apt-get install libjpeg62 libjpeg62-dev libpng12 libpng12-dev
```



Note

The numbers <62/12> might differ, depending on your distribution



Note

Starting with Ubuntu 10.10 the package is called libpng12-0, the name of the dev-package remains the same.

MySQL:

```
#> apt-get install mysql-server mysql-client libdbi0 libdbi0-dev libdbd-mysql
```

PostgreSQL:

```
#> apt-get install postgresql libdbi0 libdbi0-dev libdbd-pgsql
```

openSuSE:

Please use YaST to install at least the packages gd, gd-devel, libjpeg, libjpeg-devel, libpng, libpng-devel and, optionally, net-snmp, net-snmp-devel and perl-Net-SNMP.



Note

The devel packages might be place on the SDK DVDs.

MySQL:

Use YaST to install the packages for the RDBMS you want to use, i.e. "mysql", "mysql-devel", "mysql-client" and the libdbi packages "libdbi", "libdbi-devel", "libdbi-drivers" and "libdbi-dbd-mysql".



Note

In OpenSuSE 11 (SLES 11) the name of the package was changed from "mysql-devel" to "libmysqlclient-devel".

PostgreSQL:

Use YaST to install the packages for the RDBMS you want to use, i.e. "postgresql", "postgresql-devel", "postgresql-server" and the libdbi packages "libdbi", "libdbi-devel" and "libdbi-drivers".

Using old OpenSuSE (SLES) versions including version 10 it is most likely that there aren't any libdbi packages so you have to download and compile the sources. Replace <rdbm> with your desired RDBM like mysql or pgsql. Remember that the Oracle driver is not yet working and read the appropriate section with ocilib instead of libdbi.

1. Download and extract the tar.gz files

<http://libdbi.sourceforge.net/download.html>

<http://libdbi-drivers.sourceforge.net/download.html>

```
#> tar xvzf libdbi-0.8.3.tar.gz
#> tar xvzf libdbi-drivers-0.8.3-1.tar.gz
```

2. Install libdbi. Maybe you have to specify additional options with configure (set --prefix=/usr ...)

```
#> cd libdbi-0.8.3
#> ./configure --disable-docs
#> make
#> make install
```

3. Install libdbi-drivers

```
#> cd libdbi-drivers-0.8.3-1
#> ./configure --with-<rdbm> --disable-docs
#> make
#> make install
```



Note

Using the 64-bit-versions you have to specify the paths to the include- and lib-dir explicitly:

```
#> ./configure --with-<rdbm> \
--with-<rdbm>-incdir=/usr/include/<rdbm>/ \
--with-<rdbm>-libdir=/usr/lib64/ --disable-docs
```

Create Account Information

Become the root user.

```
$> su -l
```

Create a new *Icinga* user account and give it a password.

```
#> /usr/sbin/useradd -m icinga
#> passwd icinga
```

On some distributions you'll need to add the group in a single step:

```
#> /usr/sbin/groupadd icinga
```

For sending commands from the classic web interface to Icinga, you'll need to create a new group icinga-cmd. Add the webuser and the Icingauser to this group:

```
#> /usr/sbin/groupadd icinga-cmd
#> /usr/sbin/usermod -a -G icinga-cmd icinga
#> /usr/sbin/usermod -a -G icinga-cmd www-data
```

(or www, wwwrun, apache depending on the distribution)



Note

Some usermod-versions (e.g in OpenSuSE 11 and SLES 11, resp.) are lacking the option -a. In this case please omit the option -a.



Important

Solaris only supports groupnames with max. 8 characters, please use icingcmd instead of icinga-cmd.

Download Icinga and the Plugins

Change to your local source directory i.e. /usr/src

```
#> cd /usr/src
```

Either fetch the actual icinga-core snapshot from Icinga GIT

```
#> git clone git://git.icinga.org/icinga-core.git
#> cd icinga-core
#> git submodule init
#> git submodule update
```

or from the [Icinga Website](#). Don't forget to download the [Nagios Plugins](#)

Compile and Install Icinga with IDOUtils

Extract the Icinga source code tarball

```
#> cd /usr/src/
#> tar xvzf icinga-1.4.tar.gz
#> cd icinga-1.4
```



Note

This absolute path is meant when we refer to '/path/to/icinga-src/' in the following text.

Run the Icinga configure script and enable IDOUtils. You will get help by using the --help flag:

```
#> ./configure --with-command-group=icinga-cmd --enable-idoutils
```



Important

Compiling on Solaris might fail upon unresolved library dependencies on gethostbyname. In that case run this before running configure:

```
#> export LIBS=-lsocket -lnsl
```

With SSL-Encryption:

```
#> ./configure --with-command-group=icinga-cmd --enable-idoutils --enable-ssl
```

With Oracle Database Support:

```
#> ./configure --with-command-group=icinga-cmd --enable-idoutils --enable-oracle
```

If you didn't install Oracle libraries to PATH, you can point configure there:

```
#> ./configure --with-command-group=icinga-cmd \
--enable-idoutils --enable-oracle \
--with-oracle-lib=/path/to/instantclient
```

If you didn't install ocilib to the default path (/usr/local) you can point configure to the lib/inc directories:

```
#> ./configure --with-command-group=icinga-cmd \
--enable-idoutils --enable-oracle \
--with-ocilib-lib=/path/to/ocilib/lib \
--with-ocilib-inc=/path/to/ocilib/include
```



Note

If you want to change RDBM from Oracle to others, you need to recompile and reinstall IDOUtils!

```
#> make distclean
#> ./configure --enable-idoutils
```

Compile and Install

Compile the Icinga source code. There is also an extra option for IDOUtils (*make idoutils*) if you need to recompile only this module. To see available options, only use "make".

```
#> make all
```

Install binaries, init script, sample config files and set permissions on the external command directory. Also install the IDOUtils.

```
#> make install
#> make install-init
#> make install-config
#> make install-commandmode
#> make install-idoutils
```

or shorter

```
#> make fullinstall
```



Note

Install IDOUtils and applicable event broker modules only using the primary make install target. Do not manually copy and overwrite the existing module as this will result in a segfault on icinga core which is using idomod.o directly preventing usage of a temporary copy explicitly. This is useful for [OMD](#).

The Icinga-API will be installed during "make install" so if you are required to install it manually please try:

```
#> make install-api
```

This will be mandatory for Icinga-Web.

Don't start Icinga yet - there's still more that needs to be done...

Customise Configuration

Sample configuration files have been installed by using "make install-config" into `/usr/local/icinga/etc/`. You'll need to make just one change before you proceed...

Edit the `/usr/local/icinga/etc/objects/contacts.cfg` config file with your favourite editor and change the email address associated with the `icingaadmin` contact definition to the address you'd like to use for receiving alerts.

```
#> vi /usr/local/icinga/etc/objects/contacts.cfg
#> cd /usr/local/icinga/etc/
#> mv idomod.cfg-sample idomod.cfg
#> mv ido2db.cfg-sample ido2db.cfg
```

If you want to enable SSL-encryption and you configured the IDOUtils with `./configure --enable-ssl`, you have to change `idomod.cfg` as follows:

```
use_ssl=1
output_type=tcpsocket
output=127.0.0.1
```

(don't forget to adjust the ip-address if your database isn't located on localhost) and `ido2db.cfg` in the following way:

```
use_ssl=1
socket_type=tcp
```



Note

If SSL is enabled in `ido2db` but not in the different `idomod` clients - data from those instances will be lost - that's guaranteed! SSL configuration has to be the same on all nodes!!!

Enable the `idomod` event broker module

Edit the main config file, search for `broker_module` and uncomment the example or edit it matching your filenames in:

```
#> vi /usr/local/icinga/etc/icinga.cfg
broker_module=/usr/local/icinga/bin/idomod.o config_file=/usr/local/icinga/etc/idomod.cfg
```



Note

Starting with Icinga 1.4 you can use the new module definition in one of your object configuration files instead of specifying a broker_module entry:

```
define module{
    module_name      ido_mod
    path            /usr/local/icinga/bin/idomod.o
    module_type     neb
    args           config_file=/usr/local/icinga/etc/idomod.cfg
}
```

Creation of Database and IDOUtils



Note

If you just installed a new database system then you have to start the database server before you can create a database. In case of MySQL you might use `/etc/init.d/mysqld start`.

MySQL:

Create Database, User, Grants:

```
#> mysql -u root -p
mysql> CREATE DATABASE icinga;
GRANT USAGE ON *.* TO 'icinga'@'localhost'
    IDENTIFIED BY 'icinga'
    WITH MAX_QUERIES_PER_HOUR 0
    MAX_CONNECTIONS_PER_HOUR 0
    MAX_UPDATES_PER_HOUR 0;
GRANT SELECT , INSERT , UPDATE , DELETE
    ON icinga.* TO 'icinga'@'localhost';
FLUSH PRIVILEGES ;
quit
```

Import database scheme for MySQL:

```
#> cd /path/to/icinga-src/module/idoutils/db/mysql
#> mysql -u root -p icinga < mysql.sql
```

Edit the DB config file to customize IDOUtils

```
#> vi /usr/local/icinga/etc/ido2db.cfg
db_servertype=mysql
db_port=3306
db_user=icinga
db_pass=icinga
```

PostgreSQL:

Create database and User:

```
#> su - postgres
$ createlang plpgsql icinga;
$ psql
postgres=# CREATE USER icinga;
postgres=# ALTER USER icinga WITH PASSWORD 'icinga';
postgres=# CREATE DATABASE icinga;
```

Debian:

```
#> vi /etc/postgresql/8.x/main/pg_hba.conf
```

Fedora/RHEL/CentOS:

```
#> vi /var/lib/pgsql/data/pg_hba.conf
```

Edit the config e.g. like this (local user must be trusted)

```
# database administrative login by UNIX sockets
local    all      postgres          ident
# TYPE   DATABASE  USER      CIDR-ADDRESS  METHOD
#icinga
local    icinga    icinga           trust
# "local" is for Unix domain socket connections only
local    all      all              trust
# IPv4 local connections
host     all      all      127.0.0.1/32  trust
# IPV6 local connections
host     all      all      ::1/128       trust
```

Reload and configure database scheme.

```
#> /etc/init.d/postgresql-8.x reload
#> cd /path/to/icinga-src/module/idouutils/db/pgsql
#> psql -U icinga -d icinga < pgsql.sql
```

Edit the DB config file to customize IDOUtils

```
#> vi /usr/local/icinga/etc/ido2db.cfg
db_servertype=pgsql
db_port=5432
db_user=icinga
db_pass=icinga
```

Oracle:

Create a database schema and username/password (refer to the Oracle documentation at <http://www.oracle.com> or consult your DBA). Import the database scheme with sqlplus (or your preferred method). Copy module/idouutils/db/oracle/oracle.sql to \$ORACLE_HOME

```
#> su - oracle
$> sqlplus dbuser/dbpass
SQL> @oracle.sql
```

Edit the DB config file to customize IDOUtils. Remember that Oracle ignores the db host, instead point db_name to //DBSERVER/DBNAME

```
#> vi /usr/local/icinga/etc/ido2db.cfg
db_servertype=oracle
db_port=1521
db_user=icinga
db_pass=icinga
```

Configure the Classic Web Interface

Icinga ships with the classic web interface ("the CGIs") which can be installed via

```
#> make cgis
#> make install-cgis
#> make install-html
```

If you are interested in the new Icinga Web, please refer to [Install Icinga Web Interface](#).

Install the Icinga classic web config file in the Apache conf.d directory.

```
#> make install-webconf
```

Create an *icingaadmin* account for logging into the Icinga classic web interface. Remember the password you assign to this account - you'll need it later.

```
#> htpasswd -c /usr/local/icinga/etc/htpasswd.users icingaadmin
```

To change the password of an existing user or to add a new user, take this command:

```
#> htpasswd /usr/local/icinga/etc/htpasswd.users <USERNAME>
```



Note

Depending on your distribution/Apache-version you may have to use *htpasswd2* instead.

Reload/Restart Apache to make the new settings take effect.

Fedora/RHEL/CentOS:

```
#> service httpd restart
```

Ubuntu/openSuSE:

```
#> service apache2 restart
```

Debian:

```
#> /etc/init.d/apache2 reload
```

Compile and Install the Nagios Plugins

Extract the Nagios plugins source code tarball.

```
#> cd /usr/src
#> tar xvzf nagios-plugins-1.4.15.tar.gz
#> cd nagios-plugins-1.4.15
```

Compile and install the plugins by changing install directory to */usr/local/icinga*

```
#> ./configure --prefix=/usr/local/icinga \
--with-cgiurl=icinga/cgi-bin --with-htmurl=/icinga \
--with-nagios-user=icinga --with-nagios-group=icinga
#> make
#> make install
```

Adjusting the SELinux settings

RHEL and derived distributions like Fedora and CentOS are shipped with activated SELinux (Security Enhanced Linux) running in "enforcing" mode. This may lead to "Internal Server Error" messages when you try to invoke the Icinga-CGI.

Check if SELinux runs in enforcing mode

```
#> getenforce
```

Set SELinux in "permissive" mode

```
#> setenforce 0
```

To make this change permanent you have to adjust this setting in `/etc/selinux/config` and restart the system.

Instead of deactivating SELinux or setting it into permissive mode you can use the following commands to run the CGIs in enforcing/targeted mode:

```
#> chcon -R -t httpd_sys_script_exec_t /usr/local/icinga/sbin/
#> chcon -R -t httpd_sys_content_t /usr/local/icinga/share/
#> chcon -t httpd_sys_script_rw_t /usr/local/icinga/var/rw/icinga.cmd
```

Start IDOUtils and Icinga

IDOUtils must be started and running *before* Icinga is started.

Start IDOUtils:

Fedora/RHEL/CentOS/Ubuntu/openSuSE:

```
#> service ido2db start
```

Debian:

```
#> /etc/init.d/ido2db start
```

Stop IDOUtils:

Fedora/RHEL/CentOSUbuntu/openSuSE:

```
#> service ido2db stop
```

Debian:

```
#> /etc/init.d/ido2db stop
```

Verify the sample Icinga configuration files.

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
```

If there are no errors, start Icinga.

Fedora/RHEL/CentOS/Ubuntu/openSuSE:

```
#> service icinga start
```

Debian:

```
#> /etc/init.d/icinga start
```

Configure Icinga Startup

Add Icinga to the list of system services and have it automatically start when the system boots (make sure you have installed the init script before).

Fedora/RHEL/CentOS/openSuSE:

```
#> chkconfig --add icinga chkconfig icinga on
```

Debian/Ubuntu:

```
#> update-rc.d icinga defaults
```

Login to the Classic Web Interface

You should now be able to access the Icinga classic web interface at the URL below. You'll be prompted for the username (*icingaadmin*) and password you specified earlier.

```
http://localhost/icinga/
```

or

```
http://yourdomain.com/icinga/
```

Click on the "Service Detail" navbar link to see details of what's being monitored on your local machine. It will take a few minutes for Icinga to check all the services associated with your machine.

Other Modifications

Make sure your system's firewall rules are configured to allow access to the web server if you want to access the Icinga classic interface remotely.

```
#> iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

Setting up your mail transfer agent (MTA) like exim, sendmail or postfix to allow Icinga sending notification emails won't be explained here. Please refer to the [Nagios-Wiki](#) for more resources.

There are several tweaks for the IDOUtils (former NDOUtils of Nagios) available.

You're Done

Congratulations! You successfully installed Icinga with IDOUtils. Your journey into monitoring has just begun. You'll no doubt want to monitor more than just your local machine, so check out the chapter on "[Getting Started](#)" about "Monitoring ...".

[Prev](#)

[Up](#)

[Next](#)

[Icinga Quickstart FreeBSD](#)

[Home](#)

[Icinga and IDOUtils Quickstart on FreeBSD](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga and IDOUtils Quickstart on FreeBSD

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Icinga and IDOUtils Quickstart on FreeBSD

Introduction

This guide is intended to provide you with simple instructions on how to install Icinga from source (code) and have it monitoring your local machine within 30 minutes.

No advanced installation options are discussed here - just the basics that will work for most of the users who want to get started.

This guide will give you examples for installation on [FreeBSD](#) 8.1-RELEASE. Thanks to "[ScotchTape](#)" for the adapted instructions.

Later distributions of FreeBSD may inherit from these examples.



Note

Meanwhile there's a FreeBSD port of Icinga (`net-mgmt/icinga`) so maybe that's the one you want to install ;-).

What You'll End Up With

If you follow these instructions, here's what you'll end up with:

- Icinga and the plugins will be installed underneath `/usr/local/icinga`
- Icinga will be configured to monitor a few aspects of your local system (CPU load, disk usage, etc.)
- The Icinga classic web interface will be accessible at `http://localhost/icinga/` or `http://yourdomain.com/icinga`
- a database being filled using the IDOUtils

Prerequisites

During portions of the installation you'll need to have **root** access to your machine.

Make sure you've installed the following packages on your system before continuing.

- [Apache](#)
- GCC compiler
- C/C++ development libraries
- [GD](#) development libraries
- libdbi-drivers, a database like MySQL, or PostgreSQL

Optional

At one time or another you will likely use SNMP based checks so it might be a good idea to install the required packages now. Otherwise the plugins will not be compiled and are not available when you need them.

New features for the IDOUtils:

SSL-encryption between idomod and ido2db

If you want to use **SSL-encryption** : it is already installed.



Note

SSL has to be activated on all idomod clients, otherwise you will lose data!!!

Oracle database support

If you want to use Oracle as an RDBMS then we are sorry. Unfortunately it isn't that easy with FreeBSD.

Install the packages

You can install the packages from the ports or you take older packages (have a look at the [FreeBSD quickstart quide](#)).

Install ports

You can install these ports by running the following commands (as root) but it is recommended to use portupgrade or portmaster instead:

Please update your ports before doing so.

```
#> cd /usr/ports/devel/libtool/ && make all install clean
#> cd /usr/ports/graphics/jpeg && make all install clean
#> cd /usr/ports/graphics/png && make all install clean
#> cd /usr/ports/graphics/gd && make all install clean
```



Note

Please make sure you have Apache installed - the process will not be discussed here, however the lead is # `cd /usr/ports/www/apache22 && make clean && make`.

```
#> cd /usr/ports/devel/gmake && make all install clean
#> cd /usr/ports/devel/libltdl && make all install clean <-- unless installed already
```

MySQL

```
#> cd /usr/ports/databases/mysql51-server && make all install clean
#> cd /usr/ports/databases/libdbi-drivers && make all install clean
```

please choose the correct driver for the database used

PostgreSQL

```
#> cd /usr/ports/databases/postgresql84-server && make all install clean
#> cd /usr/ports/databases/libdbi-drivers && make all install clean
```

please choose the correct driver for the database used

Create Account Information

Become the root user.

```
$> su -l
```

Create a new *icinga* user account without a password and without the ability to log-in (set no password when asked):

```
#> adduser -D -w no -s nologin
```

For sending commands from the classic web interface to Icinga, you'll need to create a new group *icinga-cmd* and add the *webuser* (*www*) and the *Icingauser* to this group:

```
#> pw groupadd -n icinga-cmd -M icinga,www
```

Download Icinga and the plugins

Change to your local source directory i.e. `~/src`

```
#> mkdir ~/src
#> cd ~/src
```

Either fetch the current *icinga-core* snapshot from Icinga GIT

```
#> git clone git://git.icinga.org/icinga-core.git
```

or from the [Icinga Website](#).

Don't forget to download the [Nagios Plugins](#).

Compile and install Icinga and IDOUtils

Extract the Icinga source code tarball (or change directory to the GIT snapshot)

```
#> cd ~/src/
#> tar xvzf icinga-1.4.tar.gz
#> cd icinga-1.4
```



Note

This absolute path is meant if we use the description `'/path/to/icinga-src/'`.

Run the Icinga configure script. You will get help by using the `--help` flag.

```
#> ./configure --with-command-group=icinga-cmd \
--enable-idoutils CPPFLAGS=-I/usr/local/include \
CFLAGS="-I/usr/local/include -L/usr/local/lib" \
--with-dbi-lib=/usr/local/lib --with-dbi-inc=/usr/local/include
```



Important

Appending `CPPFLAGS=-I/usr/local/include` is important for the IDOUtils and the broker modul respectively.



Note

You should include `CFLAGS=...` like specified above. Otherwise you might get the following lines in `icinga.log`:

```
Error: Module '/usr/local/icinga/bin/idomod.o' is using an old or unspecified version of the event broker API. Module will be unloaded.  
Event broker module '/usr/local/icinga/bin/idomod.o' deinitialized successfully.
```

More details on this error can be found [here](#).

With SSL-encryption:

```
#> ./configure --with-command-group=icinga-cmd \
--enable-idoutils --enable-ssl CPPFLAGS=-I/usr/local/include \
--with-dbi-lib=/usr/local/lib --with-dbi-inc=/usr/local/include
```

Compile the Icinga source code. There is a separate option for IDOUtils (*make idoutils*) if you just want to recompile this module. To see possible option just call "make".

```
#> gmake all
```

Install binaries, init script, sample config files and set permissions on the external command directory.

```
#> gmake install
#> gmake install-init
#> gmake install-config
#> gmake install-commandmode
#> gmake install-idoutils
```

or shorter

```
#> make fullinstall
```

The Icinga-API will be installed during "make install" so if you are required to install it manually please try:

```
#> make install-api
```

This will be mandatory for Icinga Web.

Don't start Icinga yet - there's still more that needs to be done...

Customise Configuration

Sample configuration files have been installed using

```
#> gmake install-config
```

into `/usr/local/icinga/etc/`. You'll need to make just one change before you proceed...

Edit the `/usr/local/icinga/etc/objects/contacts.cfg` config file with your favourite editor and change the email address associated with the `icingaadmin` contact definition to the address you'd like to use for receiving alerts.

```
#> vi /usr/local/icinga/etc/objects/contacts.cfg
#> cd /usr/local/icinga/etc
#> mv idomod.cfg-sample idomod.cfg
#> mv ido2db.cfg-sample ido2db.cfg
```

If you compiled the IDOUtils with SSL then you have to activate SSL in `idomod.cfg` setting

```
use_ssl=1
output_type=tcpsocket
output=127.0.0.1
```

(adjust the IP address if your database is not running on localhost!) and in `ido2db.cfg` setting

```
use_ssl=1
socket_type=tcp
```



Note

Don't forget to recompile all other idomod clients and to activate SSL **otherwise you will lose data!!!**

Activate the idomod event broker module

Edit `/usr/local/icinga/etc/icinga.cfg`, search for "broker_module" and uncomment or insert the following line (adjust the name if necessary).

```
broker_module=/usr/local/icinga/bin/idomod.o config_file=/usr/local/icinga/etc/idomod.cfg
```



Note

Starting with Icinga 1.4 you can use the new module definition in one of your object configuration files instead of specifying a broker_module entry:

```
define module{
    module_name    ido_mod
    path          /usr/local/icinga/bin/idomod.o
    module_type   neb
    args          config_file=/usr/local/icinga/etc/idomod.cfg
}
```

Compile database and IDOUtils

MySQL:

Create database, user, and permissions



Note

If you just installed the RDBMS like described above then you have to start the database server before you can create a database. Having MySQL please use `/usr/local/etc/rc.d/mysql-server start` to start.

```
#> mysql -u root -p

mysql> CREATE DATABASE icinga;

GRANT USAGE ON *.* TO 'icinga'@'localhost' IDENTIFIED BY 'icinga'
  WITH MAX_QUERIES_PER_HOUR 0
  MAX_CONNECTIONS_PER_HOUR 0
  MAX_UPDATES_PER_HOUR 0;

GRANT SELECT , INSERT , UPDATE , DELETE
  ON icinga.* TO 'icinga'@'localhost';

FLUSH PRIVILEGES ;

quit

#> cd /path/to/icinga-src/module/idoutils/db/mysql
#> mysql -u root -p icinga < mysql.sql

#> vi /usr/local/icinga/etc/ido2db.cfg

db_servertype=mysql
db_port=3306
db_user=icinga
db_pass=icinga
```

PostgreSQL:*To Do***Install and configure the Classic Web Interface**

Icinga ships with the Classic Web Interface ("the CGIs") which can be installed via

```
#> cd /path/to/icinga-src
#> make cgis
#> make install-cgis
#> make install-html
```

If you are interested in the new Icinga Web, please refer to [Install Icinga Web Interface](#).

Install the Icinga Classic web config file in the Apache conf.d directory.

**Note**

There is currently a bug in Icinga Makefile which directly prevents this *make* command, please edit Makefile file in Icinga source directory and change the line

```
$(INSTALL) -D -m 644 sample-config/httpd.conf $(DESTDIR)$(HTTPD_CONF)/icinga.conf
to
$(INSTALL) -m 644 sample-config/httpd.conf $(DESTDIR)$(HTTPD_CONF)/icinga.conf
```

```
#> make install-webconf
```

Create an *icingaadmin* account for logging into the Icinga classic web interface. If you want to change it later, use the same command. Remember the password you assign to this account - you'll need it later.

```
#> htpasswd -c /usr/local/icinga/etc/htpasswd.users icingaadmin
```

If you want to change it later or want to add another user, use the following command:

```
#> htpasswd /usr/local/icinga/etc/htpasswd.users <USERNAME>
```

Reload/Restart Apache to make the new settings take effect.

```
#> /usr/local/etc/rc.d/apache22 reload
```

Compile and Install the Nagios-/Perl Plugins

Extract the Nagios plugins source code tarball.

```
#> cd ~/src
#> tar xvzf nagios-plugins-1.4.15.tar.gz
#> cd nagios-plugins-1.4.15
```

Compile and install the plugins by changing install directory to /usr/local/icinga

```
#> ./configure --prefix=/usr/local/icinga --with-cgiurl=/icinga/cgi-bin \
--with-htmurl=/icinga --with-nagios-user=icinga --with-nagios-group=icinga
#> make
#> make install
```



Note

There is a port for the plugins but it will install them in a different directory. You can set certain variables to tweak its make but you still have to copy some things manually later on.

Compile and install the Perl plugin:

```
#> cd /usr/ports/net-mgmt/p5-Nagios-Plugin
#> make all install clean
```

Starting IDOUtils and Icinga

IDOUtils has to be started before Icinga

Starting IDOUtils

```
#> /usr/local/etc/rc.d/ido2db start
```

Stopping IDOUtils

```
#> /usr/local/etc/rc.d/ido2db stop
```

Start Icinga

Add Icinga to the list of system services and have it automatically start when the system boots (make sure you have installed the init script before).

```
#> echo icinga_enable=\"YES\" >> /etc/rc.conf
```

Verify the sample Icinga configuration files.

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
```

If there are no errors, start Icinga.

```
#> /usr/local/etc/rc.d/icinga start
```

Login to the Classic Web Interface

You should now be able to access the Icinga classic web interface at the URL below. You'll be prompted for the username (*icingaadmin*) and password you specified earlier.

```
http://localhost/icinga/
```

or

```
http://yourdomain.com/icinga/
```

Click on the "Service Detail" navbar link to see details of what's being monitored on your local machine. It will take a few minutes for Icinga to check all the services associated with your machine.

Other Modifications

Make sure your system's firewall rules are configured to allow access to the web server if you want to access the Icinga classic interface remotely.

```
#> TCP port 80
```

Setting up your mail transfer agent (MTA) like exim, sendmail or postfix to allow Icinga sending notification emails won't be explained here.

Please refer to the [Nagios-Wiki](#) for more resources.

You're Done

Congratulations! You successfully installed Icinga. Your journey into monitoring has just begun.

You'll no doubt want to monitor more than just your local machine, so check out the chapter on "[Getting Started](#)" about "Monitoring ..."

Packages for Icinga

Compiler options for Icinga with IDOUtils

```
./configure --with-httd-conf=/usr/local/etc/apache22/Includes/ \
--with-gd-lib=/usr/local/lib/ --with-gd-inc=/usr/local/include/ \
--with-command-group=icinga-cmd --enable-idoutils \
--with-dbi-inc=/usr/local/include --with-dbu-lib=/usr/local/lib \
CPPFLAGS=-I/usr/local/include CFLAGS=-fPIC
```

Compiler options for Nagios plugins (ports)

```
make install NAGIOSUSER=icinga NAGIOSGROUP=icinga \
PREFIX=/usr/local/icinga
```

[Prev](#)

[Up](#)

[Next](#)

Icinga with IDOUtils Quickstart

[Home](#)

[Links to other published Howtos](#)



Links to other published Howtos

[Prev](#)

[Chapter 2. Getting Started](#)

[Next](#)

Links to other published Howtos

Other published Howtos

[Icinga-Reporting installation guide \(with JasperServer\)](#) (Icinga Development Team)

[Setting up DNX with Icinga](#) (Icinga Development Team)

[Setting up mod gearman with Icinga](#) (Icinga Development Team)

[Developing modules for Icinga-Web](#) (Icinga Development Team)

[Developing cronks for Icinga-Web](#) (Icinga Development Team)

[Setting up AD authentication for Icinga-Web](#) (Sebastian Waitz)

[Icinga and Oracle](#) (Icinga Development Team)

[Step By Step install using RPMS for fc13](#) (Icinga Development Team)

[Installation and configuration Icinga on CentOS](#) (HowtoForge)

Thank you all!

[Prev](#)

[Up](#)

[Next](#)

[Icinga and IDOUtils Quickstart on FreeBSD](#)

[Home](#)

[Upgrading \(to\) Icinga](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Upgrading (to) Icinga

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Upgrading (to) Icinga

Contents

[Upgrading from previous Icinga releases](#)

[Upgrading from Nagios 3.x releases](#)

[Upgrading from Nagios 2.x releases](#)

[Upgrading from an RPM installation](#)



Note

If you are using IDOUtils then you have to update it, too. Please take a look at [upgrading_idoutils](#) as well.



Note

If you are using Icinga-Web then you have to update it, too. Please take a look at [upgrading_icongaweb](#) as well.

Upgrading from previous Icinga Releases

During development there may be cases where the handling of bugfixes might touch the userspace. We try to avoid this as far as possible but sometimes it is inevitable.

Normal Icinga update Path

If there's a new Icinga version available on <http://www.icinga.org/> you should consider updating. Newer versions contain bugfixes, minor or major, which makes it even more important to get the latest and the greatest. If you already installed Icinga using the quickstart guides it's rather easy to install newer versions. It's also possible to perform that without root privileges by doing an install from source. It highly depends on your likings.

Make sure you'll have a safe backup of your Icinga installation and configs. If anything goes wrong you can restore your old setup rather easy.

Regular updates of Icinga are possible just by re-installing Icinga. Please keep in mind that `configure` requires the same parameters as before - look it up in `config.log` and store that in a safe location.

Become the icinga user. Debian/Ubuntu users might use sudo -s icinga.

```
#> su -l icinga
```

Get the latest Icinga version from <http://www.icinga.org/> and extract it. Then run configure like you did with the old version.

```
#> ./configure --with-command-group=icinga-cmd --enable-idoutils
```

Compile the source.

```
#> make all
```

Please make sure that you don't overwrite your configs - always make a backup! Don't use make install-config and/or make install-webconf if you want to manually diff the config files.

```
#> make install install-base install-cgis install-html install-init install-commandmode install-idoutils
```

or decide what you want to upgrade by just typing 'make' which shows all available options. Check your config and restart Icinga.

```
#> /etc/init.d/icinga checkconfig
#> /etc/init.d/icinga restart
```

That's it!

Icinga 0.8.0 to Icinga 1.4

Archived Logfilename

There was a small misspelling bug in Icinga 0.8.0 resulting in archived logfile names beginning with a capital "I" like in 'Icinga-\$date''. This was fixed in version 0.8.1. and newer.

If you experience this problem then please use the following script to fix the file names

```
#> cd /usr/local/icinga/var/archives
for oldfilename in `find . -name "Icinga-*"`
do
  newfilename=`echo $oldfilename | sed -e 's/Icinga/icinga/g'`
  mv $oldfilename $newfilename
done
```

Upgrading from Nagios 3.x Releases

We try to be compatible with the configuration files of the current Nagios 3.x versions so there should be very little you have to do to "upgrade" from Nagios 3.x to Icinga 1.4. Assuming you've already installed Nagios from source code as described in the Nagios quickstart guide, you can install Icinga quite easily.

If you are upgrading from Nagios 3.0.x you might need to install PHP:

Debian / Ubuntu

```
#> apt-get install php5 libapache2-mod-php5
```

Fedora / RedHat

```
#> yum install php mod_php
```

openSuSE / SLES

Use Yast to install the packages *php5* and *apache2-mod_php5* or use zypper

```
#> zypper install php5 apache2-mod_php5
```

Make sure you have a good backup of your existing Nagios installation and configuration files. If anything goes wrong or doesn't work, this will allow you to rollback to your old version.

Please install Icinga according to the [quickstart guide](#). Please note that

- the default prefix has changed to "/usr/local/icinga"
- the environment macros start with ICINGA_

PNP4Nagios is aware of that since 0.6rc1 (2009.09.20) but the macros are needed in "sync" mode only.

`check_multi` includes support since v0.21 (2010.06.03), but you have to rerun the installation using '`configure --with-nagios_name=icinga`' so that `check_multi` contains the correct values. You can use the option "-s" instead to set certain values.

- the configuration file names of the (enhanced) IDOUtils are named `idomod.cfg/ido2db.cfg` instead of `ndomod.cfg/ndo2db.cfg`

Become the icinga user. Debian/Ubuntu users might use `sudo -s icinga`.

```
$ su -l nagios
```

Get the latest Icinga version from <http://www.icinga.org/>

```
#> wget http://osdn.dl.sourceforge.net/sourceforge/icinga/icinga-1.4.tar.gz
```

Extract the tarball and change to the directory just created

```
#> tar xzf icinga-1.4.tar.gz
#> cd icinga-1.4
```

Start the Icinga-configure-script with the options you used the last time with Nagios. You'll find the call in the file config.log. Example:

```
#> ./configure --with-command-group=nagcmd
```

Compile the Icinga source code.

```
#> make all
```

Install the programs, documentation, classic web interface and the init script. Your existing config files will not be overwritten.

```
#> make cgis
#> make install
#> make install-cgis
#> make install-init
```

Instead of using "make install-config" copy your configuration files from <Nagios-path>/etc/ to <Icinga-path>/etc/. Before starting Icinga you have to alter some things:

- The filename of the main config file has changed from `nagios.cfg` to `icinga.cfg`. #> `mv nagios.cfg icinga.cfg` should be sufficient to change that
- You have to edit the main config file (`icinga.cfg`) to change the directives named "nagios_user" and "nagios_group" to "icinga_user" and "icinga_group" respectively. Depending on the paths you're using you may have to change them as well.

```
#> sed -i 's/nagios/icinga/g' ./icinga.cfg/
```

- You have to edit the CGI config file (`cgi.cfg`) to change the paths you're using.

```
#> sed -i 's/nagios/icinga/g' ./cgi.cfg/
```

Copy other relevant files from your Nagios installation to the new location. If you are unsure about the path then please have a look at `nagios.cfg` and/or `icinga.cfg`.

- `retention.dat` (it contains certain status information, comments and other "sticky" stuff)
- `nagios.log` (rename it to `icinga.log`)
- `archives/nagios-<date>.log` files (Icinga is able to process both `nagios-<date>.log` and `icinga-<date>.log` files so you don't have to rename the files)
- You don't have to copy `status.dat` and/or `objects.cache` because these files are recreated during startup. Please recreate `objects.precache` before startup [if applicable](#) instead of copying it

Verify your configuration files, correct errors (if any) and start Icinga.

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
#> /etc/init.d/icinga start
```

That's it - you're done!

Please keep in mind that

- the URL has changed to `http://localhost/icinga/` (for the classic UI)
- the name of the admin user has changed to `icingaadmin`

Upgrading from Nagios 2.x Releases

It shouldn't be too difficult to upgrade from Nagios 2.x to Icinga 1.4. The upgrade is essentially the same as what is described above for upgrading from Nagios 3.x releases. You will, however, have to change your configuration files a bit so they work with Icinga 1.4:

- The old `service_reaper_frequency` variable in the main config file has been renamed to [check_result_reaper_frequency](#).
- The old `$NOTIFICATIONNUMBER$` macro has been deprecated in favour of new `$HOSTNOTIFICATIONNUMBER$` and `$SERVICENOTIFICATIONNUMBER$` macros.
- The old `parallelize` directive in service definitions is now deprecated and no longer used, as all service checks are run in parallel.
- The old `aggregate_status_updates` option has been removed. All status file updates are now aggregated at a minimum interval of 1 second.

- Extended host and extended service definitions have been deprecated. They are still read and processed by Icinga, but it is recommended that you move the directives found in these definitions to your host and service definitions, respectively.
- The old *downtime_file* file variable in the main config file is no longer supported, as scheduled downtime entries are now saved in the [retention file](#). To preserve existing downtime entries, stop Nagios 2.x and append the contents of your old downtime file to the retention file.
- The old *comment_file* file variable in the main config file is no longer supported, as comments are now saved in the [retention file](#). To preserve existing comments, stop Nagios 2.x and append the contents of your old comment file to the retention file.

Also make sure to read the "[What's New](#)" section of the documentation. It describes all the changes that were made to the Icinga code since the latest stable release of Nagios 3.0.6. Quite a bit has changed, so make sure you read it over.

Upgrading From an RPM Installation

If you currently have an RPM- or Debian/Ubuntu APT package-based installation of Nagios and you would like to transition to installing Icinga from the official source code distribution, here's the basic process you should follow:

1. Backup your existing Nagios installation
 - Configuration files
 - Main config file (usually `nagios.cfg`)
 - Resource config file (usually `resource.cfg`)
 - CGI config file (usually `cgi.cfg`)
 - All your object definition files
 - Retention file (usually `retention.dat`)
 - Current Nagios log file (usually `nagios.log`)
 - Archived Nagios log files (usually `nagios-<date>.log`)
2. Uninstall the original RPM or APT package
3. Install Icinga from source by following the [quickstart guide](#)
4. Restore your original Nagios configuration files, retention file, and log files
5. [Verify](#) your configuration and [start](#) Icinga

Note that different RPMs or APT packages may install Icinga in different ways and in different locations. Make sure you've backed up all your critical Nagios files before removing the original RPM or APT package, so you can revert back if you encounter problems.

[Prev](#)

[Up](#)

[Next](#)

Links to other published Howtos

[Home](#)

Upgrading IDOUtils Database

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Upgrading IDOUtils Database

[Prev](#)

Chapter 2. Getting Started

[Next](#)

Upgrading IDOUtils Database

There may be a bug within the database scheme which has been fixed. If you are upgrading from an older IDOUtils version you also need to apply those fixes manually. If you are using rpm/deb packages please read the notes and/or ask the maintainer if he has added those modifications to the install routine.



Note

Depending on the changes to be done and the size of your database it may take a while to update your database. Please try to be patient and don't abort the script as it may leave the data being corrupt.

The upgrade files can be found next to the database install files in `/path/to/icinga-src/module/idoutils/db/yourrdbm/`. The syntax is as follows:

```
<rdbm>-upgrade-<version>.sql
```

where `<rdbm>` could be mysql, pgsql or oracle and `<version>` points to the version you want to upgrade to.



Note

If you are upgrading from an older version and there are other versions in between be advised that you need to apply those upgrade files with incremental steps!

E.g. you have 1.0RC1 installed and want to upgrade to 1.0.1 - you will have to upgrade to 1.0 Stable first and then apply the upgrade to 1.0.1.

1. Backup your current database before upgrading!
2. Check current IDOUtils version and the target version. Check if there are any version in between and upgrade incremental if necessary.
3. Apply the upgrade(s) using a rdbm user with appropriate rights. You may use the upgradedb script, but this is not recommended (for MySQL only).

MySQL

```
$ mysql -u root -p <dbname> < /path/to/icinga-src/module/idoutils/db/mysql/mysql-upgrade-<version>.sql
```

Postgresql

```
# su - postgres
$ psql -U icinga -d icinga < /path/to/icinga-src/module/idoutils/db/pgsql/pgsql-upgrade-<version>.sql
```

Oracle

```
# su - oracle
$ sqlplus dbuser/dbpass
SQL> @oracle-upgrade-<version>.sql
```

Upgrading IDOUtils to 1.4

Oracle

- The minimum supported version is Oracle 10g R2. Older versions may still work.
- The optional separation of data, index and lob tablespaces is introduced. Modify `oracle-upgrade-1.4.0.sql` and define your tablespaces. You can provide your existing tablespace names for all defines.

Actions:

- remove number limitations
- drop most of existing NOT NULL constraints
- label constraints
- recreate index and LOBs in new tablespaces
- set sequences NOCACHE
- changes Oracle functions to trap NO_DATA exceptions



Caution

Upgrading IDOUtils for Oracle to 1.4 requires some kind of "magic". Make sure to

1. copy the complete upgrade folder
2. edit `oracle-upgrade-1.4.0.sql` and set the tablespaces for DATA, LOB and IXT
3. run the upgrade script

Upgrading IDOUtils to 1.3

IDOUtils 1.3 re-introduces the usage of the dbversion table in IDOUtils schema. The upgrade scripts will make sure that dbversion holds the current version while ido2db compares program version with database version and will print an error into syslog if not matching.

Just apply the upgrading script for IDOUtils 1.3 in `module/idoutils/db/<rdbms>/<rdbms>-upgrade-1.3.sql` against your current schema (using incremental steps as usual, not leaving an upgrade version behind!).

After you finished the upgrade you should check the database version.

```
SQL> SELECT * FROM icinga_dbversion;
```



Note

The Oracle tablename is "dbversion" (instead of "icinga_dbversion").

Upgrading IDOUtils to 1.0.3

There were some few minor changes:

- change display_name to varchar(255) for mysql/oracle
- update pgsql schema, replace varchar(n) by text
- change configfile variables value to 1024 length for mysql/oracle

Apply the upgrading scripts for IDOUtils 1.0.3 as the general procedure applies for your RDBMS.

Upgrading IDOUtils to 1.0.2

There was a significant long lasting bug in IDOUtils which is now resolved in Icinga 1.0.2: Everytime the core was restarted, the overall amount of objects was freshly inserted instead of using the old ones for actual config, status and historical relations. E.g. having 4k objects (hosts, services, contacts, etc), that meant restarting the core twice resulted in $4k + 4k + 4k = 12k$ objects.

For config and live status data, this is by means not really important as their relations normally get cleaned when the core gets restarted. But for historical data, e.g. hostchecks before the restart were in another relation than newer hostchecks after the restart. That has led into data inconsistency which has been worked on to resolve that in an easy way - next to the IDOUtils bugfix itself.

Therefore, next to the normal SQL upgrade scripts for 1.0.2 (e.g. mysql-upgrade-1.0.2.sql) an extended SQL script has been created.

It catches up on each table and object_id with a single clustered query in order to fix the relation historical table 1..1 objects table. It also cleans up broken data caused by the restarts.

Overall this has been tested and is now available for everyone to upgrade and fix those issues.

Please apply the script as you like to - directly or step by step as commented in the script. Those scripts are provided as is without any warranty so please use at your own risk - if you only depend on live data, dropping and recreating the database will take less effort.

* <rdbms>-upgrade-1.0.2-fix-object-relations.sql

The "normal" upgrade script only is available for MySQL - the binary casts for case sensitive comparison were removed because of massive performance issues. Instead a new collation is introduced.

* mysql-upgrade-1.0.2.sql

Upgrading IDOUtils to 1.0.1

Please make sure that you already have upgraded to Icinga IDOUtils 1.0 before reading this section! There have been several (big) changes made to all supported RDBMS to please be advised to read carefully! All database scripts are now reorganized within subfolders. Changes for all RDBMS are setting more indexes and also size modifications for the command_line column in several tables which exceeded 255 characters. RDBMS specific changes and howtos are listed below:

MySQL

Change of the database engine from MYISAM to InnoDB. Reason for that mainly is row locks/transactions/rollbacks instead of a bit more speed at insert time.

The upgrade script performs an ALTER TABLE statement. If you don't like that idea, you can also do the following:

- Dump your existing database e.g.

```
#> mysqldump -u root -p icinga > icinga.sql
```

- Change all entries from "MYISAM" to "InnoDB"
- Import the adapted dataset into a new database (if you want to use the old one make sure to drop that in the first place, and recreate only the database)

PostgreSQL

The systemcommands table was missing the column named output. This will be added during upgrading.

Oracle

First of all, make sure that you set open_cursors to more than the default 50. This will increase performance at several stages. The upgrade scripts will add two new procedures needed for the delete stuff written in DML.

Furthermore there has been a rather huge change regarding the autoincrement sequence and after insert triggers (emulating the mysql autoincrement on primary key). The old routine has been completely dropped meaning all triggers and the autoincrement sequence will be removed during upgrading. As a replacement, sequences for each table will be added and used in current IDOUtils Oracle.

With existing data sets this will lead into problems during importing - the sequences start at 1 while the primary key (id) will have a maximum. For that reason there is a basic procedure available which does the following: By given a sequence name and a table name, it extracts the maximum id value +1 from imported data and alters the sequence to start with this value instead.

Please be advised to use that procedure yourself for all tables or on separated tables - it highly depends on your needs. The procedure is commented out, and provided as is without any warranty regarding data loss. Ask your DBA in case of upgrading with existing data.

Upgrading IDOUtils to 1.0

There was a unique key failure in MySQL coming through the fork causing several tables to keep duplicated and useless rows. This concerns the following tables:

- timedevents, timedeventqueue
- servicechecks
- systemcommands

If you look at the table creation e.g. servicechecks:

```
mysql> show create table icinga_servicechecks;
```

you should see something like this

```
PRIMARY KEY ('servicecheck_id'),
KEY 'instance_id' ('instance_id'),
KEY 'service_object_id' ('service_object_id'),
KEY 'start_time' ('start_time')
```

Changing this to

```
PRIMARY KEY ('servicecheck_id'),
UNIQUE KEY 'instance_id' ('instance_id','service_object_id','start_time','start_time_usec')
```

will need your attention on the following procedure!

**If you are upgrading from 1.0RC please be advised to use
module/idoutils/db/mysql/mysql-upgrade-1.0.sql - if you are using an older version, please
apply the incremental steps to get to 1.0RC first!**

Please backup your database and stop ido2db before applying this patch!

```
#> /etc/init.d/ido2db stop
#> mysql -u root -p icinga < /path/to/icinga-src/module/idoutils/db/mysql/mysql-upgrade-1.0.sql
```

The patch will do the following through MySQL statements:

- add a temporary column named 'active' to mark the updated row
- extract the needed information of two duplicate rows based on the unique constraint, update the second row and mark first=inactive, second=active
- delete all inactive marked rows
- drop false keys
- add unique key
- drop temporary column 'active'

This procedure will be executed for each table, so it might take quite a long time depending on your table size and/or db specs.

If you changed something on the keys before please make sure you'll have the same database scheme applied as in 1.0RC otherwise the script will fail.

[Prev](#)

[Up](#)

[Next](#)

[Upgrading \(to\) Icinga](#)

[Home](#)

[Monitoring Windows Machines](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Monitoring Windows Machines

[Prev](#)[Chapter 2. Getting Started](#)[Next](#)

Monitoring Windows Machines

Introduction

This document describes how you can monitor "private" services and attributes of Windows machines, such as:

- Memory usage
- CPU load
- Disk usage
- Service states
- Running processes
- etc.

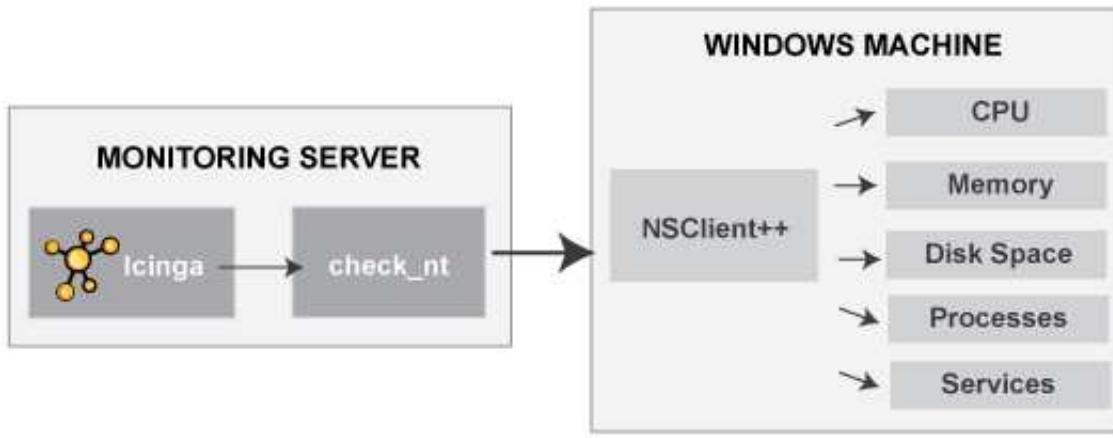
Publicly available services that are provided by Windows machines (HTTP, FTP, POP3, etc.) can be monitored easily by following the documentation on [monitoring publicly available services](#).



Note

These instructions assume that you've installed Icinga according to the [quickstart guide](#). The sample configuration entries below reference objects that are defined in the sample config files (*commands.cfg*, *templates.cfg*, etc.) that are installed if you follow the quickstart.

Overview



Monitoring private services or attributes of a Windows machine requires that you install an agent on it. This agent acts as a proxy between the Icinga plugin that does the monitoring and the actual service or attribute of the Windows machine. Without installing an agent on the Windows box, Icinga would be unable to monitor private services or attributes of the Windows box.

For this example, we will be installing the [NSClient++](#) addon on the Windows machine and using the *check_nt* plugin to communicate with the NSClient++ addon. The *check_nt* plugin should already be installed on the Icinga server if you followed the quickstart guide.

Other Windows agents (like [NC_Net](#)) could be used instead of NSClient++ if you wish - provided you change command and service definitions, etc. a bit. For the sake of simplicity we will only cover using the NSClient++ addon in these instructions.

Steps

There are several steps you'll need to follow in order to monitor a new Windows machine. They are:

1. Perform first-time prerequisites
2. Install a monitoring agent on the Windows machine
3. Create new host and service definitions for monitoring the Windows machine
4. Restart the Icinga daemon

What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A *check_nt* command definition has been added to the *commands.cfg* file. This allows you to use the *check_nt* plugin to monitor Window services.
- A Windows server host template (called *windows-server*) has already been created in the *templates.cfg* file. This allows you to add new Windows host definitions in a simple manner.

The above-mentioned config files can be found in the */usr/local/icinga/etc/objects/* directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, we'd recommend waiting until you're more familiar with configuring Icinga before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Windows boxes in no time.

Prerequisites

The first time you configure Icinga to monitor a Windows machine, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* Windows machine you monitor.

Edit the main Icinga config file.

```
#> vi /usr/local/icinga/etc/icinga.cfg
```

Remove the leading hash (#) sign from the following line in the main configuration file:

```
#cfg_file=/usr/local/icinga/etc/objects/windows.cfg
```

Save the file and exit.

What did you just do? You told Icinga to look to the */usr/local/icinga/etc/objects/windows.cfg* to find additional object definitions. That's where you'll be adding Windows host and service definitions. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* Windows machine you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Installing the Windows Agent

Before you can begin monitoring private services and attributes of Windows machines, you'll need to install an agent on those machines. We recommend using the NSClient++ addon, which can be found at <http://sourceforge.net/projects/nscplus>. These instructions will take you through a basic installation of the NSClient++ addon, as well as the configuration of Icinga for monitoring the Windows machine.

1. Download the latest stable version of the NSClient++ addon from <http://sourceforge.net/projects/nscplus>
2. Unzip the NSClient++ files into a new C:\NSClient++ directory
3. Open a command prompt and change to the C:\NSClient++ directory
4. Register the NSClient++ system service with the following command:

```
nsclient++ /install
```

5. Open the services manager and make sure the NSClientpp service is allowed to interact with the desktop (see the 'Log On' tab of the services manager). If it isn't already allowed to interact with the desktop, check the box to allow it to.



6. Edit the NSC.INI file (located in the C:\NSClient++ directory) and make the following changes:

- Uncomment all the modules listed in the [modules] section, except for CheckWMI.dll and RemoteConfiguration.dll
- Optionally require a password for clients by changing the 'password' option in the [Settings] section.
- Uncomment the 'allowed_hosts' option in the [Settings] section. Add the IP address of the Icinga server to this line, ip.add.ress/bits for a range or leave it blank to allow all hosts to connect.
- Make sure the 'port' option in the [NSClient] section is uncommented and set to '12489' (the default port).

7. Start the NSClient++ service with the following command:

```
nsclient++ /start
```

8. Success! The Windows server can now be added to the Icinga monitoring configuration...

Configuring Icinga

Now it's time to define some [object definitions](#) in your Icinga configuration files in order to monitor the new Windows machine.

Open the *windows.cfg* file for editing.

```
#> vi /usr/local/icinga/etc/objects/windows.cfg
```

Add a new [host](#) definition for the Windows machine that you're going to monitor. If this is the *first* Windows machine you're monitoring, you can simply modify the sample host definition in *windows.cfg*. Change the *host_name*, *alias*, and *address* fields to appropriate values for the Windows box.

```
define host{
    ; Inherit default values from a Windows server template
    use           windows-server ; make sure you keep this line!
    host_name     winserver
    alias         My Windows Server
    address       192.168.1.2
}
```

Good. Now you can add some service definitions (to the same configuration file) in order to tell Icinga to monitor different aspects of the Windows machine. If this is the **first** Windows machine you're monitoring, you can simply modify the sample service definitions in *windows.cfg*.



Note

Replace "winserver" in the example definitions below with the name you specified in the *host_name* directive of the host definition you just added.

Add the following service definition to monitor the version of the NSClient++ addon that is running on the Windows server. This is useful when it comes time to upgrade your Windows servers to a newer version of the addon, as you'll be able to tell which Windows machines still need to be upgraded to the latest version of NSClient++.

```
define service{
    use           generic-service
    host_name     winserver
    service_description NSClient++ Version
    check_command  check_nt!CLIENTVERSION
}
```

Add the following service definition to monitor the uptime of the Windows server.

```
define service{
    use           generic-service
    host_name     winserver
    service_description Uptime
    check_command  check_nt!UPTIME
}
```

Add the following service definition to monitor the CPU utilization on the Windows server and generate a CRITICAL alert if the 5-minute CPU load is 90% or more or a WARNING alert if the 5-minute load is 80% or greater.

```
define service{
    use           generic-service
    host_name     winserver
    service_description CPU Load
    check_command  check_nt!CPULOAD!-l 5,80,90
}
```

Add the following service definition to monitor memory usage on the Windows server and generate a CRITICAL alert if memory usage is 90% or more or a WARNING alert if memory usage is 80% or greater.

```
define service{
    use           generic-service
    host_name     winserver
    service_description Memory Usage
    check_command  check_nt!MEMUSE!-w 80 -c 90
}
```

Add the following service definition to monitor usage of the C:\ drive on the Windows server and generate a CRITICAL alert if disk usage is 90% or more or a WARNING alert if disk usage is 80% or greater.

```
define service{
    use          generic-service
    host_name   winserver
    service_description C:\ Drive Space
    check_command  check_nt!USEDISKSPACE!-l c -w 80 -c 90
}
```

Add the following service definition to monitor the W3SVC service state on the Windows machine and generate a CRITICAL alert if the service is stopped.

```
define service{
    use          generic-service
    host_name   winserver
    service_description W3SVC
    check_command  check_nt!SERVICESTATE!-d SHOWALL -l W3SVC
}
```

Add the following service definition to monitor the Explorer.exe process on the Windows machine and generate a CRITICAL alert if the process is not running.

```
define service{
    use          generic-service
    host_name   winserver
    service_description Explorer
    check_command  check_nt!PROCSTATE!-d SHOWALL -l Explorer.exe
}
```



Note

Well. To be honest it would be pretty dumb to monitor if the explorer is running but it's quite easy to test if it works like expected ;-).

That's it for now. You've added some basic services that should be monitored on the Windows box. Save the configuration file.

Password Protection

If you specified a password in the NSClient++ configuration file on the Windows machine, you'll need to modify the *check_nt* command definition to include the password. Open the *commands.cfg* file for editing.

```
#> vi /usr/local/icinga/etc/objects/commands.cfg
```

Change the definition of the *check_nt* command to include the "-s <PASSWORD>" argument (where PASSWORD is the password you specified on the Windows machine) like this:

```
define command{
    command_name  check_nt
    command_line   $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s PASSWORD -v $ARG1$ $ARG2$}
```

Save the file.

**Note**

It's a bad idea to specify passwords in the "normal" configuration files as they will be viewable via the web interface. You should use \$USERn\$ macros located in resource.cfg to store the passwords.

Restarting Icinga

You're done with modifying the Icinga configuration, so you'll need to [verify your configuration files](#) and [restart Icinga](#).

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Icinga until the verification process completes without any errors!

[Prev](#)[Up](#)[Next](#)[Upgrading IDOUtils Database](#)[Home](#)[Monitoring Linux/Unix Machines](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Monitoring Linux/Unix Machines

[Prev](#)[Chapter 2. Getting Started](#)[Next](#)

Monitoring Linux/Unix Machines

Introduction

This document describes how you can monitor "private" services and attributes of Linux/UNIX servers, such as:

- CPU load
- Memory usage
- Disk usage
- Logged in users
- Running processes
- etc.

Publicly available services that are provided by Linux servers (HTTP, FTP, SSH, SMTP, etc.) can be monitored easily by following the documentation on [monitoring publicly available services](#).



Note

These instructions assume that you've installed Icinga according to the [quickstart guide](#). The sample configuration entries below reference objects that are defined in the sample config files (*commands.cfg*, *templates.cfg*, etc.) that are installed if you follow the quickstart.

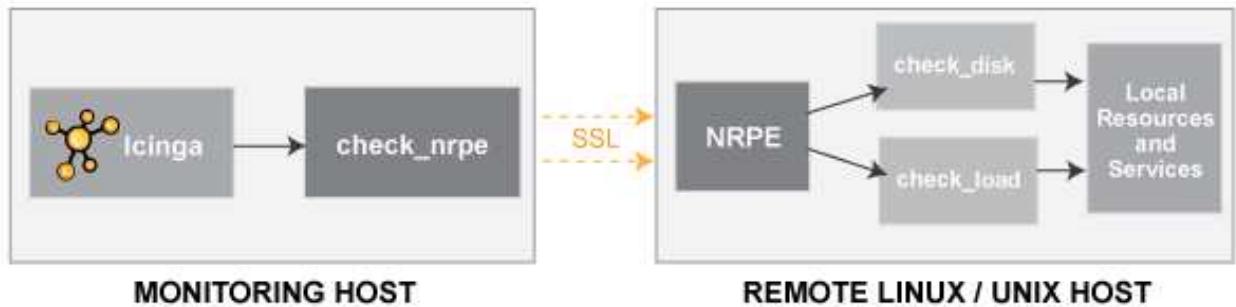
Overview



Note

This document has not been completed. We would recommend you read the documentation on the [NRPE addon](#) for instructions on how to monitor a remote Linux/Unix server.

There are several different ways to monitor attributes or remote Linux/Unix servers. One is by using shared SSH keys and the *check_by_ssh* plugin to execute plugins on remote servers. This method will not be covered here, but can result in high load on your monitoring server if you are monitoring hundreds or thousands of services. The overhead of setting up/destroying SSH connections is the cause of this.



Another common method of monitoring remote Linux/Unix hosts is to use the [NRPE addon](#). NRPE allows you to execute plugins on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host.

[Prev](#)[Up](#)[Next](#)[Monitoring Windows Machines](#)[Home](#)[Monitoring Netware Servers](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Monitoring Netware Servers**[Prev](#)[Chapter 2. Getting Started](#)[Next](#)

Monitoring Netware Servers

Introduction

This document provides information on how you can monitor Novell Netware servers.

External Resources

You can find documentation on monitoring Netware servers with Icinga at Novell's [Cool Solutions](#) site, including:

- [MRTGEXT: NLM module for MRTG and Nagios](#)
- [Nagios: Host and Service Monitoring Tool](#)
- [Nagios and NetWare: SNMP-based Monitoring](#)
- [Monitor DirXML/IDM Driver States with Nagios](#)
- [Check NDS Login ability with Nagios](#)
- [NDPS/iPrint Print Queue Monitoring by Nagios](#)
- [check_gwiaRL Plugin for Nagios 2.0](#)



Tip

When you visit Novell's [Cool Solutions](#) site, search for "Nagios" to find more articles and software components related to monitoring.

Thanks to [Christian Mies](#), [Rainer Brunold](#), and others for contributing Nagios and Netware documentation, addons, etc. on the Novell site!

[Prev](#)[Up](#)[Next](#)[Monitoring Linux/Unix Machines](#)[Home](#)[Monitoring Network Printers](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Monitoring Network Printers**[Prev](#)[Chapter 2. Getting Started](#)[Next](#)

Monitoring Network Printers

Introduction



This document describes how you can monitor the status of networked printers. Specifically, HP printers that have internal/external JetDirect cards/devices, or other print servers (like the Troy PocketPro 100S or the Netgear PS101) that support the JetDirect protocol.

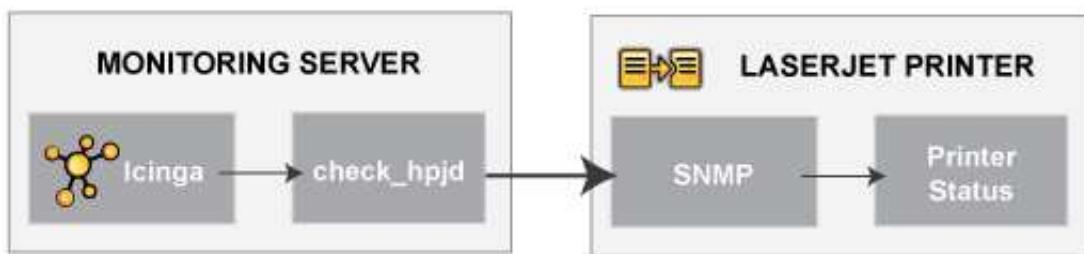
The *check_hpjd* plugin (which is part of the standard Nagios plugins distribution) allows you to monitor the status of JetDirect-capable printers which have SNMP enabled. The plugin is capable of detecting the following printer states:

- Paper Jam
- Out of Paper
- Printer Offline
- Intervention Required
- Toner Low
- Insufficient Memory
- Open Door
- Output Tray is Full
- and more...

**Note**

These instructions assume that you've installed Icinga according to the [quickstart guide](#). The sample configuration entries below reference objects that are defined in the sample config files (*commands.cfg*, *templates.cfg*, etc.) that are installed if you follow the quickstart.

Overview



Monitoring the status of a networked printer is pretty simple. JetDirect-enabled printers usually have SNMP enabled, which allows Icinga to monitor their status using the *check_hpjd* plugin.



Important

The *check_hpjd* plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in `/usr/local/icinga/libexec` before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Nagios plugins after issuing "make clean" in the source directory. Take a look at the [Quickstart Guide](#) for details.

Steps

There are several steps you'll need to follow in order to monitor a new network printer. They are:

1. Perform first-time prerequisites
2. Create new host and service definitions for monitoring the printer
3. Restart the Icinga daemon

What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A *check_hpjd* command definition has been added to the *commands.cfg* file. This allows you to use the *check_hpjd* plugin to monitor network printers.
- A printer host template (called *generic-printer*) has already been created in the *templates.cfg* file. This allows you to add new printer host definitions in a simple manner.

The above-mentioned config files can be found in the `/usr/local/icinga/etc/objects/` directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, we'd recommend waiting until you're more familiar with configuring Icinga before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network printers in no time.

Prerequisites

The first time you configure Icinga to monitor a network printer, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* printer you monitor.

Edit the main Icinga config file.

```
#> vi /usr/local/icinga/etc/icinga.cfg
```

Remove the leading hash (#) sign from the following line in the main configuration file:

```
#cfg_file=/usr/local/icinga/etc/objects/printer.cfg
```

Save the file and exit.

What did you just do? You told Icinga to look to the `/usr/local/icinga/etc/objects/printer.cfg` to find additional object definitions. That's where you'll be adding host and service definitions for the printer. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* printer you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Configuring Icinga

You'll need to create some [object definitions](#) in order to monitor a new printer.

Open the `printer.cfg` file for editing.

```
#> vi /usr/local/icinga/etc/objects/printer.cfg
```

Add a new `host` definition for the networked printer that you're going to monitor. If this is the *first* printer you're monitoring, you can simply modify the sample host definition in `printer.cfg`. Change the `host_name`, `alias`, and `address` fields to appropriate values for the printer.

```
define host{
    use          generic-printer ; Inherit default values from a template
    host_name    hplj2605dn      ; The name we're giving to this printer
    alias        HP LaserJet 2605dn ; A longer name associated with the printer
    address      192.168.1.30     ; IP address of the printer
    hostgroups   allhosts        ; Host groups this printer is associated with
}
```

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the printer. If this is the *first* printer you're monitoring, you can simply modify the sample service definition in `printer.cfg`.



Note

Replace "`hplj2605dn`" in the example definitions below with the name you specified in the `host_name` directive of the host definition you just added.

Add the following service definition to check the status of the printer. The service uses the `check_hpjd` plugin to check the status of the printer every 10 minutes by default. The SNMP community string used to query the printer is "public" in this example.

```
define service{
    use          generic-service ; Inherit values from a template
    host_name    hplj2605dn      ; The name of the host the service is associated with
    service_description Printer Status ; The service description
    check_command   check_hpjd!-C public ; The command used to monitor the service
    check_interval  10           ; Check the service every 10 minutes under normal conditions
    retry_interval  1            ; Re-check every minute until its final/hard state is determined
}
```

Add the following service definition to ping the printer every 10 minutes by default. This is useful for monitoring RTA, packet loss, and general network connectivity.

```
define service{
    use generic-service
    host_name hplj2605dn
    service_description PING
    check_command check_ping!3000.0,80%!5000.0,100%
    check_interval 10
    retry_interval 1
}
```

Save the file.

Restarting Icinga

Once you've added the new host and service definitions to the *printer.cfg* file, you're ready to start monitoring the printer. To do this, you'll need to [verify your configuration](#) and [restart Icinga](#).

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Icinga until the verification process completes without any errors!

[Prev](#)

[Up](#)

[Next](#)

[Monitoring Netware Servers](#)

[Home](#)

[Monitoring Routers and Switches](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Monitoring Routers and Switches

[Prev](#)[Chapter 2. Getting Started](#)[Next](#)

Monitoring Routers and Switches

Introduction



Switch

This document describes how you can monitor the status of network switches and routers. Some cheaper "unmanaged" switches and hubs don't have IP addresses and are essentially invisible on your network, so there's not any way to monitor them. More expensive switches and routers have addresses assigned to them and can be monitored by pinging them or using SNMP to query status information.

We'll describe how you can monitor the following things on managed switches, hubs, and routers:

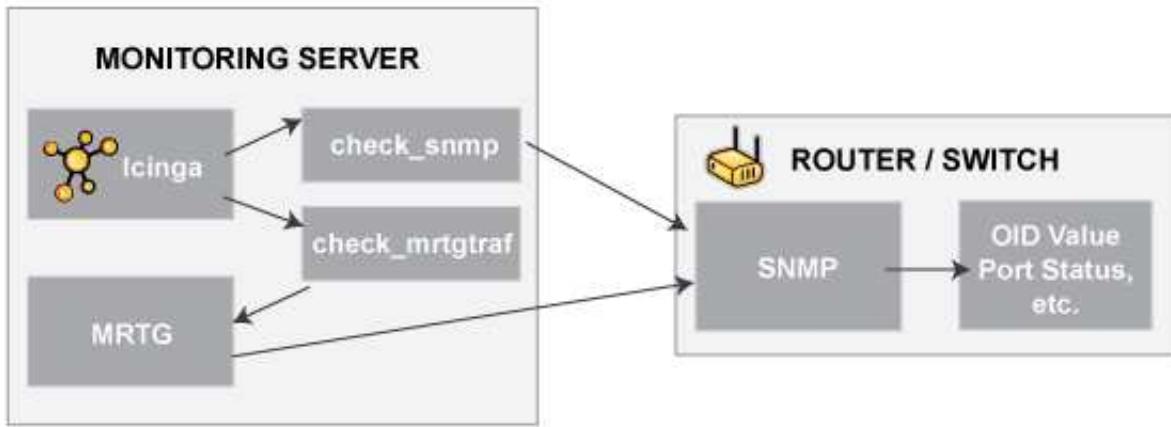
- Packet loss, round trip average
- SNMP status information
- Bandwidth / traffic rate



Note

These instructions assume that you've installed Icinga according to the [quickstart guide](#). The sample configuration entries below reference objects that are defined in the sample config files (*commands.cfg*, *templates.cfg*, etc.) that are installed when you follow the quickstart.

Overview



Monitoring switches and routers can either be easy or more involved - depending on what equipment you have and what you want to monitor. As they are critical infrastructure components, you'll no doubt want to monitor them in at least some basic manner.

Switches and routers can be monitored easily by "pinging" them to determine packet loss, RTA, etc. If your switch supports SNMP, you can monitor port status, etc. with the *check_snmp* plugin and bandwidth (if you're using MRTG) with the *check_mrtgtraf* plugin.

The *check_snmp* plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in */usr/local/icinga/libexec* before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Icinga plugins.

Steps

There are several steps you'll need to follow in order to monitor a new router or switch. They are:

1. Perform first-time prerequisites
2. Create new host and service definitions for monitoring the device
3. Restart the Icinga daemon

What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Two command definitions (*check_snmp* and *check_local_mrtgtraf*) have been added to the *commands.cfg* file. These allows you to use the *check_snmp* and *check_mrtgtraf* plugins to monitor network routers.
- A switch host template (called *generic-switch*) has already been created in the *templates.cfg* file. This allows you to add new router/switch host definitions in a simple manner.

The above-mentioned config files can be found in the */usr/local/icinga/etc/objects/* directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, we'd recommend waiting until you're more familiar with configuring Icinga before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network routers/switches in no time.

Prerequisites

The first time you configure Icinga to monitor a network switch, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* switch you monitor.

Edit the main Icinga config file.

```
#> vi /usr/local/icinga/etc/icinga.cfg
```

Remove the leading hash (#) sign from the following line in the main configuration file:

```
#cfg_file=/usr/local/icinga/etc/objects/switch.cfg
```

Save the file and exit.

What did you just do? You told Icinga to look to the */usr/local/icinga/etc/objects/switch.cfg* to find additional object definitions. That's where you'll be adding host and service definitions for routers and switches. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* router/switch you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

Configuring Icinga

You'll need to create some [object definitions](#) in order to monitor a new router/switch.

Open the *switch.cfg* file for editing.

```
#> vi /usr/local/icinga/etc/objects/switch.cfg
```

Add a new [host](#) definition for the switch that you're going to monitor. If this is the *first* switch you're monitoring, you can simply modify the sample host definition in *switch.cfg*. Change the *host_name*, *alias*, and *address* fields to appropriate values for the switch.

```
define host{
    use          generic-switch      ; Inherit default values from a template
    host_name    linksys-srw224p    ; The name we're giving to this switch
    alias        Linksys SRW224P Switch ; A longer name associated with the switch
    address      192.168.1.253       ; IP address of the switch
    hostgroups   allhosts,switches   ; Host groups this switch is associated with
}
```

Monitoring Services

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the switch. If this is the *first* switch you're monitoring, you can simply modify the sample service definition in *switch.cfg*.



Note

Replace "linksys-srw224p" in the example definitions below with the name you specified in the *host_name* directive of the host definition you just added.

Monitoring Packet Loss and RTA

Add the following service definition in order to monitor packet loss and round trip average between the Icinga host and the switch every 5 minutes under normal conditions.

```
define service{
    use generic-service ; Inherit values from a template
    host_name linksys-srw224p ; The name of the host the service is associated with
    service_description PING ; The service description
    check_command check_ping!200.0,20%!600.0,60% ; The command used to monitor the service
    check_interval 5 ; Check the service every 5 minutes under normal conditions
    retry_interval 1 ; Re-check every minute until its final/hard state is determined
}
```

This service will be:

- CRITICAL if the round trip average (RTA) is greater than 600 milliseconds or the packet loss is 60% or more
- WARNING if the RTA is greater than 200 ms or the packet loss is 20% or more
- OK if the RTA is less than 200 ms and the packet loss is less than 20%

Monitoring SNMP Status Information

If your switch or router supports SNMP, you can monitor a lot of information by using the *check_snmp* plugin. If it doesn't, skip this section.

Add the following service definition to monitor the uptime of the switch.

```
define service{
    use generic-service ; Inherit values from a template
    host_name linksys-srw224p
    service_description Uptime
    check_command check_snmp!-C public -o sysUpTime.0
}
```

In the *check_command* directive of the service definition above, the "-C public" tells the plugin that the SNMP community name to be used is "public" and the "-o sysUpTime.0" indicates which OID should be checked.

If you want to ensure that a specific port/interface on the switch is in an up state, you could add a service definition like this:

```
define service{
    use generic-service ; Inherit values from a template
    host_name linksys-srw224p
    service_description Port 1 Link Status
    check_command check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB
}
```

In the example above, the "-o ifOperStatus.1" refers to the OID for the operational status of port 1 on the switch. The "-r 1" option tells the *check_snmp* plugin to return an OK state if "1" is found in the SNMP result (1 indicates an "up" state on the port) and CRITICAL if it isn't found. The "-m RFC1213-MIB" is optional and tells the *check_snmp* plugin to only load the "RFC1213-MIB" instead of every single MIB that's installed on your system, which can help speed things up.

That's it for the SNMP monitoring example. There are a million things that can be monitored via SNMP, so it's up to you to decide what you need and want to monitor. Good luck!



Tip

You can usually find the OIDs that can be monitored on a switch by running the following command (replace 192.168.1.253 with the IP address of the switch):

```
snmpwalk -v1 -c public 192.168.1.253 -m ALL .1
```

Monitoring Bandwidth / Traffic Rate

If you're monitoring bandwidth usage on your switches or routers using [MRTG](#), you can have Icinga alert you when traffic rates exceed thresholds you specify. The `check_mrtgtraf` plugin (which is included in the Icinga plugins distribution) allows you to do this.

You'll need to let the `check_mrtgtraf` plugin know what log file the MRTG data is being stored in, along with thresholds, etc. In this example, we're monitoring one of the ports on a Linksys switch. The MRTG log file is stored in `/var/lib/mrtg/192.168.1.253_1.log`. Here's the service definition we use to monitor the bandwidth data that's stored in the log file...

```
define service{
    use generic-service ; Inherit values from a template
    host_name linksys-srw224p
    service_description Port 1 Bandwidth Usage
    check_command check_local_mrtgtraf!:/var/lib/mrtg/192.168.1.253_1.log!AVG!1000000,2000000!5000000,5000000!10
}
```

In the example above, the "`/var/lib/mrtg/192.168.1.253_1.log`" option that gets passed to the `check_local_mrtgtraf` command tells the plugin which MRTG log file to read from. The "AVG" option tells it that it should use average bandwidth statistics. The "1000000,2000000" options are the warning thresholds (in bytes) for incoming traffic rates. The "5000000,5000000" are critical thresholds (in bytes) for outgoing traffic rates. The "10" option causes the plugin to return a CRITICAL state if the MRTG log file is older than 10 minutes (it should be updated every 5 minutes).

Save the file.

Restarting Icinga

Once you've added the new host and service definitions to the `switch.cfg` file, you're ready to start monitoring the router/switch. To do this, you'll need to [verify your configuration](#) and [restart Icinga](#).

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Icinga until the verification process completes without any errors!

[Prev](#)

[Up](#)

[Next](#)

Monitoring Network Printers

[Home](#)

Monitoring Publicly Available Services

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Monitoring Publicly Available Services**[Prev](#)**Chapter 2. Getting Started**[Next](#)

Monitoring Publicly Available Services

Introduction

This document describes how you can monitor publicly available services, applications and protocols. By "public" we mean services that are accessible across the network - either the local network or the greater Internet. Examples of public services include HTTP, POP3, IMAP, FTP, and SSH. There are many more public services that you probably use on a daily basis. These services and applications, as well as their underlying protocols, can usually be monitored by Icinga without any special access requirements.

Private services, in contrast, cannot be monitored with Icinga without an intermediary agent of some kind. Examples of private services associated with hosts are things like CPU load, memory usage, disk usage, current user count, process information, etc. These private services or attributes of hosts are not usually exposed to external clients. This situation requires that an intermediary monitoring agent be installed on any host that you need to monitor such information on. More information on monitoring private services on different types of hosts can be found in the documentation on:

- [Monitoring Windows machines](#)
- [Monitoring Netware servers](#)
- [Monitoring Linux/Unix machines](#)

**Tip**

Occasionally you will find that information on private services and applications can be monitored with SNMP. The SNMP agent allows you to remotely monitor otherwise private (and inaccessible) information about the host. For more information about monitoring services using SNMP, check out the documentation on [monitoring switches and routers](#).

**Note**

These instructions assume that you've installed Icinga according to the [quickstart guide](#). The sample configuration entries below reference objects that are defined in the sample *commands.cfg* and *localhost.cfg* config files.

Plugins for Monitoring Services

When you find yourself needing to monitor a particular application, service, or protocol, chances are good that a [plugin](#) exists to monitor it. The official Nagios plugins distribution comes with plugins that can be used to monitor a variety of services and protocols. There are also a large number of contributed plugins that can be found in the *contrib/* subdirectory of the plugin distribution. The [MonitoringExchange.org](#) website hosts a number of additional plugins that have been written by users, so check it out when you have a chance.

If you don't happen to find an appropriate plugin for monitoring what you need, you can always write your own. Plugins are easy to write, so don't let this thought scare you off. Read the documentation on [developing plugins](#) for more information.

We'll walk you through monitoring some basic services that you'll probably use sooner or later. Each of these services can be monitored using one of the plugins that gets installed as part of the Nagios plugins distribution. Let's get started...

Creating a Host Definition

Before you can monitor a service, you first need to define a [host](#) that is associated with the service. You can place host definitions in any object configuration file specified by a [cfg_file](#) directive or placed in a directory specified by a [cfg_dir](#) directive. If you have already created a host definition, you can skip this step.

For this example, lets say you want to monitor a variety of services on a remote host. Let's call that host *remotehost*. The host definition can be placed in its own file or added to an already exiting object configuration file. Here's what the host definition for *remotehost* might look like:

```
define host{
    use           generic-host          ; Inherit default values from a template
    host_name     remotehost            ; The name we're giving to this host
    alias         Some Remote Host      ; A longer name associated with the host
    address       192.168.1.50          ; IP address of the host
    hostgroups    allhosts              ; Host groups this host is associated with
}
```

Now that a definition has been added for the host that will be monitored, we can start defining services that should be monitored. As with host definitions, service definitions can be placed in any object configuration file.

Creating Service Definitions

For each service you want to monitor, you need to define a [service](#) in Icinga that is associated with the host definition you just created. You can place service definitions in any object configuration file specified by a [cfg_file](#) directive or placed in a directory specified by a [cfg_dir](#) directive.

Some example service definitions for monitoring common public service (HTTP, FTP, etc) are given below.

Monitoring HTTP

Chances are you're going to want to monitor web servers at some point - either yours or someone else's. The *check_http* plugin is designed to do just that. It understands the HTTP protocol and can monitor response time, error codes, strings in the returned HTML, server certificates, and much more.

The *commands.cfg* file contains a command definition for using the *check_http* plugin. It looks like this:

```
define command{
    name          check_http
    command_name  check_http
    command_line   $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
```

A simple service definition for monitoring the HTTP service on the *remotehost* machine might look like this:

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  HTTP
    check_command  check_http
}
```

This simple service definition will monitor the HTTP service running on *remotehost*. It will produce alerts if the web server doesn't respond within 10 seconds or if it returns HTTP errors codes (403, 404, etc.). That's all you need for basic monitoring. Pretty simple, huh?



Tip

For more advanced monitoring, run the *check_http* plugin manually with *--help* as a command-line argument to see all the options you can give the plugin. This *--help* syntax works with all of the plugins we'll cover in this document.

A more advanced definition for monitoring the HTTP service is shown below. This service definition will check to see if the */download/index.php* URI contains the string "latest-version.tar.gz". It will produce an error if the string isn't found, the URI isn't valid, or the web server takes longer than 5 seconds to respond.

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  Product Download Link
    check_command  check_http!-u /download/index.php -t 5 -s "latest-version.tar.gz"
}
```

Monitoring FTP

When you need to monitor FTP servers, you can use the *check_ftp* plugin. The *commands.cfg* file contains a command definition for using the *check_ftp* plugin, which looks like this:

```
define command{
    command_name  check_ftp
    command_line   $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
```

A simple service definition for monitoring the FTP server on *remotehost* would look like this:

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  FTP
    check_command  check_ftp
}
```

This service definition will monitor the FTP service and generate alerts if the FTP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the FTP server running on port 1023 on *remotehost*. It will generate an alert if the server doesn't respond within 5 seconds or if the server response doesn't contain the string "Pure-FTPD [TLS]".

```
define service{
    use          generic-service      ; Inherit default values from a template
    host_name   remotehost
    service_description Special FTP
    check_command  check_ftp!-p 1023 -t 5 -e "Pure-FTPD [TLS]"
}
```

Monitoring SSH

When you need to monitor SSH servers, you can use the *check_ssh* plugin. The *commands.cfg* file contains a command definition for using the *check_ssh* plugin, which looks like this:

```
define command{
    command_name  check_ssh
    command_line   $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
}
```

A simple service definition for monitoring the SSH server on *remotehost* would look like this:

```
define service{
    use          generic-service      ; Inherit default values from a template
    host_name   remotehost
    service_description SSH
    check_command  check_ssh
}
```

This service definition will monitor the SSH service and generate alerts if the SSH server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the SSH server and generate an alert if the server doesn't respond within 5 seconds or if the server version string string doesn't match "OpenSSH_4.2".

```
define service{
    use          generic-service      ; Inherit default values from a template
    host_name   remotehost
    service_description SSH Version Check
    check_command  check_ssh!-t 5 -r "OpenSSH_4.2"
}
```

Monitoring SMTP

The *check_smtp* plugin can be using for monitoring your email servers. The *commands.cfg* file contains a command definition for using the *check_smtp* plugin, which looks like this:

```
define command{
    command_name  check_smtp
    command_line   $USER1$/check_smtp -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the SMTP server on *remotehost* would look like this:

```
define service{
    use          generic-service      ; Inherit default values from a template
    host_name   remotehost
    service_description SMTP
    check_command  check_smtp
}
```

This service definition will monitor the SMTP service and generate alerts if the SMTP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the SMTP server and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  SMTP Response Check
    check_command   check_smtp!-t 5 -e "mygreatmailserver.com"
}
```

Monitoring POP3

The *check_pop* plugin can be used for monitoring the POP3 service on your email servers. The *commands.cfg* file contains a command definition for using the *check_pop* plugin, which looks like this:

```
define command{
    command_name  check_pop
    command_line   $USER1$/check_pop -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the POP3 service on *remotehost* would look like this:

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  POP3
    check_command   check_pop
}
```

This service definition will monitor the POP3 service and generate alerts if the POP3 server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the POP3 service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use           generic-service      ; Inherit default values from a template
    host_name     remotehost
    service_description  POP3 Response Check
    check_command   check_pop!-t 5 -e "mygreatmailserver.com"
}
```

Monitoring IMAP

The *check_imap* plugin can be used for monitoring IMAP4 service on your email servers. The *commands.cfg* file contains a command definition for using the *check_imap* plugin, which looks like this:

```
define command{
    command_name  check_imap
    command_line   $USER1$/check_imap -H $HOSTADDRESS$ $ARG1$
}
```

A simple service definition for monitoring the IMAP4 service on *remotehost* would look like this:

```
define service{
    use generic-service          ; Inherit default values from a template
    host_name      remotehost
    service_description IMAP
    check_command   check_imap
}
```

This service definition will monitor the IMAP4 service and generate alerts if the IMAP server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the IMAP4 service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use generic-service          ; Inherit default values from a template
    host_name      remotehost
    service_description IMAP4 Response Check
    check_command   check_imap!-t 5 -e "mygreatmailserver.com"
}
```

Restarting Icinga

Once you've added the new host and service definitions to your object configuration file(s), you're ready to start monitoring them. To do this, you'll need to [verify your configuration](#) and [restart Icinga](#).

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Icinga until the verification process completes without any errors!

[Prev](#)

[Up](#)

[Next](#)

[Monitoring Routers and Switches](#)

[Home](#)

[Chapter 3. Configuring Icinga](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 3. Configuring Icinga

[Prev](#)

[Next](#)

Chapter 3. Configuring Icinga

Table of Contents

Configuration Overview
Main Configuration File Options
Object Configuration Overview
Object Definitions
Host definition
Hostgroup Definition
Service Definition
Servicegroup Definition
Contact Definition
Contactgroup Definition
Timeperiod Definition
Command Definition
Servicedependency Definition
Serviceescalation Definition
Hostdependency Definition
Hostescalation Definition
Hostextinfo Definition
Serviceextinfo Definition
Module Definition
Custom Object Variables
CGI Configuration File Options
Authentication And Authorization In The CGIs

[Prev](#)

[Next](#)

[Monitoring Publicly Available Services](#)

[Home](#)

[Configuration Overview](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Configuration Overview

[Prev](#)
[Chapter 3. Configuring Icinga](#)
[Next](#)

Configuration Overview

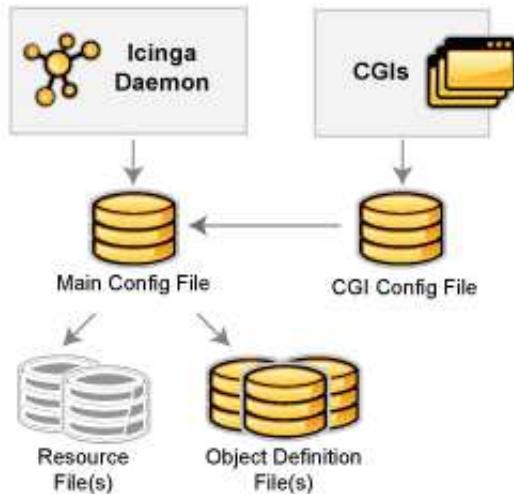
Introduction

There are several different configuration files that you're going to need to create or edit before you start monitoring anything. Be patient! Configuring Icinga can take quite a while, especially if you're first-time user. Once you figure out how things work, it'll all be well worth your time. :-)



Note

Sample configuration files are installed in the `/usr/local/icinga/etc/` directory when you follow the [quickstart installation guide](#).



Main Configuration File

The main configuration file contains a number of directives that affect how the Icinga daemon operates. This config file is read by both the Icinga daemon and the CGIs. This is where you're going to want to get started in your configuration adventures.

Documentation for the main configuration file can be found [here](#).

Resource File(s)

Resource files can be used to store user-defined macros. The main point of having resource files is to use them to store sensitive configuration information (like passwords), without making them available to the CGIs.

You can specify one or more optional resource files by using the [resource_file](#) directive in your main configuration file.

Object Definition Files

Object definition files are used to define hosts, services, hostgroups, contacts, contactgroups, commands, etc. This is where you define all the things you want monitor and how you want to monitor them.

You can specify one or more object definition files by using the [cfg_file](#) and/or [cfg_dir](#) directives in your main configuration file.

An introduction to object definitions, and how they relate to each other, can be found [here](#).

Your object definition files can in turn include other files using the [include_file](#) or [include_dir](#) directives. These can only occur outside of the actual object definitions, and behave very much like the [cfg_file=](#) and [cfg_dir=](#) directives in the main configuration file.

CGI Configuration File

The CGI configuration file contains a number of directives that affect the operation of the [CGIs](#). It also contains a reference the main configuration file, so the CGIs know how you've configured Icinga and where your object definitions are stored.

Documentation for the CGI configuration file can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

[Chapter 3. Configuring Icinga](#)

[Home](#)

[Main Configuration File Options](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Main Configuration File Options

[Prev](#)[Chapter 3. Configuring Icinga](#)[Next](#)

Main Configuration File Options

Notes

When creating and/or editing configuration files, keep the following in mind:

1. Lines that start with a '#' character are taken to be comments and are not processed
2. Variables names must begin at the start of the line - no white space is allowed before the name
3. Variable names are case-sensitive

Sample Configuration File



Tip

A sample main configuration file (`/usr/local/icinga/etc/icinga.cfg`) is installed for you when you follow the [quickstart installation guide](#).

Config File Location

The main configuration file is usually named `icinga.cfg` and located in the `/usr/local/icinga/etc` directory.

Configuration File Variables

- [Log file](#)
- [Object configuration file](#)
- [Object configuration directory](#)
- [Object cache file](#)
- [Precached object file](#)
- [Resource file](#)
- [Temp file](#)

- Temp path
- Status file
- Status file update interval
- Icinga user
- Icinga group
- Notifications option
- Service check execution option
- Passive service check acceptance option
- Host check execution option
- Passive host check acceptance option
- Event handler option
- Log rotation method
- Log archive path
- External command check option
- External command check interval
- External command file
- External command buffer slots option
- Lock file
- State retention option
- State retention file
- Sync retention file
- Automatic state retention update interval
- Use retained program state option
- Use retained scheduling info option
- Retained host/service attribute masks
- Retained process attribute masks
- Retained contact attribute masks
- Syslog logging option
- Syslog local facility logging option

- Syslog local facility logging value
- Notification logging option
- Service check retry logging option
- Host retry logging option
- Event handler logging option
- Initial state logging option
- External command logging option
- External commands user logging option
- Passive check logging option
- Current state logging option
- Long plugin output logging option
- Global host event handler
- Global service event handler
- Stalking event handlers for hosts
- Stalking event handlers for services
- Inter-check sleep time
- Service inter-check delay method
- Maximum service check spread
- Service interleave factor
- Maximum concurrent service checks
- Check result reaper frequency
- Maximum check result reaper time
- Check result path
- Maximum check result file age
- Host inter-check delay method
- Maximum host check spread
- Timing interval length
- Auto-rescheduling option
- Auto-rescheduling interval

- Auto-rescheduling window
- Aggressive host checking option
- Translate passive host checks option
- Passive host checks are SOFT option
- Predictive host dependency checks option
- Predictive service dependency checks option
- Cached host check horizon
- Cached service check horizon
- Large installation tweaks option
- Flap detection option
- Low service flap threshold
- High service flap threshold
- Low host flap threshold
- High host flap threshold
- Soft state dependencies option
- Service check timeout
- Service check timeout state
- Host check timeout
- Event handler timeout
- Notification timeout
- Obsessive compulsive service processor timeout
- Obsessive compulsive host processor timeout
- Performance data processor command timeout
- Obsess over services option
- Obsessive compulsive service processor command
- Obsess over hosts option
- Obsessive compulsive host processor command
- Performance data processing option
- Host performance data processing command

- Service performance data processing command
- Host performance data file
- Service performance data file
- Host performance data file template
- Service performance data file template
- Host performance data file mode
- Service performance data file mode
- Host performance data file processing interval
- Service performance data file processing interval
- Host performance data file processing command
- Service performance data file processing command
- Orphaned service check option
- Orphaned host check option
- Service freshness checking option
- Service freshness check interval
- Host freshness checking option
- Host freshness check interval
- Embedded Perl interpreter option
- Embedded Perl implicit use option
- Date format
- Illegal object name characters
- Illegal macro output characters
- Regular expression matching option
- True regular expression matching option
- Administrator email address
- Administrator pager
- Debug file
- Debug level
- Debug verbosity

- Maximum debug file size option
- Allow empty hostgroup assignment
- Process empty host performance results
- Process empty service performance results

Below you will find descriptions of each main Icinga configuration file option...

Log File

Format: `log_file=<file_name>`

Example: `log_file=/usr/local/icinga/var/icinga.log`

This variable specifies where Icinga should create its main log file. This should be the first variable that you define in your configuration file, as Icinga will try to write errors that it finds in the rest of your configuration data to this file. If you have [log rotation](#) enabled, this file will automatically be rotated every hour, day, week, or month.

Object Configuration File

Format: `cfg_file=<file_name>`

`cfg_file=/usr/local/icinga/etc/hosts.cfg`

Example: `cfg_file=/usr/local/icinga/etc/services.cfg`

`cfg_file=/usr/local/icinga/etc/commands.cfg`

This directive is used to specify an [object configuration file](#) containing object definitions that Icinga should use for monitoring. Object configuration files contain definitions for hosts, host groups, contacts, contact groups, services, commands, etc. You can separate your configuration information into several files and specify multiple `cfg_file=` statements to have each of them processed.

Object Configuration Directory

Format: `cfg_dir=<directory_name>`

`cfg_dir=/usr/local/icinga/etc/commands`

Example: `cfg_dir=/usr/local/icinga/etc/services`

`cfg_dir=/usr/local/icinga/etc/hosts`

This directive is used to specify a directory which contains [object configuration files](#) that Icinga should use for monitoring. All files in the directory with a `.cfg` extension are processed as object config files. Additionally, Icinga will recursively process all config files in subdirectories of the directory you specify here. You can separate your configuration files into different directories and specify multiple `cfg_dir=` statements to have all config files in each directory processed.

Object Cache File

Format: **object_cache_file=<file_name>**

Example: **object_cache_file=/usr/local/icinga/var/objects.cache**

This directive is used to specify a file in which a cached copy of [object definitions](#) should be stored. The cache file is (re)created every time Icinga is (re)started and is used by the CGIs. It is intended to speed up config file caching in the CGIs and allow you to edit the source [object config files](#) while Icinga is running without affecting the output displayed in the CGIs.

Precached Object File

Format: **precached_object_file=<file_name>**

Example: **precached_object_file=/usr/local/icinga/var/objects.precache**

This directive is used to specify a file in which a pre-processed, pre-cached copy of [object definitions](#) should be stored. This file can be used to drastically improve startup times in large/complex Icinga installations. Read more information on how to speed up start times [here](#).

Resource File

Format: **resource_file=<file_name>**

Example: **resource_file=/usr/local/icinga/etc/resource.cfg**

This is used to specify an optional resource file that can contain \$USERn\$ [macro](#) definitions. \$USERn\$ macros are useful for storing usernames, passwords, and items commonly used in command definitions (like directory paths). The CGIs will *not* attempt to read resource files, so you can set restrictive permissions (600 or 660) on them to protect sensitive information. You can include multiple resource files by adding multiple resource_file statements to the main config file - Icinga will process them all. See the sample resource.cfg file in the *sample-config/* subdirectory of the Icinga distribution for an example of how to define \$USERn\$ macros.

Temp File

Format: **temp_file=<file_name>**

Example: **temp_file=/usr/local/icinga/var/icinga.tmp**

This is a temporary file that Icinga periodically creates to use when updating comment data, status data, etc. The file is deleted when it is no longer needed.

Temp Path

Format: **temp_path=<dir_name>**

Example: **temp_path=/tmp**

This is a directory that Icinga can use as scratch space for creating temporary files used during the monitoring process. You should run *tmpwatch*, or a similiar utility, on this directory occassionally to delete files older than 24 hours.

Status File

Format: **status_file=<file_name>**

Example: **status_file=/usr/local/icinga/var/status.dat**

This is the file that Icinga uses to store the current status, comment, and downtime information. This file is used by the CGIs so that current monitoring status can be reported via a web interface. The CGIs must have read access to this file in order to function properly. This file is deleted every time Icinga stops and recreated when it starts.

Status File Update Interval

Format: **status_update_interval=<seconds>**

Example: **status_update_interval=15**

This setting determines how often (in seconds) that Icinga will update status data in the [status file](#). The minimum update interval is 1 second.

Icinga User

Format: **icinga_user=<username/UID>**

Example: **icinga_user=icinga**

This is used to set the effective user that the Icinga process should run as. After initial program startup and before starting to monitor anything, Icinga will drop its effective privileges and run as this user. You may specify either a username or a UID.

Icinga Group

Format: **icinga_group=<groupname/GID>**

Example: **icinga_group=icinga**

This is used to set the effective group that the Icinga process should run as. After initial program startup and before starting to monitor anything, Icinga will drop its effective privileges and run as this group. You may specify either a groupname or a GID.

Notifications Option

Format: **enable_notifications=<0/1>**

Example: **enable_notifications=1**

This option determines whether or not Icinga will send out [notifications](#) when it initially (re)starts. If this option is disabled, Icinga will not send out notifications for any host or service.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Disable notifications
- 1 = Enable notifications (default)

Service Check Execution Option

Format: **`execute_service_checks=<0/1>`**

Example: **`execute_service_checks=1`**

This option determines whether or not Icinga will execute service checks when it initially (re)starts. If this option is disabled, Icinga will not actively execute any service checks and will remain in a sort of "sleep" mode (it can still accept [passive checks](#) unless you've [disabled them](#)). This option is most often used when configuring backup monitoring servers, as described in the documentation on [redundancy](#), or when setting up a [distributed](#) monitoring environment.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Don't execute service checks
- 1 = Execute service checks (default)

Passive Service Check Acceptance Option

Format: **`accept_passive_service_checks=<0/1>`**

Example: **`accept_passive_service_checks=1`**

This option determines whether or not Icinga will accept [passive service checks](#) when it initially (re)starts. If this option is disabled, Icinga will not accept any passive service checks.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Don't accept passive service checks
- 1 = Accept passive service checks (default)

Host Check Execution Option

Format: **`execute_host_checks=<0/1>`**

Example: **`execute_host_checks=1`**

This option determines whether or not Icinga will execute on-demand and regularly scheduled host checks when it initially (re)starts. If this option is disabled, Icinga will not actively execute any host checks, although it can still accept [passive host checks](#) unless you've [disabled them](#)). This option is most often used when configuring backup monitoring servers, as described in the documentation on [redundancy](#), or when setting up a [distributed](#) monitoring environment.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Don't execute host checks
- 1 = Execute host checks (default)

Passive Host Check Acceptance Option

Format: **`accept_passive_host_checks=<0/1>`**

Example: **`accept_passive_host_checks=1`**

This option determines whether or not Icinga will accept [passive host checks](#) when it initially (re)starts. If this option is disabled, Icinga will not accept any passive host checks.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Don't accept passive host checks
- 1 = Accept passive host checks (default)

Event Handler Option

Format: **enable_event_handlers=<0/1>**

Example: **enable_event_handlers=1**

This option determines whether or not Icinga will run [event handlers](#) when it initially (re)starts. If this option is disabled, Icinga will not run any host or service event handlers.

**Note**

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

Values are as follows:

- 0 = Disable event handlers
- 1 = Enable event handlers (default)

Log Rotation Method

Format: **log_rotation_method=<n/h/d/w/m>**

Example: **log_rotation_method=d**

This is the rotation method that you would like Icinga to use for your log file. Values are as follows:

- n = None (don't rotate the log - this is the default)
- h = Hourly (rotate the log at the top of each hour)
- d = Daily (rotate the log at midnight each day)

- w = Weekly (rotate the log at midnight on Saturday)
- m = Monthly (rotate the log at midnight on the last day of the month)

Log Archive Path

Format: **log_archive_path=<path>**

Example: **log_archive_path=/usr/local/icinga/var/archives/**

This is the directory where Icinga should place log files that have been rotated. This option is ignored if you choose to not use the [log rotation](#) functionality.

External Command Check Option

Format: **check_external_commands=<0/1>**

Example: **check_external_commands=1**

This option determines whether or not Icinga will check the [command file](#) for commands that should be executed. This option must be enabled if you plan on using the [command CGI](#) to issue commands via the web interface. More information on external commands can be found [here](#).

- 0 = Don't check external commands
- 1 = Check external commands (default)

External Command Check Interval

Format: **command_check_interval=<xxx>[s]**

Example: **command_check_interval=1**

If you specify a number with an "s" appended to it (i.e. 30s), this is the number of *seconds* to wait between external command checks. If you leave off the "s", this is the number of "time units" to wait between external command checks. Unless you've changed the [interval_length](#) value (as defined below) from the default value of 60, this number will mean minutes.



Note

By setting this value to **-1**, Icinga will check for external commands as often as possible. Each time Icinga checks for external commands it will read and process all commands present in the [command file](#) before continuing on with its other duties. More information on external commands can be found [here](#).

External Command File

Format: **command_file=<file_name>**

Example: **command_file=/usr/local/icinga/var/rw/icinga.cmd**

This is the file that Icinga will check for external commands to process. The [command CGI](#) writes commands to this file. The external command file is implemented as a named pipe (FIFO), which is created when Icinga starts and removed when it shuts down. If the file exists when Icinga starts, the Icinga process will terminate with an error message. More information on external commands can be found [here](#).

External Command Buffer Slots

Format: **`external_command_buffer_slots=<#>`**

Example: **`external_command_buffer_slots=512`**



Note

This is an advanced feature. This option determines how many buffer slots Icinga will reserve for caching external commands that have been read from the external command file by a worker thread, but have not yet been processed by the main thread of the Icinga deamon. Each slot can hold one external command, so this option essentially determines how many commands can be buffered. For installations where you process a large number of passive checks (e.g. [distributed setups](#)), you may need to increase this number. You should consider using PNP4Nagios to graph Icinga's usage of external command buffers. You can read more on how to configure graphing [here](#).

Update Checks

Format: **`check_for_updates=<0/1>`**

Example: **`check_for_updates=1`**

This option determines whether Nagios will check for automatic updates.

Icinga ignores this option. Starting with version 1.2 you will receive a warning.

Bare Update Checks

Format: **`bare_update_checks=<0/1>`**

Example: **`bare_update_checks=0`**

This option determines what data Nagios will be sent to `api.nagios.org`.

Icinga ignores this option. Starting with version 1.2 you will receive a warning.

Lock File

Format: **`lock_file=<file_name>`**

Example: **`lock_file=/tmp/icinga.lock`**

This option specifies the location of the lock file that Icinga should create when it runs as a daemon (when started with the `-d` command line argument). This file contains the process id (PID) number of the running Icinga process.

State Retention Option

Format: **retain_state_information=<0/1>**

Example: **retain_state_information=1**

This option determines whether or not Icinga will retain state information for hosts and services between program restarts. If you enable this option, you should supply a value for the [state_retention_file](#) variable. When enabled, Icinga will save all state information for hosts and service before it shuts down (or restarts) and will read in previously saved state information when it starts up again.

- 0 = Don't retain state information
- 1 = Retain state information (default)

State Retention File

Format: **state_retention_file=<file_name>**

Example: **state_retention_file=/usr/local/icinga/var/retention.dat**

This is the file that Icinga will use for storing status, downtime, and comment information before it shuts down. When Icinga is restarted it will use the information stored in this file for setting the initial states of services and hosts before it starts monitoring anything. In order to make Icinga retain state information between program restarts, you must enable the [retain_state_information](#) option.

Sync Retention File

Format: **sync_retention_file=<file_name>**

Example: **sync_retention_file=/usr/local/icinga/var/sync.dat**

This is an advanced option which works like the [state_retention_file](#) so that you can load a subset of retention information such as status , acknowledgements, downtimes, and comments (you will have to create the contents of this file outside of Icinga). When Icinga is restarted, it reads the information in the [sync_retention_file](#) and will update the host or service specified *if the last_update time in the sync file is newer than in the state_retention_file*, otherwise information will be discarded for that host or service. After the sync file has been read, it will be removed. To disable, comment out the option. This option can be used starting with Icinga 1.0.2.

Automatic State Retention Update Interval

Format: **retention_update_interval=<minutes>**

Example: **retention_update_interval=60**

This setting determines how often (in minutes) that Icinga will automatically save retention data during normal operation. If you set this value to 0, Icinga will not save retention data at regular intervals, but it will still save retention data before shutting down or restarting. If you have disabled state retention (with the [retain_state_information](#) option), this option has no effect.

Use Retained Program State Option

Format: `use_retained_program_state=<0/1>`

Example: `use_retained_program_state=1`

This setting determines whether or not Icinga will set various program-wide state variables based on the values saved in the retention file. Some of these program-wide state variables that are normally saved across program restarts if state retention is enabled include the `enable_notifications`, `enable_flap_detection`, `enable_event_handlers`, `execute_service_checks`, and `accept_passive_service_checks` options. If you do not have `state retention` enabled, this option has no effect.

- 0 = Don't use retained program state
- 1 = Use retained program state (default)

Use Retained Scheduling Info Option

Format: `use_retained_scheduling_info=<0/1>`

Example: `use_retained_scheduling_info=1`

This setting determines whether or not Icinga will retain scheduling info (next check times) for hosts and services when it restarts. If you are adding a large number (or percentage) of hosts and services, we would recommend disabling this option when you first restart Icinga, as it can adversely skew the spread of initial checks. Otherwise you will probably want to leave it enabled.

- 0 = Don't use retained scheduling info
- 1 = Use retained scheduling info (default)

Retained Host and Service Attribute Masks

Format: `retained_host_attribute_mask=<number>`
`retained_service_attribute_mask=<number>`

Example: `retained_host_attribute_mask=0`
`retained_service_attribute_mask=0`

WARNING: This is an advanced feature. You'll need to read the Icinga source code to use this option effectively.

These options determine which host or service attributes are NOT retained across program restarts. The values for these options are a bitwise AND of values specified by the "MODATTR_" definitions in the include/common.h source code file. By default, all host and service attributes are retained.

Retained Process Attribute Masks

Format: **`retained_process_host_attribute_mask=<number>`**
 `retained_process_service_attribute_mask=<number>`
 `retained_process_host_attribute_mask=0`
 Example: **`retained_process_service_attribute_mask=0`**

WARNING: This is an advanced feature. You'll need to read the Icinga source code to use this option effectively.

These options determine which process attributes are NOT retained across program restarts. There are two masks because there are often separate host and service process attributes that can be changed. For example, host checks can be disabled at the program level, while service checks are still enabled. The values for these options are a bitwise AND of values specified by the "MODATTR_" definitions in the include/common.h source code file. By default, all process attributes are retained.

Retained Contact Attribute Masks

Format: **`retained_contact_host_attribute_mask=<number>`**
 `retained_contact_service_attribute_mask=<number>`
 `retained_contact_host_attribute_mask=0`
 Example: **`retained_contact_service_attribute_mask=0`**

WARNING: This is an advanced feature. You'll need to read the Icinga source code to use this option effectively.

These options determine which contact attributes are NOT retained across program restarts. There are two masks because there are often separate host and service contact attributes that can be changed. The values for these options are a bitwise AND of values specified by the "MODATTR_" definitions in the include/common.h source code file. By default, all process attributes are retained.

Syslog Logging Option

Format: **`use_syslog=<0/1>`**
 Example: **`use_syslog=1`**

This variable determines whether messages are logged to the syslog facility on your local host. Values are as follows:

- 0 = Don't use syslog facility
- 1 = Use syslog facility

Local Syslog Facility Option

Format: **`use_local_syslog_facility=<0/1>`**
 Example: **`use_syslog_local_facility=1`**

If you enabled `use_syslog` you can set Icinga to use a local facility instead of the default. Values are as follows:

- 0 = Don't use syslog local facility
- 1 = Use syslog local facility

This option can be used starting with Icinga 1.0.2.

Syslog Local Facility Value

Format: **`syslog_local_facility=<0|1|2|3|4|5|6|7>`**

Example: **`syslog_local_facility=1`**

If you enabled `use_syslog_local_facility` you can choose which local facility to use. Valid values are from 0 to 7. This option can be used starting with Icinga 1.0.2.

Notification Logging Option

Format: **`log_notifications=<0/1>`**

Example: **`log_notifications=1`**

This variable determines whether or not notification messages are logged. If you have a lot of contacts or regular service failures your log file will grow relatively quickly. Use this option to keep contact notifications from being logged.

- 0 = Don't log notifications
- 1 = Log notifications

Service Check Retry Logging Option

Format: **`log_service_retries=<0/1>`**

Example: **`log_service_retries=1`**

This variable determines whether or not service check retries are logged. Service check retries occur when a service check results in a non-OK state, but you have configured Icinga to retry the service more than once before responding to the error. Services in this situation are considered to be in "soft" states. Logging service check retries is mostly useful when attempting to debug Icinga or test out service [event handlers](#).

- 0 = Don't log service check retries
- 1 = Log service check retries

Host Check Retry Logging Option

Format: **`log_host_retries=<0/1>`**

Example: **`log_host_retries=1`**

This variable determines whether or not host check retries are logged. Logging host check retries is mostly useful when attempting to debug Icinga or test out host [event handlers](#).

- 0 = Don't log host check retries
- 1 = Log host check retries

Event Handler Logging Option

Format: **`log_event_handlers=<0/1>`**

Example: **`log_event_handlers=1`**

This variable determines whether or not service and host [event handlers](#) are logged. Event handlers are optional commands that can be run whenever a service or hosts changes state. Logging event handlers is most useful when debugging Icinga or first trying out your event handler scripts.

- 0 = Don't log event handlers
- 1 = Log event handlers

Initial States Logging Option

Format: **`log_initial_states=<0/1>`**

Example: **`log_initial_states=1`**

This variable determines whether or not Icinga will force all initial host and service states to be logged, even if they result in an OK state. Initial service and host states are normally only logged when there is a problem on the first check. Enabling this option is useful if you are using an application that scans the log file to determine long-term state statistics for services and hosts.

- 0 = Don't log initial states (default)
- 1 = Log initial states

External Command Logging Option

Format: **`log_external_commands=<0/1>`**

Example: **`log_external_commands=1`**

This variable determines whether or not Icinga will log [external commands](#) that it receives from the [external command file](#).



Note

This option does not control whether or not [passive service checks](#) (which are a type of external command) get logged. To enable or disable logging of passive checks, use the [log_passive_checks](#) option.

- 0 = Don't log external commands
- 1 = Log external commands (default)

External Commands User Logging Option

Format: **`log_external_commands_user=<0/1>`**

Example: **`log_external_commands_user=1`**

This option allows you to enable the logging of the current user of external commands. The syntax will be CMD;username;cmdargs instead of CMD;cmdargs written to the logs, if the external application sends that correctly. Since this will break compatibility with existing log parsers, it is intentionally disabled by default.



Note

This option is deprecated starting with Icinga 1.4 because you can use the directive [use_logging](#) to enable logging of CGI commands.

- 0 = Don't log user name using external commands (default)
- 1 = Log user name using external commands



Note

This option is available starting with Icinga 1.0.3.

Passive Check Logging Option

Format: **`log_passive_checks=<0/1>`**

Example: **`log_passive_checks=1`**

This variable determines whether or not Icinga will log [passive host and service checks](#) that it receives from the [external command file](#). If you are setting up a [distributed monitoring environment](#) or plan on handling a large number of passive checks on a regular basis, you may wish to disable this option so your log file doesn't get too large.

- 0 = Don't log passive checks
- 1 = Log passive checks (default)

Current State Logging Option

Format: **`log_current_states=<0/1>`**

Example: **`log_current_states=1`**

This variable determines whether or not Icinga will log the current states of hosts and services after a log rotation. If you set the value of `log_current_states` to 0 the current states will not be written to the main log file after a log rotation.

- 0 = Don't log current host and service states
- 1 = Log current host and service states (default)

**Note**

This option is available starting with Icinga 1.0.3.

Long Plugin Output Logging Option

Format: **log_current_states=<0/1>**

Example: **log_current_states=1**

This variable determines whether or not Icinga will log the complete output of a plugin (not only the first line). If you set the value of `log_long_plugin_output` to 1 the complete plugin output will be logged.

- 0 = Only log the first line of plugin output (default)
- 1 = Log complete plugin output

**Note**

This option is available starting with Icinga 1.0.3.

Global Host Event Handler Option

Format: **global_host_event_handler=<command>**

Example: **global_host_event_handler=log-host-event-to-db**

This option allows you to specify a host event handler command that is to be run for every host state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each host definition. The *command* argument is the short name of a command that you define in your [object configuration file](#). The maximum amount of time that this command can run is controlled by the `event_handler_timeout` option. More information on event handlers can be found [here](#).

Global Service Event Handler Option

Format: **global_service_event_handler=<command>**

Example: **global_service_event_handler=log-service-event-to-db**

This option allows you to specify a service event handler command that is to be run for every service state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each service definition. The *command* argument is the short name of a command that you define in your [object configuration file](#). The maximum amount of time that this command can run is controlled by the `event_handler_timeout` option. More information on event handlers can be found [here](#).

Event handlers for stalked hosts

Event handlers for stalked services

Format: `stalking_event_handlers_for_hosts=<0|1>`

Format: `stalking_event_handlers_for_services=<0|1>`

Example: `stalking_event_handlers_for_hosts=1`

These options allow you to specify whether or not Icinga executes event handlers for stalked hosts or services, respectively. This way it is possible to forward status information changes to external systems.

- 0 = Event handler disabled (default)
- 1 = Event handler enabled



Note

This option is available starting with Icinga 1.0.3.

Inter-Check Sleep Time

Format: `sleep_time=<seconds>`

Example: `sleep_time=1`

This is the number of seconds that Icinga will sleep before checking to see if the next service or host check in the scheduling queue should be executed. Note that Icinga will only sleep after it "catches up" with queued service checks that have fallen behind.

Service Inter-Check Delay Method

Format: `service_inter_check_delay_method=<n/d/s/x.xx>`

Example: `service_inter_check_delay_method=s`

This option allows you to control how service checks are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Icinga to calculate an average check interval and spread initial checks of all services out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally *not* recommended, as it will cause all service checks to be scheduled for execution at the same time. This means that you will generally have large CPU spikes when the services are all executed in parallel. More information on how to estimate how the inter-check delay affects service check scheduling can be found [here](#). Values are as follows:

- n = Don't use any delay - schedule all service checks to run immediately (i.e. at the same time!)
- d = Use a "dumb" delay of 1 second between service checks
- s = Use a "smart" delay calculation to spread service checks out evenly (default)

- x.xx = Use a user-supplied inter-check delay of x.xx seconds

Maximum Service Check Spread

Format: **max_service_check_spread=<minutes>**

Example: **max_service_check_spread=30**

This option determines the maximum number of minutes from when Icinga starts that all services (that are scheduled to be regularly checked) are checked. This option will automatically adjust the [service inter-check delay method](#) (if necessary) to ensure that the initial checks of all services occur within the timeframe you specify. In general, this option will not have an affect on service check scheduling if scheduling information is being retained using the [use_retained_scheduling_info](#) option. Default value is 30 (minutes).

Service Interleave Factor

Format: **service_interleave_factor=<s | x>**

Example: **service_interleave_factor=s**

This variable determines how service checks are interleaved. Interleaving allows for a more even distribution of service checks, reduced load on remote hosts, and faster overall detection of host problems. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of Icinga previous to 0.0.5 worked). Set this value to s (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. The best way to understand how interleaving works is to watch the [status CGI](#) (detailed view) when Icinga is just starting. You should see that the service check results are spread out as they begin to appear. More information on how interleaving works can be found [here](#).

- x = A number greater than or equal to 1 that specifies the interleave factor to use. An interleave factor of 1 is equivalent to not interleaving the service checks.
- s = Use a "smart" interleave factor calculation (default)

Maximum Concurrent Service Checks

Format: **max_concurrent_checks=<max_checks>**

Example: **max_concurrent_checks=20**

This option allows you to specify the maximum number of service checks that can be run in parallel at any given time. Specifying a value of 1 for this variable essentially prevents any service checks from being run in parallel. Specifying a value of 0 (the default) does not place any restrictions on the number of concurrent checks. You'll have to modify this value based on the system resources you have available on the machine that runs Icinga, as it directly affects the maximum load that will be imposed on the system (processor utilization, memory, etc.). More information on how to estimate how many concurrent checks you should allow can be found [here](#).

Check Result Reaper Frequency

Format: **check_result_reaper_frequency=<frequency_in_seconds>**

Example: **check_result_reaper_frequency=5**

This option allows you to control the frequency *in seconds* of check result "reaper" events. "Reaper" events process the results from host and service checks that have finished executing. These events constitute the core of the monitoring logic in Icinga.

Maximum Check Result Reaper Time

Format: **max_check_result_reaper_time=<seconds>**

Example: **max_check_result_reaper_time=30**

This option allows you to control the maximum amount of time *in seconds* that host and service check result "reaper" events are allowed to run. "Reaper" events process the results from host and service checks that have finished executing. If there are a lot of results to process, reaper events may take a long time to finish, which might delay timely execution of new host and service checks. This variable allows you to limit the amount of time that an individual reaper event will run before it hands control back over to Icinga for other portions of the monitoring logic.

Check Result Path

Format: **check_result_path=<path>**

Example: **check_result_path=/var/spool/icinga/checkresults**

This options determines which directory Icinga will use to temporarily store host and service check results before they are processed. This directory should not be used to store any other files, as Icinga will periodically clean this directory of old file (see the [max_check_result_file_age](#) option for more information).



Note

Make sure that only a single instance of Icinga has access to the check result path. If multiple instances of Icinga have their check result path set to the same directory, you will run into problems with check results being processed (incorrectly) by the wrong instance of Icinga!

Max Check Result File Age

Format: **max_check_result_file_age=<seconds>**

Example: **max_check_result_file_age=3600**

This options determines the maximum age in seconds that Icinga will consider check result files found in the [check_result_path](#) directory to be valid. Check result files that are older than this threshold will be deleted by Icinga and the check results they contain will not be processed. By using a value of zero (0) with this option, Icinga will process all check result files - even if they're older than your hardware :-).

Host Inter-Check Delay Method

Format: **host_inter_check_delay_method=<n/d/s/x.xx>**

Example: **host_inter_check_delay_method=s**

This option allows you to control how host checks *that are scheduled to be checked on a regular basis* are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Icinga to calculate an average check interval and spread initial checks of all hosts out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally *not* recommended. Using no delay will cause all host checks to be scheduled for execution at the same time. More information on how to estimate how the inter-check delay affects host check scheduling can be found [here](#). Values are as follows:

- n = Don't use any delay - schedule all host checks to run immediately (i.e. at the same time!)
- d = Use a "dumb" delay of 1 second between host checks
- s = Use a "smart" delay calculation to spread host checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds

Maximum Host Check Spread

Format: **max_host_check_spread=<minutes>**

Example: **max_host_check_spread=30**

This option determines the maximum number of minutes from when Icinga starts that all hosts (that are scheduled to be regularly checked) are checked. This option will automatically adjust the [host inter-check delay method](#) (if necessary) to ensure that the initial checks of all hosts occur within the timeframe you specify. In general, this option will not have an affect on host check scheduling if scheduling information is being retained using the [use_retained_scheduling_info](#) option. Default value is 30 (minutes).

Timing Interval Length

Format: **interval_length=<seconds>**

Example: **interval_length=60**

This is the number of seconds per "unit interval" used for timing in the scheduling queue, re-notifications, etc. "Units intervals" are used in the object configuration file to determine how often to run a service check, how often to re-notify a contact, etc.

Important: The default value for this is set to 60, which means that a "unit value" of 1 in the object configuration file will mean 60 seconds (1 minute). We have not really tested other values for this variable, so proceed at your own risk if you decide to do so!

Auto-Rescheduling Option

Format: **auto_reschedule_checks=<0/1>**

Example: **auto_reschedule_checks=1**

This option determines whether or not Icinga will attempt to automatically reschedule active host and service checks to "smooth" them out over time. This can help to balance the load on the monitoring server, as it will attempt to keep the time between consecutive checks consistent, at the expense of executing checks on a more rigid schedule.

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THIS OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

Auto-Rescheduling Interval

Format: **auto_rescheduling_interval=<seconds>**

Example: **auto_rescheduling_interval=30**

This option determines how often (in seconds) Icinga will attempt to automatically reschedule checks. This option only has an effect if the [auto_reschedule_checks](#) option is enabled. Default is 30 seconds.

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

Auto-Rescheduling Window

Format: **auto_rescheduling_window=<seconds>**

Example: **auto_rescheduling_window=180**

This option determines the "window" of time (in seconds) that Icinga will look at when automatically rescheduling checks. Only host and service checks that occur in the next X seconds (determined by this variable) will be rescheduled. This option only has an effect if the [auto_reschedule_checks](#) option is enabled. Default is 180 seconds (3 minutes).

WARNING: THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

Aggressive Host Checking Option

Format: **use_aggressive_host_checking=<0/1>**

Example: **use_aggressive_host_checking=0**

Icinga tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Icinga to make some smarter decisions and check hosts a bit faster. Enabling this option will increase the amount of time required to check hosts, but may improve reliability a bit. Unless you have problems with Icinga not recognizing that a host recovered, we would suggest **not** enabling this option.

- 0 = Don't use aggressive host checking (default)
- 1 = Use aggressive host checking

Translate Passive Host Checks Option

Format: **translate_passive_host_checks=<0/1>**

Example: **translate_passive_host_checks=1**

This option determines whether or not Icinga will translate DOWN/UNREACHABLE passive host check results to their "correct" state from the viewpoint of the local Icinga instance. This can be very useful in distributed and failover monitoring installations. More information on passive check state translation can be found [here](#).

- 0 = Disable check translation (default)
- 1 = Enable check translation

Passive Host Checks Are SOFT Option

Format: **passive_host_checks_are_soft=<0/1>**

Example: **passive_host_checks_are_soft=1**

This option determines whether or not Icinga will treat [passive host checks](#) as HARD states or SOFT states. By default, a passive host check result will put a host into a [HARD state type](#). You can change this behavior by enabling this option.

- 0 = Passive host checks are HARD (default)
- 1 = Passive host checks are SOFT

Predictive Host Dependency Checks Option

Format: **enable_predictive_host_dependency_checks=<0/1>**

Example: **enable_predictive_host_dependency_checks=1**

This option determines whether or not Icinga will execute predictive checks of hosts that are being depended upon (as defined in [host dependencies](#)) for a particular host when it changes state. Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found [here](#).

- 0 = Disable predictive checks
- 1 = Enable predictive checks (default)

Predictive Service Dependency Checks Option

Format: **enable_predictive_service_dependency_checks=<0/1>**

Example: **enable_predictive_service_dependency_checks=1**

This option determines whether or not Icinga will execute predictive checks of services that are being depended upon (as defined in [service dependencies](#)) for a particular service when it changes state. Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found [here](#).

- 0 = Disable predictive checks
- 1 = Enable predictive checks (default)

Cached Host Check Horizon

Format: **`cached_host_check_horizon=<seconds>`**

Example: **`cached_host_check_horizon=15`**

This option determines the maximum amount of time (in seconds) that the state of a previous host check is considered current. Cached host states (from host checks that were performed more recently than the time specified by this value) can improve host check performance immensely. Too high of a value for this option may result in (temporarily) inaccurate host states, while a low value may result in a performance hit for host checks. Use a value of 0 if you want to disable host check caching. More information on cached checks can be found [here](#).

Cached Service Check Horizon

Format: **`cached_service_check_horizon=<seconds>`**

Example: **`cached_service_check_horizon=15`**

This option determines the maximum amount of time (in seconds) that the state of a previous service check is considered current. Cached service states (from service checks that were performed more recently than the time specified by this value) can improve service check performance when a lot of [service dependencies](#) are used. Too high of a value for this option may result in inaccuracies in the service dependency logic. Use a value of 0 if you want to disable service check caching. More information on cached checks can be found [here](#).

Large Installation Tweaks Option

Format: **`use_large_installation_tweaks=<0/1>`**

Example: **`use_large_installation_tweaks=0`**

This option determines whether or not the Icinga daemon will take several shortcuts to improve performance. These shortcuts result in the loss of a few features, but larger installations will likely see a lot of benefit from doing so. More information on what optimizations are taken when you enable this option can be found [here](#).

- 0 = Don't use tweaks (default)
- 1 = Use tweaks

Child Process Memory Option

Format: **`free_child_process_memory=<0/1>`**

Example: **`free_child_process_memory=0`**

This option determines whether or not Icinga will free memory in child processes when they are fork()ed off from the main process. By default, Icinga frees memory. However, if the [use_large_installation_tweaks](#) option is enabled, it will not. By defining this option in your configuration file, you are able to override things to get the behavior you want.

- 0 = Don't free memory
- 1 = Free memory

Child Processes Fork Twice

Format: **child_processes_fork_twice=<0/1>**

Example: **child_processes_fork_twice=0**

This option determines whether or not Icinga will fork() child processes twice when it executes host and service checks. By default, Icinga fork()s twice. However, if the [use_large_installation_tweaks](#) option is enabled, it will only fork() once. By defining this option in your configuration file, you are able to override things to get the behavior you want.

- 0 = Fork() just once
- 1 = Fork() twice

Environment Macros Option

Format: **enable_environment_macros=<0/1>**

Example: **enable_environment_macros=0**

This option determines whether or not the Icinga daemon will make all standard [macros](#) available as environment variables to your check, notification, event handler, etc. commands. In large Icinga installations this can be problematic because it takes additional memory and (more importantly) CPU to compute the values of all macros and make them available to the environment.

- 0 = Don't make macros available as environment variables
- 1 = Make macros available as environment variables (default)

Flap Detection Option

Format: **enable_flap_detection=<0/1>**

Example: **enable_flap_detection=0**

This option determines whether or not Icinga will try and detect hosts and services that are "flapping". Flapping occurs when a host or service changes between states too frequently, resulting in a barrage of notifications being sent out. When Icinga detects that a host or service is flapping, it will temporarily suppress notifications for that host/service until it stops flapping. Flap detection is very experimental at this point, so use this feature with caution! More information on how flap detection and handling works can be found [here](#).



Note

If you have [state retention](#) enabled, Icinga will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the [state retention file](#)), *unless* you disable the [use_retained_program_state](#) option. If you want to change this option when state retention is active (and the [use_retained_program_state](#) is enabled), you'll have to use the appropriate [external command](#) or change it via the web interface.

- 0 = Don't enable flap detection (default)
- 1 = Enable flap detection

Low Service Flap Threshold

Format: **low_service_flap_threshold=<percent>**

Example: **low_service_flap_threshold=25.0**

This option is used to set the low threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read [this](#).

High Service Flap Threshold

Format: **high_service_flap_threshold=<percent>**

Example: **high_service_flap_threshold=50.0**

This option is used to set the high threshold for detection of service flapping. For more information on how flap detection and handling works (and how this option affects things) read [this](#).

Low Host Flap Threshold

Format: **low_host_flap_threshold=<percent>**

Example: **low_host_flap_threshold=25.0**

This option is used to set the low threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read [this](#).

High Host Flap Threshold

Format: **high_host_flap_threshold=<percent>**

Example: **high_host_flap_threshold=50.0**

This option is used to set the high threshold for detection of host flapping. For more information on how flap detection and handling works (and how this option affects things) read [this](#).

Soft State Dependencies Option

Format: **soft_state_dependencies=<0/1>**

Example: **soft_state_dependencies=0**

This option determines whether or not Icinga will use soft state information when checking [host](#) and [service dependencies](#). Normally Icinga will only use the latest hard host or service state when checking dependencies. If you want it to use the latest state (regardless of whether its a soft or hard [state type](#)), enable this option.

- 0 = Don't use soft state dependencies (default)
- 1 = Use soft state dependencies

Service Check Timeout

Format: **service_check_timeout=<seconds>**

Example: **service_check_timeout=60**

This is the maximum number of seconds that Icinga will allow service checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each service check normally finishes executing within this time limit. If a service check runs longer than this limit, Icinga will kill it off thinking it is a runaway processes.

Service Check Timeout State

Format: **service_check_timeout_state=<c/u/w/o>**

Example: **service_check_timeout_state=u**

This setting determines the state Icinga will report when a service check times out - that is does not respond within service_check_timeout seconds. This can be useful if a machine is running at too high a load and you do not want to consider a failed service check to be critical. The default was changed to service_check_timeout_state=u in Icinga1.0.1

Host Check Timeout

Format: **host_check_timeout=<seconds>**

Example: **host_check_timeout=60**

This is the maximum number of seconds that Icinga will allow host checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned and the host will be assumed to be DOWN. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each host check normally finishes executing within this time limit. If a host check runs longer than this limit, Icinga will kill it off thinking it is a runaway processes.

Event Handler Timeout

Format: **event_handler_timeout=<seconds>**

Example: **event_handler_timeout=60**

This is the maximum number of seconds that Icinga will allow [event handlers](#) to be run. If an event handler exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each event handler command normally finishes executing within this time limit. If an event handler runs longer than this limit, Icinga will kill it off thinking it is a runaway processes.

Notification Timeout

Format: **notification_timeout=<seconds>**

Example: **notification_timeout=60**

This is the maximum number of seconds that Icinga will allow notification commands to be run. If a notification command exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each notification command finishes executing within this time limit. If a notification command runs longer than this limit, Icinga will kill it off thinking it is a runaway processes.

Obsessive Compulsive Service Processor Timeout

Format: **ocsp_timeout=<seconds>**

Example: **ocsp_timeout=5**

This is the maximum number of seconds that Icinga will allow an [obsessive compulsive service processor command](#) to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Obsessive Compulsive Host Processor Timeout

Format: **ochp_timeout=<seconds>**

Example: **ochp_timeout=5**

This is the maximum number of seconds that Icinga will allow an [obsessive compulsive host processor command](#) to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Performance Data Processor Command Timeout

Format: **perfdta_timeout=<seconds>**

Example: **perfdta_timeout=5**

This is the maximum number of seconds that Icinga will allow a [host performance data processor command](#) or [service performance data processor command](#) to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Obsess Over Services Option

Format: **obsess_over_services=<0/1>**

Example: **obsess_over_services=1**

This value determines whether or not Icinga will "obsess" over service checks results and run the **obsessive compulsive service processor command** you define. We know - funny name, but it was all Ethan could think of. This option is useful for performing **distributed monitoring**. If you're not doing distributed monitoring, don't enable this option.

- 0 = Don't obsess over services (default)
- 1 = Obsess over services

Obsessive Compulsive Service Processor Command

Format: **ocsp_command=<command>**

Example: **ocsp_command=obsessive_service_handler**

This option allows you to specify a command to be run after *every* service check, which can be useful in **distributed monitoring**. This command is executed after any **event handler** or **notification** commands. The *command* argument is the short name of a **command definition** that you define in your object configuration file. The maximum amount of time that this command can run is controlled by the **ocsp_timeout** option. More information on distributed monitoring can be found [here](#). This command is only executed if the **obsess_over_services** option is enabled globally and if the *obsess_over_service* directive in the **service definition** is enabled.

Obsess Over Hosts Option

Format: **obsess_over_hosts=<0/1>**

Example: **obsess_over_hosts=1**

This value determines whether or not Icinga will "obsess" over host checks results and run the **obsessive compulsive host processor command** you define. We know - funny name, but it was all Ethan could think of. This option is useful for performing **distributed monitoring**. If you're not doing distributed monitoring, don't enable this option.

- 0 = Don't obsess over hosts (default)
- 1 = Obsess over hosts

Obsessive Compulsive Host Processor Command

Format: **ochp_command=<command>**

Example: **ochp_command=obsessive_host_handler**

This option allows you to specify a command to be run after *every* host check, which can be useful in **distributed monitoring**. This command is executed after any **event handler** or **notification** commands. The *command* argument is the short name of a **command definition** that you define in your object configuration file. The maximum amount of time that this command can run is controlled by the **ochp_timeout** option. More information on distributed monitoring

can be found [here](#). This command is only executed if the `obsess_over_hosts` option is enabled globally and if the `obsess_over_host` directive in the [host definition](#) is enabled.

Performance Data Processing Option

Format: `process_performance_data=<0/1>`

Example: `process_performance_data=1`

This value determines whether or not Icinga will process host and service check [performance data](#).

- 0 = Don't process performance data (default)
- 1 = Process performance data

Host Performance Data Processing Command

Format: `host_perfdata_command=<command>`

Example: `host_perfdata_command=process-host-perfdata`

This option allows you to specify a command to be run after *every* host check to process host [performance data](#) that may be returned from the check. The `command` argument is the short name of a [command definition](#) that you define in your object configuration file. This command is only executed if the `process_performance_data` option is enabled globally and if the `process_perf_data` directive in the [host definition](#) is enabled.

Service Performance Data Processing Command

Format: `service_perfdata_command=<command>`

Example: `service_perfdata_command=process-service-perfdata`

This option allows you to specify a command to be run after *every* service check to process service [performance data](#) that may be returned from the check. The `command` argument is the short name of a [command definition](#) that you define in your object configuration file. This command is only executed if the `process_performance_data` option is enabled globally and if the `process_perf_data` directive in the [service definition](#) is enabled.

Host Performance Data File

Format: `host_perfdata_file=<file_name>`

Example: `host_perfdata_file=/usr/local/icinga/var/host-perfdata.dat`

This option allows you to specify a file to which host [performance data](#) will be written after every host check. Data will be written to the performance file as specified by the `host_perfdata_file_template` option. Performance data is only written to this file if the `process_performance_data` option is enabled globally and if the `process_perf_data` directive in the [host definition](#) is enabled.

Service Performance Data File

Format: **service_perfdata_file=<file_name>**

Example: **service_perfdata_file=/usr/local/icinga/var/service-perfdata.dat**

This option allows you to specify a file to which service [performance data](#) will be written after every service check. Data will be written to the performance file as specified by the [service_perfdata_file_template](#) option. Performance data is only written to this file if the [process_performance_data](#) option is enabled globally and if the [process_perf_data](#) directive in the [service definition](#) is enabled.

Host Performance Data File Template

Format: **host_perfdata_file_template=<template>**

Example: **host_perfdata_file_template=[HOSTPERFDATA]\t\$TIMET\$\t\$HOSTNAME\$\\t\$HOSTEXECUTIONTIME\$\t\$HOSTOUTPUT\$\t\$HOSTPERFDATA\$**

This option determines what (and how) data is written to the [host performance data file](#). The template may contain [macros](#), special characters (\t for tab, \r for carriage return, \n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Service Performance Data File Template

Format: **service_perfdata_file_template=<template>**

Example: **service_perfdata_file_template=[SERVICEPERFDATA]\t\$TIMET\$\t\$HOSTNAME\$\\t\$SERVICEDESC\$\t\$SERVICEEXECUTIONTIME\$\t\$SERVICELATENCY\$\\t\$SERVICEOUTPUT\$\t\$SERVICEPERFDATA\$**

This option determines what (and how) data is written to the [service performance data file](#). The template may contain [macros](#), special characters (\t for tab, \r for carriage return, \n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Host Performance Data File Mode

Format: **host_perfdata_file_mode=<mode>**

Example: **host_perfdata_file_mode=a**

This option determines how the [host performance data file](#) is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

- a = Open file in append mode (default)
- w = Open file in write mode
- p = Open in non-blocking read/write mode (useful when writing to pipes)

Service Performance Data File Mode

Format: **service_perfdata_file_mode=<mode>**

Example: **service_perfdata_file_mode=a**

This option determines how the [service performance data file](#) is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

- a = Open file in append mode (default)
- w = Open file in write mode
- p = Open in non-blocking read/write mode (useful when writing to pipes)

Host Performance Data File Processing Interval

Format: **host_perfdata_file_processing_interval=<seconds>**

Example: **host_perfdata_file_processing_interval=0**

This option allows you to specify the interval (in seconds) at which the [host performance data file](#) is processed using the [host performance data file processing command](#). A value of 0 indicates that the performance data file should not be processed at regular intervals.

Service Performance Data File Processing Interval

Format: **service_perfdata_file_processing_interval=<seconds>**

Example: **service_perfdata_file_processing_interval=0**

This option allows you to specify the interval (in seconds) at which the [service performance data file](#) is processed using the [service performance data file processing command](#). A value of 0 indicates that the performance data file should not be processed at regular intervals.

Host Performance Data File Processing Command

Format: **host_perfdata_file_processing_command=<command>**

Example: **host_perfdata_file_processing_command=process-host-perfdata-file**

This option allows you to specify the command that should be executed to process the [host performance data file](#). The *command* argument is the short name of a [command definition](#) that you define in your object configuration file. The interval at which this command is executed is determined by the [host_perfdata_file_processing_interval](#) directive.

Service Performance Data File Processing Command

Format: **service_perfdata_file_processing_command=<command>**

Example: **service_perfdata_file_processing_command=process-service-perfdata-file**

This option allows you to specify the command that should be executed to process the [service performance data file](#). The *command* argument is the short name of a [command definition](#) that you define in your object configuration file. The interval at which this command is executed is determined by the [service_perfdata_file_processing_interval](#) directive.

Orphaned Service Check Option

Format: **check_for_orphaned_services=<0/1>**

Example: **check_for_orphaned_services=1**

This option allows you to enable or disable checks for orphaned service checks. Orphaned service checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the service, it is not rescheduled in the event queue. This can cause service checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a service check. If this option is enabled and Icinga finds that results for a particular service check have not come back, it will log an error message and reschedule the service check. If you start seeing service checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned services.

- 0 = Don't check for orphaned service checks
- 1 = Check for orphaned service checks (default)

Orphaned Host Check Option

Format: **check_for_orphaned_hosts=<0/1>**

Example: **check_for_orphaned_hosts=1**

This option allows you to enable or disable checks for orphaned host checks. Orphaned host checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the host, it is not rescheduled in the event queue. This can cause host checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a host check. If this option is enabled and Icinga finds that results for a particular host check have not come back, it will log an error message and reschedule the host check. If you start seeing host checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned hosts.

- 0 = Don't check for orphaned host checks
- 1 = Check for orphaned host checks (default)

Service Freshness Checking Option

Format: **check_service_freshness=<0/1>**

Example: **check_service_freshness=0**

This option determines whether or not Icinga will periodically check the "freshness" of service checks. Enabling this option is useful for helping to ensure that [passive service checks](#) are received in a timely manner. More information on freshness checking can be found [here](#).

- 0 = Don't check service freshness
- 1 = Check service freshness (default)

Service Freshness Check Interval

Format: `service_freshness_check_interval=<seconds>`

Example: `service_freshness_check_interval=60`

This setting determines how often (in seconds) Icinga will periodically check the "freshness" of service check results. If you have disabled service freshness checking (with the `check_service_freshness` option), this option has no effect. More information on freshness checking can be found [here](#).

Host Freshness Checking Option

Format: `check_host_freshness=<0/1>`

Example: `check_host_freshness=0`

This option determines whether or not Icinga will periodically check the "freshness" of host checks. Enabling this option is useful for helping to ensure that `passive host checks` are received in a timely manner. More information on freshness checking can be found [here](#).

- 0 = Don't check host freshness (default)
- 1 = Check host freshness

Host Freshness Check Interval

Format: `host_freshness_check_interval=<seconds>`

Example: `host_freshness_check_interval=60`

This setting determines how often (in seconds) Icinga will periodically check the "freshness" of host check results. If you have disabled host freshness checking (with the `check_host_freshness` option), this option has no effect. More information on freshness checking can be found [here](#).

Additional Freshness Threshold Latency Option

Format: `additional_freshness_latency=<#>`

Example: `additional_freshness_latency=15`

This option determines the number of seconds Icinga will add to any host or services freshness threshold it automatically calculates (e.g. those not specified explicitly by the user). More information on freshness checking can be found [here](#).

Embedded Perl Interpreter Option

Format: `enable_embedded_perl=<0/1>`

Example: `enable_embedded_perl=1`

This setting determines whether or not the embedded Perl interpreter is enabled on a program-wide basis. Icinga must be compiled with support for embedded Perl for this option to have an effect. More information on the embedded Perl interpreter can be found [here](#).

Embedded Perl Implicit Use Option

Format: **use_embedded_perl_implicitly=<0/1>**

Example: **use_embedded_perl_implicitly=1**

This setting determines whether or not the embedded Perl interpreter should be used for Perl plugins/scripts that do not explicitly enable/disable it. Icinga must be compiled with support for embedded Perl for this option to have an effect. More information on the embedded Perl interpreter and the effect of this setting can be found [here](#).

Date Format

Format: **date_format=<option>**

Example: **date_format=us**

This option allows you to specify what kind of date/time format Icinga should use in the web interface and date/time [macros](#). Possible options (along with example output) include:

Option	Output Format	Sample Output
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

Timezone Option

Format: **use_timezone=<tz>**

Example: **use_timezone=US/Mountain**

This option allows you to override the default timezone that this instance of Icinga runs in. Useful if you have multiple instances of Icinga that need to run from the same server, but have different local times associated with them. If not specified, Icinga will use the system configured timezone.



Note

If you use this option to specify a custom timezone, you will also need to alter the Apache configuration directives for the CGIs to specify the timezone you want.

Example:

```
<Directory "/usr/local/icinga/sbin/">
```

```
SetEnv TZ "US/Mountain"
```

```
...
```

</Directory>

Illegal Object Name Characters

Format: **illegal_object_name_chars=<chars...>**

Example: **illegal_object_name_chars='~!\$%^&*!"|'<>?,0=**

This option allows you to specify illegal characters that cannot be used in host names, service descriptions, or names of other object types. Icinga will allow you to use most characters in object definitions, but we recommend not using the characters shown in the example above. Doing so may give you problems in the web interface, notification commands, etc.

Illegal Macro Output Characters

Format: **illegal_macro_output_chars=<chars...>**

Example: **illegal_macro_output_chars='~\$%^&"|'<>**

This option allows you to specify illegal characters that should be stripped from [macros](#) before being used in notifications, event handlers, and other commands. This ALSO affects macros used in service or host check commands. You can choose to not strip out the characters shown in the example above, but we recommend you do not do this. Some of these characters are interpreted by the shell (i.e. the backtick) and can lead to security problems. The following macros are stripped of the characters you specify:

**\$HOSTOUTPUT\$, \$HOSTPERFDATA\$, \$HOSTACKAUTHOR\$, \$HOSTACKCOMMENT\$,
\$SERVICEOUTPUT\$, \$SERVICEPERFDATA\$, \$SERVICEACKAUTHOR\$, and
\$SERVICEACKCOMMENT\$**

Regular Expression Matching Option

Format: **use_regex_matching=<0/1>**

Example: **use_regex_matching=0**

This option determines whether or not various directives in your [object definitions](#) will be processed as regular expressions. More information on how this works can be found [here](#).

- 0 = Don't use regular expression matching (default)
- 1 = Use regular expression matching

True Regular Expression Matching Option

Format: **use_true_regex_matching=<0/1>**

Example: **use_true_regex_matching=0**

If you've enabled regular expression matching of various object directives using the [use_regex_matching](#) option, this option will determine when object directives are treated as regular expressions. If this option is disabled (the default), directives will only be treated as regular expressions if they contain *, ?, +, or \.. If this option is enabled, all appropriate directives will be treated as regular expression - be careful when enabling this! More information on how this works can be found [here](#).

- 0 = Don't use true regular expression matching (default)
- 1 = Use true regular expression matching

Administrator Email Address

Format: **admin_email=<email_address>**

Example: **admin_email=root@localhost.localdomain**

This is the email address for the administrator of the local machine (i.e. the one that Icinga is running on). This value can be used in notification commands by using the **\$ADMINEMAIL\$** macro.

Administrator Pager

Format: **admin_pager=<pager_number_or_pager_email_gateway>**

Example: **admin_pager=pageroot@localhost.localdomain**

This is the pager number (or pager email gateway) for the administrator of the local machine (i.e. the one that Icinga is running on). The pager number/address can be used in notification commands by using the **\$ADMINPAGER\$** macro.

Event Broker Options

Format: **event_broker_options=<#>**

Example: **event_broker_options=-1**

This option controls what (if any) data gets sent to the event broker and, in turn, to any loaded event broker modules. This is an advanced option. When in doubt, either broker nothing (if not using event broker modules) or broker everything (if using event broker modules). Possible values are shown below.

- 0 = Broker nothing
- -1 = Broker everything
- # = See BROKER_* definitions in source code (include/broker.h) for other values that can be OR'ed together

Event Broker Modules

Format: **broker_module=<modulepath> [moduleargs]**

Example: **broker_module=/usr/local/icinga/bin/idomod.o \\\n cfg_file=/usr/local/icinga/etc/idomod.cfg**

This directive is used to specify an event broker module that should be loaded by Icinga at startup. Use multiple directives if you want to load more than one module. Arguments that should be passed to the module at startup are separated from the module path by a space.

!!! WARNING !!!

Do NOT overwrite modules while they are being used by Icinga or Icinga will crash in a fiery display of SEGFAULT glory. This is a bug/limitation either in `dlopen()`, the kernel, and/or the filesystem. And maybe Icinga...

The correct/safe way of updating a module is by using one of these methods:

1. Shutdown Icinga, replace the module file, restart Icinga
2. While Icinga is running... delete the original module file, move the new module file into place, restart Icinga

Debug File

Format: **`debug_file=<file_name>`**

Example: **`debug_file=/usr/local/icinga/var/icinga.debug`**

This option determines where Icinga should write debugging information. What (if any) information is written is determined by the `debug_level` and `debug_verbosity` options. You can have Icinga automatically rotate the debug file when it reaches a certain size by using the `max_debug_file_size` option.

Debug Level

Format: **`debug_level=<#>`**

Example: **`debug_level=24`**

This option determines what type of information Icinga should write to the `debug_file`. This value is a logical OR of the values below.

- -1 = Log everything
- 0 = Log nothing (default)
- 1 = Function enter/exit information
- 2 = Config information
- 4 = Process information
- 8 = Scheduled event information
- 16 = Host/service check information
- 32 = Notification information
- 64 = Event broker information

Debug Verbosity

Format: **`debug_verbosity=<#>`**

Example: **`debug_verbosity=1`**

This option determines how much debugging information Icinga should write to the [debug file](#).

- 0 = Basic information
- 1 = More detailed information (default)
- 2 = Highly detailed information

Maximum Debug File Size

Format: **max_debug_file_size=<#>**

Example: **max_debug_file_size=1000000**

This option determines the maximum size (in bytes) of the [debug file](#). If the file grows larger than this size, it will be renamed with a .old extension. If a file already exists with a .old extension it will automatically be deleted. This helps ensure your disk space usage doesn't get out of control when debugging Icinga.

Allow Empty Hostgroup Assignment

Format: **allow_empty_hostgroup_assignment=<0|1>**

Example: **allow_empty_hostgroup_assignment=1**

This boolean option determines whether services assigned to empty host groups (host groups with no host members) will cause Icinga to exit with error on startup (or during a configuration check) or not. The default behaviour if the option is not present (or set to "0") in the main configuration file is for Icinga to exit with error if services are associated with host groups whose hostgroup definitions have no host_members associated with them.



Note

This option is available starting with Icinga 1.3.

Process empty performance results

Format: **host_perfdata_process_empty_results=<0|1>**
service_perfdata_process_empty_results=<0|1>

Example: **host_perfdata_process_empty_results=1**
service_perfdata_process_empty_results=1

These options determine whether the core will process empty perfdata results or not. This is needed for distributed monitoring, and intentionally turned on by default. If you don't require empty perfdata - saving some cpu cycles on unwanted macro calculation - you can turn that off. Be careful! Values: 1 = enable, 0 = disable



Note

These options are available starting with Icinga 1.4

[Prev](#)

[Up](#)

[Next](#)

[Configuration Overview](#)

[Home](#)

[Object Configuration Overview](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Object Configuration Overview

[Prev](#)

[Chapter 3. Configuring Icinga](#)

[Next](#)

Object Configuration Overview

What Are Objects?

Objects are all the elements that are involved in the monitoring and notification logic. Types of objects include:

- Services
- Service Groups
- Hosts
- Host Groups
- Contacts
- Contact Groups
- Commands
- Time Periods
- Notification Escalations
- Notification and Execution Dependencies

More information on what objects are and how they relate to each other can be found below.

Where Are Objects Defined?

Objects can be defined in one or more configuration files and/or directories that you specify using the [cfg_file](#) and/or [cfg_dir](#) directives in the main configuration file.

include_file / include_dir

An object definition file can include other object definition files with the use of the `include_file=<file_name>` and `include_dir=<directory_name>` directives. The former includes the single file specified, the latter will process all files ending in the `.cfg` extension in the directory specified. These directives may be repeated to include multiple files/directories.

The directives are not allowed within the actual definition of an object, and should appear before, after, or in between any object definitions. They are closely related to the [cfg_file=](#) and [cfg_dir=](#) directives in the main configuration file.

These directives may be chained; e.g. an object definition file included from the main configuration file with a [cfg_file=](#) or [cfg_dir=](#) directive can use [include_file=](#) or [include_dir=](#) to include another object definition file, which in turn can also use [include_file=](#) or [include_dir=](#) to include yet another object definition file, and so on.



Tip

When you follow [quickstart installation guide](#), several sample object configuration files are placed in `/usr/local/icinga/etc/objects/`. You can use these sample files to see how object inheritance works and learn how to define your own object definitions.

How Are Objects Defined?

Objects are defined in a flexible template format, which can make it much easier to manage your Icinga configuration in the long term. Basic information on how to define objects in your configuration files can be found [here](#).

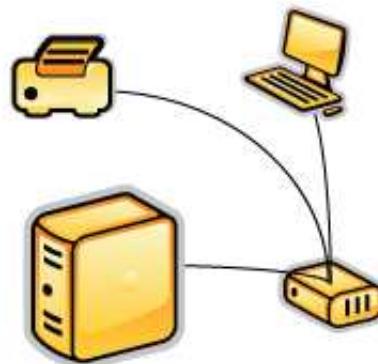
Once you get familiar with the basics of how to define objects, you should read up on [object inheritance](#), as it will make your configuration more robust for the future. Seasoned users can exploit some advanced features of object definitions as described in the documentation on [object tricks](#).

Objects Explained

Some of the main object types are explained in greater detail below...

Hosts are one of the central objects in the monitoring logic. Important attributes of hosts are as follows:

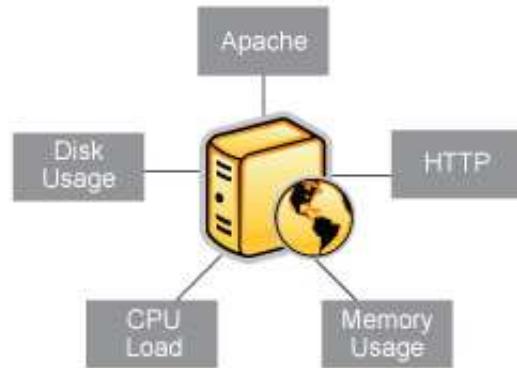
- Hosts are usually physical devices on your network (servers, workstations, routers, switches, printers, etc).
- Hosts have an address of some kind (e.g. an IP or MAC address).
- Hosts have one or more services associated with them.
- Hosts can have parent/child relationships with other hosts, often representing real-world network connections, which is used in the [network reachability](#) logic.



Host Groups are groups of one or more hosts. Host groups can make it easier to (1) view the status of related hosts in the Icinga web interface and (2) simplify your configuration through the use of [object tricks](#).

Services are one of the central objects in the monitoring logic. Services are associated with hosts and can be:

- Attributes of a host (CPU load, disk usage, uptime, etc.)
- Services provided by the host (HTTP, POP3, FTP, SSH, etc.)
- Other things associated with the host (DNS records, etc.)



Service Groups are groups of one or more services. Service groups can make it easier to (1) view the status of related services in the Icinga web interface and (2) simplify your configuration through the use of [object tricks](#).

Contacts are people involved in the notification process:

- Contacts have one or more notification methods (cellphone, pager, email, instant messaging, etc.)
- Contacts receive notifications for hosts and service they are responsible for



Contact Groups are groups of one or more contacts. Contact groups can make it easier to define all the people who get notified when certain host or service problems occur.

Timeperiods are used to control:

- When hosts and services can be monitored
- When contacts can receive notifications



Information on how timeperiods work can be found [here](#).

Commands are used to tell Icinga what programs, scripts, etc. it should execute to perform:



- Host and service checks
- Notifications
- Event handlers
- and more...

[Prev](#)

[Up](#)

[Next](#)

[Main Configuration File Options](#)

[Home](#)

[Object Definitions](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Object Definitions

[Prev](#)
[Chapter 3. Configuring Icinga](#)
[Next](#)

Object Definitions

Introduction

One of the features of Icinga's object configuration format is that you can create object definitions that inherit properties from other object definitions. An explanation of how object inheritance works can be found [here](#). We strongly suggest that you familiarise yourself with object inheritance once you read over the documentation presented below, as it will make the job of creating and maintaining object definitions much easier than it otherwise would be. Also, read up on the [object tricks](#) that offer shortcuts for otherwise tedious configuration tasks.



Note

When creating and/or editing configuration files, keep the following in mind:

1. Lines that start with a '#' character are taken to be comments and are not processed
2. Directive names are case-sensitive
3. Characters that appear after a semicolon (;) in configuration lines are treated as comments and are not processed

Retention Notes

It is important to point out that several directives in host, service, and contact definitions may not be picked up by Icinga when you change them in your configuration files. Object directives that can exhibit this behaviour are marked with an asterisk (*). The reason for this behaviour is due to the fact that Icinga chooses to honour values stored in the [state retention file](#) over values found in the config files, assuming you have [state retention](#) enabled on a program-wide basis and the value of the directive is changed during runtime with an [external command](#).

One way to get around this problem is to disable the retention of non-status information using the `retain_nonstatus_information` directive in the host, service, and contact definitions. Disabling this directive will cause Icinga to take the initial values for these directives from your config files, rather than from the state retention file when it (re)starts.

Sample Configuration Files

**Note**

Sample object configuration files are installed in the `/usr/local/icinga/etc/` directory when you follow the [quickstart installation guide](#).

Object Types[Host definitions](#)[Host group definitions](#)[Service definitions](#)[Service group definitions](#)[Contact definitions](#)[Contact group definitions](#)[Time period definitions](#)[Command definitions](#)[Service dependency definitions](#)[Service escalation definitions](#)[Host dependency definitions](#)[Host escalation definitions](#)[Extended host information definitions](#)[Extended service information definitions](#)[Module definitions](#)**Host definition****Host definition***Description:*

A host definition is used to define a physical server, workstation, device, etc. that resides on your network.

Definition Format:**Note**

"**Directives**" are required while the others are optional.

```
define host{
    host_name          host_name
    alias              alias
```

display_name	display_name
address	address
address6	address6
parents	host_names
hostgroups	hostgroup_names
check_command	command_name
initial_state	[o,d,u]
max_check_attempts	#
check_interval	#
retry_interval	#
active_checks_enabled	[0/1]
passive_checks_enabled	[0/1]
check_period	timeperiod_name
obsess_over_host	[0/1]
check_freshness	[0/1]
freshness_threshold	#
event_handler	command_name
event_handler_enabled	[0/1]
low_flap_threshold	#
high_flap_threshold	#
flap_detection_enabled	[0/1]
flap_detection_options	[o,d,u]
process_perf_data	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
contacts	contacts
contact_groups	contact_groups
notification_interval	#
first_notification_delay	#
notification_period	timeperiod_name
notification_options	[d,u,r,f,s]
notifications_enabled	[0/1]
stalking_options	[o,d,u]

```

notes           note_string
notes_url      url
action_url     url
icon_image     image_file
icon_image_alt alt_string
vrml_image     image_file
statusmap_image image_file
2d_coords      x_coord,y_coord
3d_coords      x_coord,y_coord,z_coord
}

```

Example Definition:

```

define host{
    host_name          bogus-router
    alias              Bogus Router #1
    address            192.168.1.254
    parents            server-backbone
    check_command      check-host-alive
    check_interval     5
    retry_interval     1
    max_check_attempts 5
    check_period       24x7
    process_perf_data 0
    retain_nonstatus_information 0
    contact_groups    router-admins
    notification_interval 30
    notification_period 24x7
    notification_options d,u,r
}

```

Directive Descriptions:

host_name:

This directive is used to define a short name used to identify the host. It is used in host group and service definitions to reference this particular host. Hosts can have multiple services (which are monitored) associated with them. When used properly, the \$HOSTNAME\$ [macro](#) will contain this short name.

alias:

This directive is used to define a longer name or description used to identify the host. It is provided in order to allow you to more easily identify a particular host. When used properly, the \$HOSTALIAS\$ [macro](#) will contain this alias/description.

address:

This directive is used to define the address of the host. Normally, this is an IP address, although it could really be anything you want (so long as it can be used to check the status of the host). You can use a FQDN to identify the host instead of an IP address, but if DNS services are not available this could cause problems. When used properly, the \$HOSTADDRESS\$ [macro](#) will contain this address.

**Note**

If you do not specify an address directive in a host definition, the name of the host will be used as its address. A word of caution about doing this, however - if DNS fails, most of your service checks will fail because the plugins will be unable to resolve the host name.

address6:

This directive is used to define a second address for the host. Normally, this is an IPv6 address, although it could really be anything you want (so long as it can be used to check the status of the host). You can use a FQDN to identify the host instead of an IP address, but if DNS services are not available this could cause problems. When used properly, the \$HOSTADDRESS6\$ [macro](#) will contain this address.

**Note**

If you do not specify an address6 directive in a host definition, the name of the host will be used as its address. A word of caution about doing this, however - if DNS fails, most of your service checks will fail because the plugins will be unable to resolve the host name (available starting with Icinga 1.3).

display_name:

This directive is used to define an alternate name that should be displayed in the web interface for this host. If not specified, this defaults to the value you specify for the `host_name` directive.

**Note**

The CGIs up to Icinga 1.0.1 do not use this option.

parents:

This directive is used to define a comma-delimited list of short names of the "parent" hosts for this particular host. Parent hosts are typically routers, switches, firewalls, etc. that lie between the monitoring host and a remote hosts. A router, switch, etc. which is closest to the remote host is considered to be that host's "parent". Read the "Determining Status and Reachability of Network Hosts" document located [here](#) for more information. If this host is on the same network segment as the host doing the monitoring (without any intermediate routers, etc.) the host is considered to be on the local network and will not have a parent host. Leave this value blank if the host does not have a parent host (i.e. it is on the same segment as the Icinga host). The order in which you specify parent hosts has no effect on how things are monitored.

hostgroups:

This directive is used to identify the *short name(s)* of the [hostgroup\(s\)](#) that the host belongs to. Multiple hostgroups should be separated by commas. This directive may be used as an alternative to (or in addition to) using the `members` directive in [hostgroup](#) definitions.

check_command:

This directive is used to specify the *short name* of the [command](#) that should be used to check if the host is up or down. Typically, this command would try and ping the host to see if it is "alive". The command must return a status of OK (0) or Icinga will assume the host is down. If you leave this argument blank, the host will *not* be actively checked. Thus, Icinga will likely always assume the host is up (it may show up as being in a "PENDING" state in the web interface). This is useful if you are monitoring printers or other devices that are frequently turned off. The maximum amount of time that the notification command can run is controlled by the [host_check_timeout](#) option.

initial_state:

By default Icinga will assume that all hosts are in UP states when it starts. You can override the initial state for a host by using this directive. Valid options are: **o** = UP, **d** = DOWN, and **u** = UNREACHABLE.

max_check_attempts:

This directive is used to define the number of times that Icinga will retry the host check command if it returns any state other than an OK state. Setting this value to 1 will cause Icinga to generate an alert without retrying the host check again.

**Note**

If you do not want to check the status of the host, you must still set this to a minimum value of 1. To bypass the host check, just leave the *check_command* option blank.

check_interval:

This directive is used to define the number of "time units" between regularly scheduled checks of the host. Unless you've changed the [interval_length](#) directive from the default value of 60, this number will mean minutes. More information on this value can be found in the [check scheduling](#) documentation.

retry_interval:

This directive is used to define the number of "time units" to wait before scheduling a re-check of the hosts. Hosts are rescheduled at the retry interval when they have changed to a non-UP state. Once the host has been retried **max_check_attempts** times without a change in its status, it will revert to being scheduled at its "normal" rate as defined by the **check_interval** value. Unless you've changed the [interval_length](#) directive from the default value of 60, this number will mean minutes. More information on this value can be found in the [check scheduling](#) documentation.

active_checks_enabled *:

This directive is used to determine whether or not active checks (either regularly scheduled or on-demand) of this host are enabled. Values: 0 = disable active host checks, 1 = enable active host checks.

passive_checks_enabled *:

This directive is used to determine whether or not passive checks are enabled for this host. Values: 0 = disable passive host checks, 1 = enable passive host checks.

check_period:

This directive is used to specify the short name of the [time period](#) during which active checks of this host can be made.

obsess_over_host *:

This directive determines whether or not checks for the host will be "obsessed" over using the [ochp_command](#).

check_freshness *:

This directive is used to determine whether or not [freshness checks](#) are enabled for this host. Values: 0 = disable freshness checks, 1 = enable freshness checks.

freshness_threshold:	This directive is used to specify the freshness threshold (in seconds) for this host. If you set this directive to a value of 0, Icinga will determine a freshness threshold to use automatically.
event_handler:	This directive is used to specify the <i>short name</i> of the command that should be run whenever a change in the state of the host is detected (i.e. whenever it goes down or recovers). Read the documentation on event handlers for a more detailed explanation of how to write scripts for handling events. The maximum amount of time that the event handler command can run is controlled by the event_handler_timeout option.
event_handler_enabled *:	This directive is used to determine whether or not the event handler for this host is enabled. Values: 0 = disable host event handler, 1 = enable host event handler.
low_flap_threshold:	This directive is used to specify the low state change threshold used in flap detection for this host. More information on flap detection can be found here . If you set this directive to a value of 0, the program-wide value specified by the low_host_flap_threshold directive will be used.
high_flap_threshold:	This directive is used to specify the high state change threshold used in flap detection for this host. More information on flap detection can be found here . If you set this directive to a value of 0, the program-wide value specified by the high_host_flap_threshold directive will be used.
flap_detection_enabled *:	This directive is used to determine whether or not flap detection is enabled for this host. More information on flap detection can be found here . Values: 0 = disable host flap detection, 1 = enable host flap detection.
flap_detection_options:	This directive is used to determine what host states the flap detection logic will use for this host. Valid options are a combination of one or more of the following: o = UP states, d = DOWN states, u = UNREACHABLE states.
process_perf_data *:	This directive is used to determine whether or not the processing of performance data is enabled for this host. Values: 0 = disable performance data processing, 1 = enable performance data processing.
retain_status_information:	This directive is used to determine whether or not status-related information about the host is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable status information retention, 1 = enable status information retention.
retain_nonstatus_information:	This directive is used to determine whether or not non-status information about the host is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable non-status information retention, 1 = enable non-status information retention.
contacts:	This is a list of the <i>short names</i> of the contacts that should be notified whenever there are problems (or recoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure contact groups . You must specify at least one contact or contact group in each host definition.

contact_groups:	This is a list of the <i>short names</i> of the contact groups that should be notified whenever there are problems (or recoveries) with this host. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host definition.
notification_interval:	This directive is used to define the number of "time units" to wait before re-notifying a contact that this service is <i>still</i> down or unreachable. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. If you set this value to 0, Icinga will <i>not</i> re-notify contacts about problems for this host - only one problem notification will be sent out.
first_notification_delay:	This directive is used to define the number of "time units" to wait before sending out the first problem notification when this host enters a non-UP state. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. If you set this value to 0, Icinga will start sending out notifications immediately.
notification_period:	This directive is used to specify the short name of the time period during which notifications of events for this host can be sent out to contacts. If a host goes down, becomes unreachable, or recoveries during a time which is not covered by the time period, no notifications will be sent out.
notification_options:	This directive is used to determine when notifications for the host should be sent out. Valid options are a combination of one or more of the following: d = send notifications on a DOWN state, u = send notifications on an UNREACHABLE state, r = send notifications on recoveries (OK state), f = send notifications when the host starts and stops flapping , and s = send notifications when scheduled downtime starts and ends. If you specify n (none) as an option, no host notifications will be sent out. If you do not specify any notification options, Icinga will assume that you want notifications to be sent out for all possible states. Example: If you specify d,r in this field, notifications will only be sent out when the host goes DOWN and when it recovers from a DOWN state.
notifications_enabled *:	This directive is used to determine whether or not notifications for this host are enabled. Values: 0 = disable host notifications, 1 = enable host notifications.
stalking_options:	This directive determines which host states "stalking" is enabled for. Valid options are a combination of one or more of the following: o = stalk on UP states, d = stalk on DOWN states, and u = stalk on UNREACHABLE states. More information on state stalking can be found here .
notes:	This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified host).
notes_url:	This variable is used to define an optional URL that can be used to provide more information about the host. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing host information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>). This can be very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.

action_url:

This directive is used to define an optional URL that can be used to provide more actions to be performed on the host. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing host information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. /cgi-bin/icinga/).

**Note**

Since Icinga 1.0.2 you can use multiple urls for action_url and notes_url for host and service object definitions. The syntax is as follows:

```
notes_url|action_url 'firstURL' 'secondURL' 'thirdURL'  
notes_url|action_url onlyoneURL
```

Please keep in mind that using multiple urls also mean multiple icon images. Those are hardcoded and e.g. action|notes.gif changes to 1-action|1-notes.gif and to 2-action|2-notes.gif and so on - make sure you'll have them in place when using multiple action_url|notes_url. If you are using multiple urls with different icons (1-action.gif e.g.) you can still define the last urls without single quotes. It will then be used like a single url and is referring to the normal icon (action.gif e.g.)

icon_image:

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the various places in the CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. /usr/local/icinga/share/images/logos).

icon_image_alt:

This variable is used to define an optional string that is used in the ALT tag of the image specified by the <icon_image> argument.

vrm1_image:

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host in the **statuswrl** CGI. Unlike the image you use for the <icon_image> variable, this one should probably *not* have any transparency. If it does, the host object will look a bit weird. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. /usr/local/icinga/share/images/logos).

**Note**

This directive has no effect in Icinga

statusmap_image:

This variable is used to define the name of an image that should be associated with this host in the [statusmap](#) CGI. You can specify a JPEG, PNG, and GIF image if you want, although we would strongly suggest using a GD2 format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the [pngtogd2](#) utility supplied with Thomas Boutell's [gd library](#). The GD2 images should be created in *uncompressed* format in order to minimize CPU load when the statusmap CGI is generating the network map image. The image will look best if it is 40x40 pixels in size. You can leave these option blank if you are not using the statusmap CGI. Images for hosts are assumed to be in the [logos/](#) subdirectory in your HTML images directory (i.e. `/usr/local/icinga/share/images/logos`).

2d_coords:

This variable is used to define coordinates to use when drawing the host in the [statusmap](#) CGI. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host icon that is drawn.

**Note**

Don't worry about what the maximum x and y coordinates that you can use are. The CGI will automatically calculate the maximum dimensions of the image it creates based on the largest x and y coordinates you specify.

3d_coords:

This variable is used to define coordinates to use when drawing the host in the [statuswrl](#) CGI. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.

**Note**

This directive has no effect in Icinga.

Hostgroup Definition

Host Group Definition

Description:

A host group definition is used to group one or more hosts together for simplifying configuration with [object tricks](#) or display purposes in the [CGIs](#).

Definition Format:

**Note**

"[Directives](#)" are required while the others are optional.

```
define hostgroup{
    hostgroup_name      hostgroup_name
    alias               alias
    members             hosts
    hostgroup_members   hostgroups
    notes               note_string
    notes_url           url
    action_url          url
}
```

Example Definition:

```
define hostgroup{
    hostgroup_name      novell-servers
    alias               Novell Servers
    members             netware1,netware2,netware3,netware4
}
```

Directive Descriptions:

hostgroup_name:	This directive is used to define a short name used to identify the host group.
alias:	This directive is used to define a longer name or description used to identify the host group. It is provided in order to allow you to more easily identify a particular host group.
members:	This is a list of the <i>short names</i> of hosts that should be included in this group. Multiple host names should be separated by commas. This directive may be used as an alternative to (or in addition to) the <i>hostgroups</i> directive in host definitions .
hostgroup_members:	This optional directive can be used to include hosts from other "sub" host groups in this host group. Specify a comma-delimited list of short names of other host groups whose members should be included in this group.
notes:	This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see it in the extended information CGI (when you are viewing information about the specified host).
notes_url:	This variable is used to define an optional URL that can be used to provide more information about the host group. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing hostgroup information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>). This can be very useful if you want to make detailed information on the host group, emergency contact methods, etc. available to other support staff.
action_url:	This directive is used to define an optional URL that can be used to provide more actions to be performed on the host group. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing hostgroup information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>).

Service Definition

Service Definition

Description:

A service definition is used to identify a "service" that runs on a host. The term "service" is used very loosely. It can mean an actual service that runs on the host (POP, SMTP, HTTP, etc.) or some other type of metric associated with the host (response to a ping, number of logged in users, free disk space, etc.). The different arguments to a service definition are outlined below.

Definition Format:



Note

"[Directives](#)" are required while the others are optional.

```

define service{
    host_name                      host_name
    hostgroup_name                  hostgroup_name
    service_description              service_description
    display_name                    display_name
    servicegroups                   servicegroup_names
    is_volatile                     [0|1|2]
    check_command                   command_name
    initial_state                   [o,w,u,c]
    max_check_attempts               #
    check_interval                  #
    retry_interval                  #
    active_checks_enabled           [0/1]
    passive_checks_enabled          [0/1]
    check_period                     timeperiod_name
    obsess_over_service              [0/1]
    check_freshness                 [0/1]
    freshness_threshold              #
    event_handler                   command_name
    event_handler_enabled            [0/1]
    low_flap_threshold               #
    high_flap_threshold              #
    flap_detection_enabled           [0/1]
    flap_detection_options           [o,w,c,u]
    process_perf_data                [0/1]
    retain_status_information        [0/1]
    retain_nonstatus_information     [0/1]
    notification_interval             #
    first_notification_delay         #
    notification_period               timeperiod_name
    notification_options              [w,u,c,r,f,s]
    notifications_enabled             [0/1]
    contacts                         contacts
}

```

```

contact_groups          contact_groups
stalking_options        [o,w,u,c]
notes                  note_string
notes_url              url
action_url             url
icon_image             image_file
icon_image_alt          alt_string
}

```

Example Definition:

```

define service{
    host_name          linux-server
    service_description check-disk-sdal
    check_command      check-disk! /dev/sdal
    max_check_attempts 5
    check_interval     5
    retry_interval     3
    check_period       24x7
    notification_interval 30
    notification_period 24x7
    notification_options w,c,r
    contact_groups     linux-admins
}

```

Directive Descriptions:

host_name:

This directive is used to specify the *short name(s)* of the [host\(s\)](#) that the service "runs" on or is associated with. Multiple hosts should be separated by commas.

hostgroup_name:

This directive is used to specify the *short name(s)* of the [hostgroup\(s\)](#) that the service "runs" on or is associated with. Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.

service_description:

This directive is used to define the description of the service, which may contain spaces, dashes, and colons (semicolons, apostrophes, and quotation marks should be avoided). No two services associated with the same host can have the same description. Services are uniquely identified with their *host_name* and *service_description* directives.

display_name:

This directive is used to define an alternate name that should be displayed in the web interface for this service. If not specified, this defaults to the value you specify for the *service_description* directive.



Note

The CGIs up to Icinga 1.0.1 do not use this option.

servicegroups:

This directive is used to identify the *short name(s)* of the [servicegroup\(s\)](#) that the service belongs to. Multiple servicegroups should be separated by commas. This directive may be used as an alternative to using the *members* directive in [servicegroup](#) definitions.

is_volatile:	This directive is used to denote whether the service is "volatile". Services are normally <i>not</i> volatile. More information on volatile service and how they differ from normal services can be found here . Value: 0 = service is not volatile, 1 = service is volatile, 2 = service is volatile but will respect the re-notification interval for notifications (option "2" was introduced in Icinga 1.0.2).
check_command:	This directive is used to specify the <i>short name</i> of the command that Icinga will run in order to check the status of the service. The maximum amount of time that the service check command can run is controlled by the service_check_timeout option.
initial_state:	By default Icinga will assume that all services are in OK states when it starts. You can override the initial state for a service by using this directive. Valid options are: o = OK, w = WARNING, u = UNKNOWN, and c = CRITICAL.
max_check_attempts:	This directive is used to define the number of times that Icinga will retry the service check command if it returns any state other than an OK state. Setting this value to 1 will cause Icinga to generate an alert without retrying the service check again.
check_interval:	This directive is used to define the number of "time units" to wait before scheduling the next "regular" check of the service. "Regular" checks are those that occur when the service is in an OK state or when the service is in a non-OK state, but has already been rechecked max_check_attempts number of times. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. More information on this value can be found in the check scheduling documentation.
retry_interval:	This directive is used to define the number of "time units" to wait before scheduling a re-check of the service. Services are rescheduled at the retry interval when they have changed to a non-OK state. Once the service has been retried max_check_attempts times without a change in its status, it will revert to being scheduled at its "normal" rate as defined by the check_interval value. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. More information on this value can be found in the check scheduling documentation.
active_checks_enabled *:	This directive is used to determine whether or not active checks of this service are enabled. Values: 0 = disable active service checks, 1 = enable active service checks.
passive_checks_enabled *:	This directive is used to determine whether or not passive checks of this service are enabled. Values: 0 = disable passive service checks, 1 = enable passive service checks.
check_period:	This directive is used to specify the short name of the time period during which active checks of this service can be made.
obsess_over_service *:	This directive determines whether or not checks for the service will be "obsessed" over using the ocsp_command .
check_freshness *:	This directive is used to determine whether or not freshness checks are enabled for this service. Values: 0 = disable freshness checks, 1 = enable freshness checks.
freshness_threshold:	This directive is used to specify the freshness threshold (in seconds) for this service. If you set this directive to a value of 0, Icinga will determine a freshness threshold to use automatically.

event_handler:	This directive is used to specify the <i>short name</i> of the command that should be run whenever a change in the state of the service is detected (i.e. whenever it goes down or recovers). Read the documentation on event handlers for a more detailed explanation of how to write scripts for handling events. The maximum amount of time that the event handler command can run is controlled by the event_handler_timeout option.
event_handler_enabled *:	This directive is used to determine whether or not the event handler for this service is enabled. Values: 0 = disable service event handler, 1 = enable service event handler.
low_flap_threshold:	This directive is used to specify the low state change threshold used in flap detection for this service. More information on flap detection can be found here . If you set this directive to a value of 0, the program-wide value specified by the low_service_flap_threshold directive will be used.
high_flap_threshold:	This directive is used to specify the high state change threshold used in flap detection for this service. More information on flap detection can be found here . If you set this directive to a value of 0, the program-wide value specified by the high_service_flap_threshold directive will be used.
flap_detection_enabled *:	This directive is used to determine whether or not flap detection is enabled for this service. More information on flap detection can be found here . Values: 0 = disable service flap detection, 1 = enable service flap detection.
flap_detection_options:	This directive is used to determine what service states the flap detection logic will use for this service. Valid options are a combination of one or more of the following: o = OK states, w = WARNING states, c = CRITICAL states, u = UNKNOWN states.
process_perf_data *:	This directive is used to determine whether or not the processing of performance data is enabled for this service. Values: 0 = disable performance data processing, 1 = enable performance data processing.
retain_status_information:	This directive is used to determine whether or not status-related information about the service is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable status information retention, 1 = enable status information retention.
retain_nonstatus_information:	This directive is used to determine whether or not non-status information about the service is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable non-status information retention, 1 = enable non-status information retention.
notification_interval:	This directive is used to define the number of "time units" to wait before re-notifying a contact that this service is <i>still</i> in a non-OK state. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. If you set this value to 0, Icinga will <i>not</i> re-notify contacts about problems for this service - only one problem notification will be sent out.
first_notification_delay:	This directive is used to define the number of "time units" to wait before sending out the first problem notification when this service enters a non-OK state. Unless you've changed the interval_length directive from the default value of 60, this number will mean minutes. If you set this value to 0, Icinga will start sending out notifications immediately.

notification_period:

This directive is used to specify the short name of the [time period](#) during which notifications of events for this service can be sent out to contacts. No service notifications will be sent out during times which is not covered by the time period.

notification_options:

This directive is used to determine when notifications for the service should be sent out. Valid options are a combination of one or more of the following: **w** = send notifications on a WARNING state, **u** = send notifications on an UNKNOWN state, **c** = send notifications on a CRITICAL state, **r** = send notifications on recoveries (OK state), **f** = send notifications when the service starts and stops [flapping](#), and **s** = send notifications when [scheduled downtime](#) starts and ends. If you specify **n** (none) as an option, no service notifications will be sent out. If you do not specify any notification options, Icinga will assume that you want notifications to be sent out for all possible states. Example: If you specify **w,r** in this field, notifications will only be sent out when the service goes into a WARNING state and when it recovers from a WARNING state.

notifications_enabled *:

This directive is used to determine whether or not notifications for this service are enabled. Values: 0 = disable service notifications, 1 = enable service notifications.

contacts:

This is a list of the *short names* of the [contacts](#) that should be notified whenever there are problems (or recoveries) with this service. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure [contact groups](#). You must specify at least one contact or contact group in each service definition.

contact_groups:

This is a list of the *short names* of the [contact groups](#) that should be notified whenever there are problems (or recoveries) with this service. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each service definition.

stalking_options:

This directive determines which service states "stalking" is enabled for. Valid options are a combination of one or more of the following: **o** = stalk on OK states, **w** = stalk on WARNING states, **u** = stalk on UNKNOWN states, and **c** = stalk on CRITICAL states. More information on state stalking can be found [here](#).

notes:

This directive is used to define an optional string of notes pertaining to the service. If you specify a note here, you will see it in the [extended information](#) CGI (when you are viewing information about the specified service).

notes_url:

This directive is used to define an optional URL that can be used to provide more information about the service. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing service information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. /cgi-bin/icinga/). This can be very useful if you want to make detailed information on the service, emergency contact methods, etc. available to other support staff.

**Note**

Since Icinga 1.0.2 you can use multiple urls for action_url and notes_url for host and service object definitions. The syntax is as follows:

```
notes_url|action_url 'firstURL' 'secondURL' 'thirdURL'  
notes_url|action_url onlyoneURL
```

Please keep in mind that using multiple urls also mean multiple icon images. Those are hardcoded and e.g. action|notes.gif changes to 1-action|1-notes.gif and to 2-action|2-notes.gif and so on - make sure you'll have them in place when using multiple action_url|notes_url. If you are using multiple urls with different icons (1-action.gif e.g.) you can still define the last urls without single quotes. It will then be used like a single url and is referring to the normal icon (action.gif e.g.)

action_url:

This directive is used to define an optional URL that can be used to provide more actions to be performed on the service. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing service information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. /cgi-bin/icinga/).

icon_image:

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this service. This image will be displayed in the [status](#) and [extended information](#) CGIs. The image will look best if it is 40x40 pixels in size. Images for services are assumed to be in the **logos**/ subdirectory in your HTML images directory (i.e. /usr/local/icinga/share/images/logos).

icon_image_alt:

This variable is used to define an optional string that is used in the ALT tag of the image specified by the <icon_image> argument. The ALT tag is used in the [status](#), [extended information](#) and [statusmap](#) CGIs.

Servicegroup Definition

Service Group Definition

Description:

A service group definition is used to group one or more services together for simplifying configuration with [object tricks](#) or display purposes in the [CGIs](#).

Definition Format:

 **Note**

"**Directives**" are required while the others are optional.

```
define servicegroup{
    servicegroup_name      servicegroup_name
    alias                  alias
    members                services
    servicegroup_members   servicegroups
    notes                 note_string
    notes_url              url
    action_url             url
}
```

Example Definition:

```
define servicegroup{
    servicegroup_name      dbservices
    alias                  Database Services
    members                ms1,SQL Server,ms1,SQL Server Agent,ms1,SQL DTC
}
```

Directive Descriptions:

servicegroup_name:	This directive is used to define a short name used to identify the service group.
alias:	This directive is used to define a longer name or description used to identify the service group. It is provided in order to allow you to more easily identify a particular service group.
members:	This is a list of the <i>descriptions</i> of services (and the names of their corresponding hosts) that should be included in this group. Host and service names should be separated by commas. This directive may be used as an alternative to the <i>servicegroups</i> directive in service definitions . The format of the member directive is as follows (note that a host name must precede a service name/description): <code>members=<host1>,<service1>,<host2>,<service2>,...,<hostn>,<servicen></code>
	Starting with Icinga 1.2 it is possible to use "*" as a wildcard for all hosts. Please note that it is NOT possible to exclude neither hosts nor services using the "!" at the beginning of the host or service.
servicegroup_members:	This optional directive can be used to include services from other "sub" service groups in this service group. Specify a comma-delimited list of short names of other service groups whose members should be included in this group.
notes:	This directive is used to define an optional string of notes pertaining to the service group. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified service group).
notes_url:	This directive is used to define an optional URL that can be used to provide more information about the service group. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing service group information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. /cgi-bin/icinga/). This can be very useful if you want to make detailed information on the service group, emergency contact methods, etc. available to other support staff.
action_url:	This directive is used to define an optional URL that can be used to provide more actions to be performed on the service group. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing service group information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. /cgi-bin/icinga/).

Contact Definition

Contact Definition

Description:

A contact definition is used to identify someone who should be contacted in the event of a problem on your network. The different arguments to a contact definition are described below.

Definition Format:

**Note**

"**Directives**" are required while the others are optional.

```
define contact{
```

contact_name	contact_name
alias	alias
contactgroups	contactgroup_names
host_notifications_enabled	[0/1]
service_notifications_enabled	[0/1]
host_notification_period	timeperiod_name
service_notification_period	timeperiod_name
host_notification_options	[d,u,r,f,s,n]
service_notification_options	[w,u,c,r,f,s,n]
host_notification_commands	command_name
service_notification_commands	command_name
email	email_address
pager	pager_number or pager_email_gateway
addressx	additional_contact_address
can_submit_commands	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
}	

Example Definition:

```
define contact{
    contact_name          jdoe
    alias                 John Doe
    host_notifications_enabled 1
    service_notifications_enabled 1
    service_notification_period 24x7
    host_notification_period 24x7
    service_notification_options w,u,c,r
    host_notification_options d,u,r
    service_notification_commands notify-by-email
    host_notification_commands host-notify-by-email
    email                jdoe@localhost.localdomain
    pager                555-5555@pagergateway.localhost.localdomain
    address1              xxxxx.xyyy@icq.com
    address2              555-555-5555
    can_submit_commands   1
}
```

*Directive Descriptions:***contact_name:**

This directive is used to define a short name used to identify the contact. It is referenced in [contact group](#) definitions. Under the right circumstances, the \$CONTACTNAME\$ [macro](#) will contain this value.

alias:

This directive is used to define a longer name or description for the contact. Under the rights circumstances, the \$CONTACTALIAS\$ [macro](#) will contain this value. If not specified, the *contact_name* will be used as the alias.

contactgroups:

This directive is used to identify the *short name(s)* of the [contactgroup\(s\)](#) that the contact belongs to. Multiple contactgroups should be separated by commas. This directive may be used as an alternative to (or in addition to) using the *members* directive in [contactgroup](#) definitions.

host_notifications_enabled:

This directive is used to determine whether or not the contact will receive notifications about host problems and recoveries. Values: 0 = don't send notifications, 1 = send notifications.

service_notifications_enabled:

This directive is used to determine whether or not the contact will receive notifications about service problems and recoveries. Values: 0 = don't send notifications, 1 = send notifications.

host_notification_period:

This directive is used to specify the short name of the [time period](#) during which the contact can be notified about host problems or recoveries. You can think of this as an "on call" time for host notifications for the contact. Read the documentation on [time periods](#) for more information on how this works and potential problems that may result from improper use.

service_notification_period:

This directive is used to specify the short name of the [time period](#) during which the contact can be notified about service problems or recoveries. You can think of this as an "on call" time for service notifications for the contact. Read the documentation on [time periods](#) for more information on how this works and potential problems that may result from improper use.

host_notification_commands:

This directive is used to define a list of the *short names* of the [commands](#) used to notify the contact of a *host* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the [notification_timeout](#) option.

host_notification_options:

This directive is used to define the host states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following: **d** = notify on DOWN host states, **u** = notify on UNREACHABLE host states, **r** = notify on host recoveries (UP states), **f** = notify when the host starts and stops **flapping**, and **s** = send notifications when host or service **scheduled downtime** starts and ends. If you specify **n** (none) as an option, the contact will not receive any type of host notifications.

service_notification_options:

This directive is used to define the service states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following: **w** = notify on WARNING service states, **u** = notify on UNKNOWN service states, **c** = notify on CRITICAL service states, **r** = notify on service recoveries (OK states), and **f** = notify when the service starts and stops **flapping**. If you specify **n** (none) as an option, the contact will not receive any type of service notifications.

service_notification_commands:

This directive is used to define a list of the *short names* of the **commands** used to notify the contact of a *service* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the **notification_timeout** option.

email:

This directive is used to define an email address for the contact. Depending on how you configure your notification commands, it can be used to send out an alert email to the contact. Under the right circumstances, the \$CONTACTEMAIL\$ **macro** will contain this value.

pager:

This directive is used to define a pager number for the contact. It can also be an email address to a pager gateway (i.e. pagejoe@pagenet.com). Depending on how you configure your notification commands, it can be used to send out an alert page to the contact. Under the right circumstances, the \$CONTACTPAGER\$ **macro** will contain this value.

addressx:

Address directives are used to define additional "addresses" for the contact. These addresses can be anything - cell phone numbers, instant messaging addresses, etc. Depending on how you configure your notification commands, they can be used to send out an alert to the contact. Up to six addresses can be defined using these directives (*address1* through *address6*). The \$CONTACTADDRESSx\$ **macro** will contain this value.

can_submit_commands:	This directive is used to determine whether or not the contact can submit external commands to Icinga from the CGIs. Values: 0 = don't allow contact to submit commands, 1 = allow contact to submit commands.
retain_status_information:	This directive is used to determine whether or not status-related information about the contact is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable status information retention, 1 = enable status information retention.
retain_nonstatus_information:	This directive is used to determine whether or not non-status information about the contact is retained across program restarts. This is only useful if you have enabled state retention using the retain_state_information directive. Value: 0 = disable non-status information retention, 1 = enable non-status information retention.

Contactgroup Definition

Contact Group Definition

Description:

A contact group definition is used to group one or more [contacts](#) together for the purpose of sending out alert/recovery [notifications](#).

Definition Format:



Note

"[Directives](#)" are required while the others are optional.

```
define contactgroup{
    contactgroup_name      contactgroup_name
    alias                  alias
    members                contacts
    contactgroup_members   contactgroups
}
```

Example Definition:

```
define contactgroup{
    contactgroup_name      novell-admins
    alias                  Novell Administrators
    members                jdoe,rtobert,tzach
}
```

Directive Descriptions:

- contactgroup_name:** This directive is a short name used to identify the contact group.
- alias:** This directive is used to define a longer name or description used to identify the contact group.
- members:** This directive is used to define a list of the *short names* of [contacts](#) that should be included in this group. Multiple contact names should be separated by commas. This directive may be used as an alternative to (or in addition to) using the *contactgroups* directive in [contact](#) definitions.
- contactgroup_members:** This optional directive can be used to include contacts from other "sub" contact groups in this contact group. Specify a comma-delimited list of short names of other contact groups whose members should be included in this group.

Timeperiod Definition

Time Period Definition

Description:

A time period is a list of times during various days that are considered to be "valid" times for notifications and service checks. It consists of time ranges for each day of the week that "rotate" once the week has come to an end. Different types of exceptions to the normal weekly time are supported, including: specific weekdays, days of generic months, days of specific months, and calendar dates.

Definition Format:



Note

"[Directives](#)" are required while the others are optional.

```
define timeperiod{
    timeperiod_name    timeperiod_name
    alias              alias
    [weekday]          timeranges
    [exception]        timeranges
    exclude            [timeperiod1,timeperiod2,...,timeperiodn]
}
```

Example Definitions:

```
define timeperiod{
    timeperiod_name    nonworkhours
    alias              Non-Work Hours
    sunday             00:00-24:00           ; Every Sunday of every week
    monday             00:00-09:00,17:00-24:00   ; Every Monday of every week
    tuesday            00:00-09:00,17:00-24:00   ; Every Tuesday of every week
    wednesday          00:00-09:00,17:00-24:00   ; Every Wednesday of every week
    thursday            00:00-09:00,17:00-24:00   ; Every Thursday of every week
    friday             00:00-09:00,17:00-24:00   ; Every Friday of every week
```

```

saturday          00:00-24:00           ; Every Saturday of every week
}

define timeperiod{
    timeperiod_name      misc-single-days
    alias                Misc Single Days
    1999-01-28          00:00-24:00           ; January 28th, 1999
    monday 3             00:00-24:00           ; 3rd Monday of every month
    day 2                00:00-24:00           ; 2nd day of every month
    february 10          00:00-24:00           ; February 10th of every year
    february -1          00:00-24:00           ; Last day in February of every year
    friday -2            00:00-24:00           ; 2nd to last Friday of every month
    thursday -1 november 00:00-24:00           ; Last Thursday in November of every year
}

define timeperiod{
    timeperiod_name      misc-date-ranges
    alias                Misc Date Ranges
    2007-01-01 - 2008-02-01 00:00-24:00           ; January 1st, 2007 to February 1st, 2008
    monday 3 - thursday 4 00:00-24:00           ; 3rd Monday to 4th Thursday of every month
    day 1 - 15            00:00-24:00           ; 1st to 15th day of every month
    day 20 - -1           00:00-24:00           ; 20th to the last day of every month
    july 10 - 15          00:00-24:00           ; July 10th to July 15th of every year
    april 10 - may 15    00:00-24:00           ; April 10th to May 15th of every year
    tuesday 1 april - friday 2 may   00:00-24:00 ; 1st Tuesday in April
                                         ; to 2nd Friday in May of every year
}

define timeperiod{
    timeperiod_name      misc-skip-ranges
    alias                Misc Skip Ranges
    2007-01-01 - 2008-02-01 / 3 00:00-24:00           ; Every 3 days from January 1st, 2007 to February 1st, 2008
    2008-04-01 / 7        00:00-24:00           ; Every 7 days from April 1st, 2008 (continuing forever)
    monday 3 - thursday 4 / 2 00:00-24:00           ; Every other day from 3rd Monday to 4th Thursday of every month
    day 1 - 15 / 5         00:00-24:00           ; Every 5 days from the 1st to the 15th day of every month
    july 10 - 15 / 2       00:00-24:00           ; Every other day from July 10th to July 15th of every year
    tuesday 1 april - friday 2 may / 6 00:00-24:00 ; Every 6 days from the 1st Tuesday in April
                                         ; to the 2nd Friday in May of every year
}

```

Directive Descriptions:

timeperiod_name:	This directives is the short name used to identify the time period.
alias:	This directive is a longer name or description used to identify the time period.
[weekday]:	The weekday directives ("sunday" through "saturday")are comma-delimited lists of time ranges that are "valid" times for a particular day of the week. Notice that there are seven different days for which you can define time ranges (Sunday through Saturday). Each time range is in the form of HH:MM-HH:MM , where hours are specified on a 24 hour clock. For example, 00:15-24:00 means 12:15am in the morning for this day until 12:00am midnight (a 23 hour, 45 minute total time range). If you wish to exclude an entire day from the timeperiod, simply do not include it in the timeperiod definition.
[exception]:	You can specify several different types of exceptions to the standard rotating weekday schedule. Exceptions can take a number of different forms including single days of a specific or generic month, single weekdays in a month, or single calendar dates. You can also specify a range of days/dates and even specify skip intervals to obtain functionality described by "every 3 days between these dates". Rather than list all the possible formats for exception strings, we'll let you look at the example timeperiod definitions above to see what's possible. :-) Weekdays and different types of exceptions all have different levels of precedence, so its important to understand how they can affect each other. More information on this can be found in the documentation on timeperiods .
exclude:	This directive is used to specify the short names of other timeperiod definitions whose time ranges should be excluded from this timeperiod. Multiple timeperiod names should be separated with a comma.

Command Definition

Command Definition

Description:

A command definition is just that. It defines a command. Commands that can be defined include service checks, service notifications, service event handlers, host checks, host notifications, and host event handlers. Command definitions can contain [macros](#), but you must make sure that you include only those macros that are "valid" for the circumstances when the command will be used. More information on what macros are available and when they are "valid" can be found [here](#). The different arguments to a command definition are outlined below.

Definition Format:



Note

"[Directives](#)" are required while the others are optional.

```
define command{
    command_name    command_name
    command_line    command_line
}
```

Example Definition:

```
define command{
    command_name    check_pop
    command_line    /usr/local/icinga/libexec/check_pop -H $HOSTADDRESS$}
```

Directive Descriptions:

- command_name:** This directive is the short name used to identify the command. It is referenced in [contact](#), [host](#), and [service](#) definitions (in notification, check, and event handler directives), among other places.
- command_line:** This directive is used to define what is actually executed by Icinga when the command is used for service or host checks, notifications, or [event handlers](#). Before the command line is executed, all valid [macros](#) are replaced with their respective values. See the documentation on macros for determining when you can use different macros. Note that the command line is *not* surrounded in quotes. Also, if you want to pass a dollar sign (\$) on the command line, you have to escape it with another dollar sign.

NOTE: You may not include a **semicolon** (;) in the *command_line* directive, because everything after it will be ignored as a config file comment. You can work around this limitation by setting one of the **\$USER\$** macros in your [resource file](#) to a semicolon and then referencing the appropriate \$USER\$ macro in the *command_line* directive in place of the semicolon.

If you want to pass arguments to commands during runtime, you can use **\$ARGn\$ macros** in the *command_line* directive of the command definition and then separate individual arguments from the command name (and from each other) using bang (!) characters in the object definition directive (host check command, service event handler command, etc) that references the command. More information on how arguments in command definitions are processed during runtime can be found in the documentation on [macros](#).

Servicedependency Definition

Service Dependency Definition

Description:

Service dependencies are an advanced feature of Icinga that allow you to suppress notifications and active checks of services based on the status of one or more other services. Service dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how service dependencies work (read this!) can be found [here](#).

Definition Format:



Note

"**Directives**" are required while the others are optional. However, you must supply at least one type of criteria for the definition to be of much use.

```
define servicedependency{
    dependent_host_name          host_name
    dependent_hostgroup_name      hostgroup_name
    dependent_service_description service_description
    host_name                     host_name
    hostgroup_name                hostgroup_name
    service_description           service_description
    inherits_parent                [0/1]
    execution_failure_criteria   [o,w,u,c,p,n]
    notification_failure_criteria [o,w,u,c,p,n]
    dependency_period             timeperiod_name
}
```

Example Definition:

```
define servicedependency{
    host_name          WWW1
    service_description Apache Web Server
    dependent_host_name WWW1
    dependent_service_description Main Web Site
    execution_failure_criteria n
    notification_failure_criteria w,u,c
}
```

Directive Descriptions:

dependent_host_name:

This directive is used to identify the *short name(s)* of the **host(s)** that the *dependent* service "runs" on or is associated with. Multiple hosts should be separated by commas. Leaving this directive blank can be used to create "[same host](#)" dependencies.

dependent_hostgroup_name:

This directive is used to specify the *short name(s)* of the **hostgroup(s)** that the *dependent* service "runs" on or is associated with. Multiple hostgroups should be separated by commas. The *dependent_hostgroup* may be used instead of, or in addition to, the *dependent_host* directive.

dependent_service_description:

This directive is used to identify the *description* of the *dependent service*.

host_name:	This directive is used to identify the <i>short name(s)</i> of the host(s) that the service <i>that is being depended upon</i> (also referred to as the master service) "runs" on or is associated with. Multiple hosts should be separated by commas.
hostgroup_name:	This directive is used to identify the <i>short name(s)</i> of the hostgroup(s) that the service <i>that is being depended upon</i> (also referred to as the master service) "runs" on or is associated with. Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.
service_description:	This directive is used to identify the <i>description</i> of the service <i>that is being depended upon</i> (also referred to as the master service).
inherits_parent:	This directive indicates whether or not the dependency inherits dependencies of the service <i>that is being depended upon</i> (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.
execution_failure_criteria:	This directive is used to specify the criteria that determine when the dependent service should <i>not</i> be actively checked. If the <i>master</i> service is in one of the failure states we specify, the <i>dependent</i> service will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas): o = fail on an OK state, w = fail on a WARNING state, u = fail on an UNKNOWN state, c = fail on a CRITICAL state, and p = fail on a pending state (e.g. the service has not yet been checked). If you specify n (none) as an option, the execution dependency will never fail and checks of the dependent service will always be actively checked (if other conditions allow for it to be). Example: If you specify o,c,u in this field, the <i>dependent</i> service will not be actively checked if the <i>master</i> service is in either an OK, a CRITICAL, or an UNKNOWN state.

notification_failure_criteria:	This directive is used to define the criteria that determine when notifications for the dependent service should <i>not</i> be sent out. If the <i>master</i> service is in one of the failure states we specify, notifications for the <i>dependent</i> service will not be sent to contacts. Valid options are a combination of one or more of the following: o = fail on an OK state, w = fail on a WARNING state, u = fail on an UNKNOWN state, c = fail on a CRITICAL state, and p = fail on a pending state (e.g. the service has not yet been checked). If you specify n (none) as an option, the notification dependency will never fail and notifications for the dependent service will always be sent out. Example: If you specify w in this field, the notifications for the <i>dependent</i> service will not be sent out if the <i>master</i> service is in a WARNING state.
dependency_period:	This directive is used to specify the short name of the time period during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

Serviceescalation Definition

Service Escalation Definition

Description:

Service escalations are *completely optional* and are used to escalate notifications for a particular service. More information on how notification escalations work can be found [here](#).

Definition Format:



Note

"Directives" are required while the others are optional.

```
define serviceescalation{
    host_name          host_name
    hostgroup_name     hostgroup_name
    servicegroup_name  servicegroup_name
    service_description service_description
    contacts           contacts
    contact_groups     contactgroup_name
    first_notification #
    last_notification   #
    notification_interval #
    escalation_period  timeperiod_name
    escalation_options [w,u,c,r]
    escalation_condition <condition> ( [ & / | ] <condition> )*
    first_warning_notification #
    last_warning_notification #
    first_critical_notification #
    last_critical_notification #
    first_unknown_notification #
    last_unknown_notification #
}
```

Example Definition:

```
define serviceescalation{
    host_name          nt-3
    service_description Processor Load
    first_notification 4
    last_notification   0
    notification_interval 30
    contact_groups      all-nt-admins,themanager
}
```

*Directive Descriptions:***host_name:**

This directive is used to identify the *short name(s)* of the [host\(s\)](#) that the [service](#) escalation should apply to or is associated with.

hostgroup_name:

This directive is used to specify the *short name(s)* of the [hostgroup\(s\)](#) that the service escalation should apply to or is associated with. Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.

servicegroup_name:	This directive is used to specify the <i>short name(s)</i> of the servicegroup(s) that the service escalation should apply to or is associated with. Multiple servicegroups should be separated by commas. The servicegroup_name may be used instead of, or in addition to, the service_name directive.
service_description:	This directive is used to identify the <i>description</i> of the service the escalation should apply to.
first_notification:	This directive is a number that identifies the <i>first</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third notification to go out.
last_notification:	This directive is a number that identifies the <i>last</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
contacts:	This is a list of the <i>short names</i> of the contacts that should be notified whenever there are problems (or recoveries) with this service. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure contact groups . You must specify at least one contact or contact group in each service escalation definition.
contact_groups:	This directive is used to identify the <i>short name</i> of the contact group that should be notified when the service notification is escalated. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each service escalation definition.
notification_interval:	This directive is used to determine the interval at which notifications should be made while this escalation is valid. If you specify a value of 0 for the interval, Icinga will send the first notification when this escalation definition is valid, but will then prevent any more problem notifications from being sent out for the host. Notifications are sent out again until the host recovers. This is useful if you want to stop having notifications sent out after a certain amount of time.
escalation_period:	This directive is used to specify the short name of the time period during which this escalation is valid. If this directive is not specified, the escalation is considered to be valid during all times.



Note

If multiple escalation entries for a host overlap for one or more notification ranges, the smallest notification interval from all escalation entries is used.

escalation_options:	This directive is used to define the criteria that determine when this service escalation is used. The escalation is used only if the service is in one of the states specified in this directive. If this directive is not specified in a service escalation, the escalation is considered to be valid during all service states. Valid options are a combination of one or more of the following: r = escalate on an OK (recovery) state, w = escalate on a WARNING state, u = escalate on an UNKNOWN state, and c = escalate on a CRITICAL state. Example: If you specify w in this field, the escalation will only be used if the service is in a WARNING state.
escalation_condition:	This directive is completely optional (and only available starting with Icinga 1.0.1). Details can be found here .
first_warning_notification:	This directive is a number that identifies the <i>first warning</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third warning notification to go out.
last_warning_notification:	This directive is a number that identifies the <i>last warning</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five warning notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
first_critical_notification:	This directive is a number that identifies the <i>first critical</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third critical notification to go out.
last_critical_notification:	This directive is a number that identifies the <i>last critical</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five critical notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
first_unknown_notification:	This directive is a number that identifies the <i>first unknown</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third unknown notification to go out.
last_unknown_notification:	This directive is a number that identifies the <i>last unknown</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five unknown notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).

Hostdependency Definition

Host Dependency Definition

Description:

Host dependencies are an advanced feature of Icinga that allow you to suppress notifications for hosts based on the status of one or more other hosts. Host dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how host dependencies work (read this!) can be found [here](#).

Definition Format:



Note

"**Directives**" are required while the others are optional.

```
define hostdependency{
    dependent_host_name      host_name
    dependent_hostgroup_name hostgroup_name
    host_name                host_name
    hostgroup_name           hostgroup_name
    inherits_parent          [0/1]
    execution_failure_criteria [o,d,u,p,n]
    notification_failure_criteria [o,d,u,p,n]
    dependency_period        timeperiod_name
}
```

Example Definition:

```
define hostdependency{
    host_name      WWW1
    dependent_host_name DBASE1
    notification_failure_criteria d,u
}
```

Directive Descriptions:

dependent_host_name:

This directive is used to identify the *short name(s)* of the *dependenthost(s)*. Multiple hosts should be separated by commas.

dependent_hostgroup_name:

This directive is used to identify the *short name(s)* of the *dependenthostgroup(s)*. Multiple hostgroups should be separated by commas. The *dependent_hostgroup_name* may be used instead of, or in addition to, the *dependent_host_name* directive.

host_name:	This directive is used to identify the <i>short name(s)</i> of the host(s) that is being depended upon (also referred to as the master host). Multiple hosts should be separated by commas.
hostgroup_name:	This directive is used to identify the <i>short name(s)</i> of the hostgroup(s) that is being depended upon (also referred to as the master host). Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.
inherits_parent:	This directive indicates whether or not the dependency inherits dependencies of the host <i>that is being depended upon</i> (also referred to as the master host). In other words, if the master host is dependent upon other hosts and any one of those dependencies fail, this dependency will also fail.
execution_failure_criteria:	This directive is used to specify the criteria that determine when the dependent host should <i>not</i> be actively checked. If the <i>master</i> host is in one of the failure states we specify, the <i>dependent</i> host will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas): o = fail on an UP state, d = fail on a DOWN state, u = fail on an UNREACHABLE state, and p = fail on a pending state (e.g. the host has not yet been checked). If you specify n (none) as an option, the execution dependency will never fail and the dependent host will always be actively checked (if other conditions allow for it to be). Example: If you specify u,d in this field, the <i>dependent</i> host will not be actively checked if the <i>master</i> host is in either an UNREACHABLE or DOWN state.
notification_failure_criteria:	This directive is used to define the criteria that determine when notifications for the dependent host should <i>not</i> be sent out. If the <i>master</i> host is in one of the failure states we specify, notifications for the <i>dependent</i> host will not be sent to contacts. Valid options are a combination of one or more of the following: o = fail on an UP state, d = fail on a DOWN state, u = fail on an UNREACHABLE state, and p = fail on a pending state (e.g. the host has not yet been checked). If you specify n (none) as an option, the notification dependency will never fail and notifications for the dependent host will always be sent out. Example: If you specify d in this field, the notifications for the <i>dependent</i> host will not be sent out if the <i>master</i> host is in a DOWN state.
dependency_period:	This directive is used to specify the short name of the time period during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

Hostescalation Definition

Host Escalation Definition

Description:

Host escalations are *completely optional* and are used to escalate notifications for a particular host. More information on how notification escalations work can be found [here](#).

Definition Format:



Note

"**Directives**" are required while the others are optional.

```
define hostescalation{
    host_name          host_name
    hostgroup_name     hostgroup_name
    contacts           contacts
    contact_groups     contactgroup_name
    first_notification #
    last_notification   #
    notification_interval #
    escalation_period  timeperiod_name
    escalation_options [d,u,r]
    first_down_notification #
    last_down_notification #
    first_unreachable_notification #
    last_unreachable_notification #
}
```

Example Definition:

```
define hostescalation{
    host_name          router-34
    first_notification 5
    last_notification   8
    notification_interval 60
    contact_groups      all-router-admins
}
```

Directive Descriptions:

host_name:

This directive is used to identify the *short name* of the [host](#) that the escalation should apply to.

hostgroup_name:	This directive is used to identify the <i>short name(s)</i> of the hostgroup(s) that the escalation should apply to. Multiple hostgroups should be separated by commas. If this is used, the escalation will apply to all hosts that are members of the specified hostgroup(s).
first_notification:	This directive is a number that identifies the <i>first</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the host is down or unreachable long enough for a third notification to go out.
last_notification:	This directive is a number that identifies the <i>last</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for the host. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
contacts:	This is a list of the <i>short names</i> of the contacts that should be notified whenever there are problems (or recoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure contact groups . You must specify at least one contact or contact group in each host escalation definition.
contact_groups:	This directive is used to identify the <i>short name</i> of the contact group that should be notified when the host notification is escalated. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host escalation definition.
notification_interval:	This directive is used to determine the interval at which notifications should be made while this escalation is valid. If you specify a value of 0 for the interval, Icinga will send the first notification when this escalation definition is valid, but will then prevent any more problem notifications from being sent out for the host. Notifications are sent out again until the host recovers. This is useful if you want to stop having notifications sent out after a certain amount of time.
escalation_period:	This directive is used to specify the short name of the time period during which this escalation is valid. If this directive is not specified, the escalation is considered to be valid during all times.

**Note**

If multiple escalation entries for a host overlap for one or more notification ranges, the smallest notification interval from all escalation entries is used.

escalation_options:	This directive is used to define the criteria that determine when this host escalation is used. The escalation is used only if the host is in one of the states specified in this directive. If this directive is not specified in a host escalation, the escalation is considered to be valid during all host states. Valid options are a combination of one or more of the following: r = escalate on an UP (recovery) state, d = escalate on a DOWN state, and u = escalate on an UNREACHABLE state. Example: If you specify d in this field, the escalation will only be used if the host is in a DOWN state.
first_down_notification:	This directive is a number that identifies the <i>first down</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the host is in a non-OK state long enough for a third down notification to go out.
last_down_notification:	This directive is a number that identifies the <i>first down</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five down notifications are sent out for the host. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
first_unreachable_notification:	This directive is a number that identifies the <i>first unreachable</i> notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the host is in a non-OK state long enough for a third unreachable notification to go out.
last_unreachable_notification:	This directive is a number that identifies the <i>first unreachable</i> notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five unreachable notifications are sent out for the host. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).

Hostextinfo Definition

Extended Host Information Definition

Description:

Extended host information entries are basically used to make the output from the [status](#), [statusmap](#), [statuswrl](#), and [extinfo](#) CGIs look pretty. They have no effect on monitoring and are completely optional.

**Tip**

All directives contained in extended host information definitions are also available in [host definitions](#). Thus, you can choose to define the directives below in your host definitions if it makes your configuration simpler. Separate extended host information definitions will continue to be supported for backward compatibility.

Definition Format:

**Note**

"**Directives**" are required while the others are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

```
define hostextinfo{
    host_name          host_name
    notes              note_string
    notes_url          url
    action_url         url
    icon_image         image_file
    icon_image_alt     alt_string
    vrml_image         image_file
    statusmap_image    image_file
    2d_coords          x_coord,y_coord
    3d_coords          x_coord,y_coord,z_coord
}
```

Example Definition:

```
define hostextinfo{
    host_name          netware1
    notes              This is the primary Netware file server
    notes_url          http://webserver.localhost.localdomain/hostinfo.pl?host=netware1
    icon_image         novell140.png
    icon_image_alt     IntranetWare 4.11
    vrml_image         novell140.png
    statusmap_image    novell140.gd2
    2d_coords          100,250
    3d_coords          100.0,50.0,75.0
}
```

Variable Descriptions:

host_name: This variable is used to identify the *short name* of the [host](#) which the data is associated with.

notes:	This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified host).
notes_url:	This variable is used to define an optional URL that can be used to provide more information about the host. If you specify an URL, you will see a link that says "Extra Host Notes" in the extended information CGI (when you are viewing information about the specified host). Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>). This can be very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.
action_url:	This directive is used to define an optional URL that can be used to provide more actions to be performed on the host. If you specify an URL, you will see a link that says "Extra Host Actions" in the extended information CGI (when you are viewing information about the specified host). Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>).
icon_image:	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the status and extended information CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. <code>/usr/local/icinga/share/images/logos</code>).
icon_image_alt:	This variable is used to define an optional string that is used in the ALT tag of the image specified by the <code><icon_image></code> argument. The ALT tag is used in the status , extended information and statusmap CGIs.
vrmr_image:	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host in the statuswrl CGI. Unlike the image you use for the <code><icon_image></code> variable, this one should probably <i>not</i> have any transparency. If it does, the host object will look a bit weird. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. <code>/usr/local/icinga/share/images/logos</code>).



Note

This directive has no effect in Icinga.

statusmap_image: This variable is used to define the name of an image that should be associated with this host in the [statusmap](#) CGI. You can specify a JPEG, PNG, and GIF image if you want, although we would strongly suggest using a GD2 format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the [pngtogd2](#) utility supplied with Thomas Boutell's [gd library](#). The GD2 images should be created in *uncompressed* format in order to minimize CPU load when the statusmap CGI is generating the network map image. The image will look best if it is 40x40 pixels in size. You can leave these option blank if you are not using the statusmap CGI. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. `/usr/local/icinga/share/images/logos`).

2d_coords: This variable is used to define coordinates to use when drawing the host in the [statusmap](#) CGI. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host icon that is drawn.



Note

Don't worry about what the maximum x and y coordinates that you can use are. The CGI will automatically calculate the maximum dimensions of the image it creates based on the largest x and y coordinates you specify.

3d_coords: This variable is used to define coordinates to use when drawing the host in the [statuswrl](#) CGI. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.



Note

This directive has no effect in Icinga.

Serviceextinfo Definition

Extended Service Information Definition

Description:

Extended service information entries are basically used to make the output from the [status](#) and [extinfo](#) CGIs look pretty. They have no effect on monitoring and are completely optional.

**Tip**

As of Nagios 3.x, all directives contained in extended service information definitions are also available in [service definitions](#). Thus, you can choose to define the directives below in your service definitions if it makes your configuration simpler. Separate extended service information definitions will continue to be supported for backward compatibility.

Definition Format:

**Note**

"**Directives**" are required while the others are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

```
define serviceextinfo{
    host_name          host_name
    service_description service_description
    notes              note_string
    notes_url          url
    action_url         url
    icon_image         image_file
    icon_image_alt     alt_string
}
```

Example Definition:

```
define serviceextinfo{
    host_name          linux2
    service_description Log Anomalies
    notes              Security-related log anomalies on secondary Linux server
    notes_url          http://webserver.localhost.localdomain/serviceinfo.pl?host=linux2&service=Log+Anomalies
    icon_image         security.png
    icon_image_alt     Security-Related Alerts
}
```

Directive Descriptions:

host_name:	This directive is used to identify the <i>short name</i> of the host that the service is associated with.
service_description:	This directive is description of the service which the data is associated with.
notes:	This directive is used to define an optional string of notes pertaining to the service. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified service).
notes_url:	This directive is used to define an optional URL that can be used to provide more information about the service. If you specify an URL, you will see a link that says "Extra Service Notes" in the extended information CGI (when you are viewing information about the specified service). Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>). This can be very useful if you want to make detailed information on the service, emergency contact methods, etc. available to other support staff.
action_url:	This directive is used to define an optional URL that can be used to provide more actions to be performed on the service. If you specify an URL, you will see a link that says "Extra Service Actions" in the extended information CGI (when you are viewing information about the specified service). Any valid URL can be used. If you plan on using relative paths, the base path will be the same as what is used to access the CGIs (i.e. <code>/cgi-bin/icinga/</code>).
icon_image:	This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the status and extended information CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. <code>/usr/local/icinga/share/images/logos/</code>).
icon_image_alt:	This variable is used to define an optional string that is used in the ALT tag of the image specified by the <code><icon_image></code> argument. The ALT tag is used in the status , extended information and statusmap CGIs.

Module Definition

Module Definition

Description:

A module definition is used to specify information about a module. It can be used instead of a `broker_module` entry in the main configuration file thus being more flexible (you can use `cfg_file`/`cfg_dir` entries to include it).



Note

Module definitions are available starting with Icinga 1.4.

Definition Format:

 **Note**

"**Directives**" are required while the others are optional.

```
define module{

    module_name    module name
    path          path
    args          arguments
    module_type   neb
}
```

Example Definitions:

```
define module{
    module_name    ido_mod
    path          /usr/local/icinga/bin/idomod.o
    module_type   neb
    args          config_file=/usr/local/icinga/etc/idomod.cfg
}
```

Based on the [MKLiveStatus](#) documentation the module definition would look like this:

```
define module{
    module_name    mklivestatus
    path          /usr/local/lib/mk-livestatus/livestatus.o
    module_type   neb
    args          /var/lib/nagios/rw/live
}
```

Directive Descriptions:

module_name: This directive identifies the unique name of the module so you cannot have two modules with the same module name. It is mandatory, otherwise the config will not be accepted and the module will not be loaded.

module_type: This optional directive defines the type of the module, e.g. 'neb' for event broker modules. This directive is intended to allow further filtering on the module loading.

path: This mandatory directive specifies the path to the module binary to be loaded. For event broker modules like idomod the user running the core must be allowed to read and load the module.

args: This directive defines optional arguments passed to the module. idomod needs config_file=.../idomod.cfg while other modules have their own syntax. This directive is passed as argument string to the event broker module loader if used as neb module.

**Note**

Templating should be possible but it has not been tested extensively with Icinga 1.4.

[Prev](#)[Up](#)[Next](#)[Object Configuration Overview](#)[Home](#)[Custom Object Variables](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Custom Object Variables

[Prev](#)
[Chapter 3. Configuring Icinga](#)
[Next](#)

Custom Object Variables

Introduction

Users often request that new variables be added to host, service, and contact definitions. These include variables for SNMP community, MAC address, AIM username, Skype number, and street address. The list is endless. The problem that I see with doing this is that it makes Icinga less generic and more infrastructure-specific. Icinga was intended to be flexible, which meant things needed to be designed in a generic manner. Host definitions in Icinga, for example, have a generic "address" variable that can contain anything from an IP address to human-readable driving directions - whatever is appropriate for the user's setup.

Still, there needs to be a method for admins to store information about their infrastructure components in their Icinga configuration without imposing a set of specific variables on others. Icinga attempts to solve this problem by allowing users to define custom variables in their object definitions. Custom variables allow users to define additional properties in their host, service, and contact definitions, and use their values in notifications, event handlers, and host and service checks.

Custom Variable Basics

There are a few important things that you should note about custom variables:

- Custom variable names must begin with an underscore (_) to prevent name collision with standard variables
- Custom variable names are converted to all uppercase before use
- Custom variables are [inherited](#) from object templates like normal variables
- Scripts can reference custom variable values with [macros and environment variables](#)

Examples

Here's an example of how custom variables can be defined in different types of object definitions:

```
define host{
    host_name      linuxserver
    _mac_address   00:06:5B:A6:AD:AA      ; <-- Custom MAC_ADDRESS variable
    _rack_number   R32                   ; <-- Custom RACK_NUMBER variable
    ...
}
```

```

define service{
    host_name      linuxserver
    description     Memory Usage
    _SNMP_community public           ; <-- Custom SNMP_COMMUNITY variable
    _TechContact   Jane Doe          ; <-- Custom TECHCONTACT variable
    ...
}

define contact{
    contact_name   john
    _AIM_username  john16           ; <-- Custom AIM_USERNAME variable
    _YahooID       john32           ; <-- Custom YAHOOID variable
    ...
}

```

Custom Variables As Macros

Custom variable values can be referenced in scripts and executables that Icinga runs for checks, notifications, etc. by using [macros](#) or environment variables.

In order to prevent name collision among custom variables from different object types, Icinga prepends "_HOST", "_SERVICE", or "_CONTACT" to the beginning of custom host, service, or contact variables, respectively, in macro and environment variable names. The table below shows the corresponding macro and environment variable names for the custom variables that were defined in the example above.

Object Type	Variable Name	Macro Name	Environment Variable
Host	MAC_ADDRESS	\$_HOSTMAC_ADDRESS\$	NAGIOS__HOSTMAC_ADDRESS
Host	RACK_NUMBER	\$_HOSTRACK_NUMBER\$	NAGIOS__HOSTRACK_NUMBER
Service	SNMP_COMMUNITY	\$_SERVICESNMP_COMMUNITY\$	NAGIOS__SERVICESNMP_COMMUNITY
Service	TECHCONTACT	\$_SERVICETECHCONTACT\$	NAGIOS__SERVICETECHCONTACT
Contact	AIM_USERNAME	\$_CONTACTAIM_USERNAME\$	NAGIOS__CONTACTAIM_USERNAME
Contact	YAHOOID	\$_CONTACTYAHOOID\$	NAGIOS__CONTACTYAHOOID

Custom Variables And Inheritance

Custom object variables are [inherited](#) just like standard host, service, or contact variables.

[Prev](#)
[Up](#)
[Next](#)
[Object Definitions](#)
[Home](#)
[CGI Configuration File Options](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



CGI Configuration File Options

[Prev](#)[Chapter 3. Configuring Icinga](#)[Next](#)

CGI Configuration File Options

Notes

When creating and/or editing configuration files, keep the following in mind:

1. Lines that start with a '#' character are taken to be comments and are not processed
2. Variables names must begin at the start of the line - no white space is allowed before the name
3. Variable names are case-sensitive

Sample Configuration



Tip

A sample CGI configuration file (`cgi.cfg`) is installed for you when you follow the [quickstart installation guide](#).

Config File Location

By default, Icinga expects the CGI configuration file to be named `cgi.cfg` and located in the config file directory along with the [main config file](#). If you need to change the name of the file or its location, you can configure Apache to pass an environment variable named `ICINGA_CGI_CONFIG` (which points to the correct location) to the CGIs. See the Apache documentation for information on how to do this.

Configuration File Variables

Below you will find descriptions of each main Icinga configuration file option...

Index

- [Main configuration file location](#)
- [Physical HTML path](#)
- [URL HTML path](#)

- URL Stylesheets path
- Authentication usage
- Default user name
- System/process information access
- System/process command access
- Configuration information access
- Global host information access
- Global host command access
- Global service information access
- Global service command access
- Show all services a host is authorized for
- Lock author names
- Status show long plugin output
- Statusmap CGI background image
- Default statusmap layout method
- Statuswrl CGI include world
- Default statuswrl layout method
- CGI refresh rate
- Audio alerts
- Ping syntax
- Escape HTML tags option
- Notes URL target
- Action URL target
- Tac Show Only Hard State
- Splunk integration option
- Splunk URL
- Persistent Acknowledge Comments
- Show Initial States
- Show Current States

- Show Object Type in Tab Title
- Show Service State and Notifications Number
- HTTP character set
- First Day of Week
- CGI Command Logging
- CGI Log File
- CGI Log Rotation Method
- CGI Log Archive Path
- Enforce comments on Actions
- Show Header with Tactical Information
- Show Header with Pending Counts

Main Configuration File Location

Format: **main_config_file=<file_name>**

Example: **main_config_file=/usr/local/icinga/etc/icinga.cfg**

This specifies the location of your [main configuration file](#). The CGIs need to know where to find this file in order to get information about configuration information, current host and service status, etc.

Physical HTML Path

Format: **physical_html_path=<path>**

Example: **physical_html_path=/usr/local/icinga/share**

This is the *physical* path where the HTML files for Icinga are kept on your workstation or server. Icinga assumes that the documentation and images files (used by the CGIs) are stored in subdirectories called *docs/* and *images/*, respectively.

URL HTML Path

Format: **url_html_path=<path>**

Example: **url_html_path=/icinga**

If, when accessing Icinga via a web browser, you point to an URL like <http://www.myhost.com/icinga>, this value should be */icinga*. Basically, its the path portion of the URL that is used to access the Icinga HTML pages.

URL Stylesheet Path

Format: **url_stylesheet_path=<path>/stylesheets**

Example: **url_stylesheet_path=/icinga/stylesheets**

This option allows to define an url stylesheet path other than the default. This will be useful when adding custom stylesheets in another location. If not set, the default location will be used (url_stylesheets_path=url_html_path/stylesheets).

Authentication Usage

Format: **use_authentication=<0/1>**

Example: **use_authentication=1**

This option controls whether or not the CGIs will use the authentication and authorization functionality when determining what information and commands users have access to. We would strongly suggest that you use the authentication functionality for the CGIs. If you decide not to use authentication, make sure to remove the [command CGI](#) to prevent unauthorized users from issuing commands to Icinga. The CGI will not issue commands to Icinga if authentication is disabled, but we would suggest removing it altogether just to be on the safe side. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

- 0 = Don't use authentication functionality
- 1 = Use authentication and authorization functionality (default)

Default User Name

Format: **default_user_name=<username>**

Example: **default_user_name=guest**

Setting this variable will define a default username that can access the CGIs. This allows people within a secure domain (i.e., behind a firewall) to access the CGIs without necessarily having to authenticate to the web server. You may want to use this to avoid having to use basic authentication if you are not using a secure server, as basic authentication transmits passwords in clear text over the Internet.

Important: Do *not* define a default username unless you are running a secure web server and are sure that everyone who has access to the CGIs has been authenticated in some manner! If you define this variable, anyone who has not authenticated to the web server will inherit all rights you assign to this user!

System/Process Information Access

Format: **authorized_for_system_information=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_system_information=icingaadmin,theboss**

This is a comma-delimited list of names of *authenticated users* who can view system/process information in the [extended information CGI](#). Users in this list are *not* automatically authorized to issue system/process commands. If you want users to be able to issue system/process commands as well, you must add them to the [authorized_for_system_commands](#) variable. More information on how to setup authentication and configure authorization for the CGIs can

be found [here](#).

System/Process Command Access

Format: **authorized_for_system_commands=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_system_commands=icingaadmin**

This is a comma-delimited list of names of *authenticated users* who can issue system/process commands via the [command CGI](#). Users in this list are *not* automatically authorized to view system/process information. If you want users to be able to view system/process information as well, you must add them to the [authorized_for_system_information](#) variable. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Configuration Information Access

Format: **authorized_for_configuration_information=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_configuration_information=icingaadmin**

This is a comma-delimited list of names of *authenticated users* who can view configuration information in the [configuration CGI](#). Users in this list can view information on all configured hosts, host groups, services, contacts, contact groups, time periods, and commands. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Global Host Information Access

Format: **authorized_for_all_hosts=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_all_hosts=icingaadmin,theboss**

This is a comma-delimited list of names of *authenticated users* who can view status and configuration information for all hosts. Users in this list are also automatically authorized to view information for all services. Users in this list are *not* automatically authorized to issue commands for all hosts or services. If you want users able to issue commands for all hosts and services as well, you must add them to the [authorized_for_all_host_commands](#) variable. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Global Host Command Access

Format: **authorized_for_all_host_commands=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_all_host_commands=icingaadmin**

This is a comma-delimited list of names of *authenticated users* who can issue commands for all hosts via the [command CGI](#). Users in this list are also automatically authorized to issue commands for all services. Users in this list are *not* automatically authorized to view status or configuration information for all hosts or services. If you want users able to view status and configuration information for all hosts and services as well, you must add them to the [authorized_for_all_hosts](#) variable. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Global Service Information Access

Format: **authorized_for_all_services=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_all_services=icingaadmin,theboss**

This is a comma-delimited list of names of *authenticated users* who can view status and configuration information for all services. Users in this list are *not* automatically authorized to view information for all hosts. Users in this list are *not* automatically authorized to issue commands for all services. If you want users able to issue commands for all services as well, you must add them to the [authorized_for_all_service_commands](#) variable. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Global Service Command Access

Format: **authorized_for_all_service_commands=<user1>,<user2>,<user3>,...<usern>**

Example: **authorized_for_all_service_commands=icingaadmin**

This is a comma-delimited list of names of *authenticated users* who can issue commands for all services via the [command CGI](#). Users in this list are *not* automatically authorized to issue commands for all hosts. Users in this list are *not* automatically authorized to view status or configuration information for all hosts. If you want users able to view status and configuration information for all services as well, you must add them to the [authorized_for_all_services](#) variable. More information on how to setup authentication and configure authorization for the CGIs can be found [here](#).

Show All Services A Host Is Authorized For

Format: **show_all_services_host_is_authorized_for=<0|1>**

Example: **show_all_services_host_is_authorized_for=1**

By default, a user can see all services on a host, if the user is authorized as contact for the host only. By disabling this option, the user must be an authorized contact for the service too in order to view it. Keep in mind that setting the user in [authorized_for_all_services](#) obsoletes this option (included starting with Icinga 1.0.2)

Lock Author Names

Format: **lock_author_names=[0/1]**

Example: **lock_author_names=1**

This option allows you to restrict users from changing the author name when submitting comments, acknowledgements, and scheduled downtime from the web interface. If this option is enabled, users will be unable to change the author name associated with the command request.

- 0 = Allow users to change author names when submitting commands
- 1 = Prevent users from changing author names (default)

Status Show Long Plugin Output

Format: **status_show_long_plugin_output=[0/1]**

Example: **status_show_long_plugin_output=1**

This option allows you to specify the length of status information in output of status.cgi. If you set the value to 1 it shows the full plugin output instead of the first line only. Default value is 0.

- 0 = Display only the first line of plugin output (default)
- 1 = Display the complete plugin output



Note

This option is available starting with Icinga 1.0.3.

Statusmap CGI Background Image

Format: **statusmap_background_image=<image_file>**

Example: **statusmap_background_image=smbbackground.gd2**

This option allows you to specify an image to be used as a background in the [statusmap CGI](#) if you use the user-supplied coordinates layout method. The background image is not be available in any other layout methods. It is assumed that the image resides in the HTML images path (i.e. /usr/local/icinga/share/images). This path is automatically determined by appending "/images" to the path specified by the [physical_html_path](#) directive.



Note

The image file can be in GIF, JPEG, PNG, or GD2 format. However, GD2 format (preferably in uncompressed format) is recommended, as it will reduce the CPU load when the CGI generates the map image.

Default Statusmap Layout Method

Format: **default_statusmap_layout=<layout_number>**

Example: **default_statusmap_layout=4**

This option allows you to specify the default layout method used by the [statusmap CGI](#). Valid options are:

<layout_number> Value	Layout Method
0	User-defined coordinates
1	Depth layers
2	Collapsed tree
3	Balanced tree
4	Circular
5	Circular (Marked Up)
6	Circular (Balloon)

Statuswrl CGI Include World

Format: **statuswrl_include=<vrml_file>**

Example: **statuswrl_include=myworld.wrl**

This option allows you to include your own objects in the generated VRML world. It is assumed that the file resides in the path specified by the [physical_html_path](#) directive.



Note

This file must be a fully qualified VRML world (i.e. you can view it by itself in a VRML browser).

Default Statuswrl Layout Method

Format: **default_statuswrl_layout=<layout_number>**

Example: **default_statuswrl_layout=4**

This option allows you to specify the default layout method used by the [statuswrl CGI](#). Valid options are:

<layout_number> Value	Layout Method
0	User-defined coordinates
2	Collapsed tree
3	Balanced tree
4	Circular

CGI Refresh Rate

Format: **refresh_rate=<rate_in_seconds>**

Example: **refresh_rate=90**

This option allows you to specify the number of seconds between page refreshes for the [status](#), [statusmap](#), and [extinfo](#) CGIs.

Audio Alerts

Formats:

- host_unreachable_sound=<sound_file>**
- host_down_sound=<sound_file>**
- service_critical_sound=<sound_file>**
- service_warning_sound=<sound_file>**
- service_unknown_sound=<sound_file>**

Examples:

- host_unreachable_sound=hostu.wav**
- host_down_sound=hostd.wav**
- service_critical_sound=critical.wav**
- service_warning_sound=warning.wav**
- service_unknown_sound=unknown.wav**

These options allow you to specify an audio file that should be played in your browser if there are problems when you are viewing the [status CGI](#). If there are problems, the audio file for the most critical type of problem will be played. The most critical type of problem is on or more unreachable hosts, while the least critical is one or more services in an unknown state (see the order in the example above). Audio files are assumed to be in the **media/** subdirectory in your HTML directory (i.e. `/usr/local/icinga/share/media`).

Ping Syntax

Format: **ping_syntax=<command>**

Example: **ping_syntax=/bin/ping -n -U -c 5 \$HOSTADDRESS\$**

This option determines what syntax should be used when attempting to ping a host from the WAP interface (using the [statuswml CGI](#)). You must include the full path to the ping binary, along with all required options. The `$HOSTADDRESS$` macro is substituted with the address of the host before the command is executed.

Escape HTML Tags Option

Format: **escape_html_tags=[0/1]**

Example: **escape_html_tags=1**

This option determines whether or not HTML tags in host and service (plugin) output is escaped in the CGIs. If you enable this option, your plugin output will not be able to contain clickable hyperlinks.

Notes URL Target

Format: **notes_url_target=[target]**

Example: **notes_url_target=_blank**

This option determines the name of the frame target that notes URLs should be displayed in. Valid options include *_blank*, *_self*, *_top*, *_parent*, or any other valid target name.

Action URL Target

Format: **action_url_target=[target]**

Example: **action_url_target=_blank**

This option determines the name of the frame target that action URLs should be displayed in. Valid options include *_blank*, *_self*, *_top*, *_parent*, or any other valid target name.

Tac Show Only Hard State

Format: **tac_show_only_hard_state=[0/1]**

Example: **tac_show_only_hard_state=1**

This option allows you to specify if the tactical overview should only show hard states on hosts and services. If you set this option to 1 the tactical overview shows only states on hosts and services. By default disabled, all states will be shown.

Splunk Integration Option

Format: **enable_splunk_integration=[0/1]**

Example: **enable_splunk_integration=1**

This option determines whether integration functionality with Splunk is enabled in the web interface. If enabled, you'll be presented with "Splunk It" links in various places in the CGIs (log file, alert history, host/service detail, etc). Useful if you're trying to research why a particular problem occurred. For more information on Splunk, visit <http://www.splunk.com/>.

Splunk URL

Format: **splunk_url=<path>**

Example: **splunk_url=http://127.0.0.1:8000/**

This option is used to define the base URL to your Splunk interface. This URL is used by the CGIs when creating links if the [enable_splunk_integration](#) option is enabled.

Persistent Acknowledge Comments

Format: **persistent_ack_comments=<0|1>**

Example: **persistent_ack_comments=1**

This option controls the initial state of the check box "persistent comment" to acknowledge problem hosts or services. It can be used to reinstate the behaviour of Nagios 2. Default is "0" to be compatible with Nagios 3.

- 0 = Leave check box "persistent comment" unchecked (default)
- 1 = Set check box "persistent comment"

 **Note**

This option is available starting with Icinga 1.0.3.

Show initial states

Format: **showlog_initial_states=<0|1>**

Example: **showlog_initial_states=1**

This option allows you to specify if initial states of hosts and services should be shown in showlog.cgi.

 **Note**

This option only works if the option "log_initial_states" in icinga.cfg is set to 1.

- 0 = Don't show initial states in showlog.cgi
- 1 = Show initial states in showlog.cgi (default)

 **Note**

This option is available starting with Icinga 1.3.

Show current states

Format: **showlog_current_states=<0|1>**

Example: **showlog_current_states=1**

This option allows you to specify if current states of hosts and services should be shown in showlog.cgi.

 **Note**

This option only works if the option "log_current_states" in icinga.cfg is set to 1.

- 0 = Don't show current states in showlog.cgi
- 1 = Show current states in showlog.cgi (default)

 **Note**

This option is available starting with Icinga 1.3.

Show object type in tab title

Format: **tab_friendly_titles=<0|1>**

Example: **tab_friendly_titles=1**

Activating this option changes the <title> of status.cgi and extinfo.cgi when they refer to a single host, service, or group.

They will then read:

- [Host]
- {HostGroup}
- ServiceDesc @ Host
- (ServiceGroup)

These are easier to read and find if you use (many) tabs in your browser.

- 0 = Don't show object type in tab title
- 1 = Show object type in tab title (default)



Note

This option is available starting with Icinga 1.3.

Show service state and notification number

Format: **add_notif_num_hard=n**

add_notif_num_soft=n

Example: **add_notif_num_hard=28**

Set this to an OR of the service state identifiers for which status.cgi should not only report "attempts" (e.g., "3/3" for a HARD non-OK state with max_check_attempts=3) but also the current notification number ("#0" if no problem notification has been sent yet, etc.). This is helpful to identify services which switched between different non-OK states a lot, or services which have a first_notification_delay set and are "not yet officially" considered in trouble.

Relevant values from include/statusdata.h (look them up *there* if you want to be *really* sure):

```
#define SERVICE_PENDING      1
#define SERVICE_OK           2
#define SERVICE_WARNING      4
#define SERVICE_UNKNOWN      8
#define SERVICE_CRITICAL     16
```

You'll likely want to use add_notif_num_hard=0 (default) or add_notif_num_hard=28 (warn+crit+unknown).

There's an add_notif_num_soft affecting services in a SOFT state for sake of completeness, too.



Note

These options are available starting with Icinga 1.3.

Set HTTP character set

Format: **http_charset=<char-set>**

Example: **http_charset=utf-8**

This is the character set which is sent with HTTP headers. Default is utf-8.



Note

This option is available starting with Icinga 1.3.

Set first day of week

Format: **first_day_of_week=<0|1>**

Example: **first_day_of_week=1**

This sets the first day of the week which is used in several CGI-reports. Default is 0 = sunday. 1 = monday will be suitable for countries adhering to ISO 8601.



Note

This option is available starting with Icinga 1.4.

Logging of CGI commands

Format: **use_logging=<0|1>**

Example: **use_logging=1**

This option enables the logging of CGI commands in a separate log file. Default is 0 = logging disabled, 1 = logging enabled.



Note

This option is available starting with Icinga 1.4.

CGI log file

Format: **cgi_log_file=<file_name>**

Example: **cgi_log_file=/usr/local/icinga/share/log/icinga-cgi.log**

This option sets the location of the CGI log file.

**Note**

This option is available starting with Icinga 1.4.

CGI log file rotation method

Format: **log file rotation method=<d | w | m>**

Example: **cgi_log_rotation_method=d**

This is the rotation method that you would like Icinga to use for your CGI log file. Values are as follows:

- d = Daily (rotate the log at midnight each day - this is the default)
- w = Weekly (rotate the log at midnight on Saturday)
- m = Monthly (rotate the log at midnight on the last day of the month)

**Note**

This option is available starting with Icinga 1.4.

CGI Log Archive Path

Format: **cgi_log_archive_path=<archive directory>**

Example: **cgi_log_archive_path=/usr/local/icinga/share/log**

This is the directory where Icinga should place CGI log files that have been rotated. This option is ignored if you choose to not use the [CGI log rotation](#) functionality.

**Note**

This option is available starting with Icinga 1.4.

Enforce comments on actions

Format: **enforce_comments_on_actions=<0 | 1>**

Example: **enforce_comments_on_actions=1**

Enforces the need to enter a comment on actions entered via the CGIs. 0 = do not enforce comment (default), 1 = enforce comment.

**Note**

The option `use_logging` has to be enabled, otherwise there will be no logging of comments.

**Note**

This option is available starting with Icinga 1.4.

Show header with tactical information

Format: `show_tac_header=<0|1>`

Example: `show_tac_header=0`

There's a new CGI driven top frame that is enabled by default. The new view has a similar feel to the header that is in Icinga-Web and includes important tactical and monitor performance information so you can always be informed of issues.

If you want to keep the old minimalistic look, the new CGI top view can be disabled with the directive: `show_tac_header=0`

The layout for the display of entries for each type is "X / Y / Z" where:

X = Active unacknowledged

Y = Passive unacknowledged

Z = Acknowledged

The X/Y/Z numbers themselves are click-able and will bring you to a list of all the services or hosts related to the respective entry.

Colouring for these entries has the following logical flow:

```
1+ / * / * = Full colour, full saturation
0 / 1+ / * = Full colour but desaturated, rim coloured
0 / 0 / 1+ = Rim coloured, text coloured and bold
0 / 0 / * = Grey
```

These colours are located in `html/stylesheets/tacheader.css` for easy customisation. For example, if your setup is distributed and primarily depends upon passive checks then you may prefer to make passive check issues stand out with the same colouring as active checks.

Figure 3.1. Example of new header



Note

This option is available starting with Icinga 1.4.

Show header including pending counts

Format: `show_tac_header_pending=<0|1>`

Example: `show_tac_header_pending=0`

This option enables the display of pending counts in the tac header. If your display is less than 1024x768 and this is enabled, the tactical information may not fit well in the top frame. By default it is disabled.

**Note**

This option is available starting with Icinga 1.4.1

[Prev](#)[Up](#)[Next](#)[Custom Object Variables](#)[Home](#)[Authentication And Authorization In The CGIs](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Authentication And Authorization In The CGIs

[Prev](#)
[Chapter 3. Configuring Icinga](#)
[Next](#)

Authentication And Authorization In The CGIs

Introduction

This documentation describes how the Icinga CGIs decide who has access to view monitoring and configuration information, and who can submit commands to the Icinga daemon through the web interface.

Definitions

Before continuing, it is important that you understand the meaning of and difference between authenticated users and authenticated contacts:

- An **authenticated user** is someone who has authenticated to the web server with a username and password and has been granted access to the Icinga web interface.
- An **authenticated contact** is an authenticated user whose username matches the short name of a [contact definition](#).

Setting Up Authenticated Users

Assuming you configured your web server as described in the [quickstart guide](#), it should require that you authenticate before accessing the Icinga CGIs. You should also have one user account (*icingaadmin*) that can access the CGIs.

As you define more [contacts](#) for receiving host and service notifications, you'll most likely want to let them access the Icinga web interface. You can use the following command to add additional users who can authenticate to the CGIs. Replace <username> with the actual username you want to add. In most cases, the username should match the short name of a [contact](#) that has been defined.

```
htpasswd /usr/local/icinga/etc/htpasswd.users <username>
```

Enabling Authentication/Authorization Functionality In The CGIs

The next thing you need to do is make sure that the CGIs are configured to use the authentication and authorization functionality in determining what information and/or commands users have access to. This is done by setting the [use_authentication](#) variable in the [CGI configuration file](#) to a non-zero value. Example:

```
use_authentication=1
```

Okay, you're now done with setting up basic authentication/authorization functionality in the CGIs.

Default Permissions To CGI Information

So what default permissions do users have in the CGIs by default when the authentication/authorization functionality is enabled?

CGI Data	Authenticated Contacts *	Other Authenticated Users *
Host Status Information	Yes	No
Host Configuration Information	Yes	No
Host History	Yes	No
Host Notifications	Yes	No
Host Commands	Yes	No
Service Status Information	Yes	No
Service Configuration Information	Yes	No
Service History	Yes	No
Service Notifications	Yes	No
Service Commands	Yes	No
All Configuration Information	No	No
System/Process Information	No	No
System/Process Commands	No	No

*Authenticated contacts ** are granted the following permissions for each **service** for which they are contacts (but not for services for which they are not contacts)...

- Authorization to view service status information
- Authorization to view service configuration information
- Authorization to view history and notifications for the service
- Authorization to issue service commands

*Authenticated contacts ** are granted the following permissions for each **host** for which they are contacts (but not for hosts for which they are not contacts)...

- Authorization to view host status information
- Authorization to view host configuration information
- Authorization to view history and notifications for the host

- Authorization to issue host commands
- Authorization to view status information for all services on the host
- Authorization to view configuration information for all services on the host
- Authorization to view history and notification information for all services on the host
- Authorization to issue commands for all services on the host

It is important to note that by default **no one** is authorized for the following...

- Viewing the raw log file via the [showlog CGI](#)
- Viewing Icinga process information via the [extended information CGI](#)
- Issuing Icinga process commands via the [command CGI](#)
- Viewing host group, contact, contact group, time period, and command definitions via the [configuration CGI](#)

You will undoubtedly want to access this information, so you'll have to assign additional rights for yourself (and possibly other users) as described below...

Granting Additional Permissions To CGI Information

You can grant *authenticated contacts* or other *authenticated users* permission to additional information in the CGIs by adding them to various authorization variables in the [CGI configuration file](#). We realize that the available options don't allow for getting really specific about particular permissions, but its better than nothing..

Additional authorization can be given to users by adding them to the following variables in the CGI configuration file...

- [authorized_for_system_information](#)
- [authorized_for_system_commands](#)
- [authorized_for_configuration_information](#)
- [authorized_for_all_hosts](#)
- [authorized_for_all_host_commands](#)
- [authorized_for_all_services](#)
- [authorized_for_all_service_commands](#)

CGI Authorization Requirements

If you are confused about the authorization needed to access various information in the CGIs, read the *Authorization Requirements* section for each CGI as described [here](#).

Authentication On Secured Web Servers

If your web server is located in a secure domain (i.e., behind a firewall) or if you are using SSL, you can define a default username that can be used to access the CGIs. This is done by defining the [default_user_name](#) option in the [CGI configuration file](#). By defining a default username that can access the CGIs, you can allow users to access the CGIs without necessarily having to

authenticate to the web server. You may want to use this to avoid having to use basic web authentication, as basic authentication transmits passwords in clear text over the Internet.

Important: Do *not* define a default username unless you are running a secure web server and are sure that everyone who has access to the CGIs has been authenticated in some manner. If you define this variable, anyone who has not authenticated to the web server will inherit all rights you assign to this user!

[Prev](#)[Up](#)[Next](#)[CGI Configuration File Options](#)[Home](#)[Chapter 4. Running Icinga](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 4. Running Icinga

[Prev](#)

[Next](#)

Chapter 4. Running Icinga

Table of Contents

[Verifying Your Configuration](#)
[Starting and Stopping Icinga](#)

[Prev](#)

[Next](#)

[Authentication And Authorization
In The CGIs](#)

[Home](#)

[Verifying Your Configuration](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Verifying Your Configuration

[Prev](#)

[Chapter 4. Running Icinga](#)

[Next](#)

Verifying Your Configuration

Every time you modify your configuration files, you also have to restart Icinga. It is important to run a sanity check on your configuration files because in case of an error Icinga will not be (re)started.

In order to verify your configuration, run Icinga using the **-v** command line option:

```
#> /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg
```

If you've forgotten to enter some critical data or misconfigured things, Icinga will show a warning or error message that should point you to the location of the problem.

Error messages generally print out the line in the configuration file that seems to be the source of the problem.

On errors, Icinga will often exit the pre-flight check and return to the command prompt after printing only the first error that it has encountered.

This is done because one error might lead to consecutive errors as the remainder of the configuration data is verified.

If you get any error messages you'll need to go and edit your configuration files to remedy the problem. Warning messages can *generally* be safely ignored, since they are only recommendations and not requirements.

Instead of specifying the paths to binary and config file you can issue

```
#> /etc/init.d/icinga checkconfig
```

which results in just a non-zero return code if your config contains any errors. This might be useful if you want to restart Icinga automatically.

Using

```
#> /etc/init.d/icinga show-errors
```

the command will create a file containing the errors found. If there are any the contents of the file is shown ("show-errors" starting with Icinga 1.0.2).

Once you've verified your configuration files and fixed possible errors you can go ahead with ["Starting and Stopping Icinga"](#).

[Prev](#)

[Up](#)

[Next](#)

[Chapter 4. Running Icinga](#)

[Home](#)

[Starting and Stopping Icinga](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Starting and Stopping Icinga

[Prev](#)

Chapter 4. Running Icinga

[Next](#)

Starting and Stopping Icinga

There are different ways to start, stop, and restart Icinga. Depending on your installation, the most common example will be shown... In case of using the init script make sure you have performed

```
#> make install-init
```

or

```
#> make fullinstall
```



Tip

Always make sure you're [verified your configuration](#) before you (re)start Icinga.

Starting Icinga

1. **Init Script:** The easiest way to start Icinga is by using the init script:

```
#> /etc/init.d/icinga start
```

2. **Manually:** You can start Icinga manually launching the daemon mode with the **-d** command line option:

```
#> /usr/local/icinga/bin/icinga -d /usr/local/icinga/etc/icinga.cfg
```

3. **Debugging Mode:** In the very rare case that Icinga ends silently without any hints in the various log files you can start Icinga by omitting the daemon option:

```
#> /usr/local/icinga/bin/icinga /usr/local/icinga/etc/icinga.cfg
```

This way it is started in the foreground which will lead to a lot of messages scrolling down the screen but it may result in a clue at the very end.

Restarting Icinga

Restarting/reloading is necessary if you modify your configuration files and want those changes to take effect.

1. **Init Script:** The easiest way to reload/restart Icinga is by using the init script:

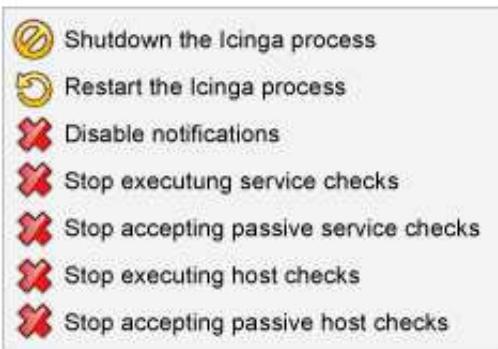
```
#> /etc/init.d/icinga reload
```

or

```
#> /etc/init.d/icinga restart
```

The difference between the two is that the latter will actually end the Icinga process and start it again. The first rereads the configuration files which is faster but in some cases may not be sufficient.

2. **Web-Interface:** You can restart Icinga using the web interface. Click on the "Process Info" navigation link and select "Restart the Icinga Process"



3. **Manually:** You can restart Icinga by sending it a SIGHUP signal:

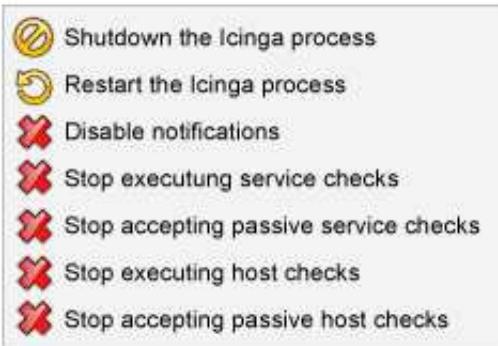
```
# kill -HUP <icinga_pid>
```

Stopping Icinga

1. **Init Script:** The easiest way to stop Icinga is by using the init script:

```
# /etc/init.d/icinga stop
```

2. **Web-Interface:** You can stop Icinga using the Web-Interface Click on the "Process Info" navigation link and select "Shutdown the Icinga Process"



3. **Manually:** You can stop by sending it a SIGTERM signal:

```
# kill <icinga_pid>
```

Logging Options in /usr/local/icinga/etc/icinga.cfg

Logging options for daemon:

If you want messages to be logged to the daemon log file (usually icinga.log). Default option is 1 (yes), the other valid option is 0 (no)

```
use_daemon_log=0/1
```

Logging options for syslog:

If you want messages to be logged to the syslog facility, as well as the Icinga log file set this option to 1. If not, set it to 0.

```
use_syslog=0/1
```

Miscellaneous options

If you experience long delays between starting Icinga and the first checks you can use several other options which are shown [here](#). How to have a look at the scheduling queue (-S) is described there as well.

[Prev](#)

[Up](#)

[Next](#)

[Verifying Your Configuration](#)

[Home](#)

[Chapter 5. The Basics](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 5. The Basics

[Prev](#)

[Next](#)

Chapter 5. The Basics

Table of Contents

[Icinga Plugins](#)

[Understanding Macros and How They Work](#)

[Standard Macros in Icinga](#)

[Host Checks](#)

[Service Checks](#)

[Active Checks](#)

[Passive Checks](#)

[State Types](#)

[Time Periods](#)

[Determining Status and Reachability of Network Hosts](#)

[Notifications](#)

[Prev](#)

[Next](#)

[Starting and Stopping Icinga](#)

[Home](#)

[Icinga Plugins](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Plugins

[Prev](#)

Chapter 5. The Basics

[Next](#)

Icinga Plugins

Introduction

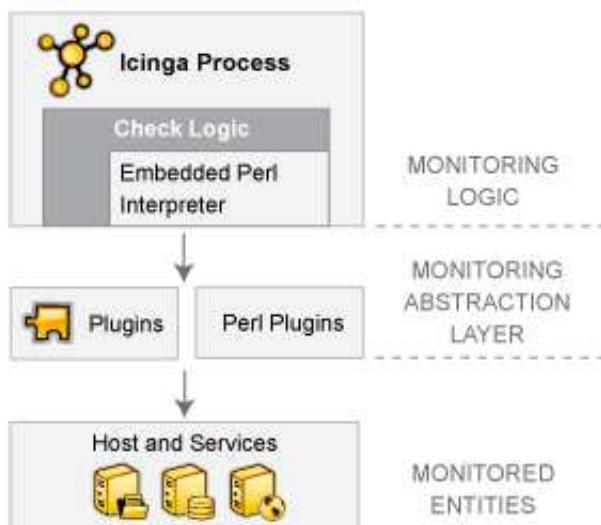
Unlike many other monitoring tools, Icinga does not include any internal mechanisms for checking the status of hosts and services on your network. Instead, Icinga relies on external programs (called plugins) to do all the dirty work.

What Are Plugins?

Plugins are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status of a host or service. Icinga uses the results from plugins to determine the current status of hosts and services on your network.

Icinga will execute a plugin whenever there is a need to check the status of a service or host. The plugin does *something* (notice the very general term) to perform the check and then simply returns the results to Icinga. Icinga will process the results that it receives from the plugin and take any necessary actions (running [event handlers](#), sending out [notifications](#), etc).

Plugins As An Abstraction Layer



Plugins act as an abstraction layer between the monitoring logic present in the Icinga daemon and the actual services and hosts that are being monitored.

The upside of this type of plugin architecture is that you can monitor just about anything you can think of. If you can automate the process of checking something, you can monitor it with Icinga. There are already a lot of plugins that have been created in order to monitor basic resources such as processor load, disk usage, ping rates, etc. If you want to monitor something else, take a look at the documentation on [writing plugins](#) and roll your own. It's simple!

The downside to this type of plugin architecture is the fact that Icinga has absolutely no idea what it is that you're monitoring. You could be monitoring network traffic statistics, data error rates, room temperature, CPU voltage, fan speed, processor load, disk space, or the ability of your super-fantastic toaster to properly brown your bread in the morning... Icinga doesn't understand the specifics of what's being monitored - it just tracks changes in the *state* of those resources. Only the plugins themselves know exactly what they're monitoring and how to perform the actual checks.

What Plugins Are Available?

There are plugins currently available to monitor many different kinds of devices and services, including:

- HTTP, POP3, IMAP, FTP, SSH, DHCP
- CPU Load, Disk Usage, Memory Usage, Current Users
- Unix/Linux, Windows, and Netware Servers
- Routers and Switches
- etc.

Obtaining Plugins

Plugins are not distributed with Icinga, but you can download the official Nagios plugins and many additional plugins created and maintained by Nagios and Icinga users from the following locations:

- Nagios Plugins Project: <http://nagiosplug.sourceforge.net/>
- Nagios Downloads Page: <http://www.nagios.org/download/>
- IcingaExchange.org: <http://www.monitoringexchange.org>

How Do I Use Plugin X?

Most all plugins will display basic usage information when you execute them using '-h' or '--help' on the command line. For example, if you want to know how the check_http plugin works or what options it accepts, you should try executing the following command:

```
$> ./check_http --help
```



Important

Always execute the plugin using the Icinga user because some plugins will create temporary files. If you're testing plugins with another user then the Icinga user may not have the permissions to overwrite existing files later on.

When testing it don't call the plugin using relative paths (i.e. `./check_test_plugin`). Always use absolute paths because that's the way Icinga does it (i.e. `/usr/local/icinga/libexec/check_test_plugin`).

Integrating a new plugin

If you want to integrate a new plugin then please read the documentation (if any) before installing it. It may contain important information about the prerequisites like additional packages, (perl) modules and on how to install the plugin together with hints for your distribution.

Sometimes you have to compile the plugin preparing the compile process using `./configure` with or without options. Please check the file `config.log` for errors regarding missing (devel) packages before issuing the actual compile command (mostly "make" or "make all"). In most cases the plugin is copied to the `plugins` directory (i.e. `/usr/local/icinga/libexec`) during "make install".

Sometimes you have to alter the plugin to reflect your environment (i.e. path to "utils.pm", ...).

After the installation of the plugin call it from the command line using the appropriate options. If this works then you can integrate it into Icinga.

Let's imagine you used the following call on the command line

```
/usr/local/icinga/libexec/sample-plugin.pl -H 192.168.1.2 -a argument1 -p parameter -n 5
```

The command definition has to contain two directives

- `command_name`: this is a short name identifying the command, let's use `check_sample`
- `command_line`: here you define the command to execute. You can specify the command you executed on the command line but that would be very inflexible. Normally the plugin directory (`/usr/local/icinga/libexec`) remains the same so we can use a `$USERn$` variable which is defined in `resource.cfg`. The IP-address changes from host to host. There is a macro called `$HOSTADDRESS$` which we can use for that purpose. The value of the arguments may vary so these should be flexible, too. This may lead to the following definition:

```
define command{
    command_name check_sample
    command_line $USER1$/sample-plugin.pl -H $HOSTADDRESS$ -a $ARG1$ -p $ARG2$ -n $ARG3$
}
```

Then we have to define the `check_command` directive which is part of the host/service definition starting with the short name followed by the arguments separated by exclamation marks (!):

```
check_command check_sample!argument1!parameter!5
```

As you can see the IP-address is not specified because it is taken from the host definition.

In addition to the [macros](#) already mentioned there are a lot of others making life easier. Please note:

- All Icinga macros use all upper case and are enclosed in dollar signs (\$)
- Most macros are only valid for some types of commands. If you try to use a macro for a type of command for which it is not valid you'll get a dollar sign (\$) instead of the expected value
- The `$USERn$` macros can be used to "hide" sensitive information like passwords because these values are not shown in the web interface. Additionally they can be used to specify special characters which otherwise may lead to problems. One example would be `USER99=;`. This way you can use a semicolon which otherwise would be treated as start of a comment within your definitions
- Native non-english speaking persons should note that `$HOSTADDRESS$` is written with a double "D"

Threshold and ranges

Some plugins support specifying ranges for the warning and critical values. Please check the documentation if this is the case for the plugin you want to use. The following is an excerpt from the [developer guidelines](#):

A range is defined as a start and end point (inclusive) on a numeric scale (possibly negative or positive infinity).

A threshold is a range with an alert level (either warning or critical).

The theory is that the plugin will do some sort of check which returns back a numerical value, or metric, which is then compared to the warning and critical thresholds.

This is the generalised format for ranges:

`[@]start:end`

Notes:

1. start = end if :end is not specified
2. start and ":" is not required if start=0
3. if range is of format "start:" and end is not specified, assume end is infinity
4. to specify negative infinity, use "~"
5. alert is raised if metric is outside start and end range (inclusive of endpoints)
6. if range starts with "@", then alert if inside this range (inclusive of endpoints)



Note

Not all plugins are coded to expect ranges in this format yet.

Example ranges

Range definition	Generate an alert if x...
10	< 0 or > 10, (outside the range of {0 .. 10})
10:	< 10, (outside {10 .. infinity})
~:10	> 10, (outside the range of {-infinity .. 10})
10:20	< 10 or > 20, (outside the range of {10 .. 20})
@10:20	<= 10 and >= 20, (inside the range of {10 .. 20})

Command line examples

Command line	Meaning
check_stuff -w10 -c20	Critical if "stuff" is over 20, else warn if over 10 (will be critical if "stuff" is less than 0)
check_stuff -w~:10 -c~:20	Same as above. Negative "stuff" is OK
check_stuff -w10: -c20	Critical if "stuff" is over 20, else warn if "stuff" is below 10 (will be critical if "stuff" is less than 0)
check_stuff -c1:	Critical if "stuff" is less than 1
check_stuff -w~:0 -c10	Critical if "stuff" is above 10; Warn if "stuff" is above zero
check_stuff -c5:6	The only noncritical range is 5:6
check_stuff -c@10:20	Critical if "stuff" is 10 to 20 [1]
check_stuff -w20:30 -c10:40	Warning if "stuff" below 20 or above 30, critical if "stuff" is below 10 or above 40 [2]



Note

[1]: The command line of the developer guidelines seems to be lacking "@" otherwise the meaning would be wrong (and there wouldn't be an example for the @ notation)

[2]: Please note that the last example shows nested ranges. This might not even work with every plugin supporting ranges. It was tested using check_snmp.

Activating the definition

Check the configuration using "/etc/init.d/icinga show-errors" and resolve any errors before issuing "/etc/init.d/icinga restart". Wait until the object is checked and look at the status details. There might be errors.

- "...resulted in a return code of 127"

This means the plugin was not found at the specified location or that a file called from within the plugin was not found. If you use \$USERn\$ macros calling the plugin then make sure that the macro really points to the location where the plugin is (is the macro defined in resource.cfg?). Notification commands often call a mail program. Make sure that the path to the mail program is correct.

- "...resulted in a return code of 13"

Mostly this indicates a permissions problem. The user might not be able to access / execute the plugin and/or other related files. This might happen if you tested a plugin as root which creates temporary files. The Icinga user is not allowed to overwrite these files.

Plugin API

You can find information on the technical aspects of plugins, as well as how to go about creating your own custom plugins [here](#).

[Prev](#)

[Up](#)

[Next](#)

[Chapter 5. The Basics](#)

[Home](#)

[Understanding Macros and How They Work](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Understanding Macros and How They Work

[Prev](#)

Chapter 5. The Basics

[Next](#)

Understanding Macros and How They Work

Macros

One of the main features that make Icinga so flexible is the ability to use macros in command definitions. Macros allow you to reference information from hosts, services, and other sources in your commands.

Macro Substitution - How Macros Work

Before Icinga executes a command, it will replace any macros it finds in the command definition with their corresponding values. This macro substitution occurs for all types of commands that Icinga executes - host and service checks, notifications, event handlers, etc.

Certain macros may themselves contain other macros. These include the \$HOSTNOTES\$, \$HOSTNOTESURL\$, \$HOSTACTIONURL\$, \$SERVICENOTES\$, \$SERVICENOTESURL\$, and \$SERVICEACTIONURL\$ macros.

Example 1: Host Address Macro

When you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. Let's try an example. Assuming we are using a host definition and a *check_ping* command defined like this:

```
define host{
    host_name      linuxbox
    address        192.168.1.2
    check_command  check_ping
    ...
}

define command{
    command_name   check_ping
    command_line   /usr/local/icinga/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%
}
```

the expanded/final command line to be executed for the host's check command would look like this:

```
$> /usr/local/icinga/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%
```

Pretty simple, right? The beauty in this is that you can use a single command definition to check an unlimited number of hosts. Each host can be checked with the same command definition because each host's address is automatically substituted in the command line before execution.

Example 2: Command Argument Macros

You can pass arguments to commands as well, which is quite handy if you'd like to keep your command definitions rather generic. Arguments are specified in the object (i.e. host or service) definition, by separating them from the command name with exclamation marks (!) like so:

```
define service{
    host_name          linuxbox
    service_description PING
    check_command      check_ping!200.0,80%!400.0,40%
    ...
}
```

In the example above, the service check command has two arguments (which can be referenced with `$ARGn$` macros). The `$ARG1$` macro will be "200.0,80%" and `$ARG2$` will be "400.0,40%" (both without quotes). Assuming we are using the host definition given earlier and a `check_ping` command defined like this:

```
define command{
    command_name      check_ping
    command_line      /usr/local/icinga/libexec/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$}
```

the expanded/final command line to be executed for the service's check command would look like this:

```
$> /usr/local/icinga/libexec/check_ping -H 192.168.1.2 -w 200.0,80% -c 400.0,40%
```



Tip

If you need to pass bang (!) characters in your command arguments, you can do so by escaping them with a backslash (\). If you need to include backslashes in your command arguments, they should also be escaped with a backslash.

On-Demand Macros

Normally when you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. For instance, if a host check command is being executed for a host named "linuxbox", all the [standard host macros](#) will refer to values for that host ("linuxbox").

If you would like to reference values for another host or service in a command (for which the command is not being run), you can use what are called "on-demand" macros. On-demand macros look like normal macros, except for the fact that they contain an identifier for the host or service from which they should get their value. Here's the basic format for on-demand macros:

- `$HOSTMACRONAME:host_name$`
- `$SERVICEMACRONAME:host_name:service_description$`

Replace `HOSTMACRONAME` and `SERVICEMACRONAME` with the name of one of the standard host or service macros found [here](#).

Note that the macro name is separated from the host or service identifier by a colon (:). For on-demand service macros, the service identifier consists of both a host name and a service description - these are separated by a colon (:) as well.

**Tip**

On-demand service macros can contain an empty host name field. In this case the name of the host associated with the service will automatically be used.

Examples of on-demand host and service macros follow:

```
$HOSTDOWNTIME:myhost$           <-- On-demand host macro
$SERVICESTATEID:novellserver:DS Database$ <-- On-demand service macro
$SERVICESTATEID::CPU Load$      <-- On-demand service macro with blank host name field
```

On-demand macros are also available for hostgroup, servicegroup, contact, and contactgroup macros. For example:

```
$CONTACTEMAIL:john$           <-- On-demand contact macro
$CONTACTGROUPMEMBERS:linux-admins$ <-- On-demand contactgroup macro
$HOSTGROUPALIAS:linux-servers$    <-- On-demand hostgroup macro
$SERVICEGROUPALIAS:DNS-Cluster$   <-- On-demand servicegroup macro
```

On-Demand Group Macros

You can obtain the values of a macro across all contacts, hosts, or services in a specific group by using a special format for your on-demand macro declaration. You do this by referencing a specific host group, service group, or contact group name in an on-demand macro, like so:

- \$HOSTMACRONAME:*hostgroup_name:delimiter\$*
- \$SERVICEMACRONAME:*servicegroup_name:delimiter\$*
- \$CONTACTMACRONAME:*contactgroup_name:delimiter\$*

Replace *HOSTMACRONAME*, *SERVICEMACRONAME*, and *CONTACTMACRONAME* with the name of one of the standard host, service, or contact macros found [here](#). The delimiter you specify is used to separate macro values for each group member.

For example, the following macro will return a comma-separated list of host state ids for hosts that are members of the *hg1* hostgroup:

```
$HOSTSTATEID:hg1:, $
```

This macro definition will return something that looks like this:

```
0,2,1,1,0,0,2
```

Custom Variable Macros

Any [custom object variables](#) that you define in host, service, or contact definitions are also available as macros. Custom variable macros are named as follows:

- \${HOSTvarname\$}
- \${SERVICEvarname\$}
- \${CONTACTvarname\$}

Take the following host definition with a custom variable called "_MACADDRESS"...

```
define host{
    host_name      linuxbox
    address        192.168.1.1
    _MACADDRESS    00:01:02:03:04:05
    ...
}
```

The `_MACADDRESS` custom variable would be available in a macro called `$HOSTMACADDRESS$`. More information on custom object variables and how they can be used in macros can be found [here](#).

Macro Cleansing

Some macros are stripped of potentially dangerous shell metacharacters before being substituted into commands to be executed. Which characters are stripped from the macros depends on the setting of the `illegal_macro_output_chars` directive. The following macros are stripped of potentially dangerous characters:

1. `$HOSTOUTPUT$`
2. `$LONGHOSTOUTPUT$`
3. `$HOSTPERFDATA$`
4. `$HOSTACKAUTHOR$`
5. `$HOSTACKCOMMENT$`
6. `$SERVICEOUTPUT$`
7. `$LONGSERVICEOUTPUT$`
8. `$SERVICEPERFDATA$`
9. `$SERVICEACKAUTHOR$`
10. `$SERVICEACKCOMMENT$`

Macros as Environment Variables

Most macros are made available as environment variables for easy reference by scripts or commands that are executed by Icinga. For purposes of security and sanity, `$USERn$` and "on-demand" host and service macros are not made available as environment variables.

Environment variables that contain standard macros are named the same as their corresponding macro names (listed [here](#)), with "NAGIOS_" prepended to their names. For example, the `$HOSTNAME$` macro would be available as an environment variable named "NAGIOS_HOSTNAME".

Available Macros

A list of all the macros that are available in Icinga, as well as a chart of when they can be used, can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

Icinga Plugins

[Home](#)

Standard Macros in Icinga

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Standard Macros in Icinga

[Prev](#)

Chapter 5. The Basics

[Next](#)

Standard Macros in Icinga

Standard macros that are available in Icinga are listed here. On-demand macros and macros for custom variables are described [here](#).

Macro Validity

Although macros can be used in all commands you define, not all macros may be "valid" in a particular type of command. For example, some macros may only be valid during service notification commands, whereas other may only be valid during host check commands. There are ten types of commands that Icinga recognizes and treats differently. They are as follows:

1. Service checks
2. Service notifications
3. Host checks
4. Host notifications
5. Service [event handlers](#) and/or a global service event handler
6. Host [event handlers](#) and/or a global host event handler
7. [OCSP](#) command
8. [OCHP](#) command
9. Service [performance data](#) commands
10. Host [performance data](#) commands

The tables below list all macros currently available in Icinga, along with a brief description of each and the types of commands in which they are valid. If a macro is used in a command in which it is invalid, it is replaced with an empty string. It should be noted that macros consist of all uppercase characters and are enclosed in \$ characters.

Macro Availability Chart

Legend:

No	The macro is not available
Yes	The macro is available

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Host Macros: ³								
\$HOSTNAME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTDISPLAYNAME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTALIAS\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTADDRESS\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTADDRESS6\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTSTATE\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$HOSTSTATEID\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTSTATE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTSTATEID\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTSTATETYPE\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$HOSTATTEMPT\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$MAXHOSTATTEMPTS\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTEVENTID\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTEVENTID\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTPROBLEMID\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTPROBLEMID\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTLATENCY\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTEXECUTIONTIME\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$HOSTDURATION\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTDURATIONSEC\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTDOWNTIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTPERCENTCHANGE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPNAME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPNAMES\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTCHECK\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTSTATECHANGE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTUP\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTDOWN\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LASTHOSTUNREACHABLE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTOUTPUT\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$LONGHOSTOUTPUT\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$HOSTPERFDATA\$	Yes	Yes	Yes ¹	Yes	Yes	Yes	Yes	Yes
\$HOSTCHECKCOMMAND\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTACKAUTHOR\$ ⁸	No	No	No	Yes	No	No	No	No
\$HOSTACKAUTHORNAME\$ ⁸	No	No	No	Yes	No	No	No	No
\$HOSTACKAUTHORALIAS\$ ⁸	No	No	No	Yes	No	No	No	No
\$HOSTACKCOMMENT\$ ⁸	No	No	No	Yes	No	No	No	No
\$HOSTACTIONURL\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTNOTESURL\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

\$HOSTNOTESS\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TOTALHOSTSERVICES\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TOTALHOSTSERVICESOK\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TOTALHOSTSERVICESWARNING\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TOTALHOSTSERVICESUNKNOWN\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TOTALHOSTSERVICESCRITICAL\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Host Group Macros:								
\$HOSTGROUPALIAS\$ ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPMEMBERS\$ ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPNOTES\$ ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPNOTESURL\$ ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTGROUPACTIONURL\$ ⁵	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Service Macros:								
\$SERVICEDESC\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEDIPLAYNAMES\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICESTATE\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$SERVICESTATEID\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICESTATE\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICESTATEID\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICESTATETYPE\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEATTEMPT\$	Yes	Yes	No	No	Yes	No	Yes	No
\$MAXSERVICEATTEMPTS\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEISVOLATILE\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEEVENTID\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICEEVENTID\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEPROBLEMID\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICEPROBLEMID\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICELATENCY\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEEXECUTIONTIME\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$SERVICEDURATION\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEDURATIONSEC\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEDOWNTIME\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEPERCENTCHANGE\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEGROUPNAME\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEGROUPNAMES\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICECHECK\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICESTATECHANGE\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICEOK\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICEWARNING\$	Yes	Yes	No	No	Yes	No	Yes	No

\$LASTSERVICEUNKNOWN\$	Yes	Yes	No	No	Yes	No	Yes	No
\$LASTSERVICECRITICAL\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEOUTPUT\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$LONGSERVICEOUTPUT\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$SERVICEPERFDATA\$	Yes ²	Yes	No	No	Yes	No	Yes	No
\$SERVICECHECKCOMMAND\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICEACKAUTHOR\$ ⁸	No	Yes	No	No	No	No	No	No
\$SERVICEACKAUTHORNAME\$ ⁸	No	Yes	No	No	No	No	No	No
\$SERVICEACKAUTHORALIAS\$ ⁸	No	Yes	No	No	No	No	No	No
\$SERVICEACKCOMMENTS\$ ⁸	No	Yes	No	No	No	No	No	No
\$SERVICEACTIONURL\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICENOTESURL\$	Yes	Yes	No	No	Yes	No	Yes	No
\$SERVICENOTES\$	Yes	Yes	No	No	Yes	No	Yes	No
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Service Group Macros:								
\$SERVICEGROUPALIAS\$ ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SERVICEGROUPMEMBERS\$ ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SERVICEGROUPNOTES\$ ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SERVICEGROUPNOTESURL\$ ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SERVICEGROUPACTIONURL\$ ⁶	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Contact Macros:								
\$CONTACTNAME\$	No	Yes	No	Yes	No	No	No	No
\$CONTACTALIAS\$	No	Yes	No	Yes	No	No	No	No
\$CONTACTEMAIL\$	No	Yes	No	Yes	No	No	No	No
\$CONTACTPAGER\$	No	Yes	No	Yes	No	No	No	No
\$CONTACTADDRESSn\$	No	Yes	No	Yes	No	No	No	No
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Contact Group Macros:								
\$CONTACTGROUPALIAS\$ ⁷	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$CONTACTGROUPMEMBERS\$ ⁷	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Summary Macros:								
\$TOTALHOSTSUP\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes

\$TOTALHOSTSDOWN\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALHOSTSUNREACHABLE\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALHOSTSDOWNUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALHOSTSUNREACHABLEUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALHOSTPROBLEMS\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALHOSTPROBLEMSUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESOK\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESWARNING\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESCRITICAL\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESUNKNOWN\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESWARNINGUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESCRITICALUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICESUNKNOWNUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICEPROBLEMS\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
\$TOTALSERVICEPROBLEMSUNHANDLED\$ ¹⁰	Yes	Yes ⁴	Yes	Yes ⁴	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Notification Macros:								
\$NOTIFICATIONTYPE\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONRECIPIENTS\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONISESCALATED\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONAUTHORS\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONAUTHORNAME\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONAUTHORALIAS\$	No	Yes	No	Yes	No	No	No	No
\$NOTIFICATIONCOMMENT\$	No	Yes	No	Yes	No	No	No	No
\$HOSTNOTIFICATIONNUMBER\$	No	Yes	No	Yes	No	No	No	No
\$HOSTNOTIFICATIONID\$	No	Yes	No	Yes	No	No	No	No
\$SERVICENOTIFICATIONNUMBER\$	No	Yes	No	Yes	No	No	No	No
\$SERVICENOTIFICATIONID\$	No	Yes	No	Yes	No	No	No	No
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Date/Time Macros:								
\$LONGDATETIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SHORTDATETIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$DATE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$ISVALIDTIME:\$ ⁹	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$NEXTVALIDTIME:\$ ⁹	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
File Macros:								
\$MAINCONFIGFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$STATUSDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$COMMENTDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes< 5 /td>
\$DOWNTIMEDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$RETENTIONDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$OBJECTCACHEFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TEMPFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$TEMPPATH\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$LOGFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$RESOURCEFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$COMMANDFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$HOSTPERFDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$SERVICEPERFDATAFILE\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers and OCSP	Host Event Handlers and OCHP	Service Perf Data	Host Perf Data
Misc Macros:								
\$PROCESSSTARTTIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$EVENTSTARTTIME\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$ADMINEMAIL\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$ADMINPAGER\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$ARGn\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
\$USERn\$	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Macro Descriptions

Host Macros: ³	
\$HOSTNAME\$	Short name for the host (i.e. "biglinuxbox"). This value is taken from the <i>host_name</i> directive in the host definition .
\$HOSTDISPLAYNAME\$	An alternate display name for the host. This value is taken from the <i>display_name</i> directive in the host definition .
\$HOSTALIAS\$	Long name/description for the host. This value is taken from the <i>alias</i> directive in the host definition .
\$HOSTADDRESS\$	Address of the host. This value is taken from the <i>address</i> directive in the host definition .

\$HOSTADDRESS6\$	Secondary/IPv6 address of the host. This value is taken from the <i>address6</i> directive in the host definition (starting with Icinga 1.3).
\$HOSTSTATE\$	A string indicating the current state of the host ("UP", "DOWN", or "UNREACHABLE").
\$HOSTSTATEID\$	A number that corresponds to the current state of the host: 0=UP, 1=DOWN, 2=UNREACHABLE.
\$LASTHOSTSTATE\$	A string indicating the last state of the host ("UP", "DOWN", or "UNREACHABLE").
\$LASTHOSTSTATEID\$	A number that corresponds to the last state of the host: 0=UP, 1=DOWN, 2=UNREACHABLE.
\$HOSTSTATETYPE\$	A string indicating the state type for the current host check ("HARD" or "SOFT"). Soft states occur when host checks return a non-OK (non-UP) state and are in the process of being retried. Hard states result when host checks have been checked a specified maximum number of times.
\$HOSTATTEMPT\$	The number of the current host check retry. For instance, if this is the second time that the host is being rechecked, this will be the number two. Current attempt number is really only useful when writing host event handlers for "soft" states that take a specific action based on the host retry number.
\$MAXHOSTATTEMPTS\$	The max check attempts as defined for the current host. Useful when writing host event handlers for "soft" states that take a specific action based on the host retry number.
\$HOSTEVENTID\$	A globally unique number associated with the host's current state. Every time a host (or service) experiences a state change, a global event ID number is incremented by one (1). If a host has experienced no state changes, this macro will be set to zero (0).
\$LASTHOSTEVENTID\$	The previous (globally unique) event number that was given to the host.

\$HOSTPROBLEMID\$	A globally unique number associated with the host's current problem state. Every time a host (or service) transitions from an UP or OK state to a problem state, a global problem ID number is incremented by one (1). This macro will be non-zero if the host is currently a non-UP state. State transitions between non-UP states (e.g. DOWN to UNREACHABLE) do not cause this problem id to increase. If the host is currently in an UP state, this macro will be set to zero (0). Combined with event handlers, this macro could be used to automatically open trouble tickets when hosts first enter a problem state.
\$LASTHOSTPROBLEMID\$	The previous (globally unique) problem number that was given to the host. Combined with event handlers, this macro could be used for automatically closing trouble tickets, etc. when a host recovers to an UP state.
\$HOSTLATENCY\$	A (floating point) number indicating the number of seconds that a <i>scheduled</i> host check lagged behind its scheduled check time. For instance, if a check was scheduled for 03:14:15 and it didn't get executed until 03:14:17, there would be a check latency of 2.0 seconds. On-demand host checks have a latency of zero seconds.
\$HOSTEXECUTIONTIME\$	A (floating point) number indicating the number of seconds that the host check took to execute (i.e. the amount of time the check was executing).
\$HOSTDURATION\$	A string indicating the amount of time that the host has spent in its current state. Format is "XXh YYm ZZs", indicating hours, minutes and seconds.
\$HOSTDURATIONSEC\$	A number indicating the number of seconds that the host has spent in its current state.
\$HOSTDOWNTIME\$	A number indicating the current "downtime depth" for the host. If this host is currently in a period of scheduled downtime , the value will be greater than zero. If the host is not currently in a period of downtime, this value will be zero.
\$HOSTPERCENTCHANGE\$	A (floating point) number indicating the percent state change the host has undergone. Percent state change is used by the flap detection algorithm.

\$HOSTGROUPNAME\$	The short name of the hostgroup that this host belongs to. This value is taken from the <i>hostgroup_name</i> directive in the hostgroup definition . If the host belongs to more than one hostgroup this macro will contain the name of just one of them.
\$HOSTGROUPNAMES\$	A comma separated list of the short names of all the hostgroups that this host belongs to.
\$LASTHOSTCHECK\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which a check of the host was last performed.
\$LASTHOSTSTATECHANGE\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time the host last changed state.
\$LASTHOSTUP\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host was last detected as being in an UP state.
\$LASTHOSTDOWN\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host was last detected as being in a DOWN state.
\$LASTHOSTUNREACHABLE\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the host was last detected as being in an UNREACHABLE state.
\$HOSTOUTPUT\$	The first line of text output from the last host check (i.e. "Ping OK").
\$LONGHOSTOUTPUT\$	The full text output (aside from the first line) from the last host check.
\$HOSTPERFDATA\$	This macro contains any performance data that may have been returned by the last host check.
\$HOSTCHECKCOMMAND\$	This macro contains the name of the command (along with any arguments passed to it) used to perform the host check.
\$HOSTACKAUTHOR\$ ⁸	A string containing the name of the user who acknowledged the host problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".

\$HOSTACKAUTHORNAME\$ ⁸	A string containing the short name of the contact (if applicable) who acknowledged the host problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$HOSTACKAUTHORALIAS\$ ⁸	A string containing the alias of the contact (if applicable) who acknowledged the host problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$HOSTACKCOMMENT\$ ⁸	A string containing the acknowledgement comment that was entered by the user who acknowledged the host problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$HOSTACTIONURL\$	Action URL for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be useful when you want to pass the host name to a web page.
\$HOSTNOTESURL\$	Notes URL for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be useful when you want to pass the host name to a web page.
\$HOSTNOTES\$	Notes for the host. This macro may contain other macros (e.g. \$HOSTNAME\$), which can be useful when you want to host-specific status information, etc. in the description.
\$TOTALHOSTSERVICES\$	The total number of services associated with the host.
\$TOTALHOSTSERVICESOK\$	The total number of services associated with the host that are in an OK state.
\$TOTALHOSTSERVICESWARNING\$	The total number of services associated with the host that are in a WARNING state.
\$TOTALHOSTSERVICESUNKNOWN\$	The total number of services associated with the host that are in an UNKNOWN state.
\$TOTALHOSTSERVICESCRITICAL\$	The total number of services associated with the host that are in a CRITICAL state.
Host Group Macros: ⁵	

\$HOSTGROUPALIAS\$ ⁵	The long name / alias of either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the alias directive in the hostgroup definition .
\$HOSTGROUPEMBERS\$ ⁵	A comma-separated list of all hosts that belong to either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro).
\$HOSTGROUPNOTES\$ ⁵	The notes associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the notes directive in the hostgroup definition .
\$HOSTGROUPNOTESURL\$ ⁵	The notes URL associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the notes_url directive in the hostgroup definition .
\$HOSTGROUPACTIONURL\$ ⁵	The action URL associated with either 1) the hostgroup name passed as an on-demand macro argument or 2) the primary hostgroup associated with the current host (if not used in the context of an on-demand macro). This value is taken from the action_url directive in the hostgroup definition .
Service Macros:	
\$SERVICEDESC\$	The long name/description of the service (i.e. "Main Website"). This value is taken from the service_description directive of the service definition .
\$SERVICEDIPLAYNAME\$	An alternate display name for the service. This value is taken from the display_name directive in the service definition .
\$SERVICESTATE\$	A string indicating the current state of the service ("OK", "WARNING", "UNKNOWN", or "CRITICAL").

\$SERVICESTATEID\$	A number that corresponds to the current state of the service: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
\$LASTSERVICESTATE\$	A string indicating the last state of the service ("OK", "WARNING", "UNKNOWN", or "CRITICAL").
\$LASTSERVICESTATEID\$	A number that corresponds to the last state of the service: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN.
\$SERVICESTATETYPE\$	A string indicating the state type for the current service check ("HARD" or "SOFT"). Soft states occur when service checks return a non-OK state and are in the process of being retried. Hard states result when service checks have been checked a specified maximum number of times.
\$SERVICEATTEMPT\$	The number of the current service check retry. For instance, if this is the second time that the service is being rechecked, this will be the number two. Current attempt number is really only useful when writing service event handlers for "soft" states that take a specific action based on the service retry number.
\$MAXSERVICEATTEMPTS\$	The max check attempts as defined for the current service. Useful when writing host event handlers for "soft" states that take a specific action based on the service retry number.
\$SERVICEISVOLATILE\$	Indicates whether the service is marked as being volatile or not: 0 = not volatile, 1 = volatile.
\$SERVICEEVENTID\$	A globally unique number associated with the service's current state. Every time a service (or host) experiences a state change, a global event ID number is incremented by one (1). If a service has experienced no state changes, this macro will be set to zero (0).
\$LASTSERVICEEVENTID\$	The previous (globally unique) event number that given to the service.

\$SERVICEPROBLEMID\$	A globally unique number associated with the service's current problem state. Every time a service (or host) transitions from an OK or UP state to a problem state, a global problem ID number is incremented by one (1). This macro will be non-zero if the service is currently a non-OK state. State transitions between non-OK states (e.g. WARNING to CRITICAL) do not cause this problem id to increase. If the service is currently in an OK state, this macro will be set to zero (0). Combined with event handlers, this macro could be used to automatically open trouble tickets when services first enter a problem state.
\$LASTSERVICEPROBLEMID\$	The previous (globally unique) problem number that was given to the service. Combined with event handlers, this macro could be used for automatically closing trouble tickets, etc. when a service recovers to an OK state.
\$SERVICELATENCY\$	A (floating point) number indicating the number of seconds that a scheduled service check lagged behind its scheduled check time. For instance, if a check was scheduled for 03:14:15 and it didn't get executed until 03:14:17, there would be a check latency of 2.0 seconds.
\$SERVICEEXECUTIONTIME\$	A (floating point) number indicating the number of seconds that the service check took to execute (i.e. the amount of time the check was executing).
\$SERVICEDURATION\$	A string indicating the amount of time that the service has spent in its current state. Format is "XXh YYm ZZs", indicating hours, minutes and seconds.
\$SERVICEDURATIONSEC\$	A number indicating the number of seconds that the service has spent in its current state.
\$SERVICEDOWNTIME\$	A number indicating the current "downtime depth" for the service. If this service is currently in a period of scheduled downtime , the value will be greater than zero. If the service is not currently in a period of downtime, this value will be zero.
\$SERVICEPERCENTCHANGE\$	A (floating point) number indicating the percent state change the service has undergone. Percent state change is used by the flap detection algorithm.

\$SERVICEGROUPNAME\$	The short name of the servicegroup that this service belongs to. This value is taken from the <code>servicegroup_name</code> directive in the servicegroup definition. If the service belongs to more than one servicegroup this macro will contain the name of just one of them.
\$SERVICEGROUPNAMES\$	A comma separated list of the short names of all the servicegroups that this service belongs to.
\$LASTSERVICECHECK\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which a check of the service was last performed.
\$LASTSERVICESTATECHANGE\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time the service last changed state.
\$LASTSERVICEOK\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service was last detected as being in an OK state.
\$LASTSERVICEWARNING\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service was last detected as being in a WARNING state.
\$LASTSERVICEUNKNOWN\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service was last detected as being in an UNKNOWN state.
\$LASTSERVICECRITICAL\$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicating the time at which the service was last detected as being in a CRITICAL state.
\$SERVICEOUTPUT\$	The first line of text output from the last service check (i.e. "Ping OK").
\$LONGSERVICEOUTPUT\$	The full text output (aside from the first line) from the last service check.
\$SERVICEPERFDATA\$	This macro contains any performance data that may have been returned by the last service check.
\$SERVICECHECKCOMMAND\$	This macro contains the name of the command (along with any arguments passed to it) used to perform the service check.

\$SERVICEACKAUTHOR\$ ⁸	A string containing the name of the user who acknowledged the service problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$SERVICEACKAUTHORNAME\$ ⁸	A string containing the short name of the contact (if applicable) who acknowledged the service problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$SERVICEACKAUTHORALIAS\$ ⁸	A string containing the alias of the contact (if applicable) who acknowledged the service problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$SERVICEACKCOMMENT\$ ⁸	A string containing the acknowledgement comment that was entered by the user who acknowledged the service problem. This macro is only valid in notifications where the \$NOTIFICATIONTYPE\$ macro is set to "ACKNOWLEDGEMENT".
\$SERVICEACTIONURL\$	Action URL for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SERVICEDESC\$), which can be useful when you want to pass the service name to a web page.
\$SERVICENOTESURL\$	Notes URL for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SERVICEDESC\$), which can be useful when you want to pass the service name to a web page.
\$SERVICENOTES\$	Notes for the service. This macro may contain other macros (e.g. \$HOSTNAME\$ or \$SERVICESTATE\$), which can be useful when you want to service-specific status information, etc. in the description
Service Group Macros: ⁶	
\$SERVICEGROUPALIAS\$ ⁶	The long name / alias of either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the alias directive in the servicegroup definition .

\$SERVICEGROUPMEMBERS\$ ⁶	A comma-separated list of all services that belong to either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro).
\$SERVICEGROUPNOTES\$ ⁶	The notes associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the <i>notes</i> directive in the servicegroup definition .
\$SERVICEGROUPNOTESURL\$ ⁶	The notes URL associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the <i>notes_url</i> directive in the servicegroup definition .
\$SERVICEGROUPNOTES\$ ⁶	The action URL associated with either 1) the servicegroup name passed as an on-demand macro argument or 2) the primary servicegroup associated with the current service (if not used in the context of an on-demand macro). This value is taken from the <i>action_url</i> directive in the servicegroup definition .
Contact Macros:	
\$CONTACTNAME\$	Short name for the contact (i.e. "jdoe") that is being notified of a host or service problem. This value is taken from the <i>contact_name</i> directive in the contact definition .
\$CONTACTALIAS\$	Long name/description for the contact (i.e. "John Doe") being notified. This value is taken from the <i>alias</i> directive in the contact definition .
\$CONTACTEMAIL\$	Email address of the contact being notified. This value is taken from the <i>email</i> directive in the contact definition .
\$CONTACTPAGER\$	Pager number/address of the contact being notified. This value is taken from the <i>pager</i> directive in the contact definition .

\$CONTACTADDRESSn\$	Address of the contact being notified. Each contact can have six different addresses (in addition to email address and pager number). The macros for these addresses are \$CONTACTADDRESS1\$ - \$CONTACTADDRESS6\$. This value is taken from the <i>addressx</i> directive in the contact definition .
\$CONTACTGROUPNAME\$	The short name of the contactgroup that this contact is a member of. This value is taken from the <i>contactgroup_name</i> directive in the contactgroup definition . If the contact belongs to more than one contactgroup this macro will contain the name of just one of them.
\$CONTACTGROUPNAMES\$	A comma separated list of the short names of all the contactgroups that this contact is a member of.
Contact Group Macros: 5	
\$CONTACTGROUPALIAS\$ 7	The long name / alias of either 1) the contactgroup name passed as an on-demand macro argument or 2) the primary contactgroup associated with the current contact (if not used in the context of an on-demand macro). This value is taken from the <i>alias</i> directive in the contactgroup definition .
\$CONTACTGROUPMEMBERS\$ 7	A comma-separated list of all contacts that belong to either 1) the contactgroup name passed as an on-demand macro argument or 2) the primary contactgroup associated with the current contact (if not used in the context of an on-demand macro).
SUMMARY Macros:	
\$TOTALHOSTSUP\$	This macro reflects the total number of hosts that are currently in an UP state.
\$TOTALHOSTSDOWN\$	This macro reflects the total number of hosts that are currently in a DOWN state.
\$TOTALHOSTSUNREACHABLE\$	This macro reflects the total number of hosts that are currently in an UNREACHABLE state.

\$TOTALHOSTSDOWNUNHANDLED\$	This macro reflects the total number of hosts that are currently in a DOWN state that are not currently being "handled". Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
\$TOTALHOSTSUNREACHABLEUNHANDLED\$	This macro reflects the total number of hosts that are currently in an UNREACHABLE state that are not currently being "handled". Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
\$TOTALHOSTPROBLEMS\$	This macro reflects the total number of hosts that are currently either in a DOWN or an UNREACHABLE state.
\$TOTALHOSTPROBLEMSUNHANDLED\$	This macro reflects the total number of hosts that are currently either in a DOWN or an UNREACHABLE state that are not currently being "handled". Unhandled host problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
\$TOTALSERVICESOK\$	This macro reflects the total number of services that are currently in an OK state.
\$TOTALSERVICESWARNING\$	This macro reflects the total number of services that are currently in a WARNING state.
\$TOTALSERVICESCRITICAL\$	This macro reflects the total number of services that are currently in a CRITICAL state.
\$TOTALSERVICESUNKNOWN\$	This macro reflects the total number of services that are currently in an UNKNOWN state.
\$TOTALSERVICESWARNINGUNHANDLED\$	This macro reflects the total number of services that are currently in a WARNING state that are not currently being "handled". Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.

\$TOTALSERVICESCRITICALUNHANDLED\$	This macro reflects the total number of services that are currently in a CRITICAL state that are not currently being "handled". Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
\$TOTALSERVICESUNKNOWNUNHANDLED\$	This macro reflects the total number of services that are currently in an UNKNOWN state that are not currently being "handled". Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
\$TOTALSERVICEPROBLEMS\$	This macro reflects the total number of services that are currently either in a WARNING, CRITICAL, or UNKNOWN state.
\$TOTALSERVICEPROBLEMSUNHANDLED\$	This macro reflects the total number of services that are currently either in a WARNING, CRITICAL, or UNKNOWN state that are not currently being "handled". Unhandled services problems are those that are not acknowledged, are not currently in scheduled downtime, and for which checks are currently enabled.
Notification Macros:	
\$NOTIFICATIONTYPE\$	A string identifying the type of notification that is being sent ("PROBLEM", "RECOVERY", "ACKNOWLEDGEMENT", "FLAPPINGSTART", "FLAPPINGSTOP", "FLAPPINGDISABLED", "DOWNTIMESTART", "DOWNTIMEEND", or "DOWNTIMECANCELLED").
\$NOTIFICATIONRECIPIENTS\$	A comma-separated list of the short names of all contacts that are being notified about the host or service.
\$NOTIFICATIONISESCALATED\$	An integer indicating whether this was sent to normal contacts for the host or service or if it was escalated. 0 = Normal (non-escalated) notification , 1 = Escalated notification.

\$NOTIFICATIONAUTHOR\$	A string containing the name of the user who authored the notification. If the \$NOTIFICATIONTYPE\$ macro is set to "DOWNTIMESTART" or "DOWNTIMEEND", this will be the name of the user who scheduled downtime for the host or service. If the \$NOTIFICATIONTYPE\$ macro is "ACKNOWLEDGEMENT", this will be the name of the user who acknowledged the host or service problem. If the \$NOTIFICATIONTYPE\$ macro is "CUSTOM", this will be name of the user who initiated the custom host or service notification.
\$NOTIFICATIONAUTHORNAME\$	A string containing the short name of the contact (if applicable) specified in the \$NOTIFICATIONAUTHOR\$ macro.
\$NOTIFICATIONAUTHORALIAS\$	A string containing the alias of the contact (if applicable) specified in the \$NOTIFICATIONAUTHOR\$ macro.
\$NOTIFICATIONCOMMENT\$	A string containing the comment that was entered by the notification author. If the \$NOTIFICATIONTYPE\$ macro is set to "DOWNTIMESTART" or "DOWNTIMEEND", this will be the comment entered by the user who scheduled downtime for the host or service. If the \$NOTIFICATIONTYPE\$ macro is "ACKNOWLEDGEMENT", this will be the comment entered by the user who acknowledged the host or service problem. If the \$NOTIFICATIONTYPE\$ macro is "CUSTOM", this will be comment entered by the user who initiated the custom host or service notification.
\$HOSTNOTIFICATIONNUMBER\$	The current notification number for the host. The notification number increases by one (1) each time a new notification is sent out for the host (except for acknowledgements). The notification number is reset to 0 when the host recovers (<i>after</i> the recovery notification has gone out). Acknowledgements do not cause the notification number to increase, nor do notifications dealing with flap detection or scheduled downtime.

\$HOSTNOTIFICATIONID\$	A unique number identifying a host notification. Notification ID numbers are unique across both hosts and service notifications, so you could potentially use this unique number as a primary key in a notification database. Notification ID numbers should remain unique across restarts of the Icinga process, so long as you have state retention enabled. The notification ID number is incremented by one (1) each time a new host notification is sent out, and regardless of how many contacts are notified.
\$SERVICENOTIFICATIONNUMBER\$	The current notification number for the service. The notification number increases by one (1) each time a new notification is sent out for the service (except for acknowledgements). The notification number is reset to 0 when the service recovers (<i>after</i> the recovery notification has gone out). Acknowledgements do not cause the notification number to increase, nor do notifications dealing with flap detection or scheduled downtime.
\$SERVICENOTIFICATIONID\$	A unique number identifying a service notification. Notification ID numbers are unique across both hosts and service notifications, so you could potentially use this unique number as a primary key in a notification database. Notification ID numbers should remain unique across restarts of the Icinga process, so long as you have state retention enabled. The notification ID number is incremented by one (1) each time a new service notification is sent out, and regardless of how many contacts are notified.
Date/Time Macros:	
\$LONGDATETIME\$	Current date/time stamp (i.e. <i>Fri Oct 13 00:30:28 CDT 2000</i>). Format of date is determined by date_format directive.
\$SHORTDATETIME\$	Current date/time stamp (i.e. <i>10-13-2000 00:30:28</i>). Format of date is determined by date_format directive.
\$DATE\$	Date stamp (i.e. <i>10-13-2000</i>). Format of date is determined by date_format directive.
\$TIME\$	Current time stamp (i.e. <i>00:30:28</i>).

\$TIMET\$	Current time stamp in time_t format (seconds since the UNIX epoch).
\$ISVALIDTIME:\$ ⁹	<p>This is a special on-demand macro that returns a 1 or 0 depending on whether or not a particular time is valid within a specified timeperiod. There are two ways of using this macro:</p> <ol style="list-style-type: none"> 1. \$ISVALIDTIME:24x7\$ will be set to "1" if the current time is valid within the "24x7" timeperiod. If not, it will be set to "0". 2. \$ISVALIDTIME:24x7:timestamp\$ will be set to "1" if the time specified by the "timestamp" argument (which must be in time_t format) is valid within the "24x7" timeperiod. If not, it will be set to "0".
\$NEXTVALIDTIME:\$ ⁹	<p>This is a special on-demand macro that returns the next valid time (in time_t format) for a specified timeperiod. There are two ways of using this macro:</p> <ol style="list-style-type: none"> 1. \$NEXTVALIDTIME:24x7\$ will return the next valid time - from and including the current time - in the "24x7" timeperiod. 2. \$NEXTVALIDTIME:24x7:timestamp\$ will return the next valid time - from and including the time specified by the "timestamp" argument (which must be specified in time_t format) - in the "24x7" timeperiod. <p>If a next valid time cannot be found in the specified timeperiod, the macro will be set to "0".</p>
File Macros:	
\$MAINCONFIGFILE\$	The location of the main config file .
\$STATUSDATAFILE\$	The location of the status data file .
\$COMMENTDATAFILE\$	The location of the comment data file.
\$DOWNTIMEDATAFILE\$	The location of the downtime data file.
\$RETENTIONDATAFILE\$	The location of the retention data file.
\$OBJECTCACHEFILE\$	The location of the object cache file.
\$TEMPFILE\$	The location of the temp file.

\$TEMPPATH\$	The directory specified by the temp path variable.
\$LOGFILE\$	The location of the log file .
\$RESOURCEFILE\$	The location of the resource file .
\$COMMANDFILE\$	The location of the command file .
\$HOSTPERFDATAFILE\$	The location of the host performance data file (if defined).
\$SERVICEPERFDATAFILES\$	The location of the service performance data file (if defined).
Misc Macros:	
\$PROCESSSTARTTIME\$	Time stamp in time_t format (seconds since the UNIX epoch) indicating when the Icinga process was last (re)started. You can determine the number of seconds that Icinga has been running (since it was last restarted) by subtracting \$PROCESSSTARTTIME\$ from \$TIMET\$.
\$EVENTSTARTTIME\$	Time stamp in time_t format (seconds since the UNIX epoch) indicating when the Icinga process starting process events (checks, etc.). You can determine the number of seconds that it took for Icinga to startup by subtracting \$PROCESSSTARTTIME\$ from \$EVENTSTARTTIME\$.
\$ADMINEMAIL\$	Global administrative email address. This value is taken from the admin_email directive.
\$ADMINPAGER\$	Global administrative pager number/address. This value is taken from the admin_pager directive.
\$ARGn\$	The <i>n</i> th argument passed to the command (notification, event handler, service check, etc.). Icinga supports up to 32 argument macros (\$ARG1\$ through \$ARG32\$).
\$USERn\$	The <i>n</i> th user-definable macro. User macros can be defined in one or more resource files . Icinga supports up to 256 user macros (\$USER1\$ through \$USER256\$).

Notes

¹ These macros are not valid for the host they are associated with when that host is being checked (i.e. they make no sense, as they haven't been determined yet).

² These macros are not valid for the service they are associated with when that service is being checked (i.e. they make no sense, as they haven't been determined yet).

³ When host macros are used in service-related commands (i.e. service notifications, event handlers, etc) they refer to the host that the service is associated with.

⁴ When host and service summary macros are used in notification commands, the totals are filtered to reflect only those hosts and services for which the contact is authorized (i.e. hosts and services they are configured to receive notifications for).

⁵ These macros are normally associated with the first/primary hostgroup associated with the current host. They could therefore be considered host macros in many cases. However, these macros are not available as on-demand host macros. Instead, they can be used as on-demand hostgroup macros when you pass the name of a hostgroup to the macro. For example: \$HOSTGROUPMEMBERS:hg1\$ would return a comma-delimited list of all (host) members of the hostgroup *hg1*.

⁶ These macros are normally associated with the first/primary servicegroup associated with the current service. They could therefore be considered service macros in many cases. However, these macros are not available as on-demand service macros. Instead, they can be used as on-demand servicegroup macros when you pass the name of a servicegroup to the macro. For example: \$SERVICEGROUPMEMBERS:sg1\$ would return a comma-delimited list of all (service) members of the servicegroup *sg1*.

⁷ These macros are normally associated with the first/primary contactgroup associated with the current contact. They could therefore be considered contact macros in many cases. However, these macros are not available as on-demand contact macros. Instead, they can be used as on-demand contactgroup macros when you pass the name of a contactgroup to the macro. For example: \$CONTACTGROUPMEMBERS:cg1\$ would return a comma-delimited list of all (contact) members of the contactgroup *cg1*.

⁸ These acknowledgement macros are deprecated. Use the more generic \$NOTIFICATIONAUTHOR\$, \$NOTIFICATIONAUTHORNAME\$, \$NOTIFICATIONAUTHORALIAS\$ or \$NOTIFICATIONCOMMENT\$ macros instead.

⁹ These macros are only available as on-demand macros - e.g. you must supply an additional argument with them in order to use them. These macros are not available as environment variables.

¹⁰ Summary macros are not available as environment variables if the [use_large_installation_tweaks](#) option is enabled, as they are quite CPU-intensive to calculate.

[Prev](#)

[Up](#)

[Next](#)

Understanding Macros and How
They Work

[Home](#)

Host Checks



Host Checks

[Prev](#)

Chapter 5. The Basics

[Next](#)

Host Checks

Introduction

The basic workings of host checks are described here...

When Are Host Checks Performed?

Hosts are checked by the Icinga daemon:

- At regular intervals, as defined by the *check_interval* and *retry_interval* options in your [host definitions](#).
- On-demand when a service associated with the host changes state.
- On-demand as needed as part of the [host reachability](#) logic.
- On-demand as needed for [predictive host dependency checks](#).

Regularly scheduled host checks are optional. If you set the *check_interval* option in your host definition to zero (0), Icinga will not perform checks of the hosts on a regular basis. It will, however, still perform on-demand checks of the host as needed for other parts of the monitoring logic.

On-demand checks are made when a service associated with the host changes state because Icinga needs to know whether the host has also changed state. Services that change state are often an indicator that the host may have also changed state. For example, if Icinga detects that the HTTP service associated with a host just changed from a CRITICAL to an OK state, it may indicate that the host just recovered from a reboot and is now back up and running.

On-demand checks of hosts are also made as part of the [host reachability](#) logic. Icinga is designed to detect network outages as quickly as possible, and distinguish between DOWN and UNREACHABLE host states. These are very different states and can help an admin quickly locate the cause of a network outage.

On-demand checks are also performed as part of the [predictive host dependency check](#) logic. These checks help ensure that the dependency logic is as accurate as possible.

Cached Host Checks

The performance of on-demand host checks can be significantly improved by implementing the use of cached checks, which allow Icinga to forgo executing a host check if it determines a relatively recent check result will do instead. More information on cached checks can be found [here](#).

Dependencies and Checks

You can define [host execution dependencies](#) that prevent Icinga from checking the status of a host depending on the state of one or more other hosts. More information on dependencies can be found [here](#).

Parallelization of Host Checks

Scheduled host checks are run in parallel. When Icinga needs to run a scheduled host check, it will initiate the host check and then return to doing other work (running service checks, etc). The host check runs in a child process that was fork()ed from the main Icinga daemon. When the host check has completed, the child process will inform the main Icinga process (its parent) of the check results. The main Icinga process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

On-demand host checks are also run in parallel if needed. As mentioned earlier, Icinga can forgo the actual execution of an on-demand host check if it can use the cached results from a relatively recent host check.

When Icinga processes the results of scheduled and on-demand host checks, it may initiate (secondary) checks of other hosts. These checks can be initiated for two reasons: [predictive dependency checks](#) and to determine the status of the host using the [network reachability](#) logic. The secondary checks that are initiated are usually run in parallel. However, there is one big exception that you should be aware of, as it can have negative effect on performance...



Note

Hosts which have their `max_check_attempts` value set to **1** can cause serious performance problems. The reason? If Icinga needs to determine their true state using the [network reachability](#) logic (to see if they're DOWN or UNREACHABLE), it will have to launch **serial** checks of all of the host's immediate parents. Just to reiterate, those checks are run *serially*, rather than in parallel, so it can cause a big performance hit. For this reason, we would recommend that you always use a value greater than 1 for the `max_check_attempts` directives in your host definitions.

Host States

Hosts that are checked can be in one of three different states:

- UP
- DOWN
- UNREACHABLE

Host State Determination

Host checks are performed by [plugins](#), which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. How does Icinga translate these plugin return codes into host states of UP, DOWN, or UNREACHABLE? Let's see...

The table below shows how plugin return codes correspond with preliminary host states. Some post-processing (which is described later) is done which may then alter the final host state.

Plugin Result	Preliminary Host State
OK	UP
WARNING	UP or DOWN*
UNKNOWN	DOWN
CRITICAL	DOWN



Note

WARNING results usually means the host is UP. However, WARNING results are interpreted to mean the host is DOWN if the [use_aggressive_host_checking](#) option is enabled.

If the preliminary host state is DOWN, Icinga will attempt to see if the host is really DOWN or if it is UNREACHABLE. The distinction between DOWN and UNREACHABLE host states is important, as it allows admins to determine root cause of network outages faster. The following table shows how Icinga makes a final state determination based on the state of the hosts parent(s). A host's parents are defined in the *parents* directive in host definition.

Preliminary Host State	Parent Host State	Final Host State
DOWN	At least one parent is UP	DOWN
DOWN	All parents are either DOWN or UNREACHABLE	UNREACHABLE

More information on how Icinga distinguishes between DOWN and UNREACHABLE states can be found [here](#).

Host State Changes

As you are probably well aware, hosts don't always stay in one state. Things break, patches get applied, and servers need to be rebooted. When Icinga checks the status of hosts, it will be able to detect when a host changes between UP, DOWN, and UNREACHABLE states and take appropriate action. These state changes result in different [state types](#) (HARD or SOFT), which can trigger [event handlers](#) to be run and [notifications](#) to be sent out. Detecting and dealing with state changes is what Icinga is all about.

When hosts change state too frequently they are considered to be "flapping". A good example of a flapping host would be server that keeps spontaneously rebooting as soon as the operating system loads. That's always a fun scenario to have to deal with. Icinga can detect when hosts start flapping, and can suppress notifications until flapping stops and the host's state stabilizes. More information on the flap detection logic can be found [here](#).

[Prev](#)
[Up](#)
[Next](#)
[Standard Macros in Icinga](#)
[Home](#)
[Service Checks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Service Checks

[Prev](#)

Chapter 5. The Basics

[Next](#)

Service Checks

Introduction

The basic workings of service checks are described here...

When Are Service Checks Performed?

Services are checked by the Icinga daemon:

- At regular intervals, as defined by the `check_interval` and `retry_interval` options in your [service definitions](#).
- On-demand as needed for [predictive service dependency checks](#).

On-demand checks are performed as part of the [predictive service dependency check](#) logic. These checks help ensure that the dependency logic is as accurate as possible. If you don't make use of [service dependencies](#), Icinga won't perform any on-demand service checks.

Cached Service Checks

The performance of on-demand service checks can be significantly improved by implementing the use of cached checks, which allow Icinga to forgo executing a service check if it determines a relatively recent check result will do instead. Cached checks will only provide a performance increase if you are making use of [service dependencies](#). More information on cached checks can be found [here](#).

Dependencies and Checks

You can define [service execution dependencies](#) that prevent Icinga from checking the status of a service depending on the state of one or more other services. More information on dependencies can be found [here](#).

Parallelization of Service Checks

Scheduled service checks are run in parallel. When Icinga needs to run a scheduled service check, it will initiate the service check and then return to doing other work (running host checks, etc). The service check runs in a child process that was fork(ed) from the main Icinga daemon. When the service check has completed, the child process will inform the main Icinga process (its parent) of the check results. The main Icinga process then handles the check results and takes appropriate action (running event handlers, sending notifications, etc.).

On-demand service checks are also run in parallel if needed. As mentioned earlier, Icinga can forgo the actual execution of an on-demand service check if it can use the cached results from a relatively recent service check.

Service States

Services that are checked can be in one of four different states:

- OK
- WARNING
- UNKNOWN
- CRITICAL

Service State Determination

Service checks are performed by [plugins](#), which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. These plugin states directly translate to service states. For example, a plugin which returns a WARNING state will cause a service to have a WARNING state.

Services State Changes

When Icinga checks the status of services, it will be able to detect when a service changes between OK, WARNING, UNKNOWN, and CRITICAL states and take appropriate action. These state changes result in different [state types](#) (HARD or SOFT), which can trigger [event handlers](#) to be run and [notifications](#) to be sent out. Service state changes can also trigger on-demand [host checks](#). Detecting and dealing with state changes is what Icinga is all about.

When services change state too frequently they are considered to be "flapping". Icinga can detect when services start flapping, and can suppress notifications until flapping stops and the service's state stabilizes. More information on the flap detection logic can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

[Host Checks](#)

[Home](#)

[Active Checks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Active Checks

[Prev](#)

Chapter 5. The Basics

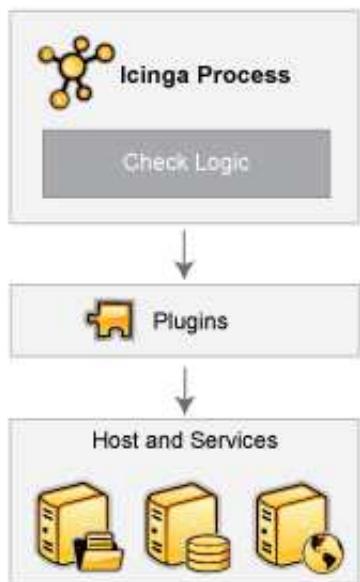
[Next](#)

Active Checks

Introduction

Icinga is capable of monitoring hosts and services in two ways: actively and passively. Passive checks are described [elsewhere](#), so we'll focus on active checks here. Active checks are the most common method for monitoring hosts and services. The main features of active checks are as follows:

- Active checks are initiated by the Icinga process
- Active checks are run on a regularly scheduled basis



How Are Active Checks Performed?

Active checks are initiated by the check logic in the Icinga daemon. When Icinga needs to check the status of a host or service it will execute a plugin and pass it information about what needs to be checked. The plugin will then check the operational state of the host or service and report the results back to the Icinga daemon. Icinga will process the results of the host or service check and take appropriate action as necessary (e.g. send notifications, run event handlers, etc).

More information on how plugins work can be found [here](#).

When Are Active Checks Executed?

Active checks are executed:

- At regular intervals, as defined by the *check_interval* and *retry_interval* options in your host and service definitions
- On-demand as needed

Regularly scheduled checks occur at intervals equaling either the *check_interval* or the *retry_interval* in your host or service definitions, depending on what [type of state](#) the host or service is in. If a host or service is in a HARD state, it will be actively checked at intervals equal to the *check_interval* option. If it is in a SOFT state, it will be checked at intervals equal to the *retry_interval* option.

On-demand checks are performed whenever Icinga sees a need to obtain the latest status information about a particular host or service. For example, when Icinga is determining the [reachability](#) of a host, it will often perform on-demand checks of parent and child hosts to accurately determine the status of a particular network segment. On-demand checks also occur in the [predictive dependency check](#) logic in order to ensure Icinga has the most accurate status information.

[Prev](#)

[Up](#)

[Next](#)

[Service Checks](#)

[Home](#)

[Passive Checks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Passive Checks

[Prev](#)

Chapter 5. The Basics

[Next](#)

Passive Checks

Introduction

In most cases you'll use Icinga to monitor your hosts and services using regularly scheduled [active checks](#). Active checks can be used to "poll" a device or service for status information every so often. Icinga also supports a way to monitor hosts and services passively instead of actively. The key features of passive checks are as follows:

- Passive checks are initiated and performed external applications/processes
- Passive check results are submitted to Icinga for processing

The major difference between active and passive checks is that active checks are initiated and performed by Icinga, while passive checks are performed by external applications.

Uses For Passive Checks

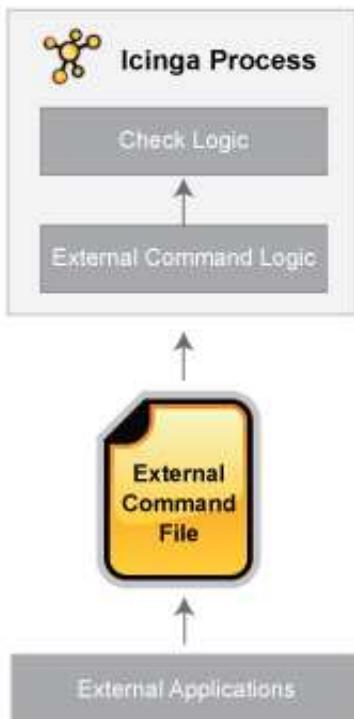
Passive checks are useful for monitoring services that are:

- Asynchronous in nature and cannot be monitored effectively by polling their status on a regularly scheduled basis
- Located behind a firewall and cannot be checked actively from the monitoring host

Examples of asynchronous services that lend themselves to being monitored passively include SNMP traps and security alerts. You never know how many (if any) traps or alerts you'll receive in a given time frame, so it's not feasible to just monitor their status every few minutes.

Passive checks are also used when configuring [distributed](#) or [redundant](#) monitoring installations.

How Passive Checks Work



Here's how passive checks work in more detail...

1. An external application checks the status of a host or service.
2. The external application writes the results of the check to the [external command file](#).
3. The next time Icinga reads the external command file it will place the results of all passive checks into a queue for later processing. The same queue that is used for storing results from active checks is also used to store the results from passive checks.
4. Icinga will periodically execute a [check result reaper event](#) and scan the check result queue. Each service check result that is found in the queue is processed in the same manner - regardless of whether the check was active or passive. Icinga may send out notifications, log alerts, etc. depending on the check result information.

The processing of active and passive check results is essentially identical. This allows for seamless integration of status information from external applications with Icinga.

Enabling Passive Checks

In order to enable passive checks in Icinga, you'll need to do the following:

- Set [accept_passive_service_checks](#) directive is set to 1.
- Set the [passive_checks_enabled](#) directive in your host and service definitions is set to 1.

If you want to disable processing of passive checks on a global basis, set the [accept_passive_service_checks](#) directive to 0.

If you would like to disable passive checks for just a few hosts or services, use the [passive_checks_enabled](#) directive in the host and/or service definitions to do so.

Submitting Passive Service Check Results

External applications can submit passive service check results to Icinga by writing a PROCESS_SERVICE_CHECK_RESULT [external command](#) to the external command file.

The format of the command is as follows:

```
[<timestamp>] PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;<return_code>;<plugin_output>
```

where...

- *timestamp* is the time in time_t format (seconds since the UNIX epoch) that the service check was performed (or submitted). Please note the single space after the right bracket.
- *host_name* is the short name of the host associated with the service in the service definition
- *svc_description* is the description of the service as specified in the service definition
- *return_code* is the return code of the check (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- *plugin_output* is the text output of the service check (i.e. the plugin output)



Note

A service must be defined in Icinga before you can submit passive check results for it! Icinga will ignore all check results for services that had not been configured before it was last (re)started.



Tip

An example shell script of how to submit passive service check results to Icinga can be found in the documentation on [volatile services](#).

Submitting Passive Host Check Results

External applications can submit passive host check results to Icinga by writing a PROCESS_HOST_CHECK_RESULT external command to the external command file.

The format of the command is as follows:

```
[<timestamp>] PROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_output>
```

where...

- *timestamp* is the time in time_t format (seconds since the UNIX epoch) that the host check was performed (or submitted). Please note the single space after the right bracket.
- *host_name* is the short name of the host (as defined in the host definition)
- *host_status* is the status of the host (0=UP, 1=DOWN, 2=UNREACHABLE)
- *plugin_output* is the text output of the host check

**Note**

A host must be defined in Icinga before you can submit passive check results for it! Icinga will ignore all check results for hosts that had not been configured before it was last (re)started.

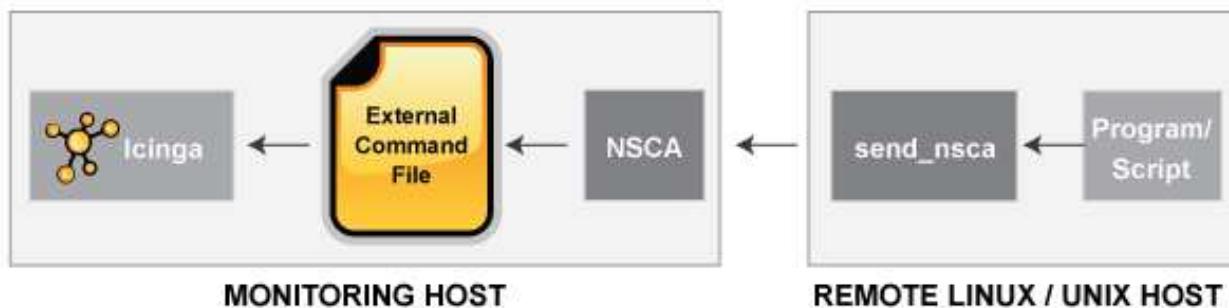
Passive Checks and Host States

Unlike with active host checks, Icinga does not (by default) attempt to determine whether or host is DOWN or UNREACHABLE with passive checks. Rather, Icinga takes the passive check result to be the actual state the host is in and doesn't try to determine the host's actual state using the [reachability logic](#). This can cause problems if you are submitting passive checks from a remote host or you have a [distributed monitoring setup](#) where the parent/child host relationships are different.

You can tell Icinga to translate DOWN/UNREACHABLE passive check result states to their "proper" state by using the [translate_passive_host_checks](#) variable. More information on how this works can be found [here](#).

**Note**

Passive host checks are normally treated as [HARD states](#), unless the [passive_host_checks_are_soft](#) option is enabled.

Submitting Passive Check Results From Remote Hosts

If an application that resides on the same host as Icinga is sending passive host or service check results, it can simply write the results directly to the external command file as outlined above. However, applications on remote hosts can't do this so easily.

In order to allow remote hosts to send passive check results to the monitoring host, the [NSCA](#) addon was developed. The NSCA addon consists of a daemon that runs on the Icinga hosts and a client that is executed from remote hosts. The daemon will listen for connections from remote clients, perform some basic validation on the results being submitted, and then write the check results directly into the external command file (as described above). More information on the NSCA addon can be found [here](#).

[Prev](#)[Up](#)[Next](#)[Active Checks](#)[Home](#)[State Types](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



State Types

[Prev](#)

Chapter 5. The Basics

[Next](#)

State Types

Introduction

The current state of monitored services and hosts is determined by two components:

- The status of the service or host (i.e. OK, WARNING, UP, DOWN, etc.)
- The *type* of state the service or host is in

There are two state types in Icinga - SOFT states and HARD states. These state types are a crucial part of the monitoring logic, as they are used to determine when [event handlers](#) are executed and when [notifications](#) are initially sent out.

This document describes the difference between SOFT and HARD states, how they occur, and what happens when they occur.

Service and Host Check Retries

In order to prevent false alarms from transient problems, Icinga allows you to define how many times a service or host should be (re)checked before it is considered to have a "real" problem. This is controlled by the `max_check_attempts` option in the host and service definitions. Understanding how hosts and services are (re)checked in order to determine if a real problem exists is important in understanding how state types work.

Soft States

Soft states occur in the following situations...

- When a service or host check results in a non-OK or non-UP state and the service check has not yet been (re)checked the number of times specified by the `max_check_attempts` directive in the service or host definition. This is called a soft error.
- When a service or host recovers from a soft error. This is considered a soft recovery.

The following things occur when hosts or services experience SOFT state changes:

- The SOFT state is logged.
- Event handlers are executed to handle the SOFT state.

SOFT states are only logged if you enabled the [log_service_retries](#) or [log_host_retries](#) options in your main configuration file.

The only important thing that really happens during a soft state is the execution of event handlers. Using event handlers can be particularly useful if you want to try and proactively fix a problem before it turns into a HARD state. The `$HOSTSTATETYPE$` or `$SERVICESTATETYPE$` macros will have a value of "SOFT" when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found [here](#).

Hard States

Hard states occur for hosts and services in the following situations:

- When a host or service check results in a non-UP or non-OK state and it has been (re)checked the number of times specified by the `max_check_attempts` option in the host or service definition. This is a hard error state.
- When a host or service transitions from one hard error state to another error state (e.g. WARNING to CRITICAL).
- When a service check results in a non-OK state and its corresponding host is either DOWN or UNREACHABLE.
- When a host or service recovers from a hard error state. This is considered to be a hard recovery.
- When a [passive host check](#) is received. Passive host checks are treated as HARD unless the [passive_host_checks_are_soft](#) option is enabled.

The following things occur when hosts or services experience HARD state changes:

- The HARD state is logged.
- Event handlers are executed to handle the HARD state.
- Contacts are notified of the host or service problem or recovery.

The `$HOSTSTATETYPE$` or `$SERVICESTATETYPE$` macros will have a value of "HARD" when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found [here](#).

Example

Here's an example of how state types are determined, when state changes occur, and when event handlers and notifications are sent out. The table below shows consecutive checks of a service over time. The service has a `max_check_attempts` value of 3.

Time	Check #	State	State Type	State Change	Notes
0	1	OK	HARD	No	Initial state of the service
1	1	CRITICAL	SOFT	Yes	First detection of a non-OK state. Event handlers execute.
2	2	WARNING	SOFT	Yes	Service continues to be in a non-OK state. Event handlers execute.
3	3	CRITICAL	HARD	Yes	Max check attempts has been reached, so service goes into a HARD state. Event handlers execute and a problem notification is sent out. Check # is reset to 1 immediately after this happens.
4	1	WARNING	HARD	Yes	Service changes to a HARD WARNING state. Event handlers execute and a problem notification is sent out.
5	1	WARNING	HARD	No	Service stabilizes in a HARD problem state. Depending on what the notification interval for the service is, another notification might be sent out.
6	1	OK	HARD	Yes	Service experiences a HARD recovery. Event handlers execute and a recovery notification is sent out.
7	1	OK	HARD	No	Service is still OK.
8	1	UNKNOWN	SOFT	Yes	Service is detected as changing to a SOFT non-OK state. Event handlers execute.
9	2	OK	SOFT	Yes	Service experiences a SOFT recovery. Event handlers execute, but notification are not sent, as this wasn't a "real" problem. State type is set HARD and check # is reset to 1 immediately after this happens.
10	1	OK	HARD	No	Service stabilizes in an OK state.

[Prev](#)[Up](#)[Next](#)

Passive Checks

[Home](#)

Time Periods



Time Periods

[Prev](#)

Chapter 5. The Basics

[Next](#)

Time Periods

Introduction



Timeperiod definitions allow you to control when various aspects of the monitoring and alerting logic can operate. For instance, you can restrict:

- When regularly scheduled host and service checks can be performed
- When notifications can be sent out
- When notification escalations can be used
- When dependencies are valid

Precedence in Time Periods

Timeperiod definitions may contain multiple types of directives, including weekdays, days of the month, and calendar dates. Different types of directives have different precedence levels and may override other directives in your timeperiod definitions. The order of precedence for different types of directives (in descending order) is as follows:

- Calendar date (2008-01-01)
- Specific month date (January 1st)
- Generic month date (Day 15)
- Offset weekday of specific month (2nd Tuesday in December)
- Offset weekday (3rd Monday)
- Normal weekday (Tuesday)

Examples of different timeperiod directives can be found [here](#).

How Time Periods Work With Host and Service Checks

Host and service definitions have an optional *check_period* directive that allows you to specify a timeperiod that should be used to restrict when regularly scheduled, active checks of the host or service can be made.

If you do not use the *check_period* directive to specify a timeperiod, Icinga will be able to schedule active checks of the host or service anytime it needs to. This is essentially a 24x7 monitoring scenario.

Specifying a timeperiod in the *check_period* directive allows you to restrict the time that Icinga perform regularly scheduled, active checks of the host or service. When Icinga attempts to reschedule a host or service check, it will make sure that the next check falls within a valid time range within the defined timeperiod. If it doesn't, Icinga will adjust the next check time to coincide with the next "valid" time in the specified timeperiod. This means that the host or service may not get checked again for another hour, day, or week, etc.



Note

On-demand checks and passive checks are not restricted by the timeperiod you specify in the *check_period* directive. Only regularly scheduled active checks are restricted.

Unless you have a good reason not to do so, I would recommend that you monitor all your hosts and services using timeperiods that cover a 24x7 time range. If you don't do this, you can run into some problems during "blackout" times (times that are not valid in the timeperiod definition):

1. The status of the host or service will appear unchanged during the blackout time.
2. Contacts will mostly likely not get re-notified of problems with a host or service during blackout times.
3. If a host or service recovers during a blackout time, contacts will not be immediately notified of the recovery.

How Time Periods Work With Contact Notifications

By specifying a timeperiod in the *notification_period* directive of a host or service definition, you can control when Icinga is allowed to send notifications out regarding problems or recoveries for that host or service. When a host notification is about to get sent out, Icinga will make sure that the current time is within a valid range in the *notification_period* timeperiod. If it is a valid time, then Icinga will attempt to notify each contact of the problem or recovery.

You can also use timeperiods to control when notifications can be sent out to individual contacts. By using the *service_notification_period* and *host_notification_period* directives in [contact definitions](#), you're able to essentially define an "on call" period for each contact. Contacts will only receive host and service notifications during the times you specify in the notification period directives.

Examples of how to create timeperiod definitions for use for on-call rotations can be found [here](#).

How Time Periods Work With Notification Escalations

Service and host [notification escalations](#) have an optional *escalation_period* directive that allows you to specify a timeperiod when the escalation is valid and can be used. If you do not use the *escalation_period* directive in an escalation definition, the escalation is considered valid at all times. If you specify a timeperiod in the *escalation_period* directive, Icinga will only use the escalation definition during times that are valid in the timeperiod definition.

How Time Periods Work With Dependencies

Service and host [dependencies](#) have an optional *dependency_period* directive that allows you to specify a timeperiod when the dependendies are valid and can be used. If you do not use the *dependency_period* directive in a dependency definition, the dependency can be used at any time. If you specify a timeperiod in the *dependency_period* directive, Icinga will only use the dependency definition during times that are valid in the timeperiod definition.

[Prev](#)[Up](#)[Next](#)[State Types](#)[Home](#)[Determining Status and
Reachability of Network Hosts](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Determining Status and Reachability of Network Hosts

[Prev](#)

Chapter 5. The Basics

[Next](#)

Determining Status and Reachability of Network Hosts

Introduction

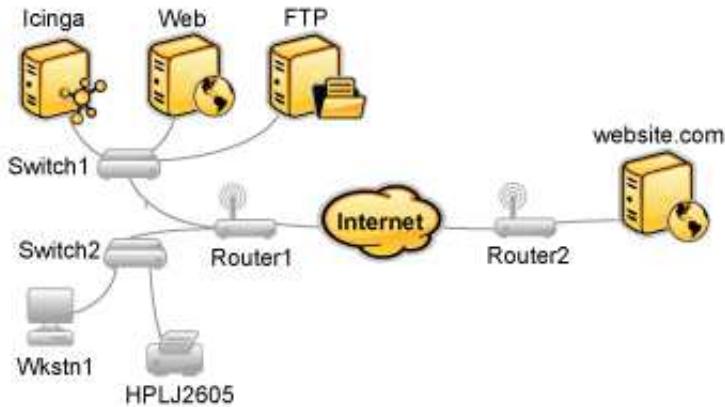
If you've ever work in tech support, you've undoubtably had users tell you "the Internet is down". As a techie, you're pretty sure that no one pulled the power cord from the Internet. Something must be going wrong somewhere between the user's chair and the Internet.

Assuming its a technical problem, you begin to search for the problem. Perhaps the user's computer is turned off, maybe their network cable is unplugged, or perhaps your organization's core router just took a dive. Whatever the problem might be, one thing is most certain - the Internet isn't down. It just happens to be unreachable for that user.

Icinga is able to determine whether the hosts you're monitoring are in a DOWN or UNREACHABLE state. These are very different (although related) states and can help you quickly determine the root cause of network problems. Here's how the reachability logic works to distinguish between these two states...

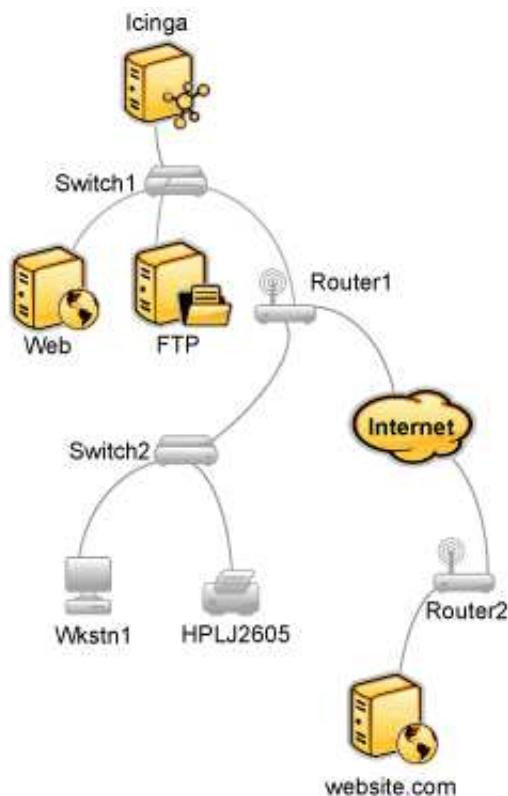
Example Network

Take a look at the simple network diagram below. For this example, lets assume you're monitoring all the hosts (server, routers, switches, etc) that are pictured. Icinga is installed and running on the *Icinga* host.



Defining Parent/Child Relationships

In order for Icinga to be able to distinguish between DOWN and UNREACHABLE states for the hosts that are being monitored, you'll need to tell Icinga how those hosts are connected to each other - from the standpoint of the Icinga daemon. To do this, trace the path that a data packet would take from the Icinga daemon to each individual host. Each switch, router, and server the packet encounters or passes through is considered a "hop" and will require that you define a parent/child host relationship in Icinga. Here's what the host parent/child relationships look like from the viewpoint of Icinga:



Now that you know what the parent/child relationships look like for hosts that are being monitored, how do you configure Icinga to reflect them? The *parents* directive in your [host definitions](#) allows you to do this. Here's what the (abbreviated) host definitions with parent/child relationships would look like for this example:

```
define host{
    host_name      Icinga ; <-- The local host has no parent - it is the topmost host
}

define host{
    host_name      Switch1
    parents        Icinga
}

define host{
    host_name      Web
    parents        Switch1
}

define host{
    host_name      FTP
    parents        Switch1
}

define host{
    host_name      Router1
    parents        Switch1
}
```

```

define host{
    host_name      Switch2
    parents        Router1
}

define host{
    host_name      Wkstn1
    parents        Switch2
}

define host{
    host_name      HPLJ2605
    parents        Switch2
}

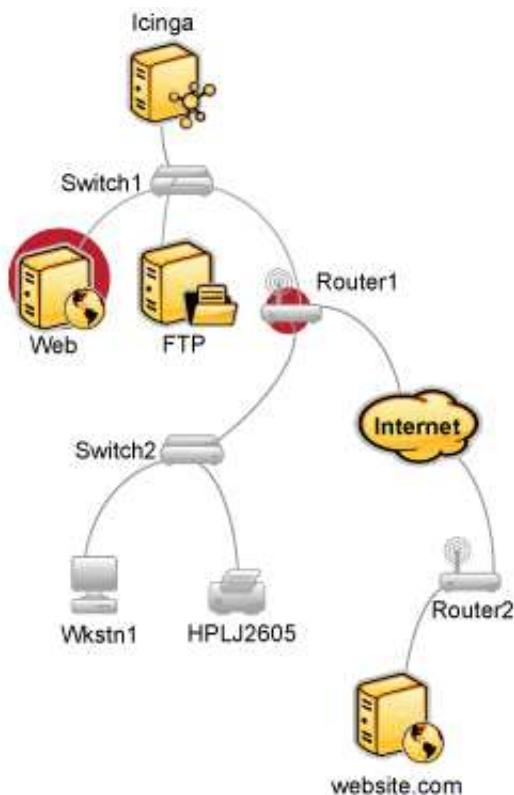
define host{
    host_name      Router2
    parents        Router1
}

define host{
    host_name      somewebsite.com
    parents        Router2
}

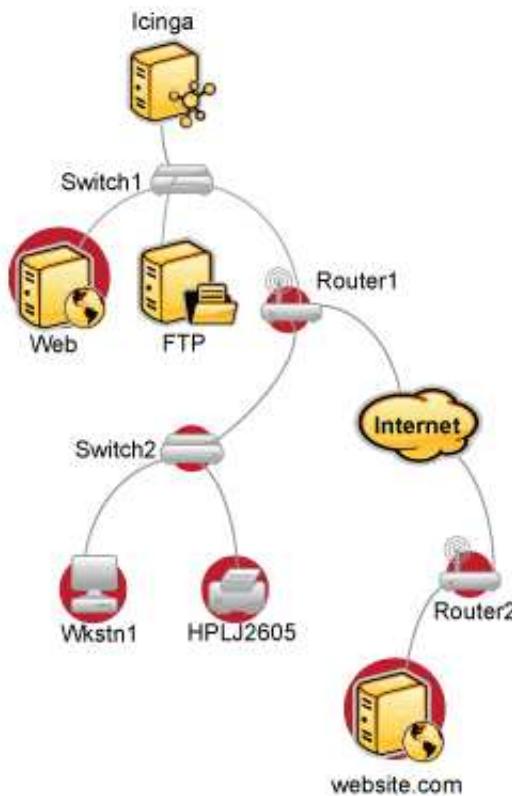
```

Reachability Logic in Action

Now that you're configured Icinga with the proper parent/child relationships for your hosts, let's see what happens when problems arise. Assume that two hosts - *Web* and *Router1* - go offline...



When hosts change state (i.e. from UP to DOWN), the host reachability logic in Icinga kicks in. The reachability logic will initiate parallel checks of the parents and children of whatever hosts change state. This allows Icinga to quickly determine the current status of your network infrastructure when changes occur.



In this example, Icinga will determine that *Web* and *Router1* are both in DOWN states because the "path" to those hosts is not being blocked.

Icinga will determine that all the hosts "beneath" *Router1* are all in an UNREACHABLE state because Icinga can't reach them. *Router1* is DOWN and is blocking the path to those other hosts. Those hosts might be running fine, or they might be offline - Icinga doesn't know because it can't reach them. Hence Icinga considers them to be UNREACHABLE instead of DOWN.

UNREACHABLE States and Notifications

By default, Icinga will notify contacts about both DOWN and UNREACHABLE host states. As an admin/tech, you might not want to get notifications about hosts that are UNREACHABLE. You know your network structure, and if Icinga notifies you that your router/firewall is down, you know that everything behind it is unreachable.

If you want to spare yourself from a flood of UNREACHABLE notifications during network outages, you can exclude the unreachable (u) option from the *notification_options* directive in your [host](#) definitions and/or the *host_notification_options* directive in your [contact](#) definitions.

[Prev](#)
[Up](#)
[Next](#)
[Time Periods](#)
[Home](#)
[Notifications](#)



Notifications

[Prev](#)

Chapter 5. The Basics

[Next](#)

Notifications



Introduction

I've had a lot of questions as to exactly how notifications work. This will attempt to explain exactly when and how host and service notifications are sent out, as well as who receives them.

Notification escalations are explained [here](#).

When Do Notifications Occur?

The decision to send out notifications is made in the service check and host check logic. The calculations for whether a notification is to be sent are only triggered when processing a host or service check corresponding to that notification; they are not triggered simply because the `<notification_interval>` has passed since a previous notification was sent. Host and service notifications occur in the following instances...

- When a hard state change occurs. More information on state types and hard state changes can be found [here](#).
- When a host or service remains in a hard non-OK state and the time specified by the `<notification_interval>` option in the host or service definition has passed since the last notification was sent out (for that specified host or service).

Who Gets Notified?

Each host and service definition has a `<contact_groups>` option that specifies what contact groups receive notifications for that particular host or service. Contact groups can contain one or more individual contacts.

When Icinga sends out a host or service notification, it will notify each contact that is a member of any contact groups specified in the `<contactgroups>` option of the service definition. Icinga realizes that a contact may be a member of more than one contact group, so it removes duplicate contact notifications before it does anything.

What Filters Must Be Passed In Order For Notifications To Be Sent?

Just because there is a need to send out a host or service notification doesn't mean that any contacts are going to get notified. There are several filters that potential notifications must pass before they are deemed worthy enough to be sent out. Even then, specific contacts may not be notified if their notification filters do not allow for the notification to be sent to them. Let's go into the filters that have to be passed in more detail...

Program-Wide Filter:

The first filter that notifications must pass is a test of whether or not notifications are enabled on a program-wide basis. This is initially determined by the `enable_notifications` directive in the main config file, but may be changed during runtime from the web interface. If notifications are disabled on a program-wide basis, no host or service notifications can be sent out - period. If they are enabled on a program-wide basis, there are still other tests that must be passed...

Service and Host Filters:

The first filter for host or service notifications is a check to see if the host or service is in a period of `scheduled downtime`. If it is in a scheduled downtime, **no one gets notified**. If it isn't in a period of downtime, it gets passed on to the next filter. As a side note, notifications for services are suppressed if the host they're associated with is in a period of scheduled downtime.

The second filter for host or service notification is a check to see if the host or service is `flapping` (if you enabled flap detection). If the service or host is currently flapping, **no one gets notified**. Otherwise it gets passed to the next filter.

The third host or service filter that must be passed is the host- or service-specific notification options. Each service definition contains options that determine whether or not notifications can be sent out for warning states, critical states, and recoveries. Similarly, each host definition contains options that determine whether or not notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, **no one gets notified**. If it does pass these options, the notification gets passed to the next filter...



Note

Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem.

The fourth host or service filter that must be passed is the time period test. Each host and service definition has a `<notification_period>` option that specifies which time period contains valid notification times for the host or service. If the time that the notification is being made does not fall within a valid time range in the specified time period, **no one gets contacted**. If it falls within a valid time range, the notification gets passed to the next filter...

**Note**

If the time period filter is not passed, Icinga will reschedule the next notification for the host or service (if its in a non-OK state) for the next valid time present in the time period. This helps ensure that contacts are notified of problems as soon as possible when the next valid time in time period arrives.

The last set of host or service filters is conditional upon two things: (1) a notification was already sent out about a problem with the host or service at some point in the past and (2) the host or service has remained in the same non-OK state that it was when the last notification went out. If these two criteria are met, then Icinga will check and make sure the time that has passed since the last notification went out either meets or exceeds the value specified by the `<notification_interval>` option in the host or service definition. If not enough time has passed since the last notification, **no one gets contacted**. If either enough time has passed since the last notification or the two criteria for this filter were not met, the notification will be sent out! Whether or not it actually is sent to individual contacts is up to another set of filters...

Contact Filters:

At this point the notification has passed the program mode filter and all host or service filters and Icinga starts to notify **all the people it should**. Does this mean that each contact is going to receive the notification? No! Each contact has their own set of filters that the notification must pass before they receive it.

**Note**

Contact filters are specific to each contact and do not affect whether or not other contacts receive notifications.

The first filter that must be passed for each contact are the notification options. Each contact definition contains options that determine whether or not service notifications can be sent out for warning states, critical states, and recoveries. Each contact definition also contains options that determine whether or not host notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, **the contact will not be notified**. If it does pass these options, the notification gets passed to the next filter...

**Note**

Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem...

The last filter that must be passed for each contact is the time period test. Each contact definition has a `<notification_period>` option that specifies which time period contains valid notification times for the contact. If the time that the notification is being made does not fall within a valid time range in the specified time period, **the contact will not be notified**. If it falls within a valid time range, the contact gets notified!

Notification Methods

You can have Icinga notify you of problems and recoveries pretty much anyway you want: pager, cellphone, email, instant message, audio alert, electric shocker, etc. How notifications are sent depend on the **notification commands** that are defined in your **object definition files**.

**Note**

If you install Icinga according to the [quickstart guide](#), it should be configured to send email notifications. You can see the email notification commands that are used by viewing the contents of the following file: `/usr/local/icinga/etc/objects/commands.cfg`.

Specific notification methods (paging, etc.) are not directly incorporated into the Icinga code as it just doesn't make much sense. The "core" of Icinga is not designed to be an all-in-one application. If service checks were embedded in Icinga' core it would be very difficult for users to add new check methods, modify existing checks, etc. Notifications work in a similiar manner. There are a thousand different ways to do notifications and there are already a lot of packages out there that handle the dirty work, so why re-invent the wheel and limit yourself to a bike tire? Its much easier to let an external entity (i.e. a simple script or a full-blown messaging system) do the messy stuff. Some messaging packages that can handle notifications for pagers and cellphones are listed below in the resource section.

Notification Type Macro

When crafting your notification commands, you need to take into account what type of notification is occurring. The `$NOTIFICATIONTYPE$` macro contains a string that identifies exactly that. The table below lists the possible values for the macro and their respective descriptions:

Value	Description
PROBLEM	A service or host has just entered (or is still in) a problem state. If this is a service notification, it means the service is either in a WARNING, UNKNOWN or CRITICAL state. If this is a host notification, it means the host is in a DOWN or UNREACHABLE state.
RECOVERY	A service or host recovery has occurred. If this is a service notification, it means the service has just returned to an OK state. If it is a host notification, it means the host has just returned to an UP state.
ACKNOWLEDGEMENT	This notification is an acknowledgement notification for a host or service problem. Acknowledgement notifications are initiated via the web interface by contacts for the particular host or service.
FLAPPINGSTART	The host or service has just started flapping .
FLAPPINGSTOP	The host or service has just stopped flapping .
FLAPPINGDISABLED	The host or service has just stopped flapping because flap detection was disabled..
DOWNTIMESTART	The host or service has just entered a period of scheduled downtime . Future notifications will be supressed.
DOWNTIMESTOP	The host or service has just exited from a period of scheduled downtime . Notifications about problems can now resume.
DOWNTIMECANCELLED	The period of scheduled downtime for the host or service was just cancelled. Notifications about problems can now resume.

Helpful Resources

There are many ways you could configure Icinga to send notifications out. Its up to you to decide which method(s) you want to use. Once you do that you'll have to install any necessary software and configure notification commands in your config files before you can use them. Here are just a few possible notification methods:

- Email
- Pager
- Phone (SMS)
- WinPopup message
- Yahoo, ICQ, or MSN instant message
- Audio alerts
- etc...

Basically anything you can do from a command line can be tailored for use as a notification command.

If you're looking for an alternative to using email for sending messages to your pager or cellphone, check out these packages. They could be used in conjunction with Icinga to send out a notification via a modem when a problem arises. That way you don't have to rely on email to send notifications out (remember, email may *not* work if there are network problems). I haven't actually tried these packages myself, but others have reported success using them...

- [Gnokii](#) (SMS software for contacting Nokia phones via GSM network)
- [QuickPage](#) (alphanumeric pager software)
- [Sendpage](#) (paging software)
- [SMS Client](#) (command line utility for sending messages to pagers and mobile phones)

If you want to try out a non-traditional method of notification, you might want to mess around with audio alerts. If you want to have audio alerts played on the monitoring server (with synthesized speech), check out [Festival](#). If you'd rather leave the monitoring box alone and have audio alerts played on another box, check out the [Network Audio System \(NAS\)](#) and [rplay](#) projects.

[Prev](#)

[Up](#)

[Next](#)

Determining Status and
Reachability of Network Hosts

[Home](#)

Chapter 6. User Interfaces



Chapter 6. User Interfaces

[Prev](#)[Next](#)

Chapter 6. User Interfaces

Table of Contents

[Icinga Classic UI: Information On The CGIs](#)

[Information On CGI parameters](#)

[Executing CGIs on the command line](#)

[Installation of the Icinga-Web Frontend](#)

[Upgrading Icinga-Web and Icinga-Web Database](#)

[Configuration Overview of Icinga-Web](#)

[Introduction to Icinga-Web](#)

[Introduction to Icinga-Web \(up to 1.2.x\)](#)

[Introduction to Icinga-Web \(>= 1.3.x\)](#)

[Integration of PNP4Nagios into Icinga-Web](#)

[Prev](#)[Next](#)[Notifications](#)[Home](#)

Icinga Classic UI: Information On
The CGIs

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Classic UI: Information On The CGIs

[Prev](#)[Chapter 6. User Interfaces](#)[Next](#)

Icinga Classic UI: Information On The CGIs

Introduction of the Icinga Classic UI

The various CGIs distributed with Icinga core are described here, along with the authorization requirements for accessing and using each CGI. By default the CGIs require that you have authenticated to the web server and are authorized to view any information you are requesting. More information on configuring authorization can be found [here](#).

The CGIs may be controlled using several parameters. More information can be found [here](#).

Index

[Status CGI](#)[Status map CGI](#)[WAP interface CGI](#)[Tactical overview CGI](#)[Network outages CGI](#)[Configuration CGI](#)[Command CGI](#)[Extended information CGI](#)[Event log CGI](#)[Alert history CGI](#)[Notifications CGI](#)[Trends CGI](#)[Availability reporting CGI](#)[Alert histogram CGI](#)

[Alert summary CGI](#)

[Changes in the Classic UI](#)

Status CGI



File Name: **[status.cgi](#)**

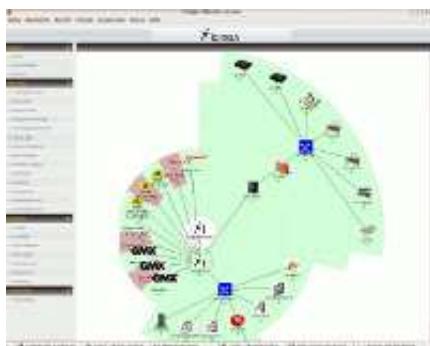
Description:

This is the most important CGI included with Icinga. It allows you to view the current status of all hosts and services that are being monitored. The status CGI can produce two main types of output - a status overview of all host groups (or a particular host group) and a detailed view of all services (or those associated with a particular host).

Authorization Requirements:

- If you are *authorized for all hosts* you can view all hosts **and** all services.
- If you are *authorized for all services* you can view all services.
- If you are an *authenticated contact* you can view all hosts and services for which you are a contact.

Status Map CGI



File Name: **[statusmap.cgi](#)**

Description:

This CGI creates a map of all hosts that you have defined on your network. The CGI uses Thomas Boutell's [gd](#) library (version 1.6.3 or higher) to create a PNG image of your network layout. The coordinates used when drawing each host (along with the optional pretty icons) are taken from [host](#) definitions. If you'd prefer to let the CGI automatically generate drawing coordinates for you, use the [default_statusmap_layout](#) directive to specify a layout algorithm that should be used.

Authorization Requirements:

- If you are *authorized for all hosts* you can view all hosts.
- If you are an *authenticated contact* you can view hosts for which you are a contact.

**Note**

Users who are not authorized to view specific hosts will see *unknown* nodes in those positions. I realize that they really shouldn't see *anything* there, but it doesn't make sense to even generate the map if you can't see all the host dependencies...

WAP Interface CGI

File Name: **statuswml.cgi**

Description:

This CGI serves as a WAP interface to network status information. If you have a WAP-enabled device (i.e. an Internet-ready cellphone), you can view status information while you're on the go. Different status views include hostgroup summary, hostgroup overview, host detail, service detail, all problems, and unhandled problems. In addition to viewing status information, you can also disable notifications and checks and acknowledge problems from your cellphone. Pretty cool, huh?

Authorization Requirements:

- If you are *authorized for system information* you can view Icinga process information.
- If you are *authorized for all hosts* you can view status data for all hosts **and** services.
- If you are *authorized for all services* you can view status data for all services.
- If you are an *authenticated contact* you can view status data for all hosts and services for which you are a contact.

Tactical Overview CGI



File Name: **tac.cgi**

Description:

This CGI is designed to serve as a "birds-eye view" of all network monitoring activity. It allows you to quickly see network outages, host status, and service status. It distinguishes between problems that have been "handled" in some way (i.e. been acknowledged, had notifications disabled, etc.) and those which have not been handled, and thus need attention. Very useful if you've got a lot of hosts/services you're monitoring and you need to keep a single screen up to alert you of problems.

Authorization Requirements:

- If you are *authorized for all hosts* you can view all hosts **and** all services.
- If you are *authorized for all services* you can view all services.
- If you are an *authenticated contact* you can view all hosts and services for which you are a contact.

Network Outages CGI



File Name: **outages.cgi**

Description:

This CGI will produce a listing of "problem" hosts on your network that are causing network outages. This can be particularly useful if you have a large network and want to quickly identify the source of the problem. Hosts are sorted based on the severity of the outage they are causing.

Authorization Requirements:

- If you are *authorized for all hosts* you can view all hosts.
- If you are an *authenticated contact* you can view hosts for which you are a contact.

Configuration CGI

File Name: **config.cgi**

Description:

This CGI allows you to view objects (i.e. hosts, host groups, contacts, contact groups, time periods, services, etc.) that you have defined in your [Object configuration file\(s\)](#).

Authorization Requirements:

- You must be *authorized for configuration information* in order to any kind of configuration information.

Command CGI

File Name: **cmd.cgi**

Description:

This CGI allows you to send commands to the Icinga process. Although this CGI has several arguments, you would be better to leave them alone. Most will change between different revisions of Icinga. Use the [extended information CGI](#) as a starting point for issuing commands.

Authorization Requirements:

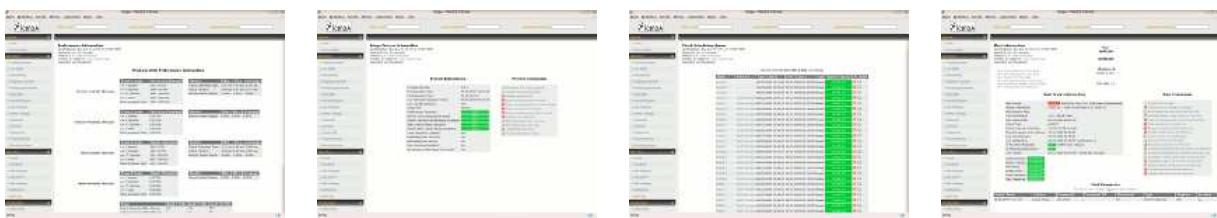
- You must be *authorized for system commands* in order to issue commands that affect the Icinga process (restarts, shutdowns, mode changes, etc.).
- If you are *authorized for all host commands* you can issue commands for all hosts **and** services.
- If you are *authorized for all service commands* you can issue commands for all services.
- If you are an *authenticated contact* you can issue commands for all hosts and services for which you are a contact.



Note

- If you have chosen not to *use authentication* with the CGIs, this CGI will *not* allow anyone to issue commands to Icinga. This is done for your own protection. We would suggest removing this CGI altogether if you decide not to use authentication with the CGIs.

Extended Information CGI



File Name: **extinfo.cgi**

Description:

This CGI allows you to view Icinga process information, host and service state statistics, host and service comments, and more. It also serves as a launching point for sending commands to Icinga via the [command CGI](#). Although this CGI has several arguments, you would be better to leave them alone - they are likely to change between different releases of Icinga. You can access this CGI by clicking on the 'Network Health' and 'Process Information' links on the side navigation bar, or by clicking on a host or service link in the output of the [status CGI](#).

Authorization Requirements:

- You must be *authorized for system information* in order to view Icinga process information.
- If you are *authorized for all hosts* you can view extended information for all hosts **and** services.
- If you are *authorized for all services* you can view extended information for all services.
- If you are an *authenticated contact* you can view extended information for all hosts and services for which you are a contact.

Event Log CGI



File Name: **showlog.cgi**

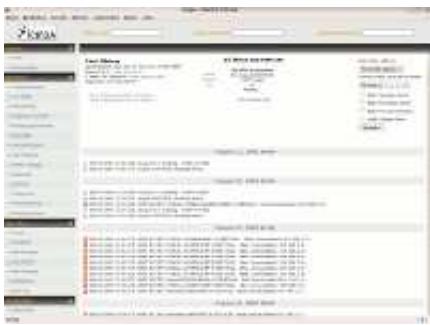
Description:

This CGI will display the [log file](#). If you have [log rotation](#) enabled, you can browse notifications present in archived log files by using the navigational links near the top of the page.

Authorization Requirements:

- You must be [authorized for system information](#) in order to view the log file.

Alert History CGI



File Name: **history.cgi**

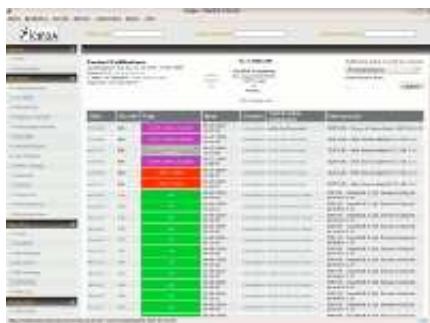
Description:

This CGI is used to display the history of problems with either a particular host or all hosts. The output is basically a subset of the information that is displayed by the [log file CGI](#). You have the ability to filter the output to display only the specific types of problems you wish to see (i.e. hard and/or soft alerts, various types of service and host alerts, all types of alerts, etc.). If you have [log rotation](#) enabled, you can browse history information present in archived log files by using the navigational links near the top of the page.

Authorization Requirements:

- If you are [authorized for all hosts](#) you can view history information for all hosts **and** all services.
- If you are [authorized for all services](#) you can view history information for all services.
- If you are an [authenticated contact](#) you can view history information for all services and hosts for which you are a contact.

Notifications CGI



File Name: **notifications.cgi**

Description:

This CGI is used to display host and service notifications that have been sent to various contacts. The output is basically a subset of the information that is displayed by the [log file CGI](#). You have the ability to filter the output to display only the specific types of notifications you wish to see (i.e. service notifications, host notifications, notifications sent to specific contacts, etc). If you have [log rotation](#) enabled, you can browse notifications present in archived log files by using the navigational links near the top of the page.

Authorization Requirements:

- If you are [authorized for all hosts](#) you can view notifications for all hosts **and** all services.
- If you are [authorized for all services](#) you can view notifications for all services.
- If you are an [authenticated contact](#) you can view notifications for all services and hosts for which you are a contact.

Trends CGI



File Name: **trends.cgi**

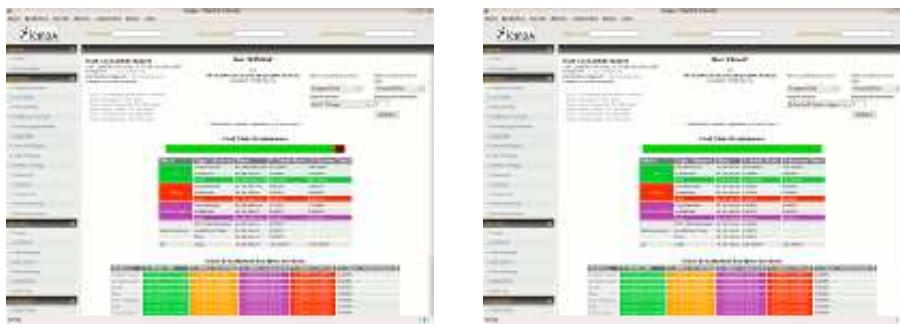
Description:

This CGI is used to create a graph of host or service states over an arbitrary period of time. In order for this CGI to be of much use, you should enable [log rotation](#) and keep archived logs in the path specified by the [log_archive_path](#) directive. The CGI uses Thomas Boutell's [gd](#) library (version 1.6.3 or higher) to create the trends image.

Authorization Requirements:

- If you are *authorized for all hosts* you can view trends for all hosts **and** all services.
- If you are *authorized for all services* you can view trends for all services.
- If you are an *authenticated contact* you can view trends for all services and hosts for which you are a contact.

Availability Reporting CGI



File Name: **avail.cgi**

Description:

This CGI is used to report on the availability of hosts and services over a user-specified period of time. In order for this CGI to be of much use, you should enable [log rotation](#) and keep archived logs in the path specified by the [log_archive_path](#) directive.

Authorization Requirements:

- If you are *authorized for all hosts* you can view availability data for all hosts **and** all services.
- If you are *authorized for all services* you can view availability data for all services.
- If you are an *authenticated contact* you can view availability data for all services and hosts for which you are a contact.

Alert Histogram CGI



File Name: **histogram.cgi**

Description:

This CGI is used to report on the availability of hosts and services over a user-specified period of time. In order for this CGI to be of much use, you should enable [log rotation](#) and keep archived logs in the path specified by the [log_archive_path](#) directive. The CGI uses Thomas Boutell's [gd](#) library (version 1.6.3 or higher) to create the histogram image.

Authorization Requirements:

- If you are [authorized for all hosts](#) you can view histograms for all hosts **and** all services.
- If you are [authorized for all services](#) you can view histograms for all services.
- If you are an [authenticated contact](#) you can view histograms for all services and hosts for which you are a contact.

Alert Summary CGI

File Name: **summary.cgi**

Description:

This CGI provides some generic reports about host and service alert data, including alert totals, top alert producers, etc.

Authorization Requirements:

- If you are [authorized for all hosts](#) you can view summary information for all hosts **and** all services.
- If you are [authorized for all services](#) you can view summary information for all services.
- If you are an [authenticated contact](#) you can view summary information for all services and hosts for which you are a contact.

Changes in the Classic UI

These changes appeared over time so they might not be available in your version of Icinga.

- The appearance of the "General" section on the left side of the screen has changed once more

General CGI



File Name: **general.cgi**

Clicking on the appropriate flag you are still able to access the documentation in the specified language.



Note

There is no option to change the language of the CGIs. This requires changes in the source code.

Now you can search again for hosts without clicking on several items.

- The classical interface refreshes in regular intervals. Sometimes this might be undesirable, i.e. when you are looking at a specific object. In this case you can turn off the automatic refresh clicking on [pause] next to "Updated every 90 seconds" in the upper left of the status frame. Clicking on [continue] restarts the refresh.

Pause CGI

Updated every 90 seconds [pause]

File Name: **pause.cgi**

Continue CGI

Update is paused [continue]

File Name: **continue.cgi**

- The "Host Detail" and "Service Detail" pages were enhanced to submit commands for more than one object (starting with Icinga 1.2). Now you can select one or more objects using the check boxes next to the objects. Activating the check box next to "Status information" selects all services for that host.

Statusinfo CGI

Status Information	
OK - load average: 0.28, 0.55, 1.15	<input type="checkbox"/>
USERS OK - 3 users currently logged in	<input type="checkbox"/>
HTTP WARNING: HTTP/1.1 403 Forbidden	<input type="checkbox"/>
PING OK - Packet loss = 0%, RTA = 0.16 ms	<input type="checkbox"/>

File Name: **statusinfo.cgi**

Commands CGI

Commands for checked services

Select command	Submit
----------------	--------

File Name: **commands.cgi**

Clicking on "Select command" shows a drop down list of available commands. After selecting an action you can press "Submit" to submit the command for the selected objects.

- "Export to CSV" has been added to several pages (starting with Icinga 1.2).
- The cells of the extinfo.cgi table have been given names. Using SSI fragments you can include JavaScript code to access the data of these cells (starting with Icinga 1.2.1).

Example code has been provided to use data from the cell "comment_data" to create a link (Thanks to Oliver Graf).

common-header.ssi:

```
<script type='text/javascript'>
function urlify() {
    var comments=document.getElementsByName('comment_data');
    var neu="";
    var neu="";
    for (i=0; i<comments.length; i++) {
        comments[i].innerHTML = comments[i].innerHTML.replace(/(\bRT#(\d+)\b)/g,"<a href='https://YOUR-SERVER/Ticket/Display.html?id=$1'>RT#$1</a>");
    }
}
window.onload=urlify;
</script>
```

[Prev](#)

[Up](#)

[Next](#)

[Chapter 6. User Interfaces](#)

[Home](#)

[Information On CGI parameters](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Information On CGI parameters

[Prev](#)

Chapter 6. User Interfaces

[Next](#)

Information On CGI parameters

Introduction

The menu on the left side of the classical web interface contains entries which offer a quick way to the information most people need. However you can change the parameters or use other options as well. Some CGIs require an objecttype ("host", "hostgroup", "service", or "servicegroup"), often followed by one or more options. The best way is to take a look at the existing URLs and change them according to your needs.

Below you will find a table showing the parameters and the CGIs they apply to. The names of the CGIs are abbreviated. An explanation along with a hint to the source code can be found [here](#).

Following this table is an explanation of the parameters (work in progress).

Executing CGIs on the command line

Instead of using your browser you may want to run the CGIs on the command line and process the results with other tools. For a first impression change to the folder containing the *.cgi modules (e.g. /usr/local/icinga/sbin), set some environment variables and execute one of the CGIs:

```
$> export REMOTE_USER=icingaadmin    # or choose the appropriate user
$> export REQUEST_METHOD='GET'      # if you enter queries
$> export QUERY_STRING='host=all'   # see the tables below

$> ./status.cgi
```

Depending on the number of hosts this may return a lot of lines containing HTML code in between which most people will find hard to read so you should add "&csvoutput" or "&jsonoutput" to QUERY_STRING along with other arguments.

Set QUERY_STRING as needed and execute the desired CGI.

If you forgot to set the environment variables you'll receive the following lines:

```
$> ./status.cgi
getcgivars(): Unsupported REQUEST_METHOD -> ''
```

I'm guessing you're trying to execute the CGI from a command line. In order to do that, you need to set the REQUEST_METHOD environment variable to either "GET", "HEAD", or "POST". When using the GET and HEAD methods, arguments can be passed to the CGI by setting the "QUERY_STRING" environment variable. If you're using the POST method, data is read from standard input. Also of note: if you've enabled authentication in the CGIs, you must set the "REMOTE_USER" environment variable to be the name of the user you're "authenticated" as.

You will find some examples [here](#).

Matrix Parameters / CGIs

Parameter	avail	cmd	config	ext	hgram	hist	notif	out	log	status	map	sum	tac	trends
ahas		X												
alerttypes												X		
archive						X	X		X					
assumeinitialstates	X													X
assumestateretention	X				X									X
assumestatesduringnotrunning	X													X
backtrack	X				X									X
breakdown					X									
broadcast_notification		X												
childoptions		X												
cmd_mod		X												
cmd_typ		X												
columns										X				
com_author		X												
com_data		X												
com_id		X												
contact							X							
createimage					X					X				X
csvoutput	X	[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]	[2]		[2]	[2]	[2]
displaytype												X		
down_id		X												
eday	X				X							X		X
ehour	X				X							X		X
embedded	X			X	X	X	X	X	X	X	X	X	X	
emin	X				X							X		X
emon	X				X							X		X
end_time		X												
esec	X				X							X		X
eyear	X				X							X		X
fixed		X												

Parameter	avail	cmd	config	ext	hgram	hist	notif	out	log	status	map	sum	tac	trends
force_check		X												
force_notification		X												
full_log_entries	X													
get_date_parts	X													
graphevents					X									
graphstatetypes					X									
host	X	X		X	X	X	X			X		X		X
hostgroup	X	X		X						X		X		
hostprops										X				
hoststates												X		
hoststatustypes	X									X				
hours		X												
includesoftstates	X													
initialassumedhoststate	X													X
initialassumedservicestate	X													X
initialstateslogged					X									
input					X									X
jsonoutput [2][3]	X		X	X			X	X	X	X		X	X	
limit												X		
minutes		X												
navbarsearch										X				
newstatesonly					X									
nodowntime						X								
noflapping						X								
nofrills						X			X					
noheader	X			X	X	X	X	X	X	X	X	X	X	X
not_dly		X												
notimebreaks						X			X					
nosystem						X								
oldestfirst						X	X		X					
performance_data		X												
persistent		X												
plugin_output		X												
plugin_state		X												
ptc		X												
report												X		
report_type	X													
rpttimeperiod	X													
sched_dly		X												
sday	X				X							X		
send_notification		X												

Parameter	avail	cmd	config	ext	hgram	hist	notif	out	log	status	map	sum	tac	trends
service	X	X		X	X	X	X			X				X
servicefilter										X				
servicegroup	X	X		X						X		X		
serviceprops										X				
servicestates												X		
servicestatusypes										X				
service_divisor								X						
shour	X				X							X		
showscheduledowntime	X													
show_log_entries	X													
smin	X				X							X		
smon	X				X							X		
sortoption				X						X				
sorttype				X						X				
ssec	X				X							X		
standardreport												X		
start_time		X												
statetype						X								
statetypes												X		
sticky_ack		X												
style										X				
syear	X				X							X		X
t1	X				X							X		X
t2	X				X							X		X
timeperiod	X				X							X		X
trigger		X												
type			X	X		X	X							

Parameter Details

Further information regarding the several parameters is given below. For a detailed understanding please have a look at the source code.

Parameter	Description	Possible Values	Example	Notes
ahas	The command will affects host and its services		ahas	
alerttypes	Show host and/or service alerts	1=host alerts; 2=service alerts; 3=host and service alerts	alerttypes=3	
archive		0-n		
assumeinitialstates		yes; no		
assumestatesduringnotrunning		yes; no		
assumestateretention		yes; no		

Parameter	Description	Possible Values	Example	Notes
backtrack	How many archive log files will be searched to determine the initial states	0-n	backtrack=1	Please keep in mind that processing of the files may take a while
breakdown	Breakdown data by timeperiod	0=monthly; 1=day of month; 2=day of week; 3=hourly	breakdown=2	
broadcast_notification	Send notification to all contacts (non-escalated and escalated)		broadcast_notification	
childoptions	Downtime handling for child hosts	0=doesn't affect child hosts; 1=schedule triggered downtime; 2=schedule non-triggered downtime	childoptions=1	
cmd_mod	Command mode			
cmd_typ	Command type	0 - 169, 999	cmd_typ=160	For details see .../include/common.h
columns	Number of overview columns	>= 1		Default is 3
com_author	Comment author	a valid user	com_author=icingadmin	May depend on the setting of "lock_author_names" in cgi.cfg
com_data	Comment contents	an urlencoded string		
com_id	Comment id			
contact	A valid contact as mail recipient			
createimage			createimage	
csvoutput	Specify whether or not the report output should be in csv format	yes;no		This option automatically sets "noheader". Added to nearly all CGIs (see note [2])
displaytype	Type of alert report	1=recent alerts; 2=alert totals; 3=top alerts; 4=hostgroup alert totals; 5=host alert totals; 6=service alert totals; 7=servicegroup alert totals		
down_id	Downtime id			
eday	End of custom timeperiod (day)			Only valid when using "custom" timeperiod
ehour	End of custom timeperiod (hour)			Only valid when using "custom" timeperiod
embedded	Omit some HTML code and SSI header/footer		embedded	
emin	End of custom timeperiod (minute)			Only valid when using "custom" timeperiod
emon	End of custom timeperiod (month)			Only valid when using "custom" timeperiod
end_time	End time of the fixed downtime			Format "MM-DD-YYYY HH:MI"

Parameter	Description	Possible Values	Example	Notes
<code>esec</code>	End of custom timeperiod (second)			Only valid when using "custom" timeperiod
<code>eyear</code>	End of custom timeperiod (year)			Only valid when using "custom" timeperiod
<code>fixed</code>	Whether or not the downtime will be fixed	0=flexible, >0=fixed		
<code>force_check</code>	The service check will be forced		forcecheck	
<code>force_notification</code>	Send notification regardless of restrictions (timeperiods or else)		force_notification	
<code>full_log_entries</code>	Show full or "condensed" log entries		full_log_entries	Default is "condensed" view
<code>get_date_parts</code>	Get custom time ranges		get_date_parts	
<code>graphevents</code>	Which objects should be graphed in what state		graphevents=112 (all service problems)	A boolean OR of: 1=Host up; 2=Host down; 4=Host unreachable; 8=Service OK; 16=Service Warning; 32=Service Unknown; 64=Service Critical
<code>graphstatetypes</code>	Graph hard and/or soft states	1=soft states; 2=hard states; 3=hard and soft states	graphstatetypes=3	
<code>host</code>	Select all hosts or a specific host whose services should be displayed	all; <host name>	host=monitor	special characters in the name have to be urlencoded (i.e. "%20" instead of a blank)
<code>hostgroup</code>	Select all hostgroups or one specific hostgroup whose hosts and services should be displayed	all; <hostgroup name>	hostgroup=linux-boxes	special characters in the name have to be urlencoded (i.e. "%20" instead of a blank)
<code>hostprops</code>	Select the hosts (including the services) matching the given bit pattern. Note that the hosts have to match ALL conditions specified		hostprops=131088 (active checks being disabled)	A boolean OR of the states specified in include/cgiutils.c [1] (HOST AND SERVICE FILTER PROPERTIES)
<code>hoststates</code>	The state the hosts should be in	1 - 7	hoststates=3 (hosts in problem state)	A boolean OR of the states: 1=DOWN; 2=UNREACHABLE; 4=UP
<code>hoststatustypes</code>	The state the host should be in	1 - 15	hoststatustypes=12 (hosts in problem state)	A boolean OR of the states: 1=Pending; 2=Up; 4=Down; 8=Unreachable
<code>hours</code>	Duration of the flexible downtime in full hours (see minutes)	>= 0		Only valid if defining a flexible downtime
<code>includesoftstates</code>	Include soft states	yes; no	includesoftstate=yes	Default: don't include soft states
<code>initialassumedhoststate</code>				
<code>initialassumedservicestate</code>				

Parameter	Description	Possible Values	Example	Notes
initialstateslogged				
input				
jsonoutput	Specify whether or not the report output should be in json format	yes;no		This option automatically sets "noheader". Added to nearly all CGIs (see note [2])
limit	Number of items to display	1-n	limit=10	Default is 25
minutes	Duration of the flexible downtime (see hours)	>= 0		
navbarsearch				
newstatesonly	Only display new states	yes; no	newstatesonly=yes	Default: display all states
nodowntime	Don't display downtimes		nodowntime	
noflapping	Don't display flapping alerts		noflapping	
nofrills	Don't display frills		nofrills	
noheader	Omit global status information and only show status details		noheader	
not_dly	Delay notification for n minutes	>= 0		
notimebreaks	Don't display timebreaks			
nosystem	Don't display system messages		nosystem	Default: display system (process) messages
oldestfirst	Reverse sort order		oldestfirst	Default: show most recent entries first
performance_data	The string sent as the performance data			
persistent	The comment will be persistent if this option is set		persistent	
plugin_output	The string sent as the plugin output			The length is limited by the value of MAX_INPUT_LENGTH (set during compile time)
plugin_state	Specify the state the plugin should have	0=OK; 1=Warning; 2=Critical; 3=Unknown	plugin_state=2	
ptc	The command will be propagated to child hosts		ptc	
report	Create report		report	
report_type	Select the report type	hostgroups; servicegroups; hosts; services	report_type=hostgroups	
rptimeperiod	Specify a timeperiod which is used for the availability report	One of the defined timeperiods		Use the short name from the timeperiod definition
sched_dly	Delay command scheduling by n minutes	>= 0		

Parameter	Description	Possible Values	Example	Notes
<code>sday</code>	Start of custom timeperiod (day)			
<code>send_notification</code>	Send a notification for the acknowledgement		send_notification	
<code>service</code>	Select all services or a specific service which should be displayed	all; <service description>	service=PING	Special characters in the name have to be urlencoded (i.e. "%20" instead of a blank)
<code>servicefilter</code>	Select only services matching the given pattern		servicefilter=Current; servicefilter=[PL]	The pattern is case sensitive. Regular expressions seem to work
<code>servicegroup</code>	Select all servicegroup or one specific servicegroup whose hosts and services should be displayed	all; <servicegroup name>	servicegroup=disk	Special characters in the name have to be urlencoded (i.e. "%20" instead of a blank)
<code>serviceprops</code>	Select the services matching the given bit pattern. Note that the services have to match ALL conditions specified		serviceprops=131088 (active checks being disabled)	A boolean OR of the states specified in include/cgiutils.c [1] (HOST AND SERVICE FILTER PROPERTIES)
<code>servicestates</code>	State the services should be in		servicestates=56 (services in problem state)	A boolean OR of the states: 8=Warning; 16=Unknown; 32=Critical; 64=OK
<code>servicestatustypes</code>	State the services should be in	1 - 31	servicestatustype=28 (services in problem state)	A boolean OR of the states: 1=Pending; 2=OK, 4=Warning; 8=Unknown; 16=Critical
<code>service_divisor</code>	Importance of services in relation to hosts	>=1		Services are 1/n important as hosts. Default: n=4
<code>shour</code>	Start of custom timeperiod (hour)			Only valid when using "custom" timeperiod
<code>showscheduledowntime</code>	Display scheduled downtimes	yes; no		Default: yes
<code>show_log_entries</code>	Display log entries		show_log_entries	Default: don't show log entries
<code>smin</code>	Start of custom timeperiod (minute)			Only valid when using "custom" timeperiod
<code>smon</code>	Start of custom timeperiod (month)			Only valid when using "custom" timeperiod
<code>sortoption</code>	Specify the column to sort on	1-n	sortoption=3	Default is column 1
<code>sorttype</code>	Specify the sort order on the column specified by "sortoption=<n>"	1=ascending; 2=descending	sorttype=2	
<code>ssec</code>	Start of custom timeperiod (second)			Only valid when using "custom" timeperiod

Parameter	Description	Possible Values	Example	Notes
<code>standardreport</code>	Standard report	1=recent alerts; 2=recent host alerts; 3=recent service alerts; 4=top host alert producers; 5=top service alert producers		
<code>start_time</code>	Start of the fixed downtime			Format "MM-DD-YYYY HH:MI" (may depend on your local settings, but unsure about that)
<code>statetype</code>	Hard and/or soft states	0=hard and soft states; 1=soft states; 2=hard states	<code>statetype=2</code>	
<code>statetypes</code>	Hard and/or soft states	1=soft states; 2=hard states; 3=hard and soft states	<code>statetypes=2</code>	
<code>sticky_ack</code>	The acknowledgement will be "sticky"		<code>sticky_ack</code>	
<code>style</code>	Specify the information to be shown	overview; detail; summary; grid; hostdetail		only applies to objecttypes "hostgroups" and "servicegroups"; hostdetail=host status details; detail=service status details; summary=status summary; grid=status grid;
<code>syear</code>	Start of custom timeperiod (year)			Only valid when using "custom" timeperiod
<code>t1</code>	Start point of a custom timeperiod		<code>t1=1296109300</code>	Unix timestamp
<code>t2</code>	End point of a custom timeperiod		<code>t2=1296189360</code>	Unix timestamp
<code>timeperiod</code>	Timeperiod which should be used for the availability report	today; yesterday; thisweek; lastweek; thismonth; lastmonth; thisquarter; lastquarter; thisyear; lastyear; last24hours; last7days; last31days; custom	<code>timeperiod=lastmonth</code>	
<code>trigger</code>	The downtime will be triggered by downtime id <n>	A valid downtime id		
<code>type</code>	Objecttype	hosts; hostgroups; services; servicegroups; contacts; contactgroups; timeperiods; commands; hostescalations; serviceescalations; hostdependencies; servicedependencies	<code>type=hosts</code>	

Abbreviations, CGIs, Links

Abbreviations used in the first table, relations to CGI and menu items, and links to the source code in <icinga-core>/cgis.

Abbreviation	CGI	Menu item(s)	Source Code
avail	avail.cgi	Availability	avail.c
cmd	cmd.cgi	N/A	cmd.c
config	config.cgi	Configuration	config.c
ext	extinfo.cgi	Comments, Downtime, Process Info, Performance Info, Scheduling Info	extinfo.c
hgram	histogram.cgi	Alert Histogram	histogram.c
hist	history.cgi	Alert History	history.c
notif	notifications.cgi	Notifications	notifications.c
out	outages.cgi	Network Outages	outages.c
log	showlog.cgi	Event Log	showlog.c
status	status.cgi	Hostgroup Overview, Servicegroup Overview, Host Problems, Service Problems	status.c
map	statusmap.cgi	Status Map	statusmap.c
summary	summary.cgi	Alert Summary	summary.c
tac	tac.cgi	N/A	tac.c
trends	trends.cgi	Trends	trends.c

Excerpt from include/cgiutils.h

```
***** HOST AND SERVICE FILTER PROPERTIES *****

#define HOST_SCHEDULED_DOWNTIME          1
#define HOST_NO_SCHEDULED_DOWNTIME        2
#define HOST_STATE_ACKNOWLEDGED         4
#define HOST_STATE_UNACKNOWLEDGED       8
#define HOST_CHECKS_DISABLED             16
#define HOST_CHECKS_ENABLED              32
#define HOST_EVENT_HANDLER_DISABLED     64
#define HOST_EVENT_HANDLER_ENABLED      128
#define HOST_FLAP_DETECTION_DISABLED   256
#define HOST_FLAP_DETECTION_ENABLED    512
#define HOST_IS_FLAPPING                1024
#define HOST_IS_NOT_FLAPPING            2048
#define HOST_NOTIFICATIONS_DISABLED    4096
#define HOST_NOTIFICATIONS_ENABLED      8192
#define HOST_PASSIVE_CHECKS_DISABLED   16384
#define HOST_PASSIVE_CHECKS_ENABLED     32768
#define HOST_PASSIVE_CHECK              65536
#define HOST_ACTIVE_CHECK                131072
#define HOST_HARD_STATE                 262144
#define HOST_SOFT_STATE                  524288

#define SERVICE_SCHEDULED_DOWNTIME        1
#define SERVICE_NO_SCHEDULED_DOWNTIME      2
#define SERVICE_STATE_ACKNOWLEDGED       4
#define SERVICE_STATE_UNACKNOWLEDGED     8
#define SERVICE_CHECKS_DISABLED           16
#define SERVICE_CHECKS_ENABLED             32
#define SERVICE_EVENT_HANDLER_DISABLED   64
#define SERVICE_EVENT_HANDLER_ENABLED    128
```

```
#define SERVICE_FLAP_DETECTION_ENABLED 256
#define SERVICE_FLAP_DETECTION_DISABLED 512
#define SERVICE_IS_FLAPPING 1024
#define SERVICE_IS_NOT_FLAPPING 2048
#define SERVICE_NOTIFICATIONS_DISABLED 4096
#define SERVICE_NOTIFICATIONS_ENABLED 8192
#define SERVICE_PASSIVE_CHECKS_DISABLED 16384
#define SERVICE_PASSIVE_CHECKS_ENABLED 32768
#define SERVICE_PASSIVE_CHECK 65536
#define SERVICE_ACTIVE_CHECK 131072
#define SERVICE_HARD_STATE 262144
#define SERVICE_SOFT_STATE 524288
```

[1] Logical OR means that the numbers are added and only the objects will be displayed satisfying ALL conditions.

[2] Available starting with Icinga 1.4.

[3] avail, log, notif, out, status, sum: all views/reports support jsonoutput; config: all types except command expansion; ext: all views except hostgroup/servicegroup info (always without performance data); tac: data output in json format. More details in the [Icinga wiki](#).

[Prev](#)

[Up](#)

[Next](#)

Icinga Classic UI: Information On
The CGIs

[Home](#)

Executing CGIs on the command
line

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Executing CGIs on the command line

[Prev](#)

Chapter 6. User Interfaces

[Next](#)

Executing CGIs on the command line

Introduction

In most cases you will use your browser to retrieve the information you need. There may be situations where you want to process the data with other tools to create wiki entries, send mails and so on. Together with the [information on the CGI parameters](#) you can call the CGIs from the command line.

Prerequisites

Before actually calling the CGIs you have to set three environment variables:

- REMOTE_USER

This variable is set to the user which has permission to retrieve the information. In most cases this will be "icingaadmin" (set `REMOTE_USER='icingaadmin'`)

- REQUEST_METHOD

`set REQUEST_METHOD='GET'`. Possible values are "GET", "POST" and "HEAD"

- QUERY_STRING

Instead of passing arguments to the CGIs via the command line you have to set the variable "QUERY_STRING" with the appropriate values.



Note

Most people find it difficult to read HTML output so it may be a good idea to add "jsonoutput" or "csvoutput" to this variable
`(QUERY_STRING=' jsonoutput&... ' or QUERY_STRING=' jsonoutput ')`.

If you forgot to set the environment variables you'll receive the following lines:

```
$> ./status.cgi
getcgivars(): Unsupported REQUEST_METHOD -> ''

I'm guessing you're trying to execute the CGI from a command line.
In order to do that, you need to set the REQUEST_METHOD environment
variable to either "GET", "HEAD", or "POST". When using the
GET and HEAD methods, arguments can be passed to the CGI
by setting the "QUERY_STRING" environment variable. If you're
using the POST method, data is read from standard input. Also of
note: if you've enabled authentication in the CGIs, you must set the
"REMOTE_USER" environment variable to be the name of the user you're
"authenticated" as.
```

Examples



Note

The CGIs are called from within the folder which contains the *.cgi files (e.g. /usr/local/icinga/sbin). This is not necessary but for the sake of simplicity. Unless otherwise specified the REQUEST_METHOD is set to 'GET'.

Tactical overview

```
$> set QUERY_STRING='jsonoutput'
$> ./tac.cgi
```

All hosts in DOWN state

```
$> set QUERY_STRING='jsonoutput&style=hostdetail&hoststatustypes=4'
$> ./status.cgi
```

All hosts in DOWN state being unacknowledged *and* not in a downtime

```
$> set QUERY_STRING='jsonoutput&style=hostdetail&hoststatustypes=4&hostprops=10'
$> ./status.cgi
```

All service in non-OK state

```
$> set QUERY_STRING='jsonoutput&style=detail&servicestatustypes=28'
$> ./status.cgi
```

All services being in CRITICAL state and being passive checks

```
$> set QUERY_STRING='jsonoutput&style=detail&servicestatustypes=28&serviceprops=65536'
$> ./status.cgi
```

Comments for all objects

```
$> set QUERY_STRING='jsonoutput&type=3'
$> ./extinfo.cgi
```

Trends for router_02 specifying a custom timeperiod using unix timestamps

```
$> set QUERY_STRING='jsonoutput&host=router_02&timeperiod=custom&t1=130748400&t2=1307570400'
$> ./extinfo.cgi
```

Trends for router_02 specifying a custom timeperiod using date and time

```
$> set QUERY_STRING='jsonoutput&host=router_02&timeperiod=custom\
&sday=6&smon=6&syear=2011&shour=0&smin=0&ssec=0\
&eday=7&emon=6&eyear=2011&ehour=0&emin=0&esec=0'
$> ./extinfo.cgi
```

(to be continued)

[Prev](#)

[Up](#)

[Next](#)

[Information On CGI parameters](#)

[Home](#)

[Installation of the Icinga-Web Frontend](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Installation of the Icinga-Web Frontend

[Prev](#)

Chapter 6. User Interfaces

[Next](#)

Installation of the Icinga-Web Frontend

Introducion

The new Icinga Web is under heavy development so please keep in mind that some information in this howto might change without any further notice. If you require more detailed information about installing, please check doc/INSTALL.

More information about the overall architecture can be found on our website: <http://www.icinga.org/architecture/>. If you want know more about Icinga web development and the module architecture, please check out the development wiki of Icinga Web: [Icinga-Wiki](#)

This installation guide describes the installation of Icinga-Web with MySQL as underlying database. Icinga-Web also supports Oracle and PostgreSQL as database backends.

1. Prerequisites

Based on the fact that you have a running MySQL and PHP (with PEAR and CLI) environment and Icinga, Icinga-API and IDOUtils are running as well, you can continue with the second step. Otherwise:

Ubuntu / Debian

Make sure you have a repository/packages for PHP 5.2.x - RHEL/CentOS (CentOS <= 5.4) only support 5.1.6 out of the box.

```
#> apt-get install php5 php5-cli php-pear php5-xmlrpc php5-xsl php5-pdo php5-gd php5-ldap php5-mysql
```



Note

For Ubuntu you install the required PDO extensions with php5 and php5-mysql! There is no php5-pdo package available.

Fedora / RHEL / CentOS

```
#> yum install php php-cli php-pear php-xmlrpc php-xsl php-pdo php-gd php-ldap php-mysql
```

**Note**

Currently, RHEL and CentOS provide old php 5.1.6 and pcre 6.6 packages. For running Icinga web you need at least php 5.2.3 and pcre 7.6 - you can use an external repository, pre-built packages or compile php and pcre yourself. Precompiled PCRE and PHP packages can be installed e.g. from [Les RPM de Remi](#) or <http://www.jasonlitka.com/category/yum-repo-news/>

OpenSuSE

Please use yast to install the packages "php5", "php5-peach", "php5-xmlrpc", "php5-xsl", "php5-<rdbm>", "php5-soap", "php5-gettext", "php5-ldap", "php5-gd" and "php5-mysql". The CLI is contained in the php5 package.

**Note**

At least in SLES10 SP2 the function "hash_hmac" is missing.

Installing Icinga with IDOUtils and Icinga-API is described in the [Icinga with IDOUtils Quickstart Guide](#) and the [Icinga-API](#) section.

2. The installation

Please download the archive from <http://sourceforge.net/projects/icinga/files/> or take your clone from the icinga-web.git to get the freshest branch:

```
#> git clone git://git.icinga.org/icinga-web.git
```

Unpack your tarball:

```
#> tar xzvf icinga-web-1.4.0.tar.gz
```

Then change to the directory:

```
#> cd icinga-web-1.4.0
```

Icinga-Web provides several configure options e.g.

```
#> ./configure
--prefix=/usr/local/icinga-web
--with-web-user=www-data
--with-web-group=www-data
--with-web-path=/icinga-web
--with-web-apache-path=/etc/apache2/conf.d
--with-db-type=mysql
--with-db-host=localhost
--with-db-port=3306
--with-db-name=icinga_web
--with-db-user=icinga_web
--with-db-pass=icinga_web
--with-icinga-api=/usr/local/icinga/share/icinga-api
--with-api-type=APICON API type (default CONNECTION_IDO)
--with-api-subtype=TYPE DB driver or network connection
--with-api-host=HOST Host to connect (DB or other) (default localhost)
--with-api-port=PORT Port for connection (default 3306)
--with-api-socket=PATH Path to socket (default none)
```

**Note**

Keep in mind that you configure the Icinga Web database not the Icinga IDOUtils database! User and group name of the web process depend on the distribution used.

Please use:

```
#> ./configure --help
```

to see all configure options.

**Note**

Using no options the installer expects the icinga-API to be found at /usr/local/icinga/share/icinga-api.

Per default the Icinga-Webinterface will be installed to /usr/local/icinga-web using:

```
#> ./configure
#> make install
```

Install the new Apache configuration

```
#> make install-apache-config
```

If you don't need an alias, you can alternatively use the old behaviour setting a symlink instead:

```
#> make install-javascript
```

Make reports after install:

```
#> make install-done
```

```
Installation of icinga-web succeeded.
Please check the new Apache2 configuration (etc/apache2/icinga-web.conf).
```

Other useful target:

```
#> make icinga-reset-password
```

Reset password for any account on icinga-web.

3. PHP dependencies

Test the php dependencies with:

```
#> make testdeps
```

All required tests should pass successfully. Maybe you have to alter the php.ini for the framework.

In case of the gpc_magic_quote setting, you have to disable both entries (apache and cli php.ini). If you use php < 5.3.0, you have to set "safe_mode" to "off". The locations depend on the distributions used.

```
#> vi /etc/php5/apache/php.ini
magic_quotes_gpc = off
safe_mode = off

#> vi /etc/php5/cli/php.ini
magic_quotes_gpc = off
```



Note

If one of these files is missing you'll get an agavi error complaining about the setting of "magic_quotes_qpc" because the default is "ON".

4. Database creation

Icinga Web requires its own database e.g. icinga_web. You can use the one from Icinga IDOUtils but it is recommended to keep this separated for upgrading purposes.

Create a database user

The user must have default data privileges like SELECT, UPDATE, INSERT, DELETE.

```
SQL> GRANT USAGE ON *.* TO 'icinga_web'@'localhost' IDENTIFIED BY 'icinga_web';
SQL> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER, INDEX ON icinga_web.* TO 'icinga_web'@'localhost';
```

MySQL

```
# mysql -u root -p
mysql> CREATE DATABASE icinga_web;
GRANT USAGE ON *.* TO 'icinga_web'@'localhost' IDENTIFIED BY 'icinga_web' WITH MAX_QUERIES_PER_HOUR 0 MAX_CONNECTIONS_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0;
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ALTER, INDEX ON icinga_web.* TO 'icinga_web'@'localhost';
quit
```

Create Database

Icinga Web ships with Doctrine so you can initialise or drop the database directly using 'make'.

```
#> make db-initialize      - creates a spick-and-span database
#> make db-drop            - drops your database with a security query to avoid casualties
#> make db-doc2sql          - if you require plain SQL scripts, use this option to generate SQL from Doctrine. This can be useful for packaging or manual db install
```

To use the database creation commands you have to grant privileges to the user who will execute the commands on your dbms. If the user defined via configure is a low-privileged one, 'make' asks about a more privileged one e.g. a root user. If this does not work for you, alter 'etc/build.properties' to match the requirements of a root user.

So a simple database install looks like this:

```
#> make db-initialize
```

Manually create Database

If you require manual database creation e.g. for package building, you can extract the plain SQL scripts using:

```
#> make db-doc2sql
```

and then use the generated scripts on the freshly created database.

5. Settings

There are two different sections:

- * Settings of Icinga Web, especially database settings
- * Settings of the Icinga API which is mandatory as data source

Settings of Icinga-Web

Normally you can set the database credentials during configure, but if you want to recheck or even change them, please adapt those to your needs.

```
#> vi app/config/databases.xml
```



Note

Optional: Your specific Icinga database settings can be set in `app/config/.site.xml` and will be used in the first place. This won't get overwritten during upgrade process.

```
<databases default="icinga_web">
    <database name="icinga_web" class="AgaviDoctrineDatabase">
        <!--
            Doctrine dsn strings:
            http://www.doctrine-project.org/documentation/manual/1_1/en/introduction-to-connections
        -->
        <ae:parameter name="dsn">mysql://icinga_web:icinga_web@127.0.0.1:3306/icinga_web</ae:parameter>

        <!-- Generic credentials -->
        <!-- <ae:parameter name="username">icinga_web</ae:parameter> -->
        <!-- <ae:parameter name="password">icinga_web</ae:parameter> -->

        <!-- DB encoding type -->
        <ae:parameter name="charset">utf8</ae:parameter>

        <!--
            Doctrine_Manager configuration
        -->
        <ae:parameter name="manager_attributes">
            <!-- This allows lazy loading of the models -->
            <ae:parameter name="Doctrine_Core::ATTR_MODEL_LOADING">CONSERVATIVE</ae:parameter>
        </ae:parameter>

        <!-- The path to our models -->
        <ae:parameter name="load_models">%core.module_dir%/AppKit/lib/database/models/generated</ae:parameter>
        <ae:parameter name="models_directory">%core.module_dir%/AppKit/lib/database/models</ae:parameter>
    </database>
```

Settings of Icinga-API

Those settings are directly within Icinga-Web which provides the given information in an appropriate way to the Icinga-API. Keep a look into `app/modules/Web/config/icinga-io.xml.xml` for the IcingaApi factories: `IcingaData` and `IcingaCommand`.

```
#> vi app/modules/Web/config/icinga-io.xml
```



Note

Optional: Your specific Icinga connection settings like api credentials config can be set in `app/modules/Web/config/icinga-io.site.xml` and will be used in the first place. This won't get overwritten during upgrade process.

In `IcingaData` you can set the credentials for the Icinga API e.g. using the Icinga IDOUtils database.

Please keep in mind that you have to install IDOUtils before (according to the [Icinga IDOUtils Quickstart Guide](#))

```
<!--
      See doc/icinga-api-types.txt for options
-->
<setting name="api.interfaces.data">
    <!-- IcingaApi connection interface -->
    <ae:parameter name="api_type">IcingaApi::CONNECTION_IDO</ae:parameter>

    <!-- Suits for all interfaces -->
    <ae:parameter name="config_type">mysql</ae:parameter>
    <ae:parameter name="config_host">localhost</ae:parameter>
    <ae:parameter name="config_port">3306</ae:parameter>

    <!-- ###BEGIN_CONNECTION_IDO## -->
    <!-- Database specific (IcingaApi::CONNECTION_IDO) -->
    <ae:parameter name="config_database">icinga</ae:parameter>
    <ae:parameter name="config_user">icinga</ae:parameter>
    <ae:parameter name="config_password">icinga</ae:parameter>
    <ae:parameter name="config_table_prefix">icinga_</ae:parameter>
    <!-- ###END_CONNECTION_IDO## -->

</setting>
```

In IcingaCommand you can specify the credentials for sending commands via Icinga Core command pipe - local or remote via SSH. Depending on 'enabled' those interfaces can be used or not (default is the local pipe).

```
<setting name="api.interfaces.command">
    <ae:parameter name="pipe1">
        <ae:parameter name="type">IcingaApi::COMMAND_PIPE</ae:parameter>
        <ae:parameter name="enabled">true</ae:parameter>

        <ae:parameter name="pipe">/usr/local/icinga/var/rw/icinga.cmd</ae:parameter>

        <ae:parameter name="instance">default</ae:parameter>
        <ae:parameter name="broadcast">false</ae:parameter>
    </ae:parameter>

    <!--

        * This examples show how to send commands to specific instances
        * Use 'instance' to send commands to this instance only
        * Use 'broadcast' to send all commands to this instances!

    <ae:parameter name="pipe1-instance-test">
        <ae:parameter name="type">IcingaApi::COMMAND_PIPE</ae:parameter>
        <ae:parameter name="enabled">true</ae:parameter>

        <ae:parameter name="pipe">/usr/local/icinga/var/rw/icinga.cmd</ae:parameter>

        <ae:parameter name="instance">test</ae:parameter>
        <ae:parameter name="broadcast">false</ae:parameter>
    </ae:parameter>

    <ae:parameter name="pipe1-broadcast">
        <ae:parameter name="type">IcingaApi::COMMAND_PIPE</ae:parameter>
        <ae:parameter name="enabled">false</ae:parameter>

        <ae:parameter name="pipe">/usr/local/icinga/var/rw/icinga.cmd</ae:parameter>

        <ae:parameter name="instance"></ae:parameter>
        <ae:parameter name="broadcast">true</ae:parameter>
    </ae:parameter>

    -->

    <ae:parameter name="ssh1">
        <ae:parameter name="type">IcingaApi::COMMAND_SSH</ae:parameter>
        <ae:parameter name="enabled">false</ae:parameter>

        <ae:parameter name="ssh_bin">/usr/bin/ssh</ae:parameter>
        <ae:parameter name="ssh_user">icinga</ae:parameter>
        <ae:parameter name="ssh_host">127.0.0.1</ae:parameter>
        <ae:parameter name="ssh_port">22</ae:parameter>
        <ae:parameter name="ssh_timeout">20</ae:parameter>
        <ae:parameter name="ssh_pipe">/usr/local/icinga/var/rw/icinga.cmd</ae:parameter>

        <ae:parameter name="instance">default</ae:parameter>
        <ae:parameter name="broadcast">false</ae:parameter>
    </ae:parameter>
</setting>
```



Note

After changing those configs you need to clear the config cache again!

```
#> rm -rf app/cache/config/*.php
or /path/to/clearcache.sh
#> /usr/local/icinga-web/bin/clearcache.sh
```

6. Apache settings

This should be prepared:

- mod_rewrite enabled, maybe you have to create a link:

```
#> ln -s /etc/apache2/mods-available/rewrite.load /etc/apache2/mods-enabled/rewrite.load
```

Using OpenSuSE or SLES you can use "a2enmod rewrite" to activate the module. If that doesn't work you have to edit the file "/etc/sysconfig/apache2". The module "rewrite" has to be appended to the line "APACHE_MODULES=..." .

Using Debian or Ubuntu "a2enmod rewrite" activates the module as well.

The httpd of RHEL / Fedora / CentOS already supports rewriting so you don't have to change anything in this regard.

- Any htaccess enabled alias settings

Edit your .htaccess in /usr/local/icinga-web/pub:

At line 14, change the RewriteBase direction to suit to your needs:

```
DirectoryIndex index.php

Options -MultiViews -Indexes +FollowSymLinks
Order allow,deny
Allow from all

<IfModule mod_rewrite.c>
    RewriteEngine On

    # This depends on your path
    # on independent hosts the base is ''
    RewriteBase /icinga-web/

    # If the requested URL does not exist (it's likely an agavi route),
    # pass it as path info to index.php, the Agavi dispatch script.
    RewriteRule ^$ index.php?/ [QSA,L]
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteRule (.*) index.php?/$1 [QSA,L]
</IfModule>

<IfModule mod_deflate.c>
    SetOutputFilter DEFLATE

    BrowserMatch ^Mozilla/4 gzip-only-text/html
    BrowserMatch ^Mozilla/4\.0[678] no-gzip

    BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
    BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html

    Header append Vary User-Agent env=!dont-vary
</IfModule>

<IfDefine APACHE2>
    AcceptPathInf On
```

```
</IfDefine>

#<IfModule mod_auth_basic.c>
#    AuthType Basic
#    AuthName "My http basic auth realm"
#    AuthUserFile /path/to/my/htusers
#    require valid-user
#</IfModule>
```

Go to the webservers configuration directory. Look if the created config by **make install-apache-config** suits to your configuration. Or create new aliases (maybe in /etc/apache2/conf.d/icinga-web.conf) :

```
#> cat /etc/apache2/icinga-web.conf

# icinga-web apache configuration
# - Enable all options .htaccess
# - Add extjs library to alias
#
Alias /icinga-web/js/ext3 /usr/local/icinga-web/lib/ext3
Alias /icinga-web /usr/local/icinga-web/pub

<Directory /usr/local/icinga-web/lib/ext3>
    Order allow,deny
    Allow from all
</Directory>

<Directory /usr/local/icinga-web/pub>
    DirectoryIndex index.php
    Options FollowSymLinks
    AllowOverride all
    Order allow,deny
    Allow from all
</Directory>
```

Clear cache:

```
#> rm /usr/local/icinga-web/app/cache/config/*.php
or /path/to/clearcache.sh
#> /usr/local/icinga-web/bin/clearcache.sh
```

Restart your Webserver:

```
#> service apache2 restart
or
#> /etc/init.d/apache2 restart
or
#> /etc/init.d/httpd restart
```

7. Use It!

Please ensure that your RDBMS, Apache, IDOUtils and Icinga are running!

Go to the webpath (<http://localhost/icinga-web/>) and check if the webinterface starts without exceptions (database connections web and api). You can login with user 'root' and password 'password'.

Have fun! :-)

8. Test & Errors ?

The following collection shows some useful information how to resolve errors - and what you should provide when reporting an error to the mailing lists or <http://www.icinga-portal.org>

- * Always provide the version - current tar.gz or GIT?
- * Add your browser and the version
- * If the problem is related to the data source, provide extensive information about API, IDOUtils, Core (version, debug logs)

Where to look?

- * Apache Error Logs, PHP Errors, PHP files cannot be found
- * /var/log/messages, /path/to/icinga/var/icinga.log Changes on Icinga Web Config (e.g. API IDOUtils credentials changed) are not used?
- * Delete the config cache in app/cache/config/*.php

use /path/to/clearcache.sh

```
#> /usr/local/icinga-web/bin/clearcache.sh
```

Icinga Web shows a blank page?

- * Apache Error Logs => mod_rewrite enabled, php dependencies ok? 'make testdeps'. Using Debian you may find something like the following: ".htaccess: Invalid command 'RewriteEngine', perhaps misspelled or defined by a module not included in the server configuration"

- * .htaccess/vhost config => paths not correct?

Icinga Web does not show any data?

- * Apache Error logs tell that IcingaApi.php was not found => recheck the install guide if Icinga API is installed and properly configured

- * DB access denied => check if the icinga web database connection settings are correct

IDOUtils DB does not get filled with data?

- * "Error writing to data sink" => check IDOUtils (ido2db runs 2x - ok?), ido2db.cfg debug_level=-1, debug_verbosity=2, restart IDOUtils and look for errors in ido2db.debug

- * Nothing there => Check icinga.log if IDOMOD gets loaded if not enable the Event Broker Module in icinga.cfg like described in the Icinga Core with [IDOUtils Quickstart Guide](#)

- * IDOUtils DB-Schema is the current one? => If not check the upgrade path and docs of IDOUtils

- * Sockets correctly defined? => unix-socket or tcp-socket, the last one with or without SSL

Testing the Web:

```
$> cd etc/tests/
$> php icingaWebTesting.php
```



Note

If you are using your root account for testing then make sure in advance that the specified web user has a valid shell. Otherwise some tests might fail. Depending on the version used you might experience wrong permissions on /usr/local/icinga-web/app/data/log resulting in Icinga-Web showing "loading" coming to no end!



Note

Remember - changing the php settings in php.ini requires an Apache reload/restart!

* PHP Fatal error: Allowed memory size of ... bytes exhausted (tried to allocate ... bytes) => Check your php.ini (both apache2 and cli) and adjust memory_limit to 128M or higher.

* PHP Fatal error: Uncaught exception 'AgaviCacheException' with message 'Failed to write cache file

/usr/local/icinga-web/app/cache/config/config_handlers.xml_development_xxxx.php" generated from configuration file

/usr/local/icinga-web/app/config/config_handlers.xml". Please make sure you have set correct write permissions for directory /usr/local/icinga-web/app/cache.... => Check /etc/php5/apache/php.ini, set safe_mode = off.

* Could not connect to API. The API Connector returned the following message: getConnection failed: Database connection failed: SQLSTATE[28000] [1045] Access denied for user 'icinga'@'localhost' (using password: YES))

=> check your IDOUtils DB credentials in ido2db.cfg and add those to Icinga Web configuration as preferred DB credentials for IDO (see above). With 1.0.3 you can set those values directly during configure

* touch: cannot touch '/usr/local/icinga-web/.../cache/testfile.txt': Permission denied => Your configuration in the xml files will be pre-cached by the framework. It therefore needs special permissions on the caching directories. By running icingaWebTesting.php in etc/tests you will be asked to automatically fix this.

* PHP Fatal error: Uncaught exception '...' with message 'Couldn't locate driver named mysql' => Make sure that the php pdo is installed and loaded in an appropriate way, even if make testdeps tells everything is fine.

* Login is not shown => enable short_open_tag in your php.ini

=> edit open_basedir in your php.ini and add icinga web location and parent location of icinga api (e.g. /usr/local/icinga/share/).

* Empty Icinga Web? => If mod_rewrite is enabled and 'index.php' appears in the request url the portal does not work. Try to remove the index.php from your url and everything should work

- * Login possible but Icinga Web keeps loading data...
- * Request failed, Ressource /icinga-web/appkit/ext/application State could not be loaded
- is the url correct? => mod_rewrite enabled ?
- * Counts in the Status Cronk do not match your configuration? => Check with the Backend,
e.g. IDOUtils DB, selecting the counts for the status tables.
- * No Data shown in Cronks? => Make sure that all permissions are set correctly, especially
the log/

If you have any updates on that please do not hesitate to report back! :-)

[Prev](#)

[Up](#)

[Next](#)

Executing CGIs on the command
line

[Home](#)

Upgrading Icinga-Web and
Icinga-Web Database

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Upgrading Icinga-Web and Icinga-Web Database

[Prev](#)
[Chapter 6. User Interfaces](#)
[Next](#)

Upgrading Icinga-Web and Icinga-Web Database

Upgrading Icinga-Web

If there's a new Icinga-Web version available on <http://www.icinga.org/> you should consider updating. Newer versions contain bugfixes, minor or major, which makes it even more important to get the latest and the greatest. If you already installed Icinga-Web; using the installing guide it's rather easy to install newer version.

Make sure you'll have a safe backup of your Icinga-Web installation and your configs (if you stored your customised configuration in *.site.xml- Files, they won't be overwritten during the upgrade process!). If anything goes wrong you can restore your old setup rather easy.

Regular updates of Icinga-Web are possible by just by re-installing Icinga-Web. Please keep in mind that `configure` requires the same parameters as before - look it up in `config.log` and save that in a safe location.

Please make sure, that you have already installed the appropriate Icinga-API version. Installing the Icinga-API is described in the [Icinga-API](#) section.

The Upgrade

Please download the archive from <http://sourceforge.net/projects/icinga/files/> or take your clone from the `icinga-web.git` to get the freshest branch:

```
#> git clone git://git.icinga.org/icinga-web.git
```

Unpack your tarball:

```
#> tar xzvf icinga-web-1.4.0.tar.gz
```

Then change to the directory:

```
#> cd icinga-web-1.4.0
```

Icinga-Web provides several configure options e.g.

```
#> ./configure
--prefix=/usr/local/icinga-web
--with-web-user=www-data
--with-web-group=www-data
--with-web-path=/icinga-web
--with-web-apache-path=/etc/apache2/conf.d
--with-db-type=mysql
--with-db-host=localhost
```

```
--with-db-port=3306
--with-db-name=icinga_web
--with-db-user=icinga_web
--with-db-pass=icinga_web
--with-icinga-api=/usr/local/icinga/share/icinga-api
--with-api-type=APICON API type (default CONNECTION_IDO)
--with-api-subtype=TYPE DB driver or network connection
--with-api-host=HOST Host to connect (DB or other) (default localhost)
--with-api-port=PORT Port for connection (default 3306)
--with-api-socket=PATH Path to socket (default none)
```

**Note**

Keep in mind that you configure the Icinga Web database not the Icinga IDOUtils database! User and group name of the web process depend on the distribution used.

Please use:

```
#> ./configure --help
```

to see all configure options.

**Note**

Using no options the installer expects the Icinga-API to be found at /usr/local/icinga/share/icinga-api.

Per default the Icinga-Web will be updated to /usr/local/icinga-web using:

```
#> ./configure
#> make upgrade
```

**Note**

After changing any configs you need to clear the config cache!

```
#> rm -rf app/cache/config/*.php
or /path/to/clearcache.sh
#> /usr/local/icinga-web/bin/clearcache.sh
```

Any errors? Take a look [here](#).

Upgrading the Icinga-Web Database

This is optional, but there may be a bug within the database scheme which has been fixed. If you are upgrading from an older Icinga-Web database version you also need to apply those fixes manually. If you are using rpm/deb packages please read the notes and/or ask the maintainer if he has added those modifications to the install routine.

**Note**

Depending on the changes to be done and the size of your database it may take a while to update your database. Please try to be patient and don't abort the script as it may leave the data being corrupt.

The upgrade files can be found next to the database install files in `/path/to/icinga-web/etc/schema/updates/`. The syntax is as follows:

```
<rdbm>_<old-version>_to_<new-version>.sql
```

where `<rdbm>` could be mysql, pgsql or oracle and `<newversion>` points to the version you want to upgrade to.



Note

If you are upgrading from an older version and there are other versions in between be advised that you need to apply those upgrade files with incremental steps!

1. Backup your current database before upgrading!
2. Check current Icinga-Web database version and the target version. Check if there are any version in between and upgrade incremental if necessary.
3. Apply the upgrade(s) using a rdbm user with appropriate rights.

MySQL

```
$ mysql -u root -p icinga_web < /path/to/icinga-web/etc/schema/updates/mysql_<oldversion>_to_<newversion>.sql
```

Postgresql

```
#> su - postgres
$ psql -U icinga_web -d icinga_web < /path/to/icinga-web/etc/schema/updates/pgsql_<oldversion>_to_<newversion>.sql
```

Oracle

```
#> su - oracle
$ sqlplus dbuser/dbpass
SQL> @oracle_<oldversion>_to_<newversion>.sql
```

That's all.

[Prev](#)
[Up](#)
[Next](#)
[Installation of the Icinga-Web Frontend](#)
[Home](#)
[Configuration Overview of Icinga-Web](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Configuration Overview of Icinga-Web

[Prev](#)

Chapter 6. User Interfaces

[Next](#)

Configuration Overview of Icinga-Web

Where are my config files?

We're trying to reduce global configuration files. Icinga operates in modules, every module has its own configuration. This is also done with the libraries. If you need information about the cronk library, take a look into `app/modules/Cronks/lib` (for js things `app/modules/Cronks/lib/js`).

A sample module looks like this:

```
tree -d -L 1 app/modules/AppKit/
app/modules/AppKit/
|-- actions
|-- config
|-- lib
|-- models
|-- templates
|-- validate
|-- views
```

Index

- [Global Configuration Section](#)
- [Session Cookie Lifetime](#)
- [Change default Timezone](#)
- [Module Configuration](#)
- [Authentication](#)
- [Icinga-API connection](#)
- [Customised Configuration](#)

Global Configuration Section

app/config

Here you find the global configuration files for e.g. the Web session, the Icinga web path and the database information.

The main interesting files:

- `database.xml` - holds the connection information for your icinga-web database
- `factories.xml` - holds the config for your web session, e.g. the `session_cookie_lifetime-parameter`
- `icinga.xml` - holds the config for your Icinga Web root-dir, Web path.

Session Cookie Lifetime

Example: change the `session_cookie_lifetime`

The Session Lifetime is the time in seconds until the Icinga Web session expires. It can be configured on global level in the file `app/config/factories.xml`.

```
#> vi app/config/factories.xml
<ae:parameter name="session_cookie_lifetime">3600</ae:parameter>
```

Please make your customised settings in `factories.site.xml`.

Change Icinga-Web timezone

Example: Change the default timezone for Icinga-Web

If the time(zone) in Icinga-Web differs from your system time(zone), please check the `date.timezone` entry in `app/modules/AppKit/config/module.xml` (e.g. 'Europe/Berlin')

```
#> vi app/modules/AppKit/config/module.xml
<ae:parameter name="date.timezone">GMT</ae:parameter>
```

Module Configuration

app/modules/AppKit

Here lives the framework spun around: Authentication, Menus and so on.

Authentication

Example: LDAP authentication

Look into `app/modules/AppKit/config/auth.xml`.

There is a provider like this:

```
<ae:parameter name="msad-ldap1">
    <ae:parameter name="auth_module">AppKit</ae:parameter>
    <ae:parameter name="auth_provider">Auth.Provider.LDAP</ae:parameter>
    <ae:parameter name="auth_enable">true</ae:parameter>
    <ae:parameter name="auth_authoritative">true</ae:parameter>
    <ae:parameter name="auth_create">true</ae:parameter>
    <ae:parameter name="auth_update">true</ae:parameter>

    <ae:parameter name="auth_map">
        <ae:parameter name="user_firstname">givenName</ae:parameter>
        <ae:parameter name="user_lastname">sn</ae:parameter>
        <ae:parameter name="user_email">mail</ae:parameter>
    </ae:parameter>

    <ae:parameter name="ldap_dsn">ldap://ad.icinga.org</ae:parameter>
    <ae:parameter name="ldap_basedn">DC=ad,DC=icinga,DC=org</ae:parameter>
    <ae:parameter name="ldap_binddn">ldap@AD.icinga.org</ae:parameter>
    <ae:parameter name="ldap_bindpw"><! [CDATA[XXXXXXXXXX]]></ae:parameter>
    <ae:parameter name="ldap_userattr">uid</ae:parameter>
    <ae:parameter name="ldap_filter_user"><! [CDATA[ (&(sAMAccountName=__USERNAME__)) ]]></ae:parameter>
</ae:parameter>
```

The auth.xml holds the documentation for the global config. The LDAP authentication should be done with some basic LDAP knowledge.

You can also duplicate the provider to provide more authentication bases. You can split the auth into authentication and authorising parts if you want. If you have new providers, please share them and let the auth database grow!

Please store your custom authentication configuration in auth.site.xml !

app/modules/Cronks

All the cronks are implemented there: Grids, iframes. All of them are simple html sites which hold ExtJS component codes. If you need to add a new cronk, this module is your friend.

If you want to develop a new cronk take a look at [HowToDevelopCronks](#)

To change the configuration, go to the Cronks module.

```
#> ls app/modules/Cronks/config
autoload.xml  config_handlers.xml  cronks.xml  module.xml  validators.xml
```

- module.xml - to define new categories in which the cronks appears, module.xml holds all the information
- cronks.xml - to make new cronks accessible or define new iframe cronks

app/modules/Web

Or better: **Icinga**. This module holds all Icinga related stuff, IcingaAPI2Json, status information. Also the IcingaAPI connection is configured here.

Icinga-API connection settings

Example: Change the IcingaApi connection settings

Take a look at app/modules/Web/config/icinga-io.xml, here you'll find the default settings for the IcingaApi connection.

```
#> vi app/modules/Web/config/icinga-io.xml

<setting name="api.interfaces.data">
    <!-- IcingaApi connection interface -->
    <ae:parameter name="api_type">IcingaApi::CONNECTION_IDO</ae:parameter>

    <!-- Suits for all interfaes -->
    <ae:parameter name="config_type">mysql</ae:parameter>
    <ae:parameter name="config_host">localhost</ae:parameter>
    <ae:parameter name="config_port">3306</ae:parameter>

    <!-- ###BEGIN_CONNECTION_IDO### -->
    <!-- Database specific (IcingaApi::CONNECTION_IDO) -->
    <ae:parameter name="config_database">icinga</ae:parameter>
    <ae:parameter name="config_user">icinga</ae:parameter>
    <ae:parameter name="config_password">icinga</ae:parameter>
    <ae:parameter name="config_table_prefix">icinga_</ae:parameter>
    <!-- ###END_CONNECTION_IDO## -->
```

If you want to change these settings to your needs, please edit app/modules/Web/config/icinga-io.site.xml.

Customised Configuration

Notes

When creating and/or editing configuration files, keep the following in mind:

1. Lines that start with a <!-- and end with --> character are taken to be comments and are not processed
2. Variable names are case-sensitive
3. Your specific Icinga-Web configuration settings like api, authentication and database settings can be set in app/modules/Appkit(/Cronks/Web)/config/*.site.xml files and will be used in the first place. They won't get overwritten during the next upgrade process.

These files already exist and you're free to use them for your specific settings:

```
app/modules/Cronks/config/cronks.site.xml
app/modules/Web/config/icinga-io.site.xml
app/modules/AppKit/config/auth.site.xml
app/config/icinga.site.xml
app/config/databases.site.xml
app/config/settings.site.xml
app/config/translation.site.xml
app/config/factories.site.xml
```

The templates for Grids and TO's (tactical overview) are allowed to copy to *.site.xml in their location:

The Grids:

```
app/modules/Cronks/data/xml/grid/icinga-hostgroup-summary-template.xml
app/modules/Cronks/data/xml/grid/icinga-host-history-template.xml
app/modules/Cronks/data/xml/grid/icinga-host-template.xml
...
```

and the TO's:

```
app/modules/Cronks/data/xml/to/icinga-tactical-overview-groupstat.xml
app/modules/Cronks/data/xml/to/icinga-tactical-overview-presets.xml
app/modules/Cronks/data/xml/to/icinga-tactical-overview-template-charts.xml
...
```

Please edit app/modules/Cronks/config/cronks.xml and add your created *.site.xml to make the new cronks accessible.

**Note**

After changing those configs you need to clear the config cache!

```
#> rm -rf app/cache/config/*.php
```

or

```
#> /usr/local/icinga-web/bin/clearcache.sh
```

You need more information? Please have a look at our [Development Wiki](#).

[Prev](#)[Up](#)[Next](#)

Upgrading Icinga-Web and
Icinga-Web Database

[Home](#)

Introduction to Icinga-Web

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Introduction to Icinga-Web

[Prev](#)

Chapter 6. User Interfaces

[Next](#)

Introduction to Icinga-Web

Overview

This introduction takes you on a short tour showing some aspects of Icinga-Web. It is not complete so there will be things which aren't covered. We're working on this.

Due to constant development some things change over time resulting in different appearance and additional options. So far we offer two small introductions, one for Icinga-Web up to version 1.2.x, and one starting with [version 1.3](#).

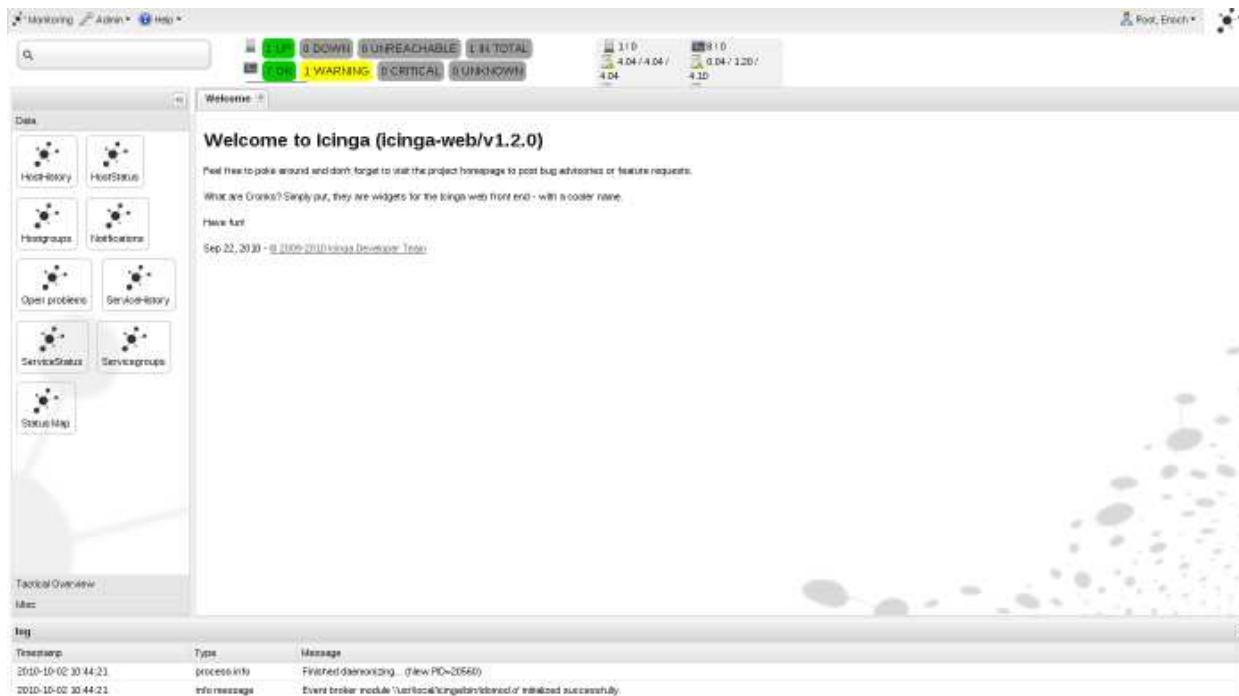
Introduction to Icinga-Web (up to 1.2.x)

If you followed the quickstart guides ([IDOUtils](#) and [Icinga-Web](#)) then please open your browser and enter `http://<icinga server>/icinga-web`. You will get the login screen

Figure 6.1. Icinga-Web login screen

You should be able to login using "root" and "password". This will take you to an overview page showing the state of the hosts and services being monitored

Figure 6.2. Icinga-Web overview



Main Screen

This is where everything comes together: You can drag and drop new windows over here resulting in a new opened tab. The views are customizable for the current user (they remain persistent) by dragging the column heading to the new place. Rightclicking on the heading allows to change the sort order or to hide columns. Selected search results will be opened here too, next to opening available cronks.

Figure 6.3. Icinga-Web main screen

Open problems											
	Instance	Host	Service	Host status	Service status	Last check	Last check	Attempt (H)	Attempt (S)	Hostcheck	Servicecheck
<input type="checkbox"/>	defau	gmx-smtp		DOWN	undefined	2010-09-2	2010-10-0	10 / 10	N/A	PING CRITI	
<input type="checkbox"/>	defau	gmx-smtp	SMTP	DOWN	CRITICAL	2010-09-2	2010-09-2	10 / 10	1 / 4	PING CRITI	CRITICAL -
<input type="checkbox"/>	defau	google-sm		DOWN	undefined	2010-09-2	2010-10-0	1 / 10	N/A	PING CRITI	
<input type="checkbox"/>	defau	google-sm	SMTP	DOWN	CRITICAL	2010-09-2	2010-09-2	1 / 10	1 / 4	PING CRITI	CRITICAL -
<input type="checkbox"/>	defau	web_de-s		DOWN	undefined	2010-09-2	2010-10-0	1 / 10	N/A	CRITICAL -	
<input type="checkbox"/>	defau	web_de-s	SMTP	DOWN	CRITICAL	2010-09-2	2010-09-2	1 / 10	1 / 4	CRITICAL -	CRITICAL -
<input type="checkbox"/>	defau	yahoo-sm		DOWN	undefined	2010-09-2	2010-10-0	1 / 10	N/A	PING CRITI	
<input type="checkbox"/>	defau	yahoo-sm	SMTP	DOWN	CRITICAL	2010-09-2	2010-09-2	1 / 10	1 / 4	PING CRITI	CRITICAL -

At the bottom, there are navigation buttons for back, forward, and search, along with a page number indicator (1 of 1) and a link to display topics (1 - 8 of 8).

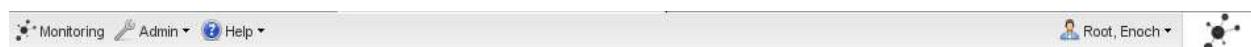
Status Cronk

The status cronk shows host and service counters for their respective states. If count is zero, the colour remains grey. Clicking on states opens a new tab in the main view showing the current clicked state only. Next to that, general process information is provided, just as

- Hosts | Services (active/passive)
- Host | Service execution time (min/avg/max)
- Host | Services latency (min/avg/max)

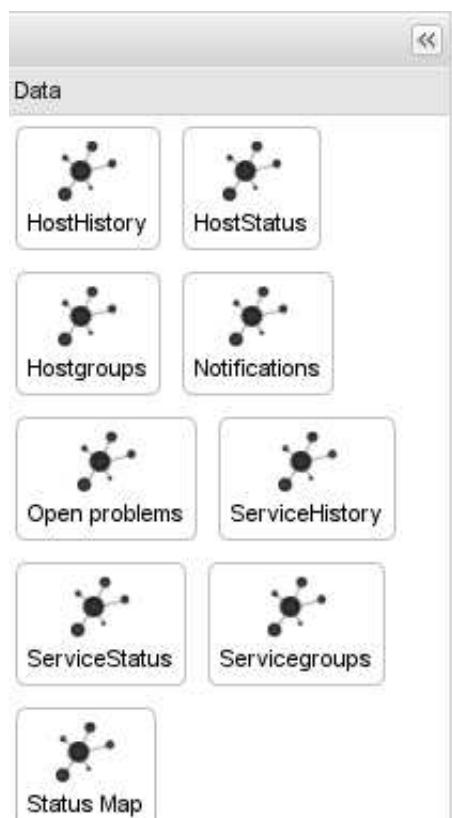
Figure 6.4. Icinga-Web status cronk*Top menu*

The top menu consists of general information about Icinga and the administration items for users, groups and logs. On the top right, you can see the user currently logged in and edit its preferences or log out of Icinga Web.

Figure 6.5. Icinga-Web top menu*Left menu*

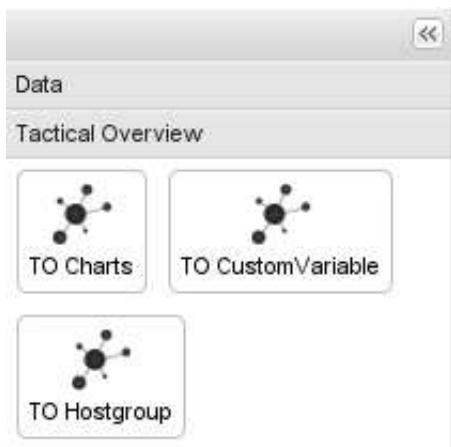
In the left (also hideable) menu you can select among different widgets (we call them "cronks"). You can either doubleclick on each cronk or drag it with the cursor into the main view. The categories are separated into

- "Data" for getting status, historical and configuration data

Figure 6.6. Icinga-Web data cronks

- "Tactical Overview" keeps general charts and customized ones (custom variables)

Figure 6.7. Icinga-Web tactical overview crons



- "Misc" contains several useful crons, just as iframe for external websites or the 1,2,3 columns for main view

Figure 6.8. Icinga-Web misc crons



Search

The search form shows results while typing. The results will be shown in a separate inlay window. By clicking/selecting the results, a new tab in the main view opens with more information. Please note that the search is currently case sensitive ("ping" vs. "PING").

Figure 6.9. Icinga-Web live search

The screenshot shows the Icinga-Web interface. At the top, there's a navigation bar with 'Monitoring', 'Admin', and 'Help' buttons. Below the navigation is a search bar with the placeholder 'Search: PING'. The main area has a 'Welcome' banner with a message 'Feel free to have a look around!' and a date 'Sep 22, 2010'. To the left is a sidebar with links like 'HostHistory', 'HostStatus', 'Hostgroups', 'Notifications', 'Open problems', 'ServiceHistory', 'ServiceStatus', 'Servicegroups', and 'Status Map'. The central part of the screen displays a table titled 'Type: service (20 Items)'. The table lists services with their names, types (PING), and statuses (all shown as 'OK'). A 'Close' button is at the bottom right of the table. The top right corner shows user information ('Root, Enoch') and a system ID ('149881').

Log

At the bottom, the current Icinga log is being shown. The log is refreshed automatically and can be hidden in order to allocate more space for the main view.

Figure 6.10. Icinga-Web log

log		
Timestamp	Type	Message
2010-09-29 01:32:28	process info	Auto-save of retention data completed successfully.
2010-09-29 01:25:29	info message	HOST FLAPPING ALERT: monitor;STOPPED; Host appears to have stopped flapping (3.8% change < 5.0% threshold)
2010-09-29 01:10:19	service OK	SERVICE ALERT: monitor;Current Load;OK;SOFT;3;OK - load average: 0.83, 3.60, 2.41
2010-09-29 01:09:19	service warning	SERVICE ALERT: monitor;Current Load;WARNING;SOFT;2;WARNING - load average: 1.71, 4.33, 2.55
2010-09-29 01:08:19	service warning	SERVICE ALERT: monitor;Current Load;WARNING;SOFT;1;WARNING - load average: 4.13, 5.20, 2.70

Cronks and Views

Icinga Web allows to open different cronks in order to view data, set filters for different views, send commands. The following list summarizes the possibilities in general (several Cronks can provide more, like sending commands).

Figure 6.11. Icinga-Web cronk bar



- Manual Refresh
- Settings

Enable/Disable Auto-Refresh

Get this <item> by url

Figure 6.12. Icinga-Web cronk bar



- Filter

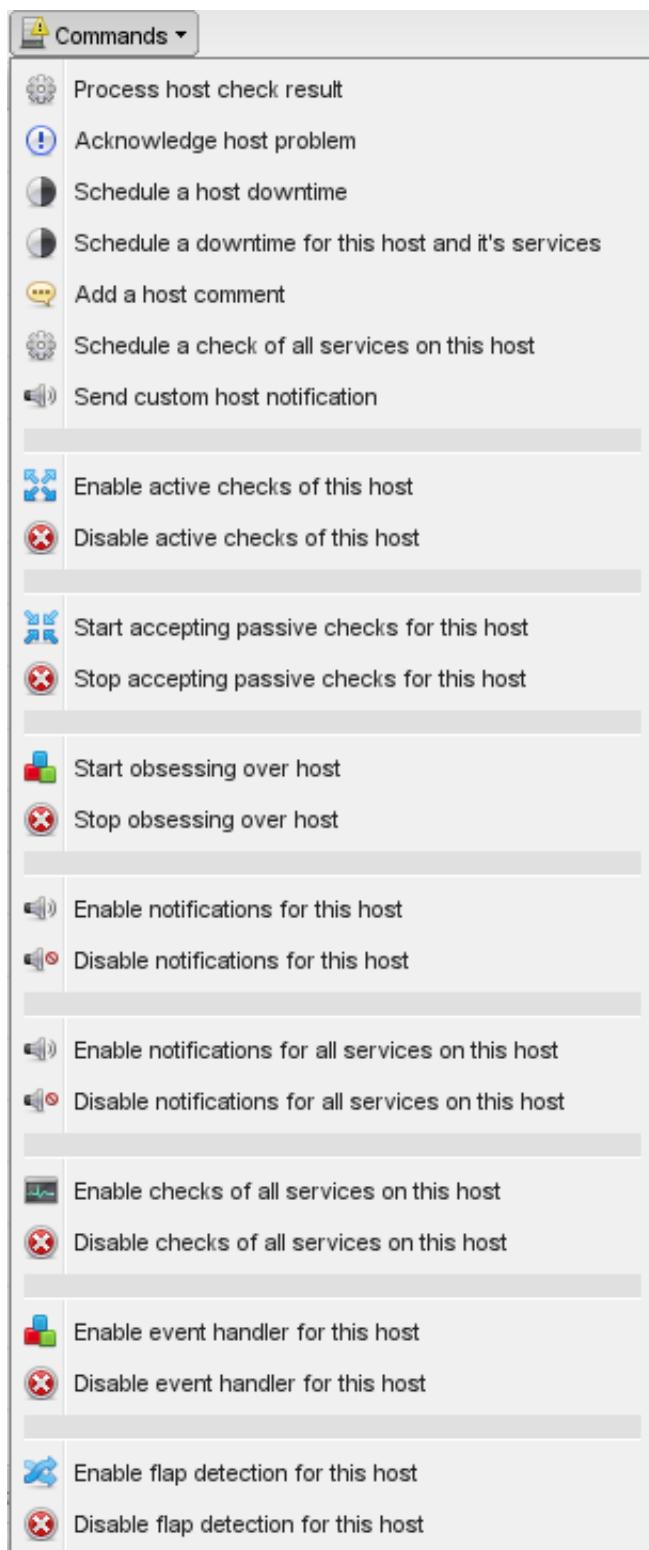
Modify/Remove

Commands

There are different commands available (check the chapter [External Commands](#) for more information on commands). Select the items which should be affected and then select the command to be executed.

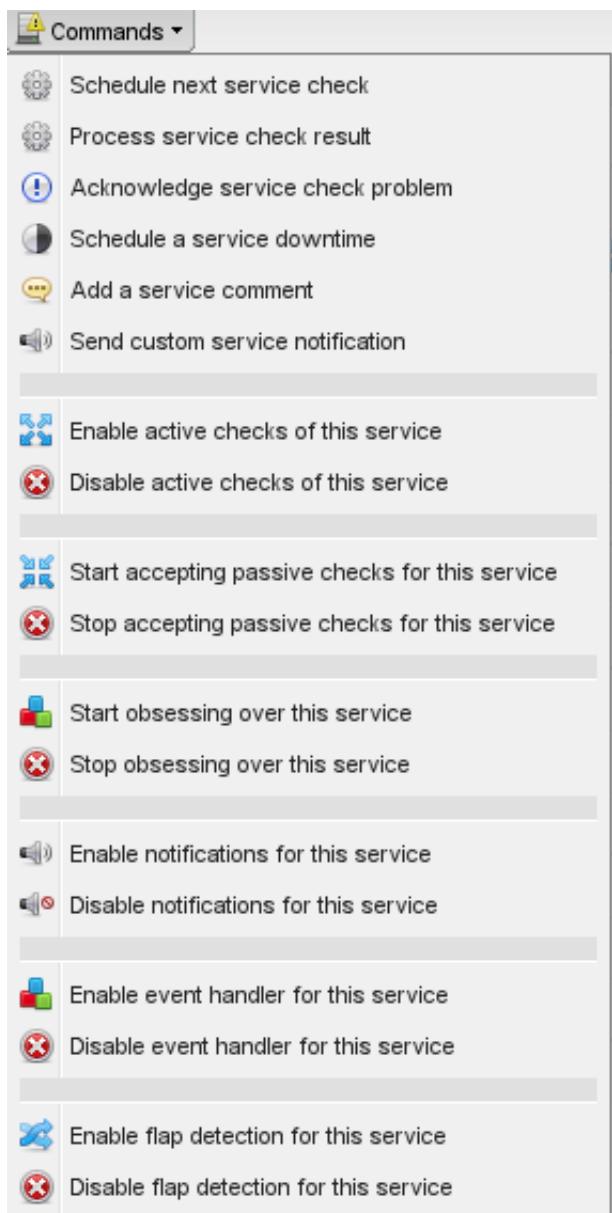
- Host Commands

Figure 6.13. Icinga-Web host commands



- Service Commands

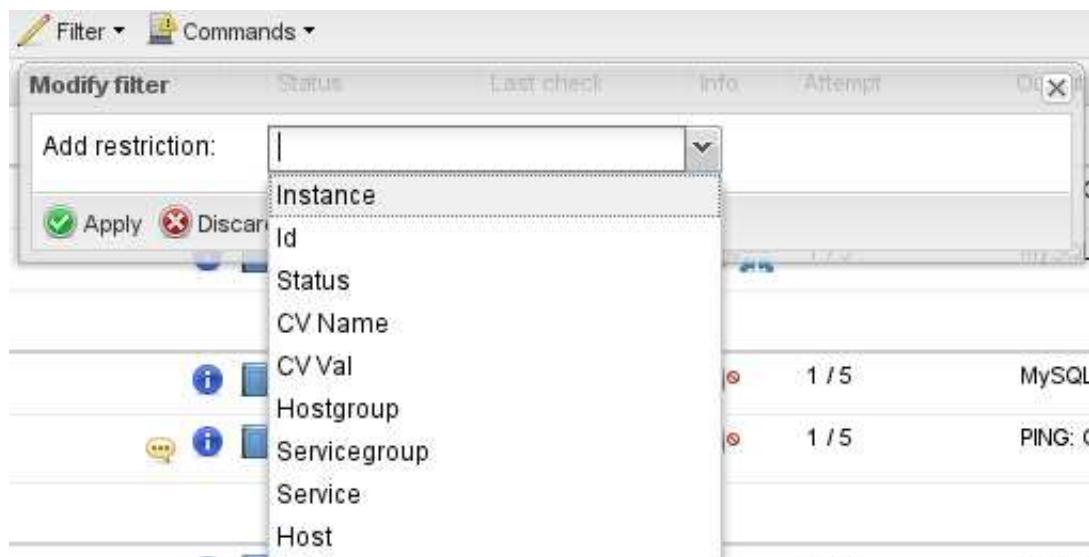
Figure 6.14. Icinga-Web service commands



Filters

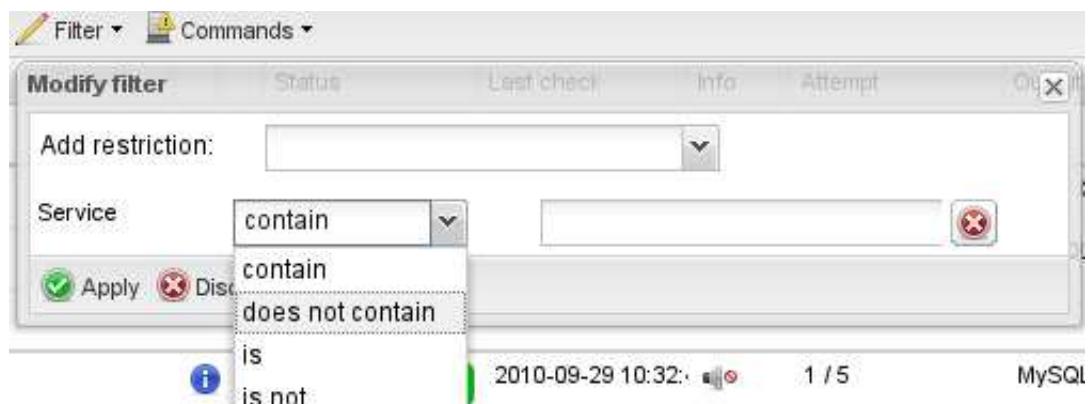
Icinga Web allows to set filters and create customized views which can be saved as own persistent cronk too. First, select "Filter" and "Modify". Add a restriction selected from the dropdown menu (this differs for various cronks). You can set more restrictions by repeating this procedure.

Figure 6.15. Icinga-Web filter restriction



Afterwards, specify on which condition the filter should match on the selected restriction (contain, does not contain, is, is not) and add a value into the form. Proposals are shown while you are typing.

Figure 6.16. Icinga-Web filter condition



Apply the filter to the current cronk. If needed, you can modify the filter afterwards and add/remove restrictions and conditions. An active filter is marked red.

Figure 6.17. Icinga-Web filter active



Administration

Enter the administration panel by selecting "Admin" from the top menu, and select "Users", "Groups" or "Logs".

Figure 6.18. Icinga-Web top menu admin



Users

You can add, remove and edit users.

Figure 6.19. Icinga-Web user admin

Available users						+ / -
ID	username	lastname	firstname	email	active	
1	root	Root	Enoch	root@localhost.local	<input checked="" type="checkbox"/>	
2	guest	Doe	John	john.dow@icinga.org	<input checked="" type="checkbox"/>	
3	demo	user	demo	test@demo.de	<input checked="" type="checkbox"/>	

Doubleclick a user to open a new inlay window which lets you edit very detailed options. The same options apply when adding a new user. You can modify the size by dragging the cursor in the corners.

- General information
- Change password (and optional AuthKey for API)
- Permissions e.g. which group membership
- Principals for special roles

Figure 6.20. Icinga-Web edit user

Edit user

General information

User name:	guest		
Name:	John	Surname:	Doe
Email:	john.dow@icinga.org		
Disabled:	<input type="checkbox"/>		
Auth via:	internal		

Change Password

Password:	
Confirm password:	
Authkey for Api (optional):	

Meta information

Created:	2010-09-01 16:14:16
Modified:	2010-09-02 08:13:02

Permissions

Groups

appkit_admin (AppKit admin) :	<input type="checkbox"/>
appkit_user (Appkit user test) :	<input type="checkbox"/>
guest (Unauthorized Guest) :	<input type="checkbox"/>
icinga_user (The default representation of a ICINGA user) :	<input checked="" type="checkbox"/>

Principals

- + -**
- **principals**
 - **credential**
 - 🔑 icinga.demoMode

Save

Groups

You can add, remove and edit groups. You can also modify group inheritance within the group tree on the right side.

Figure 6.21. Icinga-Web group admin

The screenshot shows the Icinga-Web group administration interface. On the left, there is a table titled "Available groups" with the following data:

ID	groupname	description	isActive
3	appkit_admin	AppKit admin	<input checked="" type="checkbox"/>
2	appkit_user	Appkit user test	<input checked="" type="checkbox"/>
4	guest	Unauthorized Guest	<input checked="" type="checkbox"/>
1	icinga_user	The default representat	<input checked="" type="checkbox"/>

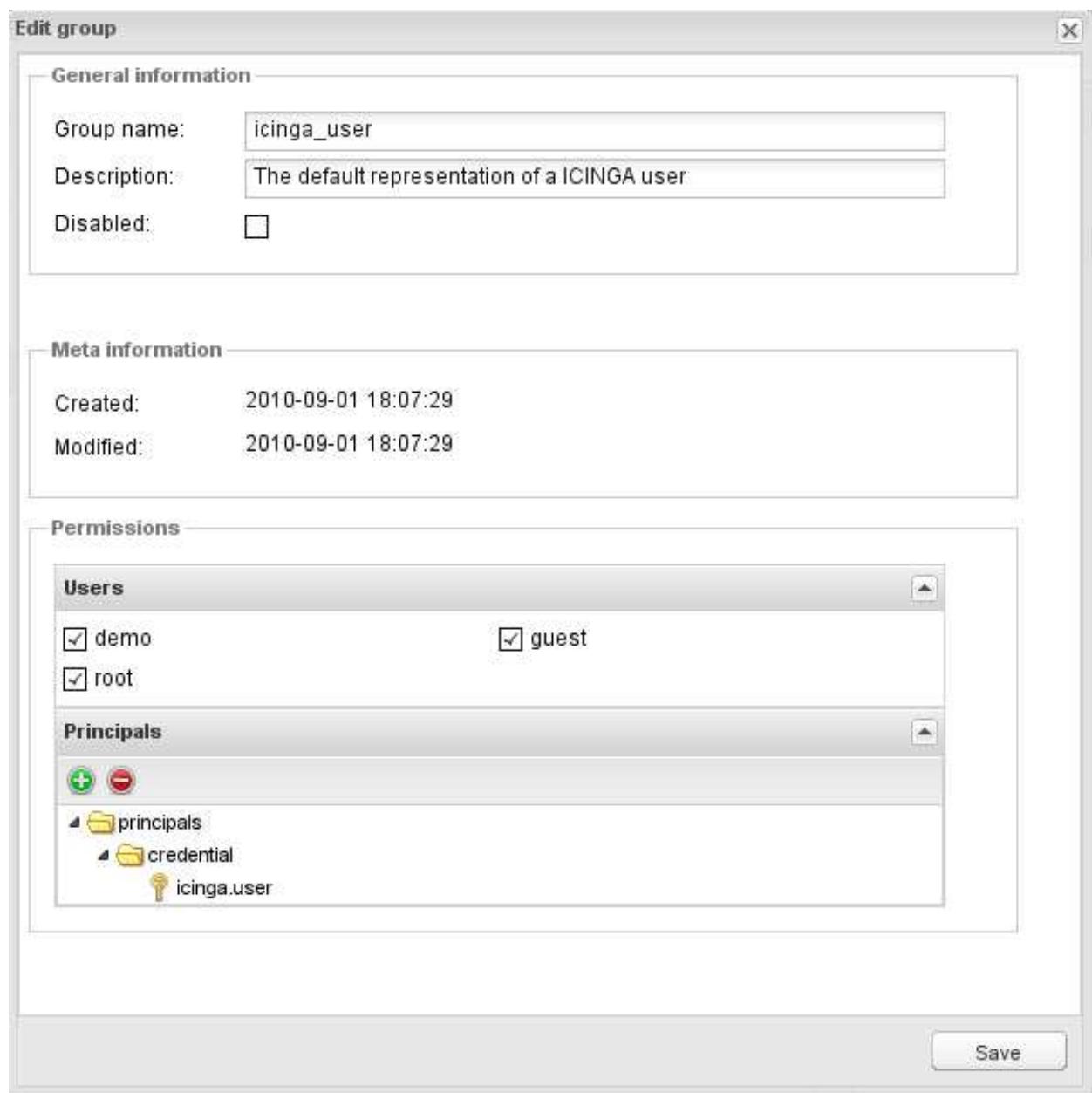
On the right, there is a "Group inheritance" tree view:

- Root
 - appkit_user
 - guest
 - icinga_user

Doubleclick a group to open a new inlay window which lets you edit very detailed options. The same options apply when adding a new group. You can modify the size by dragging the cursor in the corners.

- General information + Permissions (which users are part of this group)
- Principals for special roles

Figure 6.22. Icinga-Web groups



Principals

Within the user and group administration panel, you can add/remove/edit principals. You can see default available principals in the picture:

Figure 6.23. Icinga-Web principals

Select a principal (Press Ctrl for multiple selects)		
Principal	Description	Type
IcingaHostgroup	Limit data access to specific hostgroups	icinga
IcingaServicegroup	Limit data access to specific servicegroups	icinga
IcingaHostCustomVariablePair	Limit data access to specific custom variables	icinga
IcingaServiceCustomVariablePair	Limit data access to specific custom variables	icinga
IcingaContactgroup	Limit data access to users contact group membership	icinga
IcingaCommandRo	Limit access to commands	icinga
appkit.access	Access to login-page (which, actually, means no access)	credential
icinga.user	Access to icinga	credential
appkit.admin.groups	Access to group editor	credential
appkit.admin.users	Access to user editor	credential
appkit.admin	Access to admin panel	credential
appkit.user.dummy	Basic right for users	credential
appkit.api.access	Access to web-based api adapter	credential
icinga.demoMode	Hide features like password reset which are not wanted in demo syst	credential

Logs

You can view several logs here, in order to catch up on possible errors.

Figure 6.24. Icinga-Web logs

Time	Message	Severity
No data to display		

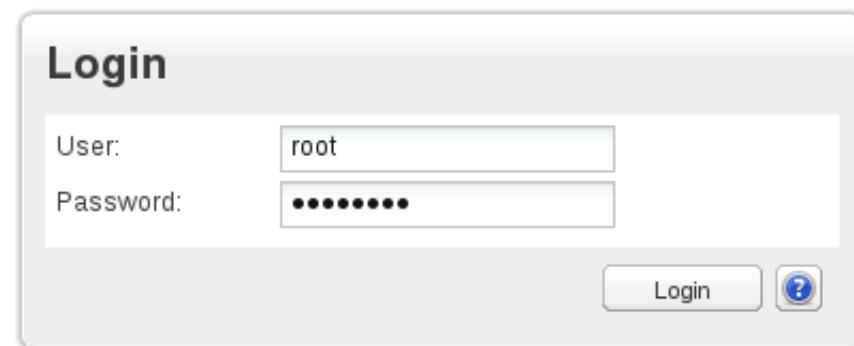
Available logs

- debug
- icinga-web

Introduction to Icinga-Web (>= 1.3.x)

If you followed the quickstart guides ([IDOUtils](#) and [Icinga-Web](#)) then please open your browser and enter `http://<icinga server>/icinga-web`. You will get the login screen

Figure 6.25. Icinga-Web login screen



You should be able to login using "root" and "password". This will take you to an overview page showing the state of the hosts and services being monitored

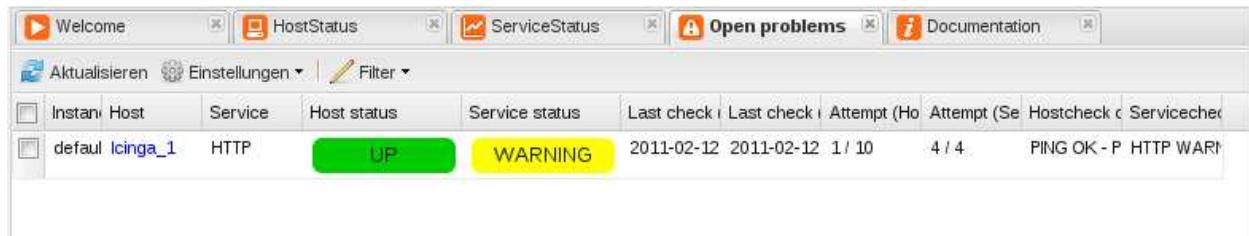
Figure 6.26. Icinga-Web overview



Main Screen

This is where everything comes together: You can drag and drop new windows over here resulting in a new opened tab. The views are customizable for the current user (they remain persistent) by dragging the column heading to the new place. Rightclicking on the heading allows to change the sort order or to hide columns. Selected search results will be opened here too, next to opening available cronks.

Figure 6.27. Icinga-Web main screen



Status Cronk

The status cronk shows host and service counters for their respective states. If count is zero, the colour remains grey. Clicking on states opens a new tab in the main view showing the current clicked state only. Next to that, general process information is provided, just as

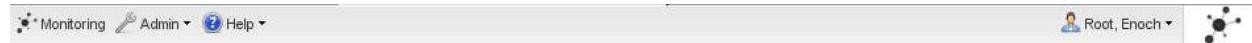
- Hosts | Services (active/passive)
- Host | Service execution time (min/avg/max)
- Host | Services latency (min/avg/max)

Figure 6.28. Icinga-Web status cronk



Top menu

The top menu consists of general information about Icinga and the administration items for users, groups and logs. On the top right, you can see the user currently logged in and edit its preferences or log out of Icinga Web.

Figure 6.29. Icinga-Web top menu*Left menu*

In the left (also hideable) menu you can select among different widgets (we call them "cronks"). You can either doubleclick on each cronk or drag it with the cursor into the main view. The categories are separated into

- "Data" for getting status, historical and configuration data

Figure 6.30. Icinga-Web data cronks

- "Tactical Overview" keeps general charts and customized ones (custom variables)

Figure 6.31. Icinga-Web tactical overview cronks

- "Misc" contains several useful cronks, just as iframe for external websites or the 1,2,3 columns for main view

Figure 6.32. Icinga-Web misc cronks



Search

The search form shows results while typing. The results will be shown in a separate inlay window. By clicking/selecting the results, a new tab in the main view opens with more information. Please note that the search is currently case sensitive ("ping" vs. "PING").

Figure 6.33. Icinga-Web live search

Type	Name	Status	Info	Attempt	Out
host	HAM_00(127.0.0.1) Hamburg Router	UP		1 / 10	PING
host	HAM_01(127.0.0.1) Hamburg S01	UP		1 / 10	PING
host	HAM_02(127.0.0.1) Hamburg S02	UP		1 / 10	PING
host	HAM_03(127.0.0.1) Hamburg S03	UP		1 / 10	PING
host	HAM_04(127.0.0.1) Hamburg S04	UP		1 / 10	PING
host	HAM_05(127.0.0.1) Hamburg S05	UP		1 / 10	PING
host	HAM_06(127.0.0.1) Hamburg S06	UP		1 / 10	PING
host	HAM_07(127.0.0.1) Hamburg S07	UP		1 / 10	PING
host	HAM_08(127.0.0.1)	UP		1 / 10	PING

Log

The log view has moved to an own cronk so it's not shown at the bottom anymore.

Figure 6.34. Icinga-Web log

Instance	Timestamp	Type	Message
default	2011-02-12 12:23:24	process info	Finished daemonizing... (New PID=3995)
default	2011-02-12 12:23:24	info message	Event broker module '/usr/local/icinga/bin/idomod.o' initialized successfully.
default	2011-02-12 12:18:35	process info	Caught SIGTERM, shutting down...
default	2011-02-12 12:18:35	process info	Successfully shutdown... (PID=4218)
default	2011-02-12 12:18:34	info message	idomod: Error writing to data sink! Some output may get lost...
default	2011-02-12 12:18:34	info message	idomod: Please check remote ido2db log, database connection or SSL Parameters

Cronks and Views

Icinga Web allows to open different cronks in order to view data, set filters for different views, send commands. The following list summarizes the possibilities in general (several Cronks can provide more, like sending commands).

Figure 6.35. Icinga-Web cronk bar



- Manual Refresh
- Settings
 - Enable/Disable Auto-Refresh
 - Get this <item> by url

Figure 6.36. Icinga-Web cronk bar



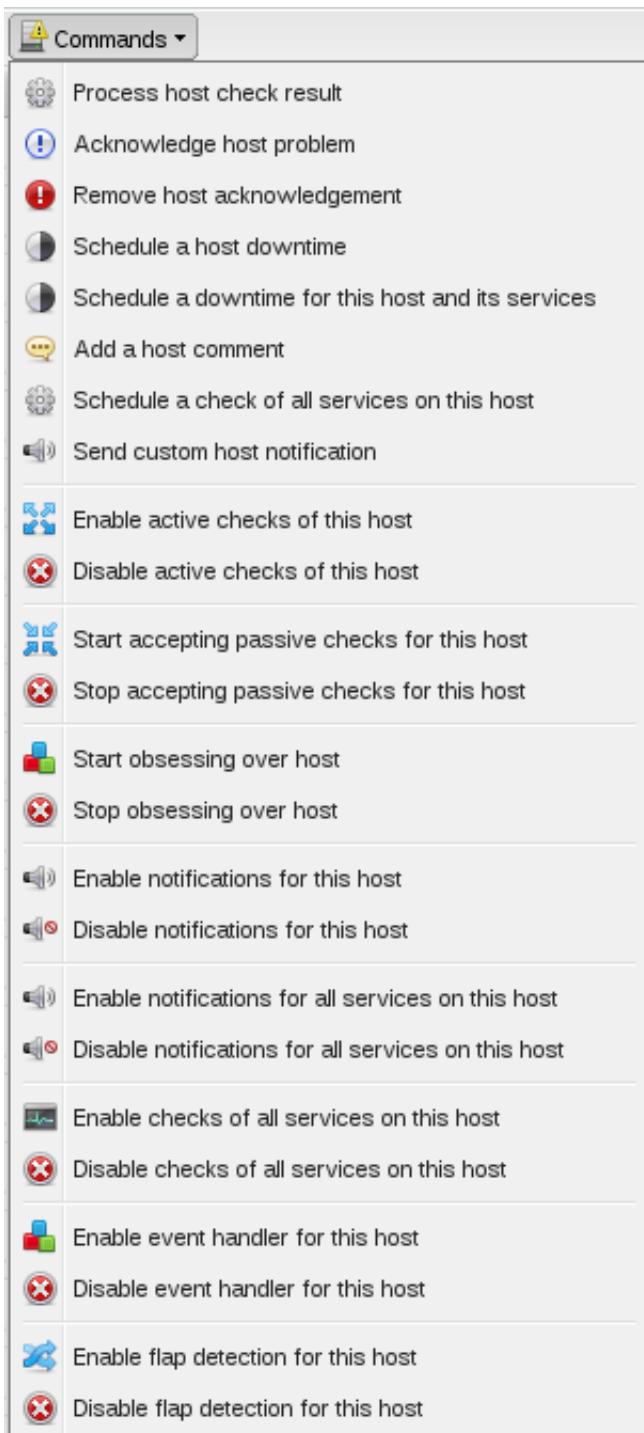
- Filter
 - Modify/Remove

Commands

There are different commands available (check the chapter [External Commands](#) for more information on commands). Select the items which should be affected and then select the command to be executed.

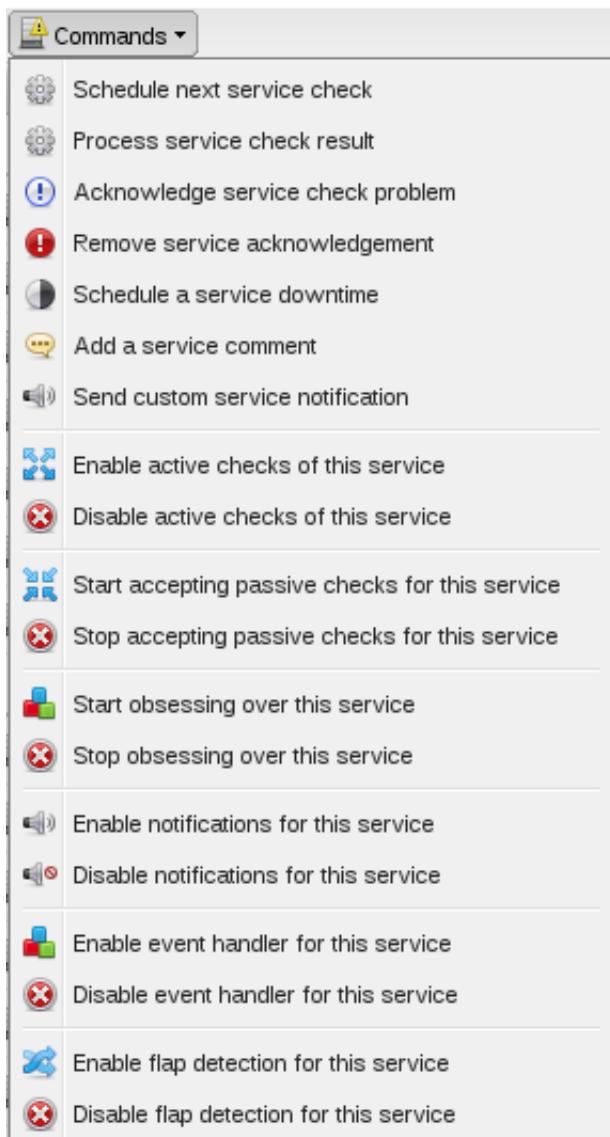
- Host Commands

Figure 6.37. Icinga-Web host commands



- Service Commands

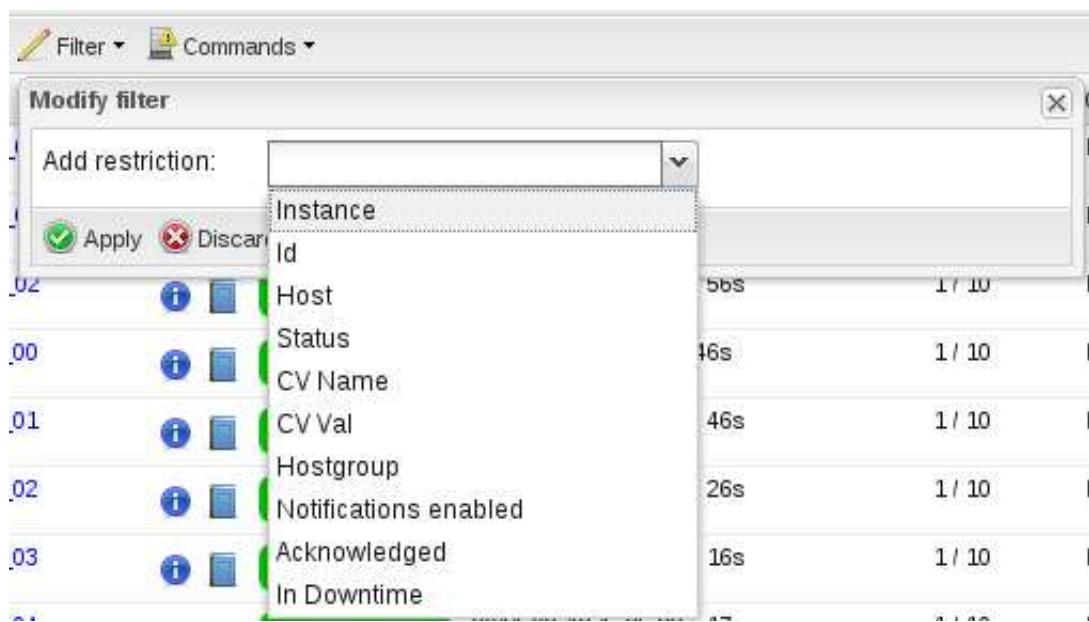
Figure 6.38. Icinga-Web service commands



Filters

Icinga Web allows to set filters and create customized views which can be saved as own persistent cronk too. First, select "Filter" and "Modify". Add a restriction selected from the dropdown menu (this differs for various cronks). You can set more restrictions by repeating this procedure.

Figure 6.39. Icinga-Web filter restriction



Afterwards, specify on which condition the filter should match on the selected restriction (contain, does not contain, is, is not) and add a value into the form. Proposals are shown while you are typing.

Figure 6.40. Icinga-Web filter condition



Apply the filter to the current cronk. If needed, you can modify the filter afterwards and add/remove restrictions and conditions. An active filter is marked red.

Figure 6.41. Icinga-Web filter active



Administration

Enter the administration panel by selecting "Admin" from the top menu, and select "Users", "Groups" or "Logs".

Figure 6.42. Icinga-Web top menu admin



Users

You can add, remove and edit users.

Figure 6.43. Icinga-Web user admin

Available users						+ / -
ID	username	lastname	firstname	email	active	
1	root	Root	Enoch	root@localhost.local		
2	guest	Doe	John	john.dow@icinga.org		
3	demo	user	demo	test@demo.de		

Doubleclick a user to open a new inlay window which lets you edit very detailed options. The same options apply when adding a new user. You can modify the size by dragging the cursor in the corners.

- General information
- Change password (and optional AuthKey for API)
- Permissions e.g. which group membership
- Principals for special roles

Figure 6.44. Icinga-Web edit user

Edit user

General information

User name:	guest		
Name:	John	Surname:	Doe
Email:	john.dow@icinga.org		
Disabled:	<input type="checkbox"/>		
Auth via:	internal		

Change Password

Password:	
Confirm password:	
Authkey for Api (optional):	

Meta information

Created:	2010-09-01 16:14:16
Modified:	2010-09-02 08:13:02

Permissions

Groups

appkit_admin (AppKit admin) :	<input type="checkbox"/>
appkit_user (Appkit user test) :	<input type="checkbox"/>
guest (Unauthorized Guest) :	<input type="checkbox"/>
icinga_user (The default representation of a ICINGA user) :	<input checked="" type="checkbox"/>

Principals

- + -**
- **principals**
 - **credential**
 - 🔑 **icinga.demoMode**

Save

Groups

You can add, remove and edit groups. You can also modify group inheritance within the group tree on the right side.

Figure 6.45. Icinga-Web group admin

The screenshot shows the Icinga-Web group administration interface. On the left, there is a table titled "Available groups" with the following data:

ID	groupname	description	isActive
3	appkit_admin	AppKit admin	<input checked="" type="checkbox"/>
2	appkit_user	Appkit user test	<input checked="" type="checkbox"/>
4	guest	Unauthorized Guest	<input checked="" type="checkbox"/>
1	icinga_user	The default representat	<input checked="" type="checkbox"/>

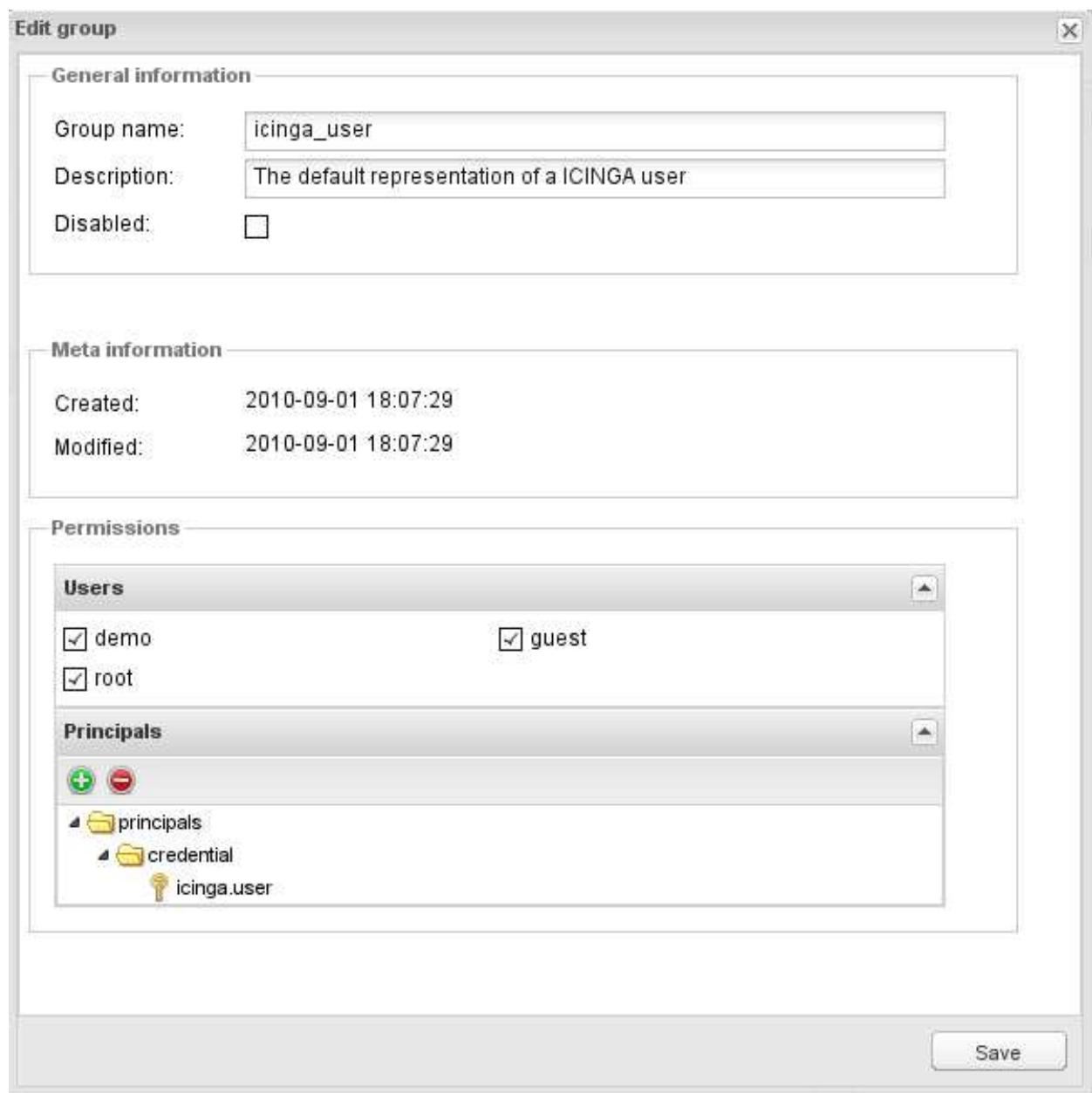
On the right, there is a "Group inheritance" tree view:

- Root
 - appkit_user
 - guest
 - icinga_user

Doubleclick a group to open a new inlay window which lets you edit very detailed options. The same options apply when adding a new group. You can modify the size by dragging the cursor in the corners.

- General information + Permissions (which users are part of this group)
- Principals for special roles

Figure 6.46. Icinga-Web groups



Principals

Within the user and group administration panel, you can add/remove/edit principals. You can see default available principals in the picture:

Figure 6.47. Icinga-Web principals

Select a principal (Press Ctrl for multiple selects)		
Principal	Description	Type
IcingaHostgroup	Limit data access to specific hostgroups	icinga
IcingaServicegroup	Limit data access to specific servicegroups	icinga
IcingaHostCustomVariablePair	Limit data access to specific custom variables	icinga
IcingaServiceCustomVariablePair	Limit data access to specific custom variables	icinga
IcingaContactgroup	Limit data access to users contact group membership	icinga
IcingaCommandRo	Limit access to commands	icinga
appkit.access	Access to login-page (which, actually, means no access)	credential
icinga.user	Access to icinga	credential
appkit.admin.groups	Access to group editor	credential
appkit.admin.users	Access to user editor	credential
appkit.admin	Access to admin panel	credential
appkit.user.dummy	Basic right for users	credential
appkit.api.access	Access to web-based api adapter	credential
icinga.demoMode	Hide features like password reset which are not wanted in demo syst	credential

Logs

You can view several logs here, in order to catch up on possible errors.

Figure 6.48. Icinga-Web logs

Time	Message	Severity
No data to display		

Available logs

- debug
- icinga-web

Task

Several changes require clearing the cache. Instead of using the command line you can issue the command via the "Tasks" menu.

Figure 6.49. Icinga-Web Tasks

Clear cache

Clear the agavi configuration cache to apply new xml configuration.

[Prev](#)

[Up](#)

[Next](#)

Configuration Overview of
Icinga-Web

[Home](#)

Integration of PNP4Nagios into
Icinga-Web

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Integration of PNP4Nagios into Icinga-Web

[Prev](#)[Chapter 6. User Interfaces](#)[Next](#)

Integration of PNP4Nagios into Icinga-Web

Here we'll give you some instructions on how to integrate PNP4Nagios into the Icinga-Web frontend. If you want to integrate PNP4Nagios into the Icinga Classic UI please follow the PNP4Nagios [documentation](#).

Figure 6.50. PNP4Nagios integrated in Icinga-Web

Instance	Service	Perfdata	Status	Last check	Info	Attempt	Output
default	SSH		OK	2010-09-01 16:06:39		1/4	SSH OK - OpenSSH_5.4 (protocol
default	Swap Usage		OK	2010-09-01 16:07:14		1/4	SWAP OK - 96% free (3779 MB o
default	Total Processes		OK	2010-09-01 16:02:52		1/4	PROCS OK: 203 processes with
default	Current Load		OK	2010-09-01 16:06:29		1/4	OK - load average: 0.55, 0.41, 0.
default	Current Users		OK	2010-09-01 16:04:07		1/4	USERS OK - 4 users currently lo
default	HTTP		WARNING	2010-09-01 16:04:44		4/4	HTTP WARNING: HTTP/1.1 403
default	PING		OK	2010-09-01 16:05:22		1/4	PING OK - Packet loss = 0%, RT
default	Root Partition		OK	2010-09-01 16:05:59		1/4	DISK OK - free space: / 21780 M

Install PNP4Nagios

1. Install PNP4Nagios as described in the PNP4Nagios [documentation](#)
2. Change the PNP4Nagios configuration to match your Icinga configuration. You probably may want to change these:

```
#> vi npcd.cfg
user = icinga
group = icinga
log_file = /var/log/icinga/npcd.log
perfdata_spool_dir = /var/icinga/spool/
perfdata_file = /var/icinga/perfdata.dump

#> vi process_perfdata.cfg
LOG_FILE = /var/log/icinga/perfdata.log

#> vi config.php
$conf['nagios_base'] = "/icinga/cgi-bin";
```

Create a configuration to include PNP4Nagios host pages in Icinga-Web

1. Create a new grid view

Make a copy of the default icinga-host-template.xml in app/modules/Cronks/data/xml/grid under your Icinga-Web installation path, example:

```
#> cp /usr/local/icinga-web/app/modules/Cronks/data/xml/grid/icinga-host-template.xml \
/usr/local/icinga-web/app/modules/Cronks/data/xml/grid/icinga-my-host-template.xml
```

In the new file we just created add a new field definition:

```
<field name="npn4nagios_host_link">
  <!-- datasource maps a data field from api call -->
  <datasource>
    <parameter name="field">HOST_NAME</parameter>
  </datasource>
  <display>
```

```

<parameter name="visible">true</parameter>
<parameter name="label">Perfdata</parameter>
<parameter name="width">55</parameter>

<parameter name="Ext.grid.Column">
    <parameter name="menuDisabled">true</parameter>
    <parameter name="fixed">true</parameter>
</parameter>

<parameter name="jsFunc">
    <!-- function to display column with icon in host status grid view -->
    <parameter>
        <parameter name="namespace">Cronk.grid.ColumnRenderer</parameter>
        <parameter name="function">columnImage</parameter>
        <parameter name="type">renderer</parameter>

        <parameter name="arguments">
            <parameter name="image">images/icons/application_view_gallery.png</parameter>
            <parameter name="css">x-icinga-grid-link</parameter>
            <parameter name="attr">
                <parameter name="qtip">Show host perfdata for this host</parameter>
            </parameter>
        </parameter>
    </parameter>
</parameter>

<!-- create cell click event for the previously defined column -->
<parameter>
    <parameter name="namespace">Cronk.grid.IcingaColumnRenderer</parameter>
    <parameter name="function">iFrameCronk</parameter>
    <parameter name="type">cellclick</parameter>
    <parameter name="arguments">
        <parameter name="title">Host perfdata for {host_name}</parameter>
        <parameter name="url"><![CDATA[ /pnp4nagios/index.php/graph?host={host_name}&srv=_HOST_ ]]></parameter>
        <parameter name="activateOnClick">true</parameter>
    </parameter>
</parameter>
</parameter>
</display>

<filter>
    <parameter name="enabled">false</parameter>
</filter>

<order>
    <parameter name="enabled">false</parameter>
</order>
</field>

```

2. Add the new grid view to the "Data" cronk-container

Edit cronks.xml in directory app/modules/Cronks/config/ under your icinga-web add a section:

```

<ae:parameter name="gridMyHostView">
    <ae:parameter name="module">Cronks</ae:parameter>
    <ae:parameter name="action">System.ViewProc</ae:parameter>
    <ae:parameter name="hide">false</ae:parameter>
    <ae:parameter name="description">Viewing Host status in a grid including perfdata link</ae:parameter>
    <ae:parameter name="name">MyHostStatus</ae:parameter>
    <ae:parameter name="image">cronks.Stats</ae:parameter>
    <ae:parameter name="categories">data</ae:parameter>
    <ae:parameter name="ae:parameter">
        <ae:parameter name="template">icinga-my-host-template</ae:parameter>
    </ae:parameter>
</ae:parameter>

```

Create a configuration to include pnp4nagios service pages in icinga-web

1. Create a new grid view

Make a copy of the default icinga-service-template.xml in app/modules/Cronks/data/xml/grid under your icinga-web installation path.

```
cp /usr/local/icinga-web/app/modules/Cronks/data/xml/grid/icinga-service-template.xml \
/usr/local/icinga-web/app/modules/Cronks/data/xml/grid/icinga-my-service-template.xml
```

In the new file we just created add a new field definition:

```

<field name="pnp4nagios_service_link">
    <!-- datasource maps a data field from api call -->
    <datasource>
        <parameter name="field">SERVICE_NAME</parameter>
    </datasource>

    <display>
        <parameter name="visible">true</parameter>
        <parameter name="label">Perfdata</parameter>
        <parameter name="width">55</parameter>
    </display>

```

```

<parameter name="Ext.grid.Column">
    <parameter name="menuDisabled">true</parameter>
    <parameter name="fixed">true</parameter>
</parameter>

<parameter name="jsFunc">
    <!-- function to display column with icon in host status grid view -->
    <parameter>
        <parameter name="namespace">Cronk.grid.ColumnRenderer</parameter>
        <parameter name="function">columnImage</parameter>
        <parameter name="type">renderer</parameter>

        <parameter name="arguments">
            <parameter name="image">images/icons/application_view_gallery.png</parameter>
            <parameter name="css">x-icinga-grid-link</parameter>
            <parameter name="attr">
                <parameter name="tip">Show perftdata for this service</parameter>
            </parameter>
        </parameter>
    </parameter>
</parameter>

<!-- create cell click event for the previously defined column -->
<parameter>
    <parameter name="namespace">Cronk.grid.IcingaColumnRenderer</parameter>
    <parameter name="function">iFrameCronk</parameter>
    <parameter name="type">cellclick</parameter>
    <parameter name="arguments">
        <parameter name="title">Service perftdata for {service_name} on {host_name}</parameter>
        <parameter name="url"><![CDATA[/{pn4nagios}/index.php/graph?host={host_name}&srv={service_name}]]></parameter>
        <parameter name="activateOnClick">true</parameter>
    </parameter>
</parameter>
</parameter>
</display>

<filter>
    <parameter name="enabled">false</parameter>
</filter>

<order>
    <parameter name="enabled">false</parameter>
</order>
</field>

```

2. Add the new grid view to the "Data" Cronk-Container

Edit cronks.xml in directory app/modules/Cronks/config/ under your icinga-web add a section:

```

<ae:parameter name="gridMyServiceView">
    <ae:parameter name="module">Cronks</ae:parameter>
    <ae:parameter name="action">System.ViewProc</ae:parameter>
    <ae:parameter name="hide">false</ae:parameter>
    <ae:parameter name="description">Viewing service status in a grid including perftdata link</ae:parameter>
    <ae:parameter name="name">MyServiceStatus</ae:parameter>
    <ae:parameter name="image">cronks.Stats2</ae:parameter>
    <ae:parameter name="categories">data</ae:parameter>
    <ae:parameter name="ae:parameter">
        <ae:parameter name="template">icinga-my-service-template</ae:parameter>
    </ae:parameter>
</ae:parameter>

```

3. Using your new created grid view as default (if you want)

Please backup your original view first:

```
#> cp data/xml/grid/icinga-service-template.xml data/xml/grid/icinga-service-template.bak
```

then

```
#> cp data/xml/grid/icinga-my-service-template.xml data/xml/grid/icinga-service-template.xml
```

Clear the cache, like described below. Performance graphs are now in your default "serviceStatus" Cronk!



Note

If you edit any *.xml-file you have to clear the cache afterwards!

```
#> rm -f app/cache/config/*.php
```

or /path/to/clearcache.sh

```
#> /usr/local/icinga-web/bin/clearcache.sh
```

That's all, you're done!

[Prev](#)

[Up](#)

[Next](#)

[Introduction to Icinga-Web](#)

[Home](#)

[Chapter 7. Advanced Topics](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 7. Advanced Topics

[Prev](#)

[Next](#)

Chapter 7. Advanced Topics

Table of Contents

- [External Commands](#)
 - [Event Handlers](#)
 - [Volatile Services](#)
 - [Service and Host Freshness Checks](#)
 - [Distributed Monitoring](#)
 - [Redundant and Failover Network Monitoring](#)
 - [Detection and Handling of State Flapping](#)
 - [Notification Escalations](#)
 - [Escalation Condition](#)
 - [On-Call Rotations](#)
 - [Monitoring Service and Host Clusters](#)
 - [Host and Service Dependencies](#)
 - [State Stalking](#)
 - [Performance Data](#)
 - [Scheduled Downtime](#)
 - [Using The Embedded Perl Interpreter](#)
 - [Adaptive Monitoring](#)
 - [Predictive Dependency Checks](#)
 - [Cached Checks](#)
 - [Passive Host State Translation](#)
 - [Service and Host Check Scheduling](#)
 - [Custom CGI Headers and Footers](#)
 - [Object Inheritance](#)
 - [Time-Saving Tricks For Object Definitions](#)
-

[Prev](#)

[Next](#)

Integration of PNP4Nagios into
Icinga-Web

[Home](#)

External Commands

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



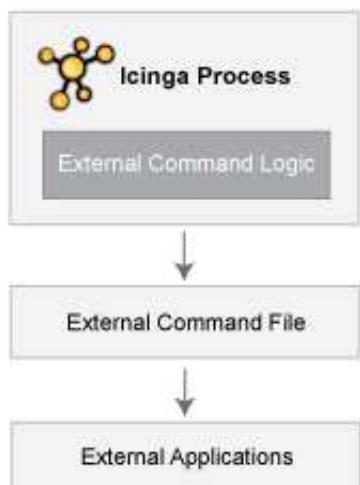
External Commands

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

External Commands

Introduction

Icinga can process commands from external applications (including the CGIs) and alter various aspects of its monitoring functions based on the commands it receives. External applications can submit commands by writing to the [command file](#), which is periodically processed by the Icinga daemon.



Enabling External Commands

In order to have Icinga process external commands, make sure you do the following:

- Enable external command checking with the [check_external_commands](#) option.
- Set the frequency of command checks with the [command_check_interval](#) option.
- Specify the location of the command file with the [command_file](#) option.
- Setup proper permissions on the directory containing the external command file, as described in the [quickstart guide](#).

When Does Icinga Check For External Commands?

- At regular intervals specified by the [command_check_interval](#) option in the main configuration file
- Immediately after [event handlers](#) are executed. This is in addition to the regular cycle of external command checks and is done to provide immediate action if an event handler submits commands to Icinga.

Using External Commands

External commands can be used to accomplish a variety of things while Icinga is running. Examples of what can be done include temporarily disabling notifications for services and hosts, temporarily disabling service checks, forcing immediate service checks, adding comments to hosts and services, etc.

Command Format

External commands that are written to the [command file](#) have the following format...

```
[ time ] command_id; command_arguments
```

...where *time* is the time (in *time_t* format) that the external application submitted the external command to the command file. The values for the *command_id* and *command_arguments* arguments will depend on what command is being submitted to Icinga.

A full listing of external commands that can be used can be found in the [list of external commands](#) in the development section.

[Prev](#)[Up](#)[Next](#)[Chapter 7. Advanced Topics](#)[Home](#)[Event Handlers](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Event Handlers

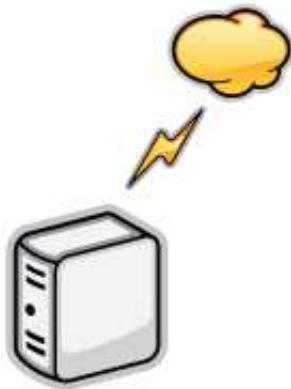
[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Event Handlers

Introduction



Event handlers are optional system commands (scripts or executables) that are run whenever a host or service state change occurs. They are executed on the system where the check is scheduled (initiated).

An obvious use for event handlers is the ability for Icinga to proactively fix problems before anyone is notified. Some other uses for event handlers include:

- Restarting a failed service
- Entering a trouble ticket into a helpdesk system
- Logging event information to a database
- Cycling power on a host*
- etc.

* Cycling power on a host that is experiencing problems with an automated script should not be implemented lightly. Consider the consequences of this carefully before implementing automatic reboots. :-)

When Are Event Handlers Executed?

Event handlers are executed when a service or host:

- Is in a SOFT problem state
- Initially goes into a HARD problem state
- Initially recovers from a SOFT or HARD problem state

SOFT and HARD states are described in detail [here](#).

Event Handler Types

There are different types of optional event handlers that you can define to handle host and state changes:

- Global host event handler
- Global service event handler
- Host-specific event handlers
- Service-specific event handlers

Global host and service event handlers are run for *every* host or service state change that occurs, immediately prior to any host- or service-specific event handler that may be run. You can specify global event handler commands by using the [global_host_event_handler](#) and [global_service_event_handler](#) options in your main configuration file.

Individual hosts and services can have their own event handler command that should be run to handle state changes. You can specify an event handler that should be run by using the [event_handler](#) directive in your [host](#) and [service](#) definitions. These host- and service-specific event handlers are executed immediately after the (optional) global host or service event handler is executed.

Enabling Event Handlers

Event handlers can be enabled or disabled on a program-wide basis by using the [enable_event_handlers](#) in your main configuration file.

Host- and service-specific event handlers can be enabled or disabled by using the [event_handler_enabled](#) directive in your [host](#) and [service](#) definitions. Host- and service-specific event handlers will not be executed if the global [enable_event_handlers](#) option is disabled.

Event Handler Execution Order

As already mentioned, global host and service event handlers are executed immediately before host- or service-specific event handlers.

Event handlers are executed for HARD problem and recovery states immediately after notifications are sent out.

Writing Event Handler Commands

Event handler commands will likely be shell or perl scripts, but they can be any type of executable that can run from a command prompt. At a minimum, the scripts should take the following [macros](#) as arguments:

For Services: **`$SERVICESTATE$`**, **`$SERVICESTATETYPE$`**, **`$SERVICEATTEMPT$`**

For Hosts: **`$HOSTSTATE$`**, **`$HOSTSTATETYPE$`**, **`$HOSTATTEMPT$`**

The scripts should examine the values of the arguments passed to it and take any necessary action based upon those values. The best way to understand how event handlers work is to see an example. Lucky for you, one is provided [below](#).



Tip

Additional sample event handler scripts can be found in the *contrib/eventhandlers/* subdirectory of the Icinga distribution. Some of these sample scripts demonstrate the use of [external commands](#) to implement a [redundant](#) and [distributed](#) monitoring environments.

Permissions For Event Handler Commands

Event handler commands will normally execute with the same permissions as the user under which Icinga is running on your machine. This can present a problem if you want to write an event handler that restarts system services, as root privileges are generally required to do these sorts of tasks.

Ideally you should evaluate the types of event handlers you will be implementing and grant just enough permissions to the Icinga user for executing the necessary system commands. You might want to try using [sudo](#) to accomplish this.

Service Event Handler Example

The example below assumes that you are monitoring the HTTP server on the local machine and have specified *restart-httdp* as the event handler command for the HTTP service definition. Also, I will be assuming that you have set the *max_check_attempts* option for the service to be a value of 4 or greater (i.e. the service is checked 4 times before it is considered to have a real problem). An abbreviated example service definition might look like this...

```
define service{
    host_name           somehost
    service_description HTTP
    max_check_attempts 4
    event_handler       restart-httdp
    ...
}
```

Once the service has been defined with an event handler, we must define that event handler as a command. An example command definition for *restart-httdp* is shown below. Notice the macros in the command line that I am passing to the event handler script - these are important!

```
define command{
    command_name      restart-httdp
    command_line     /usr/local/icinga/libexec/eventhandlers/restart-httdp $SERVICESTATE$ $SERVICESTATETYPE$ $SERVICEATTEMPT$
```

Now, let's actually write the event handler script (this is the */usr/local/icinga/libexec/eventhandlers/restart-httdp* script).

```
#!/bin/sh
#
# Event handler script for restarting the web server on the local machine
#
# Note: This script will only restart the web server if the service is
#        retried 3 times (in a "soft" state) or if the web service somehow
#        manages to fall into a "hard" error state.
#
```

```

# What state is the HTTP service in?
case "$1" in
OK)
    # The service just came back up, so don't do anything...
    ;;
WARNING)
    # We don't really care about warning states, since the service is probably still running...
    ;;
UNKNOWN)
    # We don't know what might be causing an unknown error, so don't do anything...
    ;;
CRITICAL)
    # Aha! The HTTP service appears to have a problem - perhaps we should restart the server...

    # Is this a "soft" or a "hard" state?
    case "$2" in
        # We're in a "soft" state, meaning that Icinga is in the middle of retrying the
        # check before it turns into a "hard" state and contacts get notified...
        SOFT)

            # What check attempt are we on? We don't want to restart the web server on the first
            # check, because it may just be a fluke!
            case "$3" in
                # Wait until the check has been tried 3 times before restarting the web server.
                # If the check fails on the 4th time (after we restart the web server), the state
                # type will turn to "hard" and contacts will be notified of the problem.
                # Hopefully this will restart the web server successfully, so the 4th check will
                # result in a "soft" recovery. If that happens no one gets notified because we
                # fixed the problem!
                3)
                    echo -n "Restarting HTTP service (3rd soft critical state)..."
                    # Call the init script to restart the HTTPD server
                    /etc/rc.d/init.d/httpd restart
                    ;;
                    esac
                ;;

            # The HTTP service somehow managed to turn into a hard error without getting fixed.
            # It should have been restarted by the code above, but for some reason it didn't.
            # Let's give it one last try, shall we?
            # Note: Contacts have already been notified of a problem with the service at this
            # point (unless you disabled notifications for this service)
            HARD)
                echo -n "Restarting HTTP service..."
                # Call the init script to restart the HTTPD server
                /etc/rc.d/init.d/httpd restart
                ;;
                esac
            ;;

        esac
    ;;
esac
exit 0

```

The sample script provided above will attempt to restart the web server on the local machine in two different instances:

- After the service has been rechecked for the 3rd time and is in a SOFT CRITICAL state
- After the service first goes into a HARD CRITICAL state

The script should theoretically restart and web server and fix the problem before the service goes into a HARD problem state, but we include a fallback case in the event it doesn't work the first time. It should be noted that the event handler will only be executed the first time that the service falls into a HARD problem state. This prevents Icinga from continuously executing the script to restart the web server if the service remains in a HARD problem state. You don't want that. :-)

That's all there is to it! Event handlers are pretty simple to write and implement, so give it a try and see what you can do.

[Prev](#)[Up](#)[Next](#)[External Commands](#)[Home](#)[Volatile Services](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Volatile Services**[Prev](#)**Chapter 7. Advanced Topics**[Next](#)

Volatile Services

Introduction

Icinga has the ability to distinguish between "normal" services and "volatile" services. The `is_volatile` option in each service definition allows you to specify whether a specific service is volatile or not. For most people, the majority of all monitored services will be non-volatile (i.e. "normal"). However, volatile services can be very useful when used properly...

What Are They Useful For?

Volatile services are useful for monitoring...

- Things that automatically reset themselves to an "OK" state each time they are checked
- Events such as security alerts which require attention every time there is a problem (and not just the first time)

What's So Special About Volatile Services?

Volatile services differ from "normal" services in three important ways. *Each time* they are checked when they are in a `hard` non-OK state, and the check returns a non-OK state (i.e. no state change has occurred)...

- The non-OK service state is logged
- Contacts are notified about the problem (if that's [what should be done](#)).

**Note**

Notification intervals are ignored for volatile services.

- The [event handler](#) for the service is run (if one has been defined)

These events normally only occur for services when they are in a non-OK state and a hard state change has just occurred. In other words, they only happen the first time that a service goes into a non-OK state. If future checks of the service result in the same non-OK state, no hard state change occurs and none of the events mentioned take place again.

**Tip**

If you are only interested in logging, consider using [stalking](#) options instead.

The Power Of Two

If you combine the features of volatile services and [passive service checks](#), you can do some very useful things. Examples of this include handling SNMP traps, security alerts, etc.

How about an example... Let's say you're running [PortSentry](#) to detect port scans on your machine and automatically firewall potential intruders. If you want to let Icinga know about port scans, you could do the following...

Icinga Configuration:

- Create a service definition called *Port Scans* and associate it with the host that PortSentry is running on.
- Set the *max_check_attempts* directive in the service definition to 1. This will tell Icinga to immediately force the service into a [hard state](#) when a non-OK state is reported.
- Set the *active_checks_enabled* directive in the service definition to 0. This prevents Icinga from actively checking the service.
- Set the *passive_checks_enabled* directive in the service definition to 1. This enables passive checks for the service.
- Set this *is_volatile* directive in the service definition to 1.

PortSentry Configuration:

Edit your PortSentry configuration file (*portsentry.conf*) and define a command for the *KILL_RUN_CMD* directive as follows:

```
KILL_RUN_CMD="/usr/local/Icinga/libexec/eventhandlers/submit_check_result host_name 'Port Scans' 2 'Port scan from host $TARGET$ on port $PORT$. Host has been firewalled.'"
```

Make sure to replace *host_name* with the short name of the host that the service is associated with.

Port Scan Script:

Create a shell script in the */usr/local/icinga/libexec/eventhandlers* directory named *submit_check_result*. The contents of the shell script should be something similar to the following...

```
#!/bin/sh

# Write a command to the Icinga command file to cause
# it to process a service check result

echocmd="/bin/echo"

CommandFile="/usr/local/icinga/var/rw/nagios.cmd"

# get the current date/time in seconds since UNIX epoch
datetime='date +%s'

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
'$echocmd $cmdline >> $CommandFile'
```

What will happen when PortSentry detects a port scan on the machine in the future?

- PortSentry will firewall the host (this is a function of the PortSentry software)
- PortSentry will execute the *submit_check_result* shell script and send a passive check result to Icinga
- Icinga will read the external command file and see the passive service check submitted by PortSentry
- Icinga will put the *Port Scans* service in a hard CRITICAL state and send notifications to contacts

Pretty neat, huh?

[Prev](#)

[Up](#)

[Next](#)

[Event Handlers](#)

[Home](#)

[Service and Host Freshness Checks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Service and Host Freshness Checks**[Prev](#)[Chapter 7. Advanced Topics](#)[Next](#)

Service and Host Freshness Checks

Introduction

Icinga supports a feature that does "freshness" checking on the results of host and service checks. The purpose of freshness checking is to ensure that host and service checks are being provided passively by external applications on a regular basis.

Freshness checking is useful when you want to ensure that [passive checks](#) are being received as frequently as you want. This can be very useful in [distributed](#) and [failover](#) monitoring environments.



How Does Freshness Checking Work?

Icinga periodically checks the freshness of the results for all hosts services that have freshness checking enabled.

- A freshness threshold is calculated for each host or service.
- For each host/service, the age of its last check result is compared with the freshness threshold.
- If the age of the last check result is greater than the freshness threshold, the check result is considered "stale".
- If the check results is found to be stale, Icinga will force an [active check](#) of the host or service by executing the command specified by in the host or service definition.

**Tip**

An active check is executed even if active checks are disabled on a program-wide or host- or service-specific basis.

For example, if you have a freshness threshold of 60 for one of your services, Icinga will consider that service to be stale if its last check result is older than 60 seconds.

Enabling Freshness Checking

Here's what you need to do to enable freshness checking...

- Enable freshness checking on a program-wide basis with the `check_service_freshness` and `check_host_freshness` directives.
- Use `service_freshness_check_interval` and `host_freshness_check_interval` options to tell Icinga how often it should check the freshness of service and host results.
- Enable freshness checking on a host- and service-specific basis by setting the `check_freshness` option in your host and service definitions to a value of 1.
- Configure freshness thresholds by setting the `freshness_threshold` option in your host and service definitions.
- Configure the `check_command` option in your host or service definitions to reflect a valid command that should be used to actively check the host or service when it is detected as stale.
- The `check_period` option in your host and service definitions is used when Icinga determines when a host or service can be checked for freshness, so make sure it is set to a valid timeperiod.

**Tip**

If you do not specify a host- or service-specific `freshness_threshold` value (or you set it to zero), Icinga will automatically calculate a threshold automatically, based on a how often you monitor that particular host or service. We would recommend that you explicitly specify a freshness threshold, rather than let Icinga pick one for you.

Example

An example of a service that might require freshness checking might be one that reports the status of your nightly backup jobs. Perhaps you have an external script that submit the results of the backup job to Icinga once the backup is completed. In this case, all of the checks/results for the service are provided by an external application using passive checks. In order to ensure that the status of the backup job gets reported every day, you may want to enable freshness checking for the service. If the external script doesn't submit the results of the backup job, you can have Icinga fake a critical result by doing something like this...

Here's what the definition for the service might look like (some required options are omitted)...

```
define service{
    host_name          backup-server
    service_description ArcServe Backup Job
    active_checks_enabled 0           ; active checks are NOT enabled
    passive_checks_enabled 1          ; passive checks are enabled (this is how results are reported)
    check_freshness     1
    freshness_threshold 93600        ; 26 hour threshold, since backups may not always finish at the same time
    check_command       no-backup-report ; this command is run only if the service results are "stale"
    ...other options...
}
```

Notice that active checks are disabled for the service. This is because the results for the service are only made by an external application using passive checks. Freshness checking is enabled and the freshness threshold has been set to 26 hours. This is a bit longer than 24 hours because backup jobs sometimes run late from day to day (depending on how much data there is to backup, how much network traffic is present, etc.). The *no-backup-report* command is executed only if the results of the service are determined to be stale. The definition of the *no-backup-report* command might look like this...

```
define command{
    command_name      no-backup-report
    command_line      /usr/local/icinga/libexec/check_dummy 2 "CRITICAL: Results of backup job were not reported!"
```

If Icinga detects that the service results are stale, it will run the *no-backup-report* command as an active service check. This causes the *check_dummy* plugin to be executed, which returns a critical state to Icinga. The service will then go into to a critical state (if it isn't already there) and someone will probably get notified of the problem.

[Prev](#)
[Up](#)
[Next](#)
[Volatile Services](#)
[Home](#)
[Distributed Monitoring](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Distributed Monitoring

[Prev](#)[Chapter 7. Advanced Topics](#)[Next](#)

Distributed Monitoring

Introduction

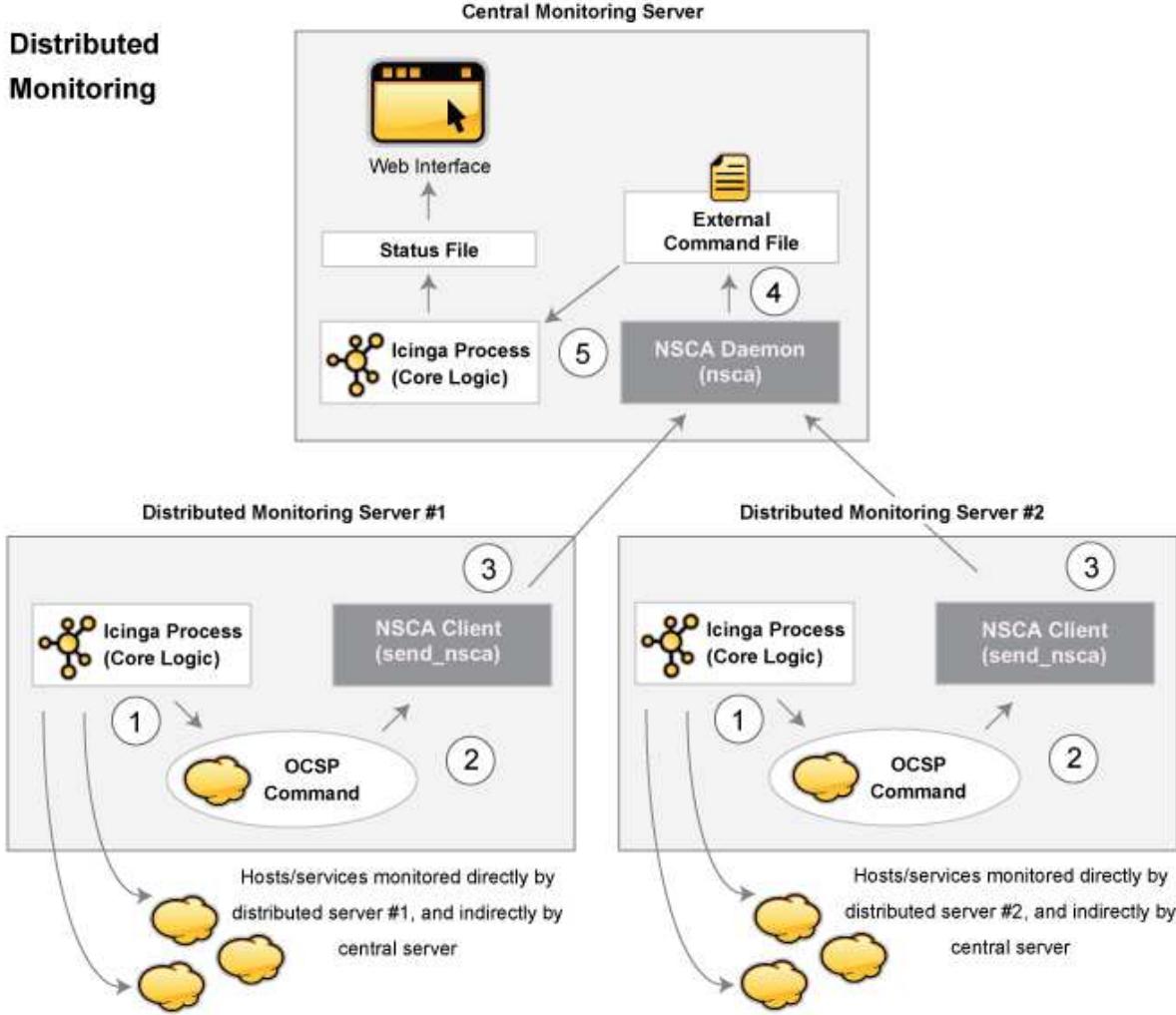
Icinga can be configured to support distributed monitoring of network services and resources. We'll try to briefly explain how this can be accomplished...

Goals

The goal in the distributed monitoring environment that we will describe is to offload the overhead (CPU usage, etc.) of performing service checks from a "central" server onto one or more "distributed" servers. Most small to medium sized shops will not have a real need for setting up such an environment. However, when you want to start monitoring hundreds or even thousands of *hosts* (and several times that many services) using Icinga, this becomes quite important.

Reference Diagram

The diagram below should help give you a general idea of how distributed monitoring works with Icinga. We'll be referring to the items shown in the diagram as we explain things...



Central Server vs. Distributed Servers

When setting up a distributed monitoring environment with Icinga, there are differences in the way the central and distributed servers are configured. We'll show you how to configure both types of servers and explain what effects the changes being made have on the overall monitoring. For starters, let's describe the purpose of the different types of servers...

The function of a *distributed server* is to actively perform checks all the services you define for a "cluster" of hosts. We use the term "cluster" loosely - it basically just means an arbitrary group of hosts on your network. Depending on your network layout, you may have several clusters at one physical location, or each cluster may be separated by a WAN, its own firewall, etc. The important thing to remember is that for each cluster of hosts (however you define that), there is one distributed server that runs Icinga and monitors the services on the hosts in the cluster. A distributed server is usually a bare-bones installation of Icinga. It doesn't have to have the web interface installed, send out notifications, run event handler scripts, or do anything other than execute service checks if you don't want it to. More detailed information on configuring a distributed server comes later...

The purpose of the *central server* is to simply listen for service check results from one or more distributed servers. Even though services are occasionally actively checked from the central server, the active checks are only performed in dire circumstances, so let's just say that the central server only accepts passive checks for now. Since the central server is obtaining **passive service check** results from one or more distributed servers, it serves as the focal point for all

monitoring logic (i.e. it sends out notifications, runs event handler scripts, determines host states, has the web interface installed, etc).

Obtaining Service Check Information From Distributed Monitors

Okay, before we go jumping into configuration detail we need to know how to send the service check results from the distributed servers to the central server. We've already discussed how to submit passive check results to Icinga from same host that Icinga is running on (as described in the documentation on [passive checks](#)), but we haven't given any info on how to submit passive check results from other hosts.

In order to facilitate the submission of passive check results to a remote host, the [nsca addon](#) was written. The addon consists of two pieces. The first is a client program (send_nsca) which is run from a remote host and is used to send the service check results to another server. The second piece is the nsca daemon (nsca) which either runs as a standalone daemon or under inetd and listens for connections from client programs. Upon receiving service check information from a client, the daemon will sumbit the check information to Icinga (on the central server) by inserting a *PROCESS_SVC_CHECK_RESULT* command into the [external command file](#), along with the check results. The next time Icinga checks for [external commands](#), it will find the passive service check information that was sent from the distributed server and process it. Easy, huh?

Distributed Server Configuration

So how exactly is Icinga configured on a distributed server? Basically, its just a bare-bones installation. You don't need to install the web interface or have notifications sent out from the server, as this will all be handled by the central server.

Key configuration changes:

- Only those services and hosts which are being monitored directly by the distributed server are defined in the [object configuration file](#).
- The distributed server has its [enable_notifications](#) directive set to 0. This will prevent any notifications from being sent out by the server.
- The distributed server is configured to [obsess over services](#).
- The distributed server has an [ocsp command](#) defined (as described below).

In order to make everything come together and work properly, we want the distributed server to report the results of *all* service checks to Icinga. We could use [event handlers](#) to report *changes* in the state of a service, but that just doesn't cut it. In order to force the distributed server to report all service check results, you must enabled the [obsess_over_services](#) option in the main configuration file and provide a [ocsp_command](#) to be run after every service check. We will use the ocsp command to send the results of all service checks to the central server, making use of the send_nsca client and nsca daemon (as described above) to handle the transmission.

In order to accomplish this, you'll need to define an ocsp command like this:

ocsp_command=submit_check_result

The command definition for the *submit_check_result* command looks something like this:

```
define command{
    command_name    submit_check_result
    command_line    /usr/local/icinga/libexec/eventhandlers/submit_check_result $HOSTNAME$ '$SERVICEDESC$' '$SERVICESTATE$' '$SERVICEOUTPUT$'
}
```

The *submit_check_result* shell scripts looks something like this (replace *central_server* with the IP address of the central server):

```
#!/bin/sh

# Arguments:
# $1 = host_name (Short name of host that the service is
#           associated with)
# $2 = svc_description (Description of the service)
# $3 = state_string (A string representing the status of
#           the given service - "OK", "WARNING", "CRITICAL"
#           or "UNKNOWN")
# $4 = plugin_output (A text string that should be used
#           as the plugin output for the service checks)
#
# Convert the state string to the corresponding return code
return_code=-1

case "$3" in
    OK)
        return_code=0
        ;;
    WARNING)
        return_code=1
        ;;
    CRITICAL)
        return_code=2
        ;;
    UNKNOWN)
        return_code=-1
        ;;
esac

# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server
/bin/printf "%s\t%s\t%s\n" "$1" "$2" "$return_code" "$4" | /usr/local/icinga/bin/send_nsca -H central_server -c /usr/local/icinga/etc/send_nsca.cfg
```

The script above assumes that you have the *send_nsca* program and it configuration file (*send_nsca.cfg*) located in the */usr/local/icinga/bin/* and */usr/local/icinga/etc/* directories, respectively.

That's it! We've sucessfully configured a remote host running Icinga to act as a distributed monitoring server. Let's go over exactly what happens with the distributed server and how it sends service check results to Icinga (the steps outlined below correspond to the numbers in the reference diagram above):

1. After the distributed server finishes executing a service check, it executes the command you defined by the *ocsp_command* variable. In our example, this is the */usr/local/icinga/libexec/eventhandlers/submit_check_result* script. Note that the definition for the *submit_check_result* command passed four pieces of information to the script: the name of the host the service is associated with, the service description, the return code from the service check, and the plugin output from the service check.
2. The *submit_check_result* script pipes the service check information (host name, description, return code, and output) to the *send_nsca* client program.
3. The *send_nsca* program transmits the service check information to the *nsca* daemon on the central monitoring server.
4. The *nsca* daemon on the central server takes the service check information and writes it to the external command file for later pickup by Icinga.
5. The Icinga process on the central server reads the external command file and processes the passive service check information that originated from the distributed monitoring server.

Central Server Configuration

We've looked at how distributed monitoring servers should be configured, so let's turn to the central server. For all intensive purposes, the central is configured as you would normally configure a standalone server. It is setup as follows:

- The central server has the web interface installed (optional, but recommended)
- The central server has its `enable_notifications` directive set to 1. This will enable notifications. (optional, but recommended)
- The central server has `active service checks` disabled (optional, but recommended - see notes below)
- The central server has `external command checks` enabled (required)
- The central server has `passive service checks` enabled (required)

There are three other very important things that you need to keep in mind when configuring the central server:

- The central server must have service definitions for *all services* that are being monitored by all the distributed servers. Icinga will ignore passive check results if they do not correspond to a service that has been defined.
- If you're only using the central server to process services whose results are going to be provided by distributed hosts, you can simply disable all active service checks on a program-wide basis by setting the `execute_service_checks` directive to 0. If you're using the central server to actively monitor a few services on its own (without the aid of distributed servers), the `active_checks_enabled` option of the definitions for service being monitored by distributed servers should be set to 0. This will prevent Icinga from actively checking those services.

It is important that you either disable all service checks on a program-wide basis or disable the `enable_active_checks` option in the definitions for each service that is monitored by a distributed server. This will ensure that active service checks are never executed under normal circumstances. The services will keep getting rescheduled at their normal check intervals (3 minutes, 5 minutes, etc...), but they won't actually be executed. This rescheduling loop will just continue all the while Icinga is running. We'll explain why this is done in a bit...

That's it! Easy, huh?

Problems With Passive Checks

For all intensive purposes we can say that the central server is relying solely on passive checks for monitoring. The main problem with relying completely on passive checks for monitoring is the fact that Icinga must rely on something else to provide the monitoring data. What if the remote host that is sending in passive check results goes down or becomes unreachable? If Icinga isn't actively checking the services on the host, how will it know that there is a problem?

Fortunately, there is a way we can handle these types of problems...

Freshness Checking

Icinga supports a feature that does "freshness" checking on the results of service checks. More information on freshness checking can be found [here](#). This feature gives some protection against situations where remote hosts may stop sending passive service checks into the central monitoring server. The purpose of "freshness" checking is to ensure that service checks are either being provided passively by distributed servers on a regular basis or performed actively by the central server if the need arises. If the service check results provided by the distributed servers get "stale", Icinga can be configured to force active checks of the service from the central monitoring host.

So how do you do this? On the central monitoring server you need to configure services that are being monitoring by distributed servers as follows...

- The *check_freshness* option in the service definitions should be set to 1. This enables "freshness" checking for the services.
- The *freshness_threshold* option in the service definitions should be set to a value (in seconds) which reflects how "fresh" the results for the services (provided by the distributed servers) should be.
- The *check_command* option in the service definitions should reflect valid commands that can be used to actively check the service from the central monitoring server.

Icinga periodically checks the "freshness" of the results for all services that have freshness checking enabled. The *freshness_threshold* option in each service definition is used to determine how "fresh" the results for each service should be. For example, if you set this value to 300 for one of your services, Icinga will consider the service results to be "stale" if they're older than 5 minutes (300 seconds). If you do not specify a value for the *freshness_threshold* option, Icinga will automatically calculate a "freshness" threshold by looking at either the *check_interval* or *retry_interval* options (depending on what [type of state](#) the service is in). If the service results are found to be "stale", Icinga will run the service check command specified by the *check_command* option in the service definition, thereby actively checking the service.

Remember that you have to specify a *check_command* option in the service definitions that can be used to actively check the status of the service from the central monitoring server. Under normal circumstances, this check command is never executed (because active checks were disabled on a program-wide basis or for the specific services). When freshness checking is enabled, Icinga will run this command to actively check the status of the service *even if active checks are disabled on a program-wide or service-specific basis*.

If you are unable to define commands to actively check a service from the central monitoring host (or it turns out to be a major pain), you could simply define all your services with the *check_command* option set to run a dummy script that returns a critical status. Here's an example... Let's assume you define a command called 'service-is-stale' and use that command name in the *check_command* option of your services. Here's what the definition would look like...

```
define command{
    command_name service-is-stale
    command_line /usr/local/icinga/libexec/check_dummy 2 "CRITICAL: Service results are stale"
}
```

When Icinga detects that the service results are stale and runs the **service-is-stale** command, the *check_dummy* plugin is executed and the service will go into a critical state. This would likely cause notifications to be sent out, so you'll know that there's a problem.

Performing Host Checks

At this point you know how to obtain service check results passivly from distributed servers. This means that the central server is not actively checking services on its own. But what about host checks? You still need to do them, so how?

Since host checks usually compromise a small part of monitoring activity (they aren't done unless absolutely necessary), we'd recommend that you perform host checks actively from the central server. That means that you define host checks on the central server the same way that you do on the distributed servers (and the same way you would in a normal, non-distributed setup).

Passive host checks are available (read [here](#)), so you could use them in your distributed monitoring setup, but they suffer from a few problems. The biggest problem is that Icinga does not translate passive host check problem states (DOWN and UNREACHABLE) when they are processed. This means that if your monitoring servers have a different parent/child host structure (and they will, if your monitoring servers are in different locations), the central monitoring server will have an inaccurate view of host states.

If you do want to send passive host checks to a central server in your distributed monitoring setup, make sure:

- The central server has [passive host checks](#) enabled (required)
- The distributed server is configured to [obsess over hosts](#).
- The distributed server has an [ochp command](#) defined.

The ochp command, which is used for processing host check results, works in a similar manner to the ocsp command, which is used for processing service check results (see documentation above). In order to make sure passive host check results are up to date, you'll want to enable [freshness checking](#) for hosts (similar to what is described above for services).

[Prev](#)

[Up](#)

[Next](#)

Service and Host Freshness
Checks

[Home](#)

Redundant and Failover Network
Monitoring

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Redundant and Failover Network Monitoring**[Prev](#)**Chapter 7. Advanced Topics**[Next](#)

Redundant and Failover Network Monitoring

Introduction

This section describes a few scenarios for implementing redundant monitoring hosts in various types of network layouts. With redundant hosts, you can maintain the ability to monitor your network when the primary host that runs Icinga fails or when portions of your network become unreachable.



Note

If you are just learning how to use Icinga, we would suggest not trying to implement redundancy until you have become familiar with the [prerequisites](#) that have been laid out. Redundancy is a relatively complicated issue to understand, and even more difficult to implement properly.

Index

[Prerequisites](#)[Sample scripts](#)[Scenario 1 - Redundant monitoring](#)[Scenario 2 - Failover monitoring](#)

Prerequisites

Before you can even think about implementing redundancy with Icinga, you need to be familiar with the following...

- Implementing [event handlers](#) for hosts and services
- Issuing [external commands](#) to Icinga via shell scripts
- Executing plugins on remote hosts using either the [NRPE addon](#) or some other method
- Checking the status of the Icinga process with the *check_nagios* plugin

Sample Scripts

All of the sample scripts that we use in this documentation can be found in the *eventhandlers/* subdirectory of the Icinga distribution. You'll probably need to modify them to work on your system...

Scenario 1 - Redundant Monitoring

Introduction

This is an easy (and naive) method of implementing redundant monitoring hosts on your network and it will only protect against a limited number of failures. More complex setups are necessary in order to provide smarter redundancy, better redundancy across different network segments, etc.

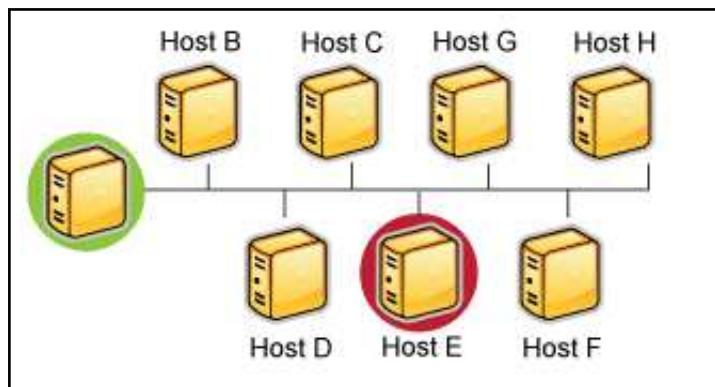
Goals

The goal of this type of redundancy implementation is simple. Both the "master" and "slave" hosts monitor the same hosts and service on the network. Under normal circumstances only the "master" host will be sending out notifications to contacts about problems. We want the "slave" host running Icinga to take over the job of notifying contacts about problems if:

1. The "master" host that runs Icinga is down or..
2. The Icinga process on the "master" host stops running for some reason

Network Layout Diagram

The diagram below shows a very simple network setup. For this scenario we will be assuming that hosts A and E are both running Icinga and are monitoring all the hosts shown. Host A will be considered the "master" host and host E will be considered the "slave" host.



Initial Program Settings

The slave host (host E) has its initial [enable_notifications](#) directive disabled, thereby preventing it from sending out any host or service notifications. You also want to make sure that the slave host has its [check_external_commands](#) directive enabled. That was easy enough...

Initial Configuration

Next we need to consider the differences between the [object configuration file\(s\)](#) on the master and slave hosts...

We will assume that you have the master host (host A) setup to monitor services on all hosts shown in the diagram above. The slave host (host E) should be setup to monitor the same services and hosts, with the following additions in the configuration file...

- The host definition for host A (in the host E configuration file) should have a host [event handler](#) defined. Lets say the name of the host event handler is [handle-master-host-event](#).
- The configuration file on host E should have a service defined to check the status of the Icinga process on host A. Lets assume that you define this service check to run the `check_nagios` plugin on host A. This can be done by using one of the methods described in [this FAQ](#) (update this!).
- The service definition for the Icinga process check on host A should have an [event handler](#) defined. Lets say the name of the service event handler is [handle-master-proc-event](#).

It is important to note that host A (the master host) has no knowledge of host E (the slave host). In this scenario it simply doesn't need to. Of course you may be monitoring services on host E from host A, but that has nothing to do with the implementation of redundancy...

Event Handler Command Definitions

We need to stop for a minute and describe what the command definitions for the event handlers on the slave host look like. Here is an example...

```
define command{
    command_name handle-master-host-event
    command_line /usr/local/icinga/libexec/eventhandlers/handle-master-host-event $HOSTSTATE$ $HOSTSTATETYPE$
}

define command{
    command_name handle-master-proc-event
    command_line /usr/local/icinga/libexec/eventhandlers/handle-master-proc-event $SERVICESTATE$ $SERVICESTATETYPE$
```

This assumes that you have placed the event handler scripts in the `/usr/local/icinga/libexec/eventhandlers` directory. You may place them anywhere you wish, but you'll need to modify the examples that are given here.

Event Handler Scripts

Okay, now lets take a look at what the event handler scripts look like...

Host Event Handler ([handle-master-host-event](#)):

```
#!/bin/sh

# Only take action on hard host states...
case "$2" in
HARD)
    case "$1" in
DOWN)
        # The master host has gone down!
        # We should now become the master host and take
        # over the responsibilities of monitoring the
        # network, so enable notifications...
        /usr/local/icinga/libexec/eventhandlers/enable_notifications
        ;;
UP)
        # The master host has recovered!
        # We should go back to being the slave host and
        # let the master host do the monitoring, so
        # disable notifications...
        /usr/local/icinga/libexec/eventhandlers/disable_notifications
        ;;
esac
esac
```

```

    esac
    ;;
esac
exit 0

```

Service Event Handler ([handle-master-proc-event](#)):

```

#!/bin/sh

# Only take action on hard service states...
case "$2" in
HARD)
    case "$1" in
    CRITICAL)
        # The master Icinga process is not running!
        # We should now become the master host and
        # take over the responsibility of monitoring
        # the network, so enable notifications...
        /usr/local/icinga/libexec/eventhandlers/enable_notifications
        ;;
    WARNING)
    UNKNOWN)
        # The master Icinga process may or may not
        # be running.. We won't do anything here, but
        # to be on the safe side you may decide you
        # want the slave host to become the master in
        # these situations...
        ;;
    OK)
        # The master Icinga process running again!
        # We should go back to being the slave host,
        # so disable notifications...
        /usr/local/icinga/libexec/eventhandlers/disable_notifications
        ;;
    esac
    ;;
esac
exit 0

```

What This Does For Us

The slave host (host E) initially has notifications disabled, so it won't send out any host or service notifications while the Icinga process on the master host (host A) is still running.

The Icinga process on the slave host (host E) becomes the master host when...

- The master host (host A) goes down and the [handle-master-host-event](#) host event handler is executed.
- The Icinga process on the master host (host A) stops running and the [handle-master-proc-event](#) service event handler is executed.

When the Icinga process on the slave host (host E) has notifications enabled, it will be able to send out notifications about any service or host problems or recoveries. At this point host E has effectively taken over the responsibility of notifying contacts of host and service problems!

The Icinga process on host E returns to being the slave host when...

- Host A recovers and the [handle-master-host-event](#) host event handler is executed.
- The Icinga process on host A recovers and the [handle-master-proc-event](#) service event handler is executed.

When the Icinga process on host E has notifications disabled, it will not send out notifications about any service or host problems or recoveries. At this point host E has handed over the responsibilities of notifying contacts of problems to the Icinga process on host A. Everything is now as it was when we first started!

Time Lags

Redundancy in Icinga is by no means perfect. One of the more obvious problems is the lag time between the master host failing and the slave host taking over. This is affected by the following...

- The time between a failure of the master host and the first time the slave host detects a problem
- The time needed to verify that the master host really does have a problem (using service or host check retries on the slave host)
- The time between the execution of the event handler and the next time that Icinga checks for external commands

You can minimize this lag by...

- Ensuring that the Icinga process on host E (re)checks one or more services at a high frequency. This is done by using the *check_interval* and *retry_interval* arguments in each service definition.
- Ensuring that the number of host rechecks for host A (on host E) allow for fast detection of host problems. This is done by using the *max_check_attempts* argument in the host definition.
- Increase the frequency of [external command](#) checks on host E. This is done by modifying the [command_check_interval](#) option in the main configuration file.

When Icinga recovers on the host A, there is also some lag time before host E returns to being a slave host. This is affected by the following...

- The time between a recovery of host A and the time the Icinga process on host E detects the recovery
- The time between the execution of the event handler on host B and the next time the Icinga process on host E checks for external commands

The exact lag times between the transfer of monitoring responsibilities will vary depending on how many services you have defined, the interval at which services are checked, and a lot of pure chance. At any rate, its definitely better than nothing.

Special Cases

Here is one thing you should be aware of... If host A goes down, host E will have notifications enabled and take over the responsibilities of notifying contacts of problems. When host A recovers, host E will have notifications disabled. If - when host A recovers - the Icinga process on host A does not start up properly, there will be a period of time when neither host is notifying contacts of problems! Fortunately, the service check logic in Icinga accounts for this. The next time the Icinga process on host E checks the status of the Icinga process on host A, it will find that it is not running. Host E will then have notifications enabled again and take over all responsibilities of notifying contacts of problems.

The exact amount of time that neither host is monitoring the network is hard to determine. Obviously, this period can be minimized by increasing the frequency of service checks (on host E) of the Icinga process on host A. The rest is up to pure chance, but the total "blackout" time shouldn't be too bad.

Scenario 2 - Failover Monitoring

Introduction

Failover monitoring is similar to, but slightly different than redundant monitoring (as discussed above in [scenario 1](#)).

Goals

The basic goal of failover monitoring is to have the Icinga process on the slave host sit idle while the Icinga process on the master host is running. If the process on the master host stops running (or if the host goes down), the Icinga process on the slave host starts monitoring everything.

While the method described in [scenario 1](#) will allow you to continue receive notifications if the master monitoring hosts goes down, it does have some pitfalls. The biggest problem is that the slave host is monitoring the same hosts and servers as the master *at the same time as the master!* This can cause problems with excessive traffic and load on the machines being monitored if you have a lot of services defined. Here's how you can get around that problem...

Initial Program Settings

Disable active service checks and notifications on the slave host using the [execute_service_checks](#) and [enable_notifications](#) directives. This will prevent the slave host from monitoring hosts and services and sending out notifications while the Icinga process on the master host is still up and running. Make sure you also have the [check_external_commands](#) directive enabled on the slave host.

Master Process Check

Set up a cron job on the slave host that periodically (say every minute) runs a script that checks the status of the Icinga process on the master host (using the [check_nrpe](#) plugin on the slave host and the [nrpe daemon](#) and [check_nagios](#) plugin on the master host). The script should check the return code of the [check_nrpe plugin](#). If it returns a non-OK state, the script should send the appropriate commands to the [external command file](#) to enable both notifications and active service checks. If the plugin returns an OK state, the script should send commands to the external command file to disable both notifications and active checks.

By doing this you end up with only one process monitoring hosts and services at a time, which is much more efficient than monitoring everything twice.

Also of note, you *don't* need to define host and service handlers as mentioned in [scenario 1](#) because things are handled differently.

Additional Issues

At this point, you have implemented a very basic failover monitoring setup. However, there is one more thing you should consider doing to make things work smoother.

The big problem with the way things have been setup thus far is the fact that the slave host doesn't have the current status of any services or hosts at the time it takes over the job of monitoring. One way to solve this problem is to enable the [ocsp command](#) on the master host

and have it send all service check results to the slave host using the [nsca addon](#). The slave host will then have up-to-date status information for all services at the time it takes over the job of monitoring things. Since active service checks are not enabled on the slave host, it will not actively run any service checks. However, it will execute host checks if necessary. This means that both the master and slave hosts will be executing host checks as needed, which is not really a big deal since the majority of monitoring deals with service checks.

That's pretty much it as far as setup goes.

[Prev](#)[Up](#)[Next](#)[Distributed Monitoring](#)[Home](#)[Detection and Handling of State Flapping](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Detection and Handling of State Flapping

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Detection and Handling of State Flapping

Introduction

Icinga supports optional detection of hosts and services that are "flapping". Flapping occurs when a service or host changes state too frequently, resulting in a storm of problem and recovery notifications. Flapping can be indicative of configuration problems (i.e. thresholds set too low), troublesome services, or real network problems.

How Flap Detection Works

Before we get into this, it is time to say that flapping detection has been a little difficult to implement. How exactly does one determine what "too frequently" means in regards to state changes for a particular host or service? When Ethan Galstad first started thinking about implementing flap detection he tried to find some information on how flapping could/should be detected. He couldn't find any information about what others were using (where they using any?), so he decided to settle with what seemed to him to be a reasonable solution...

Whenever Icinga checks the status of a host or service, it will check to see if it has started or stopped flapping. It does this by:

- Storing the results of the last 21 checks of the host or service
- Analyzing the historical check results and determine where state changes/transitions occur
- Using the state transitions to determine a percent state change value (a measure of change) for the host or service
- Comparing the percent state change value against low and high flapping thresholds

A host or service is determined to have *started* flapping when its percent state change first exceeds a *high* flapping threshold.

A host or service is determined to have *stopped* flapping when its percent state goes below a *low* flapping threshold (assuming that it was previously flapping).

Example

Let's describe in more detail how flap detection works with services...

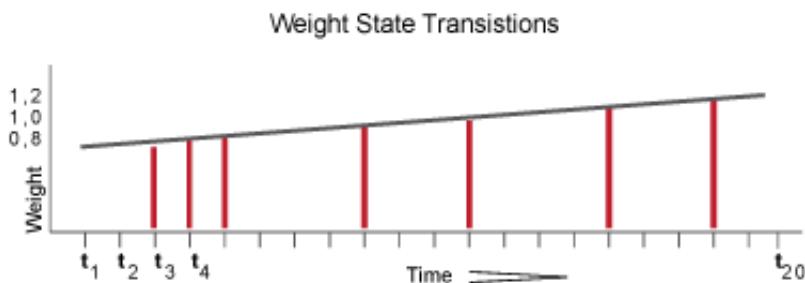
The image below shows a chronological history of service states from the most recent 21 service checks. OK states are shown in green, WARNING states in yellow, CRITICAL states in red, and UNKNOWN states in orange.



The historical service check results are examined to determine where state changes/transitions occur. State changes occur when an archived state is different from the archived state that immediately precedes it chronologically. Since we keep the results of the last 21 service checks in the array, there is a possibility of having at most 20 state changes. In this example there are 7 state changes, indicated by blue arrows in the image above.

The flap detection logic uses the state changes to determine an overall percent state change for the service. This is a measure of volatility/change for the service. Services that never change state will have a 0% state change value, while services that change state each time they're checked will have 100% state change. Most services will have a percent state change somewhere in between.

When calculating the percent state change for the service, the flap detection algorithm will give more weight to new state changes compare to older ones. Specifically, the flap detection routines are currently designed to make the newest possible state change carry 50% more weight than the oldest possible state change. The image below shows how recent state changes are given more weight than older state changes when calculating the overall or total percent state change for a particular service.



Using the images above, lets do a calculation of percent state change for the service. You will notice that there are a total of 7 state changes (at t_3 , t_4 , t_5 , t_9 , t_{12} , t_{16} , and t_{19}). Without any weighting of the state changes over time, this would give us a total state change of 35%:

$$(7 \text{ observed state changes} / \text{possible 20 state changes}) * 100 = 35 \%$$

Since the flap detection logic will give newer state changes a higher rate than older state changes, the actual calculated percent state change will be slightly less than 35% in this example. Let's say that the weighted percent of state change turned out to be 31%...

The calculated percent state change for the service (31%) will then be compared against flapping thresholds to see what should happen:

- If the service was *not* previously flapping and 31% is *equal to or greater than* the high flap threshold, Icinga considers the service to have just started flapping.
- If the service *was* previously flapping and 31% is *less than* the low flap threshold, Icinga considers the service to have just stopped flapping.

If neither of those two conditions are met, the flap detection logic won't do anything else with the service, since it is either not currently flapping or it is still flapping.

Flap Detection for Services

Icinga checks to see if a service is flapping whenever the service is checked (either actively or passively).

The flap detection logic for services works as described in the example above.

Flap Detection for Hosts

Host flap detection works in a similar manner to service flap detection, with one important difference: Icinga will attempt to check to see if a host is flapping whenever:

- The host is checked (actively or passively)
- Sometimes when a service associated with that host is checked. More specifically, when at least x amount of time has passed since the flap detection was last performed, where x is equal to the average check interval of all services associated with the host.

Why is this done? With services we know that the minimum amount of time between consecutive flap detection routines is going to be equal to the service check interval. However, you might not be monitoring hosts on a regular basis, so there might not be a host check interval that can be used in the flap detection logic. Also, it makes sense that checking a service should count towards the detection of host flapping. Services are attributes of or things associated with host after all... At any rate, that's the best method I could come up with for determining how often flap detection could be performed on a host, so there you have it.

Flap Detection Thresholds

Icinga uses several variables to determine the percent state change thresholds it uses for flap detection. For both hosts and services, there are *global* high and low thresholds and *host-* or *service-specific* thresholds that you can configure. Icinga will use the global thresholds for flap detection if you do not specify host- or service-specific thresholds.

The table below shows the global and host- or service-specific variables that control the various thresholds used in flap detection.

Object Type	Global Variables	Object-Specific Variables
Host	<code>low_host_flap_threshold</code>	<code>low_flap_threshold</code>
	<code>high_host_flap_threshold</code>	<code>high_flap_threshold</code>
Service	<code>low_service_flap_threshold</code>	<code>low_flap_threshold</code>
	<code>high_service_flap_threshold</code>	<code>high_flap_threshold</code>

States Used For Flap Detection

Normally Icinga will track the results of the last 21 checks of a host or service, regardless of the check result (host/service state), for use in the flap detection logic.



Tip

You can exclude certain host or service states from use in flap detection logic by using the `flap_detection_options` directive in your host or service definitions. This directive allows you to specify what host or service states (i.e. "UP", "DOWN", "OK", "CRITICAL") you want to use for flap detection. If you don't use this directive, all host or service states are used in flap detection.

Flap Handling

When a service or host is first detected as flapping, Icinga will:

1. Log a message indicating that the service or host is flapping.
2. Add a non-persistent comment to the host or service indicating that it is flapping.
3. Send a "flapping start" notification for the host or service to appropriate contacts.
4. Suppress other notifications for the service or host (this is one of the filters in the [notification logic](#)).

When a service or host stops flapping, Icinga will:

1. Log a message indicating that the service or host has stopped flapping.
2. Delete the comment that was originally added to the service or host when it started flapping.
3. Send a "flapping stop" notification for the host or service to appropriate contacts.
4. Remove the block on notifications for the service or host (notifications will still be bound to the normal [notification logic](#)).

Enabling Flap Detection

In order to enable the flap detection features in Icinga, you'll need to:

- Set `enable_flap_detection` directive is set to 1.
- Set the `flap_detection_enabled` directive in your host and service definitions is set to 1.

If you want to disable flap detection on a global basis, set the `enable_flap_detection` directive to 0.

If you would like to disable flap detection for just a few hosts or services, use the `flap_detection_enabled` directive in the host and/or service definitions to do so.

[Prev](#)

[Up](#)

[Next](#)

Redundant and Failover Network
Monitoring

[Home](#)

Notification Escalations

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Notification Escalations

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

Notification Escalations

Introduction



Icinga supports optional escalation of contact notifications for hosts and services. Escalation of host and service notifications is accomplished by defining [host escalations](#) and [service escalations](#) in your [object configuration file\(s\)](#).



Note

The examples we provide below all make use of service escalation definitions, but host escalations work the same way. Except, of course, that they're for hosts instead of services. :-)

When Are Notifications Escalated?

Notifications are escalated *if and only if* one or more escalation definitions matches the current notification that is being sent out. If a host or service notification *does not* have any valid escalation definitions that applies to it, the contact group(s) specified in either the host group or service definition will be used for the notification. Look at the example below:

```
define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 3
    last_notification   5
    notification_interval 90
    contact_groups     nt-admins, managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 6
```

```

last_notification      10
notification_interval 60
contact_groups         nt-admins, managers, everyone
}

```

Notice that there are "holes" in the notification escalation definitions. In particular, notifications 1 and 2 are not handled by the escalations, nor are any notifications beyond 10. For the first and second notification, as well as all notifications beyond the tenth one, the *default* contact groups specified in the service definition are used. For all the examples we'll be using, we'll be assuming that the default contact groups for the service definition is called *nt-admins*.

Contact Groups

When defining notification escalations, it is important to keep in mind that any contact groups that were members of "lower" escalations (i.e. those with lower notification number ranges) should also be included in "higher" escalation definitions. This should be done to ensure that anyone who gets notified of a problem *continues* to get notified as the problem is escalated. Example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 3
    last_notification   5
    notification_interval 90
    contact_groups     nt-admins, managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 6
    last_notification   0
    notification_interval 60
    contact_groups     nt-admins, managers, everyone
}

```

The first (or "lowest") escalation level includes both the *nt-admins* and *managers* contact groups. The last (or "highest") escalation level includes the *nt-admins*, *managers*, and *everyone* contact groups. Notice that the *nt-admins* contact group is included in both escalation definitions. This is done so that they continue to get paged if there are still problems after the first two service notifications are sent out. The *managers* contact group first appears in the "lower" escalation definition - they are first notified when the third problem notification gets sent out. We want the *managers* group to continue to be notified if the problem continues past five notifications, so they are also included in the "higher" escalation definition.

Overlapping Escalation Ranges

Notification escalation definitions can have notification ranges that overlap. Take the following example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 3
    last_notification   5
    notification_interval 20
    contact_groups     nt-admins, managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 1
    last_notification   10
    notification_interval 30
    contact_groups     nt-admins, managers
}

```

```

service_description      HTTP
first_notification      4
last_notification        0
notification_interval   30
contact_groups          on-call-support
}

```

In the example above:

- The *nt-admins* and *managers* contact groups get notified on the third notification
- All three contact groups get notified on the fourth and fifth notifications
- Only the *on-call-support* contact group gets notified on the sixth (or higher) notification

Recovery Notifications

Recovery notifications are slightly different than problem notifications when it comes to escalations. Take the following example:

```

define serviceescalation{
    host_name              webserver
    service_description     HTTP
    first_notification      3
    last_notification       5
    notification_interval   20
    contact_groups          nt-admins,managers
}

define serviceescalation{
    host_name              webserver
    service_description     HTTP
    first_notification      4
    last_notification        0
    notification_interval   30
    contact_groups          on-call-support
}

```

If, after three problem notifications, a recovery notification is sent out for the service, who gets notified? The recovery is actually the fourth notification that gets sent out. However, the escalation code is smart enough to realize that only those people who were notified about the problem on the third notification should be notified about the recovery. In this case, the *nt-admins* and *managers* contact groups would be notified of the recovery.

Notification Intervals

You can change the frequency at which escalated notifications are sent out for a particular host or service by using the *notification_interval* option of the hostgroup or service escalation definition. Example:

```

define serviceescalation{
    host_name              webserver
    service_description     HTTP
    first_notification      3
    last_notification       5
    notification_interval   45
    contact_groups          nt-admins,managers
}

define serviceescalation{
    host_name              webserver
    service_description     HTTP
    first_notification      6
}

```

```

last_notification      0
notification_interval 60
contact_groups         nt-admins, managers, everyone
}

```

In this example we see that the default notification interval for the services is 240 minutes (this is the value in the service definition). When the service notification is escalated on the 3rd, 4th, and 5th notifications, an interval of 45 minutes will be used between notifications. On the 6th and subsequent notifications, the notification interval will be 60 minutes, as specified in the second escalation definition.

Since it is possible to have overlapping escalation definitions for a particular hostgroup or service, and the fact that a host can be a member of multiple hostgroups, Icinga has to make a decision on what to do as far as the notification interval is concerned when escalation definitions overlap. In any case where there are multiple valid escalation definitions for a particular notification, Icinga will choose the smallest notification interval. Take the following example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 3
    last_notification   5
    notification_interval 45
    contact_groups     nt-admins, managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 4
    last_notification   0
    notification_interval 60
    contact_groups     nt-admins, managers, everyone
}

```

We see that the two escalation definitions overlap on the 4th and 5th notifications. For these notifications, Icinga will use a notification interval of 45 minutes, since it is the smallest interval present in any valid escalation definitions for those notifications.

One last note about notification intervals deals with intervals of 0. An interval of 0 means that Icinga should only sent a notification out for the first valid notification during that escalation definition. All subsequent notifications for the hostgroup or service will be suppressed. Take this example:

```

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 3
    last_notification   5
    notification_interval 45
    contact_groups     nt-admins, managers
}

define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 4
    last_notification   6
    notification_interval 0
    contact_groups     nt-admins, managers, everyone
}

```

```
define serviceescalation{
    host_name          webserver
    service_description HTTP
    first_notification 7
    last_notification   0
    notification_interval 30
    contact_groups     nt-admins,managers
}
```

In the example above, the maximum number of problem notifications that could be sent out about the service would be four. This is because the notification interval of 0 in the second escalation definition indicates that only one notification should be sent out (starting with and including the 4th notification) and all subsequent notifications should be repressed. Because of this, the third service escalation definition has no effect whatsoever, as there will never be more than four notifications.

Time Period Restrictions

Under normal circumstances, escalations can be used at any time that a notification could normally be sent out for the host or service. This "notification time window" is determined by the *notification_period* directive in the [host](#) or [service](#) definition.

You can optionally restrict escalations so that they are only used during specific time periods by using the *escalation_period* directive in the host or service escalation definition. If you use the *escalation_period* directive to specify a [timeperiod](#) during which the escalation can be used, the escalation will only be used during that time. If you do not specify any *escalation_period* directive, the escalation can be used at any time within the "notification time window" for the host or service.



Note

Escalated notifications are still subject to the normal time restrictions imposed by the *notification_period* directive in a host or service definition, so the [timeperiod](#) you specify in an escalation definition should be a subset of that larger "notification time window".

State Restrictions

If you would like to restrict the escalation definition so that it is only used when the host or service is in a particular state, you can use the *escalation_options* directive in the host or service escalation definition. If you do not use the *escalation_options* directive, the escalation can be used when the host or service is in any state.

[Prev](#)
[Up](#)
[Next](#)
[Detection and Handling of State Flapping](#)
[Home](#)
[Escalation Condition](#)



Escalation Condition

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Escalation Condition

Introduction

Starting with Icinga 1.0.1 a patch is implemented which introduces the ability to define an escalation_condition (similar to escalation_options [w,u,c,r]). An escalation with a defined condition will only be escalated if the current state of a particular host/service fits the condition. One possible example of use for this could be the following scenario:

Imagine two different escalations for the same service *foo*. One of them should only escalate when service *bar* is OK, the other should escalate if *bar* is CRITICAL or WARNING. Now think about *foo* being the main service offered by a company and the admin has to react immediately if it is down. *bar* could be a service indicating if the admin is in the office or at home and the escalation would react as following:

- If the admin is in the office, send an email first, after 5 minutes send an SMS
- If the admin is at home, send an SMS first and after 30 minutes a second SMS to the admin and the head of department

This should be achieved without reloading or restarting the Icinga service.

Syntax

The escalation_condition option is completely optional and can be defined for host escalations as well as for service escalations. The syntax is:

```
escalation_condition <condition> ( [ & / | ] <condition> )*
```

where <condition> is either: host hostname = [u,d,o] or service hostname.service_description = [w,u,c,o].

As you can see, the escalation_condition accepts a list of one or more conditions separated by "&" (logical AND) or "|" (logical OR). The meanings of [w,u,c,o,d] differ a bit from the ones used for escalation_options:

- w = WARNING
- u = UNKNOWN

- c = CRITICAL
- o = OK for services or UP for hosts (one could think of ONLINE)
- d = Down for hosts

Example

```
define serviceescalation {
    host_name           localhost
    service_description HTTP
    first_notification 5
    contact_groups     admins, managers
    escalation_condition host linux=d | service linux.SSH=w,c
}
```

This example escalation would be escalated if the HOST 'linux' is DOWN or the Service 'linux.SSH' is WARNING or CRITICAL.

[Thanks to: Vitali Voroth, DECOIT GmbH * <http://www.decoit.de>]

[Prev](#)

[Up](#)

[Next](#)

[Notification Escalations](#)

[Home](#)

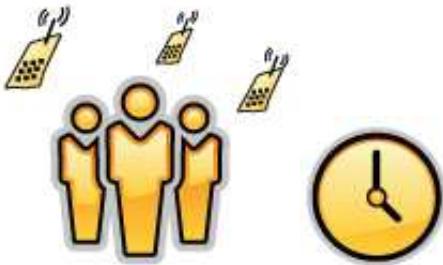
[On-Call Rotations](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**On-Call Rotations**[Prev](#)**Chapter 7. Advanced Topics**[Next](#)

On-Call Rotations

Introduction



Admins often have to shoulder the burden of answering pagers, cell phone calls, etc. when they least desire them. No one likes to be woken up at 4 am to fix a problem. But its often better to fix the problem in the middle of the night, rather than face the wrath of an unhappy boss when you stroll in at 9 am the next morning.

For those lucky admins who have a team of gurus who can help share the responsibility of answering alerts, on-call rotations are often setup. Multiple admins will often alternate taking notifications on weekends, weeknights, holidays, etc.

We'll show you how you can create `timeperiod` definitions in a way that can facilitate most on-call notification rotations. These definitions won't handle human issues that will inevitably crop up (admins calling in sick, swapping shifts, or throwing their pagers into the river), but they will allow you to setup a basic structure that should work the majority of the time.

Scenario 1: Holidays and Weekends

Two admins - John and Bob - are responsible for responding to Icinga alerts. John receives all notifications for weekdays (with 24 hour days), excluding holidays; Bob handles notifications during the weekends and holidays. Lucky Bob. Here's how you can define this type of rotation using `timeperiods`...

First, define 3 `timeperiods` that contains time ranges for holidays, weekdays, and weekends:

```
define timeperiod{
    name                weekdays
    timeperiod_name     weekdays
    monday              00:00-24:00
    tuesday             00:00-24:00
    wednesday           00:00-24:00
```

```

thursday          00:00-24:00
friday           00:00-24:00
}

define timeperiod{
    name            weekends
    timeperiod_name
    saturday        00:00-24:00
    sunday         00:00-24:00
}

define timeperiod{
    name            holidays
    timeperiod_name
    january 1      00:00-24:00 ; New Year's Day
    2008-03-23     00:00-24:00 ; Easter (2008)
    2009-04-12     00:00-24:00 ; Easter (2009)
    monday -1 may  00:00-24:00 ; Memorial Day (Last Monday in May)
    july 4          00:00-24:00 ; Independence Day
    monday 1 september 00:00-24:00 ; Labor Day (1st Monday in September)
    thursday 4 november 00:00-24:00 ; Thanksgiving (4th Thursday in November)
    december 25     00:00-24:00 ; Christmas
    december 31     17:00-24:00 ; New Year's Eve (5pm onwards)
}

```

Next, define a timeperiod for John's on-call times that include weekdays, but excludes the dates/times defined in the holidays timeperiod above:

```

define timeperiod{
    timeperiod_name      john-oncall
    use                 weekdays      ; Include weekdays
    exclude             holidays       ; Exclude holiday dates/times defined elsewhere
}

```

You can now reference this timeperiod in John's contact definition:

```

define contact{
    contact_name        john
    ...
    host_notification_period   john-oncall
    service_notification_period  john-oncall
}

```

Define a new timeperiod for Bob's on-call times that include weekends and the dates/times defined in the holidays timeperiod above:

```

define timeperiod{
    timeperiod_name      bob-oncall
    use                 weekends,holidays ; Include weekend and holiday date/times defined elsewhere
}

```

You can now reference this timeperiod in Bob's contact definition:

```

define contact{
    contact_name        bob
    ...
    host_notification_period   bob-oncall
    service_notification_period  bob-oncall
}

```

Scenario 2: Alternating Days

In this scenario John and Bob alternate handling alerts every other day - regardless of whether its a weekend, weekday, or holiday.

Define a timeperiod for when John should receive notifications. Assuming today's date is August 1st, 2007 and John is handling notifications starting today, the definition would look like this:

```
define timeperiod{
    timeperiod_name          john-oncall
    2007-08-01 / 2           00:00-24:00      ; Every two days, starting August 1st, 2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the days that John doesn't, so his first on-call day starts tomorrow (August 2nd, 2007).

```
define timeperiod{
    timeperiod_name          bob-oncall
    2007-08-02 / 2           00:00-24:00      ; Every two days, starting August 2nd, 2007
}
```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```
define contact{
    contact_name              john
    ...
    host_notification_period   john-oncall
    service_notification_period john-oncall
}

define contact{
    contact_name              bob
    ...
    host_notification_period   bob-oncall
    service_notification_period bob-oncall
}
```

Scenario 3: Alternating Weeks

In this scenario John and Bob alternate handling alerts every other week. John handles alerts Sunday through Saturday one week, and Bob handles alerts for the following seven days. This continues in perpetuity.

Define a timeperiod for when John should receive notifications. Assuming today's date is Sunday, July 29th, 2007 and John is handling notifications this week (starting today), the definition would look like this:

```
define timeperiod{
    timeperiod_name          john-oncall
    2007-07-29 / 14 00:00-24:00      ; Every 14 days (two weeks), starting Sunday, July 29th, 2007
    2007-07-30 / 14 00:00-24:00      ; Every other Monday starting July 30th, 2007
    2007-07-31 / 14 00:00-24:00      ; Every other Tuesday starting July 31st, 2007
    2007-08-01 / 14 00:00-24:00      ; Every other Wednesday starting August 1st, 2007
    2007-08-02 / 14 00:00-24:00      ; Every other Thursday starting August 2nd, 2007
    2007-08-03 / 14 00:00-24:00      ; Every other Friday starting August 3rd, 2007
    2007-08-04 / 14 00:00-24:00      ; Every other Saturday starting August 4th, 2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the weeks that John doesn't, so his first on-call day starts next Sunday (August 5th, 2007).

```
define timeperiod{
    timeperiod_name      bob-oncall
    2007-08-05 / 14 00:00-24:00 ; Every 14 days (two weeks), starting Sunday, August 5th, 2007
    2007-08-06 / 14 00:00-24:00 ; Every other Monday starting August 6th, 2007
    2007-08-07 / 14 00:00-24:00 ; Every other Tuesday starting August 7th, 2007
    2007-08-08 / 14 00:00-24:00 ; Every other Wednesday starting August 8th, 2007
    2007-08-09 / 14 00:00-24:00 ; Every other Thursday starting August 9th, 2007
    2007-08-10 / 14 00:00-24:00 ; Every other Friday starting August 10th, 2007
    2007-08-11 / 14 00:00-24:00 ; Every other Saturday starting August 11th, 2007
}
```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```
define contact{
    contact_name          john
    ...
    host_notification_period   john-oncall
    service_notification_period   john-oncall
}

define contact{
    contact_name          bob
    ...
    host_notification_period   bob-oncall
    service_notification_period   bob-oncall
}
```

Scenario 4: Vacation Days

In this scenarios, John handles notifications for all days except those he has off. He has several standing days off each month, as well as some planned vacations. Bob handles notifications when John is on vacation or out of the office.

First, define a timeperiod that contains time ranges for John's vacation days and days off:

```
define timeperiod{
    name                  john-out-of-office
    timeperiod_name       john-out-of-office
    day 15                00:00-24:00 ; 15th day of each month
    day -1                00:00-24:00 ; Last day of each month (28th, 29th, 30th, or 31st)
    day -2                00:00-24:00 ; 2nd to last day of each month (27th, 28th, 29th, or 30th)
    january 2              00:00-24:00 ; January 2nd each year
    june 1 - july 5        00:00-24:00 ; Yearly camping trip (June 1st - July 5th)
    2007-11-01 - 2007-11-10 00:00-24:00 ; Vacation to the US Virgin Islands (November 1st-10th, 2007)
}
```

Next, define a timeperiod for John's on-call times that excludes the dates/times defined in the timeperiod above:

```
define timeperiod{
    timeperiod_name       john-oncall
    monday                00:00-24:00
    tuesday               00:00-24:00
    wednesday             00:00-24:00
    thursday              00:00-24:00
    friday                00:00-24:00
    exclude               john-out-of-office ; Exclude dates/times John is out
}
```

You can now reference this timeperiod in John's contact definition:

```
define contact{
    contact_name          john
    ...
    host_notification_period   john-oncall
    service_notification_period   john-oncall
}
```

Define a new timeperiod for Bob's on-call times that include the dates/times that John is out of the office:

```
define timeperiod{
    timeperiod_name          bob-oncall
    use                      john-out-of-office      ; Include holiday date/times that John is out
}
```

You can now reference this timeperiod in Bob's contact definition:

```
define contact{
    contact_name            bob
    ...
    host_notification_period bob-oncall
    service_notification_period bob-oncall
}
```

Other Scenarios

There are a lot of other on-call notification rotation scenarios that you might have. The date exception directive in [timeperiod definitions](#) is capable of handling most dates and date ranges that you might need to use, so check out the different formats that you can use. If you make a mistake when creating timeperiod definitions, always err on the side of giving someone else more on-call duty time. :-)

[Prev](#)

[Up](#)

[Next](#)

[Escalation Condition](#)

[Home](#)

[Monitoring Service and Host Clusters](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Monitoring Service and Host Clusters

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

Monitoring Service and Host Clusters

Introduction

Several people have asked how to go about monitoring clusters of hosts or services, so this little documentation should provide you with some information on how to do this. It's fairly straightforward, so hopefully you find things easy to understand...

First off, we need to define what we mean by a "cluster". The simplest way to understand this is with an example. Let's say that your organization has five hosts which provide redundant DNS services to your organization. If one of them fails, it's not a major catastrophe because the remaining servers will continue to provide name resolution services. If you're concerned with monitoring the availability of DNS service to your organization, you will want to monitor five DNS servers. This is what is considered to be a *service* cluster. The service cluster consists of five separate DNS services that you are monitoring. Although you do want to monitor each individual service, your main concern is with the overall status of the DNS service cluster, rather than the availability of any one particular service.

If your organization has a group of hosts that provide a high-availability (clustering) solution, that would be considered to be a *host* cluster. If one particular host fails, another will step in to take over all the duties of the failed server. As a side note, check out the [High-Availability Linux Project](#) for information on providing host and service redundancy with Linux.

Plan of Attack

There are several ways you could potentially monitor service or host clusters. We'll describe one method to do that. Monitoring service or host clusters involves two things:

- Monitoring individual cluster elements
- Monitoring the cluster as a collective entity

Monitoring individual host or service cluster elements is easier than you think. In fact, you're probably already doing it. For service clusters, just make sure that you are monitoring each service element of the cluster. If you've got a cluster of five DNS servers, make sure you have five separate service definitions (probably using the `check_dns` plugin). For host clusters, make sure you have configured appropriate host definitions for each member of the cluster (you'll also have to define at least one service to be monitored for each of the hosts). **Important:** You're going to want to disable notifications for the individual cluster elements (host or service definitions). Even though no notifications will be sent about the individual elements, you'll still get a visual display of the individual host or service status in the [status CGI](#). This will be useful for pinpointing the source of problems within the cluster in the future.

Monitoring the overall cluster can be done by using the previously cached results of cluster elements. Although you could re-check all elements of the cluster to determine the cluster's status, why waste bandwidth and resources when you already have the results cached? Where are the results cached? Cached results for cluster elements can be found in the [status file](#) (assuming you are monitoring each element). The *check_cluster* plugin is designed specifically for checking cached host and service states in the status file. **Important:** Although you didn't enable notifications for individual elements of the cluster, you will want them enabled for the overall cluster status check.

Using the *check_cluster* Plugin

The *check_cluster* plugin is designed to report the overall status of a host or service cluster by checking the status information of each individual host or service cluster elements.

More to come... The *check_cluster* plugin can be found in the contrib directory of the Nagios Plugins release at <http://sourceforge.net/projects/nagiosplug>.

Monitoring Service Clusters

Let's say you have three DNS servers that provide redundant services on your network. First off, you need to be monitoring each of these DNS servers separately before you can monitor them as a cluster. We'll assume that you already have three separate services (all called "DNS Service") associated with your DNS hosts (called "host1", "host2" and "host3").

In order to monitor the services as a cluster, you'll need to create a new "cluster" service. However, before you do that, make sure you have a service cluster check command configured. Let's assume that you have a command called *check_service_cluster* defined as follows:

```
define command{
    command_name      check_service_cluster
    command_line      /usr/local/icinga/libexec/check_cluster --service -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
}
```

Now you'll need to create the "cluster" service and use the *check_service_cluster* command you just created as the cluster's check command. The example below gives an example of how to do this. The example below will generate a CRITICAL alert if 2 or more services in the cluster are in a non-OK state, and a WARNING alert if only 1 of the services is in a non-OK state. If all the individual service members of the cluster are OK, the cluster check will return an OK state as well.

```
define service{
    ...
    check_command  check_service_cluster!"DNS Cluster"!0!$SERVICESTATEID:host1:DNS Service$$SERVICESTATEID:host2:DNS Service$$SERVICESTATEID:host3:DNS Service$
    ...
}
```

It is important to notice that we are passing a comma-delimited list of *on-demand* service state [macros](#) to the \$ARG4\$ macro in the cluster check command. That's important! Icinga will fill those on-demand macros in with the current service state IDs (numerical values, rather than text strings) of the individual members of the cluster.

Monitoring Host Clusters

Monitoring host clusters is very similiar to monitoring service clusters. Obviously, the main difference is that the cluster members are hosts and not services. In order to monitor the status of a host cluster, you must define a service that uses the *check_cluster* plugin. The service should *not* be associated with any of the hosts in the cluster, as this will cause problems with notifications for the cluster if that host goes down. A good idea might be to associate the service with the host that Icinga is running on. After all, if the host that Icinga is running on goes down, then Icinga isn't running anymore, so there isn't anything you can do as far as monitoring (unless you've setup [redundant monitoring hosts](#))...

Anyway, let's assume that you have a *check_host_cluster* command defined as follows:

```
define command{
    command_name  check_host_cluster
    command_line   /usr/local/icinga/libexec/check_cluster --host -l $ARG1$ -w $ARG2$ -c $ARG3$ -d $ARG4$
}
```

Let's say you have three hosts (named "host1", "host2" and "host3") in the host cluster. If you want Icinga to generate a warning alert if one host in the cluster is not UP or a critical alert if two or more hosts are not UP, the the service you define to monitor the host cluster might look something like this:

```
define service{
    ...
    check_command  check_host_cluster!"Super Host Cluster"!0!1!$HOSTSTATEID:host1$,HOSTSTATEID:host2$,HOSTSTATEID:host3$...
}
```

It is important to notice that we are passing a comma-delimited list of *on-demand* host state **macros** to the \$ARG4\$ macro in the cluster check command. That's important! Icinga will fill those on-demand macros in with the current host state IDs (numerical values, rather than text strings) of the individual members of the cluster.

That's it! Icinga will periodically check the status of the host cluster and send notifications to you when its status is degraded (assuming you've enabled notification for the service). Note that for thehost definitions of each cluster member, you will most likely want to disable notifications when the host goes down . Remeber that you don't care as much about the status of any individual host as you do the overall status of the cluster. Depending on your network layout and what you're trying to accomplish, you may wish to leave notifications for unreachable states enabled for the host definitions.

[Prev](#)

[Up](#)

[Next](#)

[On-Call Rotations](#)

[Home](#)

[Host and Service Dependencies](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Host and Service Dependencies**[Prev](#)[Chapter 7. Advanced Topics](#)[Next](#)

Host and Service Dependencies

Introduction

Service and host dependencies are an advanced feature of Icinga that allow you to control the behavior of hosts and services based on the status of one or more other hosts or services. We'll explain how dependencies work, along with the differences between host and service dependencies.

Service Dependencies Overview

There are a few things you should know about service dependencies:

1. A service can be dependent on one or more other services
2. A service can be dependent on services which are not associated with the same host
3. Service dependencies are not inherited (unless specifically configured to)
4. Service dependencies can be used to cause service check execution and service notifications to be suppressed under different circumstances (OK, WARNING, UNKNOWN, and/or CRITICAL states)
5. Service dependencies might only be valid during specific [timeperiods](#)

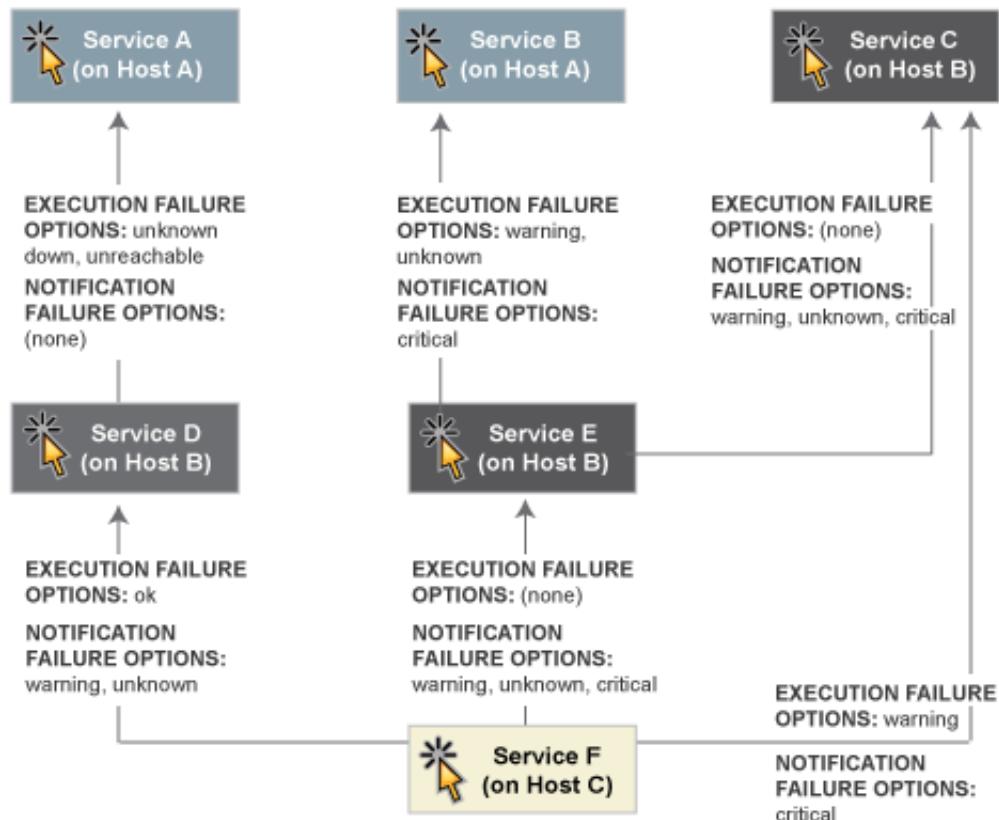
Defining Service Dependencies

First, the basics. You create service dependencies by adding [service dependency definitions](#) in your [object config file\(s\)](#). In each definition you specify the *dependent* service, the service you are *depending on*, and the criteria (if any) that cause the execution and notification dependencies to fail (these are described later).

You can create several dependencies for a given service, but you must add a separate service dependency definition for each dependency you create.

Example Service Dependencies

The image below shows an example logical layout of service notification and execution dependencies. Different services are dependent on other services for notifications and check execution.



In this example, the dependency definitions for *Service F* on *Host C* would be defined as follows:

```

define servicedependency{
    host_name                  Host B
    service_description        Service D
    dependent_host_name       Host C
    dependent_service_description Service F
    execution_failure_criteria o
    notification_failure_criteria w,u
}

define servicedependency{
    host_name                  Host B
    service_description        Service E
    dependent_host_name       Host C
    dependent_service_description Service F
    execution_failure_criteria n
    notification_failure_criteria w,u,c
}

define servicedependency{
    host_name                  Host B
    service_description        Service C
    dependent_host_name       Host C
    dependent_service_description Service F
    execution_failure_criteria w
    notification_failure_criteria c
}
  
```

The other dependency definitions shown in the image above would be defined as follows:

```

define servicedependency{
    host_name                  Host A
    service_description        Service A
    dependent_host_name       Host B
    dependent_service_description Service D
  
```

```

execution_failure_criteria      u
notification_failure_criteria   n
}

define servicedependency{
    host_name                  Host A
    service_description         Service B
    dependent_host_name        Host B
    dependent_service_description Service E
    execution_failure_criteria w,u
    notification_failure_criteria c
}

define servicedependency{
    host_name                  Host B
    service_description         Service C
    dependent_host_name        Host B
    dependent_service_description Service E
    execution_failure_criteria n
    notification_failure_criteria w,u,c
}

```

How Service Dependencies Are Tested

Before Icinga executes a service check or sends notifications out for a service, it will check to see if the service has any dependencies. If it doesn't have any dependencies, the check is executed or the notification is sent out as it normally would be. If the service *does* have one or more dependencies, Icinga will check each dependency entry as follows:

1. Icinga gets the current status * of the service that is being *depended upon*.
2. Icinga compares the current status of the service that is being *depended upon* against either the execution or notification failure options in the dependency definition (whichever one is relevant at the time).
3. If the current status of the service that is being *depended upon* matches one of the failure options, the dependency is said to have failed and Icinga will break out of the dependency check loop.
4. If the current state of the service that is being *depended upon* does not match any of the failure options for the dependency entry, the dependency is said to have passed and Icinga will go on and check the next dependency entry.

This cycle continues until either all dependencies for the service have been checked or until one dependency check fails.



Note

*One important thing to note is that by default, Icinga will use the most current [hard state](#) of the service(s) that is/are being depended upon when it does the dependency checks. If you want Icinga to use the most current state of the services (regardless of whether its a soft or hard state), enable the [soft_state_dependencies](#) option.

Execution Dependencies

Execution dependencies are used to restrict when [active checks](#) of a service can be performed. [Passive checks](#) are not restricted by execution dependencies.

If *all* of the execution dependency tests for the service *passed*, Icinga will execute the check of the service as it normally would. If even just one of the execution dependencies for a service fails, Icinga will temporarily prevent the execution of checks for that (dependent) service. At some point in the future the execution dependency tests for the service may all pass. If this happens, Icinga will start checking the service again as it normally would. More information on the check scheduling logic can be found [here](#).

In the example above, **Service E** would have failed execution dependencies if **Service B** is in a WARNING or UNKNOWN state. If this was the case, the service check would not be performed and the check would be scheduled for (potential) execution at a later time.

Notification Dependencies

If *all* of the notification dependency tests for the service *passed*, Icinga will send notifications out for the service as it normally would. If even just one of the notification dependencies for a service fails, Icinga will temporarily repress notifications for that (dependent) service. At some point in the future the notification dependency tests for the service may all pass. If this happens, Icinga will start sending out notifications again as it normally would for the service. More information on the notification logic can be found [here](#).

In the example above, **Service F** would have failed notification dependencies if **Service C** is in a CRITICAL state, *and/or* **Service D** is in a WARNING or UNKNOWN state, *and/or* if **Service E** is in a WARNING, UNKNOWN, or CRITICAL state. If this were the case, notifications for the service would not be sent out.

Dependency Inheritance

As mentioned before, service dependencies are *not* inherited by default. In the example above you can see that Service F is dependent on Service E. However, it does not automatically inherit Service E's dependencies on Service B and Service C. In order to make Service F dependent on Service C we had to add another service dependency definition. There is no dependency definition for Service B, so Service F is *not* dependent on Service B.

If you *do* wish to make service dependencies inheritable, you must use the *inherits_parent* directive in the [service dependency](#) definition. When this directive is enabled, it indicates that the dependency inherits dependencies of the service *that is being depended upon* (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.

In the example above, imagine that you want to add a new dependency for service F to make it dependent on service A. You could create a new dependency definition that specified service F as the *dependent* service and service A as being the *master* service (i.e. the service *that is being dependend on*). You could alternatively modify the dependency definition for services D and F to look like this:

```
define servicedependency{
    host_name           Host B
    service_description Service D
    dependent_host_name Host C
    dependent_service_description Service F
    execution_failure_criteria o
    notification_failure_criteria n
    inherits_parent      1
}
```

Since the *inherits_parent* directive is enabled, the dependency between services A and D will be tested when the dependency between services F and D are being tested.

Dependencies can have multiple levels of inheritance. If the dependency definition between A and D had its *inherits_parent* directive enable and service A was dependent on some other service (let's call it service G), the service F would be dependent on services D, A, and G (each with potentially different criteria).

Host Dependencies

As you'd probably expect, host dependencies work in a similar fashion to service dependencies. The difference is that they're for hosts, not services.



Tip

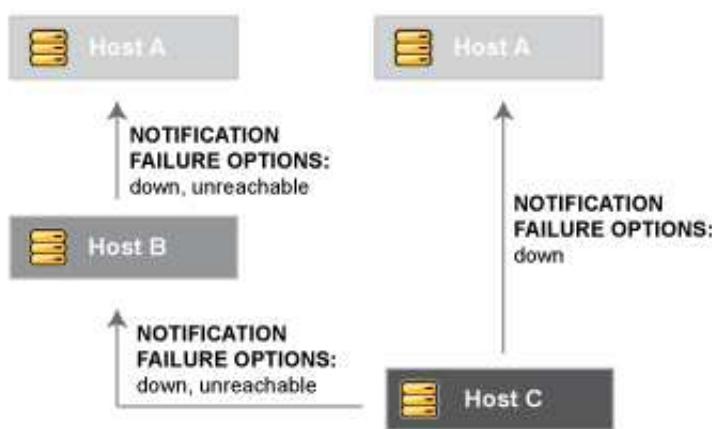
Do not confuse host dependencies with parent/child host relationships. You should be using parent/child host relationships (defined with the *parents* directive in `host` definitions) for most cases, rather than host dependencies. A description of how parent/child host relationships work can be found in the documentation on [network reachability](#).

Here are the basics about host dependencies:

1. A host can be dependent on one or more other host
2. Host dependencies are not inherited (unless specifically configured to)
3. Host dependencies can be used to cause host check execution and host notifications to be suppressed under different circumstances (UP, DOWN, and/or UNREACHABLE states)
4. Host dependencies might only be valid during specific [timeperiods](#)

Example Host Dependencies

The image below shows an example of the logical layout of host notification dependencies. Different hosts are dependent on other hosts for notifications.



In the example above, the dependency definitions for *Host C* would be defined as follows:

```

define hostdependency{
    host_name                  Host A
    dependent_host_name        Host C
    notification_failure_criteria d
}

define hostdependency{
    host_name                  Host B
    dependent_host_name        Host C
    notification_failure_criteria d,u
}

```

As with service dependencies, host dependencies are not inherited. In the example image you can see that Host C does not inherit the host dependencies of Host B. In order for Host C to be dependent on Host A, a new host dependency definition must be defined.

Host notification dependencies work in a similiar manner to service notification dependencies. If *all* of the notification dependency tests for the host *pass*, Icinga will send notifications out for the host as it normally would. If even just one of the notification dependencies for a host fails, Icinga will temporarily repress notifications for that (dependent) host. At some point in the future the notification dependency tests for the host may all pass. If this happens, Icinga will start sending out notifications again as it normally would for the host. More information on the notification logic can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

Monitoring Service and Host
Clusters

[Home](#)

State Stalking

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**State Stalking**[Prev](#)**Chapter 7. Advanced Topics**[Next](#)

State Stalking

Introduction

State "stalking" is a feature which is probably not going to be used by most users. When enabled, it allows you to log changes in the output service and host checks even if the state of the host or service does not change. When stalking is enabled for a particular host or service, Icinga will watch that host or service very carefully and log any changes it sees in the output of check results. As you'll see, it can be very helpful to you in later analysis of the log files.

How Does It Work?

Under normal circumstances, the result of a host or service check is only logged if the host or service has changed state since it was last checked. There are a few exceptions to this, but for the most part, that's the rule.

If you enable stalking for one or more states of a particular host or service, Icinga will log the results of the host or service check if the output from the check differs from the output from the previous check. Take the following example of eight consecutive checks of a service:

Service Check #:	Service State:	Service Check Output:	Logged Normally	Logged With Stalking
x	OK	RAID array optimal	-	-
x+1	OK	RAID array optimal	-	-
x+2	WARNING	RAID array degraded (1 drive bad, 1 hot spare rebuilding)	✓	✓
x+3	CRITICAL	RAID array degraded (2 drives bad, 1 host spare online, 1 hot spare rebuilding)	✓	✓
x+4	CRITICAL	RAID array degraded (3 drives bad, 2 hot spares online)	-	✓
x+5	CRITICAL	RAID array failed	-	✓
x+6	CRITICAL	RAID array failed	-	-
x+7	CRITICAL	RAID array failed	-	-

Given this sequence of checks, you would normally only see two log entries for this catastrophe. The first one would occur at service check x+2 when the service changed from an OK state to a WARNING state. The second log entry would occur at service check x+3 when the service changed from a WARNING state to a CRITICAL state.

For whatever reason, you may like to have the complete history of this catastrophe in your log files. Perhaps to help explain to your manager how quickly the situation got out of control, perhaps just to laugh at it over a couple of drinks at the local pub...

Well, if you had enabled stalking of this service for CRITICAL states, you would have events at x+4 and x+5 logged in addition to the events at x+2 and x+3. Why is this? With state stalking enabled, Icinga would have examined the output from each service check to see if it differed from the output of the previous check. If the output differed and the state of the service didn't change between the two checks, the result of the newer service check would get logged.

A similiar example of stalking might be on a service that checks your web server. If the check_http plugin first returns a WARNING state because of a 404 error and on subsequent checks returns a WARNING state because of a particular pattern not being found, you might want to know that. If you didn't enable state stalking for WARNING states of the service, only the first WARNING state event (the 404 error) would be logged and you wouldn't have any idea (looking back in the archived logs) that future WARNING states were not due to a 404, but rather some text pattern that could not be found in the returned web page.

Should I Enable Stalking?

First, you must decide if you have a real need to analyze archived log data to find the exact cause of a problem. You may decide you need this feature for some hosts or services, but not for all. You may also find that you only have a need to enable stalking for some host or service states, rather than all of them. For example, you may decide to enable stalking for WARNING and CRITICAL states of a service, but not for OK and UNKNOWN states.

The decision to enable state stalking for a particular host or service will also depend on the plugin that you use to check that host or service. If the plugin always returns the same text output for a particular state, there is no reason to enable stalking for that state.

How Do I Enable Stalking?

You can enable state stalking for hosts and services by using the *stalking_options* directive in [host and service definitions](#).

How Does Stalking Differ From Volatile Services?

[Volatile services](#) are similar, but will cause notifications and event handlers to run. Stalking is purely for logging purposes.

Caveats

You should be aware that there are some potential pitfalls with enabling stalking. These all relate to the reporting functions found in various [CGIs](#) (histogram, alert summary, etc.). Because state stalking will cause additional alert entries to be logged, the data produced by the reports will show evidence of inflated numbers of alerts.

As a general rule, I would suggest that you *not* enable stalking for hosts and services without thinking things through. Still, it's there if you need and want it.

[Prev](#)

[Up](#)

[Next](#)

[Host and Service Dependencies](#)

[Home](#)

[Performance Data](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Performance Data

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Performance Data

Introduction

Icinga is designed to allow [plugins](#) to return optional performance data in addition to normal status data, as well as allow you to pass that performance data to external applications for processing. A description of the different types of performance data, as well as information on how to go about processing that data is described below...

Types of Performance Data

There are two basic categories of performance data that can be obtained from Icinga:

1. Check performance data
2. Plugin performance data

Check performance data is internal data that relates to the actual execution of a host or service check. This might include things like service check latency (i.e. how "late" was the service check from its scheduled execution time) and the number of seconds a host or service check took to execute. This type of performance data is available for all checks that are performed. The [\\$HOSTEXECUTIONTIME\\$](#) and [\\$SERVICEEXECUTIONTIME\\$](#) macros can be used to determine the number of seconds a host or service check was running and the [\\$HOSTLATENCY\\$](#) and [\\$SERVICELATENCY\\$](#) macros can be used to determine how "late" a regularly-scheduled host or service check was.

Plugin performance data is external data specific to the plugin used to perform the host or service check. Plugin-specific data can include things like percent packet loss, free disk space, processor load, number of current users, etc. - basically any type of metric that the plugin is measuring when it executes. Plugin-specific performance data is optional and may not be supported by all plugins. Plugin-specific performance data (if available) can be obtained by using the [\\$HOSTPERFDATA\\$](#) and [\\$SERVICEPERFDATA\\$](#) macros. Read on for more information on how plugins can return performance data to Icinga for inclusion in the \$HOSTPERFDATA\$ and \$SERVICEPERFDATA\$ macros.

Plugin Performance Data

At a minimum, Icinga plugins must return a single line of human-readable text that indicates the status of some type of measurable data. For example, the `check_ping` plugin might return a line of text like the following:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms
```

With this simple type of output, the entire line of text is available in the `$HOSTOUTPUT$` or `$SERVICEOUTPUT$` [macros](#) (depending on whether this plugin was used as a host check or service check).

Plugins can return optional performance data in their output by sending the normal, human-readable text string that they usually would, followed by a pipe character (`|`), and then a string containing one or more performance data metrics. Let's take the `check_ping` plugin as an example and assume that it has been enhanced to return percent packet loss and average round trip time as performance data metrics. Sample output from the plugin might look like this:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80
```

When Icinga sees this plugin output format it will split the output into two parts:

1. Everything before the pipe character is considered to be the "normal" plugin output and will be stored in either the `$HOSTOUTPUT$` or `$SERVICEOUTPUT$` macro
2. Everything after the pipe character is considered to be the plugin-specific performance data and will be stored in the `$HOSTPERFDATA$` or `$SERVICEPERFDATA$` macro

In the example above, the `$HOSTOUTPUT$` or `$SERVICEOUTPUT$` macro would contain *"PING ok - Packet loss = 0%, RTA = 0.80 ms"* (without quotes) and the `$HOSTPERFDATA$` or `$SERVICEPERFDATA$` macro would contain *"percent_packet_loss=0, rta=0.80"* (without quotes).

Multiple lines of performance data (as well as normal text output) can be obtained from plugins, as described in the [plugin API documentation](#).



Note

The Icinga daemon doesn't directly process plugin performance data, so it doesn't really care what the performance data looks like. There aren't really any inherent limitations on the format or content of the performance data. However, if you are using an external addon to process the performance data (i.e. `PerfParse`), the addon may be expecting that the plugin returns performance data in a specific format. Check the documentation that comes with the addon for more information.

Processing Performance Data

If you want to process the performance data that is available from Icinga and the plugins, you'll need to do the following:

1. Enable the [process_performance_data](#) option.
2. Configure Icinga so that performance data is either written to files and/or processed by executing commands.

Read on for information on how to process performance data by writing to files or executing commands.

Processing Performance Data Using Commands

The most flexible way to process performance data is by having Icinga execute commands (that you specify) to process or redirect the data for later processing by external applications. The commands that Icinga executes to process host and service performance data are determined by

the [host_perfdata_command](#) and [service_perfdata_command](#) options, respectively.

An example command definition that redirects service check performance data to a text file for later processing by another application is shown below:

```
define command{
    command_name    store-service-perfdata
    command_line    /bin/echo -e "$!LASTSERVICECHECK\$!\$HOSTNAME\$!\$SERVICEDESC\$!\$STATE\$!\$SERVICESTATEATTEMPTS\$!\$SERVICESTATETYPES\$!\$SERVICEEXECUTIONTIME\$!\$SERVICELATENCY\$!\$SERVICEOUTPUT\$!\$SERVICEPERFDATA\$" >> /usr/local/icinga/var/service-perfdata.dat
}
```



Tip

This method, while flexible, comes with a relatively high CPU overhead. If you're processing performance data for a large number of hosts and services, you'll probably want Icinga to write performance data to files instead. This method is described in the next section.

Writing Performance Data To Files

You can have Icinga write all host and service performance data directly to text files using the [host_perfdata_file](#) and [service_perfdata_file](#) options. The format in which host and service performance data is written to those files is determined by the [host_perfdata_file_template](#) and [service_perfdata_file_template](#) options.

An example file format template for service performance data might look like this:

```
service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\"$HOSTNAME$\t$SERVICEDESC$\t$SERVICEEXECUTIONTIME$\t$SERVICELATENCY$\t$SERVICEOUTPUT$\t$SERVICEPERFDATA$
```



Note

The above is just one example of how to define a template. This definition will NOT work for PNP!

By default, the text files will be opened in "append" mode. If you need to change the modes to "write" or "non-blocking read/write" (useful when writing to pipes), you can use the [host_perfdata_file_mode](#) and [service_perfdata_file_mode](#) options.

Additionally, you can have Icinga periodically execute commands to periodically process the performance data files (e.g. rotate them) using the [host_perfdata_file_processing_command](#) and [service_perfdata_file_processing_command](#) options. The interval at which these commands are executed are governed by the [host_perfdata_file_processing_interval](#) and [service_perfdata_file_processing_interval](#) options, respectively.

[Prev](#)

[Up](#)

[Next](#)

[State Stalking](#)

[Home](#)

[Scheduled Downtime](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Scheduled Downtime

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Scheduled Downtime

Introduction

Icinga allows you to schedule periods of planned downtime for hosts and service that you're monitoring. This is useful in the event that you actually know you're going to be taking a server down for an upgrade, etc.



Scheduling Downtime

You can schedule downtime for hosts and service through the [extinfo CGI](#) (either when viewing host or service information). Click in the "Schedule downtime for this host/service" link to actually schedule the downtime.

Once you schedule downtime for a host or service, Icinga will add a comment to that host/service indicating that it is scheduled for downtime during the period of time you indicated. When that period of downtime passes, Icinga will automatically delete the comment that it added. Nice, huh?

Fixed vs. Flexible Downtime

When you schedule downtime for a host or service through the web interface you'll be asked if the downtime is fixed or flexible. Here's an explanation of how "fixed" and "flexible" downtime differs:

"Fixed" downtime starts and stops at the exact start and end times that you specify when you schedule it. Okay, that was easy enough...

"Flexible" downtime is intended for times when you know that a host or service is going to be down for X minutes (or hours), but you don't know exactly when that'll start. When you schedule flexible downtime, Icinga will start the scheduled downtime sometime between the start and end times you specified. The downtime will last for as long as the duration you specified when you scheduled the downtime. This assumes that the host or service for which you scheduled flexible downtime either goes down (or becomes unreachable) or goes into a non-OK state sometime between the start and end times you specified. The time at which a host or service transitions to a problem state determines the time at which Icinga actually starts the downtime. The downtime will then last for the duration you specified, even if the host or service recovers before the downtime expires. This is done for a very good reason. As we all know, you might think you've got a problem fixed, but then have to restart a server ten times before it actually works right. Smart, eh?

Triggered Downtime

When scheduling host or service downtime you have the option of making it "triggered" downtime. What is triggered downtime, you ask? With triggered downtime the start of the downtime is triggered by the start of some other scheduled host or service downtime. This is extremely useful if you're scheduling downtime for a large number of hosts or services and the start time of the downtime period depends on the start time of another downtime entry. For instance, if you schedule flexible downtime for a particular host (because it's going down for maintenance), you might want to schedule triggered downtime for all of that host's "children".

How Scheduled Downtime Affects Notifications

When a host or service is in a period of scheduled downtime, Icinga will not allow normal notifications to be sent out for the host or service. However, a "DOWNTIMESTART" notification will get sent out for the host or service, which will serve to put any admins on notice that they won't receive upcoming problem alerts.

When the scheduled downtime is over, Icinga will allow normal notifications to be sent out for the host or service again. A "DOWNTIMEEND" notification will get sent out notifying admins that the scheduled downtime is over, and they will start receiving normal alerts again.

If the scheduled downtime is cancelled prematurely (before it expires), a "DOWNTIMECANCELLED" notification will get sent out to the appropriate admins.

Overlapping Scheduled Downtime

I like to refer to this as the "Oh crap, it's not working" syndrome. You know what I'm talking about. You take a server down to perform a "routine" hardware upgrade, only to later realize that the OS drivers aren't working, the RAID array blew up, or the drive imaging failed and left your original disks useless to the world. Moral of the story is that any routine work on a server is quite likely to take three or four times as long as you had originally planned...

Let's take the following scenario:

1. You schedule downtime for host A from 7:30pm-9:30pm on a Monday
2. You bring the server down about 7:45pm Monday evening to start a hard drive upgrade
3. After wasting an hour and a half battling with SCSI errors and driver incompatibilities, you finally get the machine to boot up

4. At 9:15 you realize that one of your partitions is either hosed or doesn't seem to exist anywhere on the drive
5. Knowing you're in for a long night, you go back and schedule additional downtime for host A from 9:20pm Monday evening to 1:30am Tuesday Morning.

If you schedule overlapping periods of downtime for a host or service (in this case the periods were 7:40pm-9:30pm and 9:20pm-1:30am), Icinga will wait until the last period of scheduled downtime is over before it allows notifications to be sent out for that host or service. In this example notifications would be suppressed for host A until 1:30am Tuesday morning.

[Prev](#)[Up](#)[Next](#)[Performance Data](#)[Home](#)[Using The Embedded Perl Interpreter](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Using The Embedded Perl Interpreter

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Using The Embedded Perl Interpreter

Introduction

Icinga can be compiled with support for an embedded Perl interpreter. This allows Icinga to execute Perl plugins much more efficiently than it otherwise would, so it may be of interest to you if you rely heavily on plugins written in Perl.

Without the embedded Perl interpreter, Icinga executes Perl (and non-Perl) plugins by forking and executing the plugins as an external command. When the embedded Perl interpreter is used, Icinga can execute Perl plugins by simply making a library call.

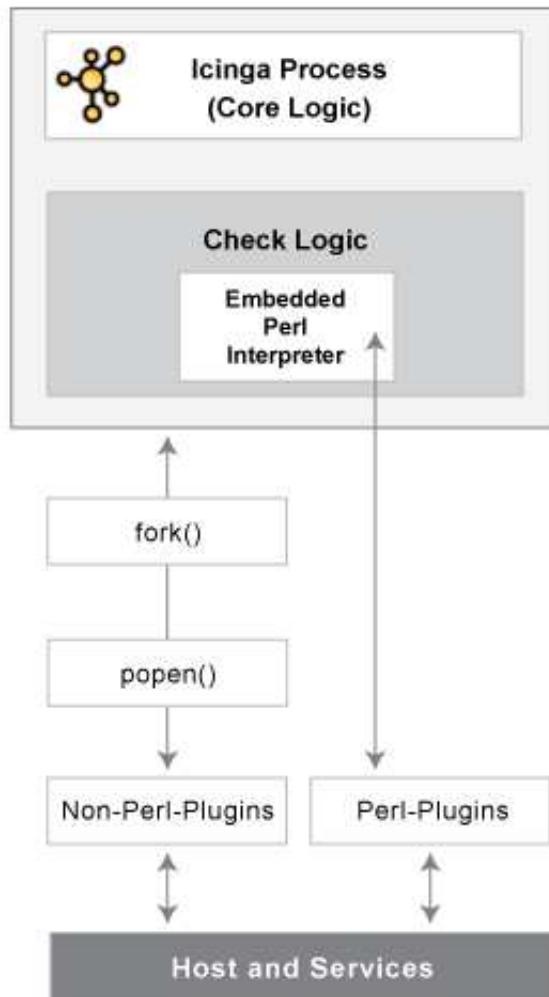


Tip

The embedded Perl interpreter works with all Perl scripts that Icinga executes - not just plugins. This documentation discusses the embedded Perl interpreter in relation to plugins used for host and service checks, but it applies just the same to other types of Perl scripts you may be using for other types of commands (e.g. notification scripts, event handler scripts, etc.).

Stephen Davies contributed the original embedded Perl interpreter code several years back. Stanley Hopcroft has been the primary person helping to improve the embedded Perl interpreter code quite a bit and has commented on the advantages/disadvantages of using it. He has also given several helpful hints on creating Perl plugins that work properly with the embedded interpreter.

It should be noted that "ePN", as used in this documentation, refers to embedded Perl Icinga, or if you prefer, Icinga compiled with an embedded Perl interpreter.



Advantages

Some advantages of ePN (embedded Perl Icinga) include:

- Icinga will spend much less time running your Perl plugins because it no longer forks to execute the plugin (each time loading the Perl interpreter). Instead, it executes your plugin by making a library call.
- It greatly reduces the system impact of Perl plugins and/or allows you to run more checks with Perl plugin than you otherwise would be able to. In other words, you have less incentive to write plugins in other languages such as C/C++, or Expect/TCL, that are generally recognised to have development times at least an order of magnitude slower than Perl (although they do run about ten times faster also - TCL being an exception).
- If you are not a C programmer, then you can still get a huge amount of mileage out of Icinga by letting Perl do all the heavy lifting without having Icinga slow right down. Note however, that the ePN will not speed up your plugin (apart from eliminating the interpreter load time). If you want fast plugins then consider Perl XSUBS (XS), or C *after* you are sure that your Perl is tuned and that you have a suitable algorithm (Benchmark.pm is *invaluable* for comparing the performance of Perl language elements).
- Using the ePN is an excellent opportunity to learn more about Perl.

Disadvantages

The disadvantages of ePN (embedded Perl Icinga) are much the same as Apache mod_perl (i.e. Apache with an embedded interpreter) compared to a plain Apache:

- A Perl program that works *fine* with plain Icinga may *not* work with the ePN. You may have to modify your plugins to get them to work.
- Perl plugins are harder to debug under an ePN than under a plain Icinga.
- Your ePN will have a larger SIZE (memory footprint) than a plain Icinga.
- Some Perl constructs cannot be used or may behave differently than what you would expect.
- You may have to be aware of 'more than one way to do it' and choose a way that seems less attractive or obvious.
- You will need greater Perl knowledge (but nothing very esoteric or stuff about Perl internals - unless your plugin uses XSUBS).

Using The Embedded Perl Interpreter

If you want to use the embedded Perl interpreter to run your Perl plugins and scripts, here's what you'll need to do:

1. Compile Icinga with support for the embedded Perl interpreter (see instructions below).
2. Enable the [enable_embedded_perl](#) option in the main configuration file.
3. Set the [use_embedded_perl_implicitly](#) option to fit your needs. This option determines whether or not the Perl interpreter should be used by default for individual Perl plugins and scripts.

4. Optionally enable or disable certain Perl plugins and scripts from being run using the embedded Perl interpreter. This can be useful if certain Perl scripts have problems being running under the Perl interpreter. See instructions below for more information on doing this.

Compiling Icinga With Embedded Perl

If you want to use the embedded Perl interpreter, you'll first need to compile Icinga with support for it. To do this, simply run the configure script with the addition of the **--enable-embedded-perl** option. If you want the embedded interpreter to cache internally compiled scripts, add the **--with-perlcache** option as well. Example:

```
./configure --enable-embedded-perl --with-perlcache otheroptions...
```

Once you've rerun the configure script with the new options, make sure to recompile Icinga.

Plugin-Specific Use of the Perl Interpreter

Beginning with Icinga 1.4, you can specify which Perl plugins or scripts should or should not be run under the embedded Perl interpreter. This is particularly useful if you have troublesome Perl scripts which do not work well with the Perl interpreter.

To *explicitly* tell Icinga whether or not to use the embedded Perl interpreter for a particular perl script, add one of the following entries to your Perl script/plugin...

To tell Icinga to use the Perl interpreter for a particular script, add this line to the Perl script:

```
# icinga: +epn
```

To tell Icinga to NOT use the embedded Perl interpreter for a particular script, add this line to the Perl script:

```
# icinga: -epn
```

Either line must be located within the first 10 lines of a script for Icinga to detect it.



Note

"icinga: +/-epn" is supported starting with Icinga 1.2.1. Before that you had to specify "nagios: +/-epn" which is still possible for compatibility reasons.



Tip

If you do not *explicitly* use the method above to tell Icinga whether an individual plugin can be run under the Perl interpreter, Icinga will make will a decision for you. This decision process is controlled by the [use_embedded_perl_implicitly](#) variable. If the value is set to 1, all Perl plugins/scripts (that do not explicitly enable/disable the ePN) will be run under the Perl interpreter. If the value is 0, they will NOT be run under the Perl interpreter.

Developing Plugins For Use With Embedded Perl

Information on developing plugins for use with the embedded Perl interpreter can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

[Scheduled Downtime](#)

[Home](#)

[Adaptive Monitoring](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Adaptive Monitoring

[Prev](#)[Chapter 7. Advanced Topics](#)[Next](#)

Adaptive Monitoring

Introduction

Icinga allows you to change certain commands and host and service check attributes during runtime. I'll refer to this feature as "adaptive monitoring". Please note that the adaptive monitoring features found in Icinga will probably not be of much use to 99% of users, but they do allow you to do some neat things.

What Can Be Changed?

The following service check attributes can be changed during runtime:

- Check command (and command arguments)
- Check interval
- Max check attempts
- Check timeperiod
- Event handler command (and command arguments)

The following host check attributes can be changed during runtime:

- Check command (and command arguments)
- Check interval
- Max check attempts
- Check timeperiod
- Event handler command (and command arguments)

The following global attributes can be changed during runtime:

- Global host event handler command (and command arguments)
- Global service event handler command (and command arguments)

External Commands For Adaptive Monitoring

In order to change global or host- or service-specific attributes during runtime, you must submit the appropriate [external command](#) to Icinga via the [external command file](#). The table below lists the different attributes that may be changed during runtime, along with the external command to accomplish the job.

A full listing of external commands that can be used for adaptive monitoring can be found in the [list of external commands](#).



Note

- When changing check commands, check timeperiods, or event handler commands, it is important to note that the new values for these options must have been defined before Icinga was started. Any request to change a command or timeperiod to one which had not been defined when Icinga was started is ignored.
- You can specify command arguments along with the actual command name - just separate individual arguments from the command name (and from each other) using bang (!) characters. More information on how arguments in command definitions are processed during runtime can be found in the documentation on [macros](#).

[Prev](#)
[Up](#)
[Next](#)
[Using The Embedded Perl Interpreter](#)
[Home](#)
[Predictive Dependency Checks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Predictive Dependency Checks

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

Predictive Dependency Checks

Introduction

Host and service [dependencies](#) can be defined to allow you greater control over when checks are executed and when notifications are sent out. As dependencies are used to control basic aspects of the monitoring process, it is crucial to ensure that status information used in the dependency logic is as up to date as possible.

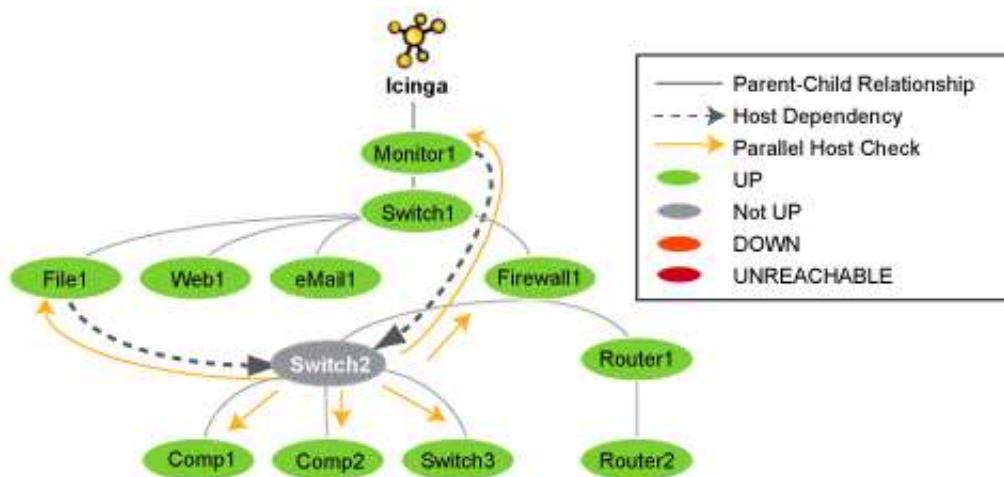
Icinga allows you to enable predictive dependency checks for hosts and services to ensure that the dependency logic will have the most up-to-date status information when it comes to making decisions about whether to send out notifications or allow active checks of a host or service.

How Do Predictive Checks Work?

The image below shows a basic diagram of hosts that are being monitored by Icinga, along with their parent/child relationships and dependencies.

The *Switch2* host in this example has just changed state from an UP state to a problem state. Icinga needs to determine whether the host is DOWN or UNREACHABLE, so it will launch parallel checks of *Switch2*'s immediate parents (*Firewall1*) and children (*Comp1*, *Comp2*, and *Switch3*). This is a normal function of the [host reachability](#) logic.

You will also notice that *Switch2* is depending on *Monitor1* and *File1* for either notifications or check execution (which one is unimportant in this example). If predictive host dependency checks are enabled, Icinga will launch parallel checks of *Monitor1* and *File1* at the same time it launches checks of *Switch2*'s immediate parents and children. Icinga does this because it knows that it will have to test the dependency logic in the near future (e.g. for purposes of notification) and it wants to make sure it has the most current status information for the hosts that take part in the dependency.



That's how predictive dependency checks work. Simple, eh?



Note

Predictive service dependency checks work in a similar manner to what is described above. Except, of course, they deal with services instead of hosts.

Enabling Predictive Checks

Predictive dependency checks involve rather little overhead, so I would recommend that you enable them. In most cases, the benefits of having accurate information for the dependency logic outweighs the extra overhead imposed by these checks.

Enabling predictive dependency checks is easy:

- Predictive host dependency checks are controlled by the `enable_predictive_host_dependency_checks` option.
- Predictive service dependency checks are controlled by the `enable_predictive_service_dependency_checks` option.

Cached Checks

Predictive dependency checks are on-demand checks and are therefore subject to the rules of [cached checks](#). Cached checks can provide you with performance improvements by allowing Icinga to forgo running an actual host or service check if it can use a relatively recent check result instead. More information on cached checks can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

Adaptive Monitoring

[Home](#)

[Cached Checks](#)



Cached Checks

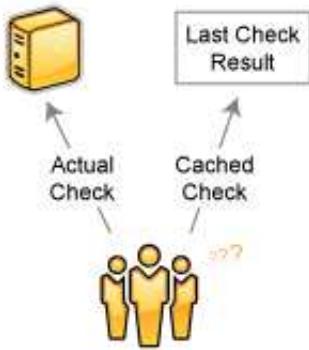
[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Cached Checks

Introduction



The performance of Icinga' monitoring logic can be significantly improved by implementing the use of cached checks. Cached checks allow Icinga to forgo executing a host or service check command if it determines a relatively recent check result will do instead.

For On-Demand Checks Only

Regularly scheduled host and service checks will not see a performance improvement with use of cached checks. Cached checks are only useful for improving the performance of on-demand host and service checks. Scheduled checks help to ensure that host and service states are updated regularly, which may result in a greater possibility their results can be used as cached checks in the future.

For reference, on-demand host checks occur...

- When a service associated with the host changes state.
- As needed as part of the [host reachability](#) logic.
- As needed for [predictive host dependency checks](#).

And on-demand service checks occur...

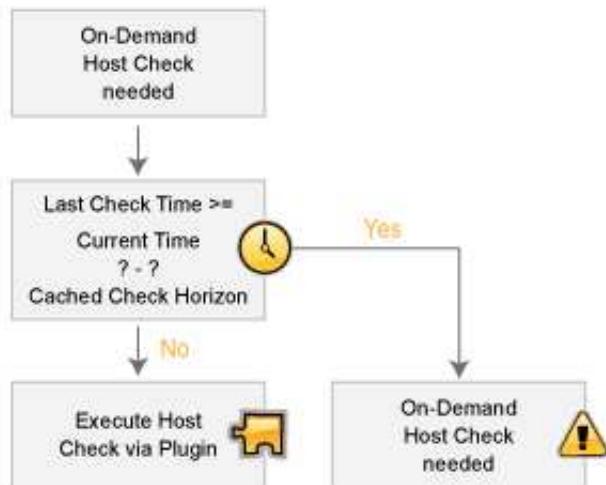
- As needed for [predictive service dependency checks](#).



Note

Unless you make use of service dependencies, Icinga will not be able to use cached check results to improve the performance of service checks. Don't worry about that - its normal. Cached host checks are where the big performance improvements lie, and everyone should see a benefit there.

How Caching Works



When Icinga needs to perform an on-demand host or service check, it will make a determination as to whether it can use a cached check result or if it needs to perform an actual check by executing a plugin. It does this by checking to see if the last check of the host or service occurred within the last X minutes, where X is the cached host or service check horizon.

If the last check was performed within the timeframe specified by the cached check horizon variable, Icinga will use the result of the last host or service check and will *not* execute a new check. If the host or service has not yet been checked, or if the last check falls outside of the cached check horizon timeframe, Icinga will execute a new host or service check by running a plugin.

What This Really Means

Icinga performs on-demand checks because it needs to know the current state of a host or service *at that exact moment* in time. Utilizing cached checks allows you to make Icinga think that recent check results are "good enough" for determining the current state of hosts, and that it doesn't need to go out and actually re-check the status of that host or service.

The cached check horizon tells Icinga how recent check results must be in order to reliably reflect the current state of a host or service. For example, with a cached check horizon of 30 seconds, you are telling Icinga that if a host's state was checked sometime in the last 30 seconds, the result of that check should still be considered the current state of the host.

The number of cached check results that Icinga can use versus the number of on-demand checks it has to actually execute can be considered the cached check "hit" rate. By increasing the cached check horizon to equal the regular check interval of a host, you could theoretically achieve a cache hit rate of 100%. In that case all on-demand checks of that host would use cached check results. What a performance improvement! But is it really? Probably not.

The reliability of cached check result information decreases over time. Higher cache hit rates require that previous check results are considered "valid" for longer periods of time. Things can change quickly in any network scenario, and there's no guarantee that a server that was functioning properly 30 seconds ago isn't on fire right now. There's the tradeoff - reliability versus speed. If you have a large cached check horizon, you risk having unreliable check result values being used in the monitoring logic.

Icinga will eventually determine the correct state of all hosts and services, so even if cached check results prove to unreliably represent their true value, Icinga will only work with incorrect information for a short period of time. Even short periods of unreliable status information can prove to be a nuisance for admins, as they may receive notifications about problems which no longer exist.

There is no standard cached check horizon or cache hit rate that will be acceptable to every Icinga user. Some people will want a short horizon timeframe and a low cache hit rate, while others will want a larger horizon timeframe and a larger cache hit rate (with a low reliability rate). Some users may even want to disable cached checks altogether to obtain a 100% reliability rate. Testing different horizon timeframes, and their effect on the reliability of status information, is the only way that an individual user will find the "right" value for their situation. More information on this is discussed below.

Configuration Variables

The following variables determine the timeframes in which a previous host or service check result may be used as a cached host or service check result:

- The [cached_host_check_horizon](#) variable controls cached host checks.
- The [cached_service_check_horizon](#) variable controls cached service checks.

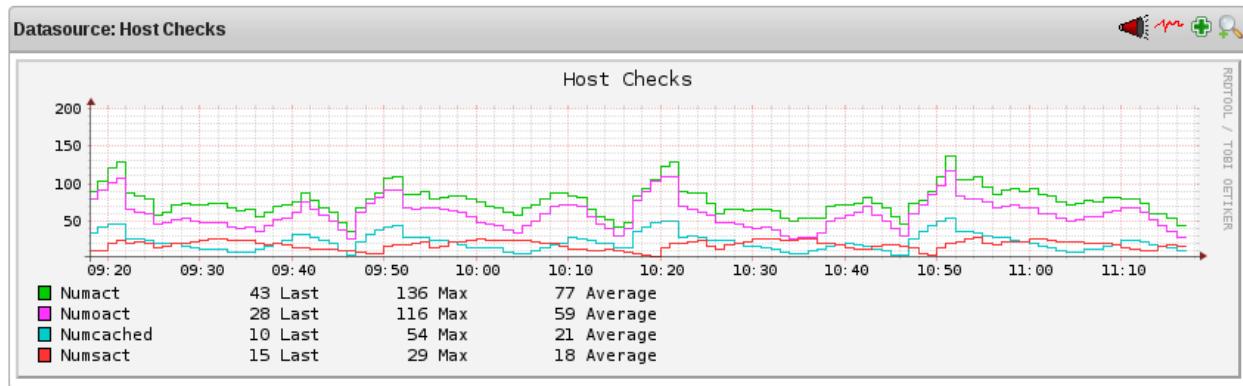
Optimizing Cache Effectiveness

In order to make the most effective use of cached checks, you should:

- Schedule regular checks of your hosts
- Use [PNP4Nagios](#) to graph statistics for 1) on-demand checks and 2) cached checks
- Adjust cached check horizon variables to fit your needs

You can schedule regular checks of your hosts by specifying a value greater than 0 for `check_interval` option in your [host definitions](#). If you do this, make sure that you set the `max_check_attempts` option to a value greater than 1, or it will cause a big performance hit. This potential performance hit is described in detail [here](#).

Figure 7.1. Cached checks



A good way to determine the proper value for the cached check horizon option is to compare how many on-demand checks Icinga has to actually run versus how many it can use cached values for. The [icingastats](#) utility can produce information on cached checks, which can then be [graphed with PNP4Nagios](#). An example graph that compares cached vs. actual on-demand checks is shown above.

The monitoring installation which produced the graphs had:

- A total of 110 hosts, all of which were checked at regular intervals
- An average (regularly scheduled) host check interval of 30 minutes with a 5 minute retry interval
- A [cached_host_check_horizon](#) of 15 seconds

The graph shows how many regularly scheduled host checks compared to how many cached host checks have occurred. In this example, an average of 77 host checks occur every five minutes. 59 of these (76%) are on-demand checks.

It also shows how many cached host checks have occurred over time. In this example an average of 21 cached host checks occurs every five minutes.

Remember, cached checks are only available for on-demand checks. Based on the 5 minute averages from the graphs, we see that Icinga is able to use cached host check results every 21 out of 59 times an on-demand check has to be run. That may not seem much, but these graphs represent a small monitoring environment. Consider that 21 out of 59 is nearly 36% and you can start to see how this could significantly help improve host check performance in large environments. That percentage could be higher if the cached host check horizon variable value was increased, but that would reduce the reliability of the cached host state information.

Once you've had a few hours or days worth of PNP4Nagios graphs, you should see how many host and service checks were done by executing plugins versus those that used cached check results. Use that information to adjust the cached check horizon variables appropriately for your situation. Continue to monitor the PNP4Nagios graphs over time to see how changing the horizon variables affected cached check statistics. Rinse and repeat as necessary.

[Prev](#)
[Up](#)
[Next](#)
[Predictive Dependency Checks](#)
[Home](#)
[Passive Host State Translation](#)



Passive Host State Translation

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

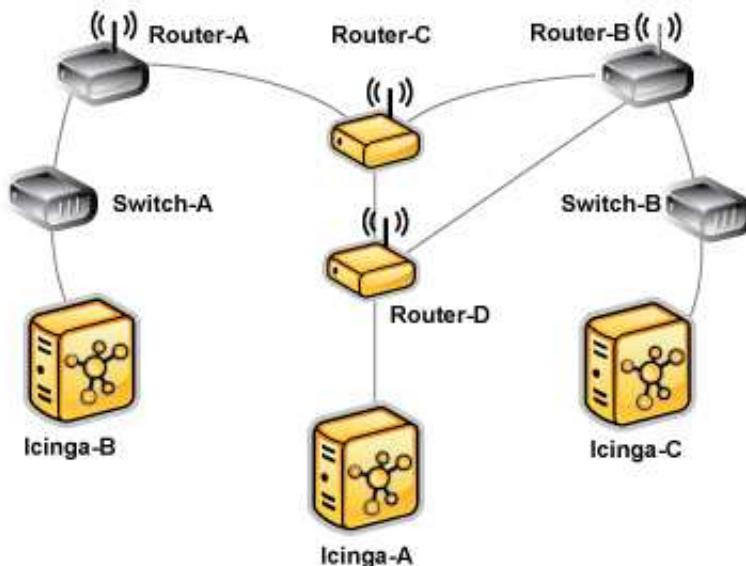
Passive Host State Translation

Introduction

When Icinga receives passive host checks from remote sources (i.e other Icinga instances in distributed or failover setups), the host state reported by the remote source may not accurately reflect the state of the host from Icinga' view. As distributed and failover monitoring installations are fairly common, it is important to provide a mechanism for ensuring accurate host states between different instances of Icinga.

Different World Views

The image below shows a simplified view of a failover monitoring setup.



- *Icinga-A* is the primary monitoring server, and is actively monitoring all switches and routers.
- *Icinga-B* and *Icinga-C* are backup monitoring servers, and are receiving passive check results from *Icinga-A*
- Both *Router-C* and *Router-D* have suffered failures and are offline.

What states are *Router-C* and *Router-D* currently in? The answer depends on which Icinga instance you ask.

- *Icinga-A* sees *Router-D* as DOWN and *Router-C* as UNREACHABLE
- *Icinga-B* should see *Router-C* as DOWN and *Router-D* as UNREACHABLE
- *Icinga-C* should see both routers as being DOWN.

Each Icinga instance has a different view of the network. The backup monitoring servers should not blindly accept passive host states from the primary monitoring server, or they will have incorrect information on the current state of the network.

Without translating passive host check results from the primary monitoring server (*Icinga-A*), *Icinga-C* would see *Router-D* as UNREACHABLE, when it is really DOWN based on its viewpoint. Similarly, the DOWN/UNREACHABLE states (from the viewpoint of *Icinga-A*) for *Router-C* and *Router-D* should be flipped from the viewpoint of *Icinga-B*.



Note

There may be some situations where you do not want Icinga to translate DOWN/UNREACHABLE states from remote sources to their "correct" state from the viewpoint of the local Icinga instance. For example, in distributed monitoring environments you may want the central Icinga instance to know how distributed instances see their respective portions of the network.

Enabling State Translation

By default, Icinga will *not* automatically translate DOWN/UNREACHABLE states from passive check results. You will need to enable this feature if you need and want it.

The automatic translation of passive host check states is controlled by the `translate_passive_host_checks` variable. Enable it and Icinga will automatically translate DOWN and UNREACHABLE states from remote sources to their correct state for the local instance of Icinga.

[Prev](#)

[Up](#)

[Next](#)

[Cached Checks](#)

[Home](#)

[Service and Host Check Scheduling](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Service and Host Check Scheduling

[Prev](#)[Chapter 7. Advanced Topics](#)[Next](#)

Service and Host Check Scheduling

Introduction

There were a lot of questions regarding how service checks are scheduled in certain situations, along with how the scheduling differs from when the checks are actually executed and their results are processed. We'll try to go into a little more detail on how this all works...

Configuration Options

Before we begin, there are several configuration options that affect how service checks are scheduled, executed, and processed. For starters, each [service definition](#) contains three options that determine when and how each specific service check is scheduled and executed. Those three options are:

- *check_interval*
- *retry_interval*
- *check_period*

There are also four configuration options in the [main configuration file](#) that affect service checks. These include:

- *service_inter_check_delay_method*
- *service_interleave_factor*
- *max_concurrent_checks*
- *check_result_reaper_frequency*



Note

The last directive affects host checks as well.

We'll go into more detail on how all these options affect service check scheduling as we progress. First off, let's see how services are initially scheduled when Icinga first starts or restarts...

Initial Scheduling

When Icinga (re)starts, it will attempt to schedule the initial check of all services in a manner that will minimize the load imposed on the local and remote hosts. This is done by spacing the initial service checks out, as well as interleaving them. The spacing of service checks (also known as the inter-check delay) is used to minimize/equalize the load on the local host running Icinga and the interleaving is used to minimize/equalize load imposed on remote hosts. Both the inter-check delay and interleave functions are discussed below.

Even though service checks are initially scheduled to balance the load on both the local and remote hosts, things will eventually give in to the ensuing chaos and be a bit random. Reasons for this include the fact that services are not all checked at the same interval, some services take longer to execute than others, host and/or service problems can alter the timing of one or more service checks, etc. At least we try to get things off to a good start. Hopefully the initial scheduling will keep the load on the local and remote hosts fairly balanced as time goes by...



Note

If you want to view the initial service check scheduling information, start Icinga using the `-s` command line option. Doing so will display basic scheduling information (inter-check delay, interleave factor, first and last service check time, etc) and will create a new status log that shows the exact time that all services are initially scheduled. Because this option will overwrite the status log, you should not use it when another copy of Icinga is running. Icinga does *not* start monitoring anything when this argument is used.

Inter-Check Delay

As mentioned before, Icinga attempts to equalize the load placed on the machine that is running Icinga by equally spacing out initial service checks. The spacing between consecutive service checks is called the inter-check delay. By giving a value to the `service_inter_check_delay_method` variable in the main config file, you can modify how this delay is calculated. We will discuss how the "smart" calculation works, as this is the setting you will want to use for normal operation.

When using the "smart" setting of the `service_inter_check_delay_method` variable, Icinga will calculate an inter-check delay value by using the following calculation:

$$\text{inter-check delay} = (\text{average check interval for all services}) / (\text{total number of services})$$

Let's take an example. Say you have 1,000 services that each have a normal check interval of 5 minutes (obviously some services are going to be checked at different intervals, but let's look at an easy case...). The total check interval time for all services is 5,000 (1,000 * 5). That means that the average check interval for each service is 5 minutes (5,000 / 1,000). Given that information, we realize that (on average) we need to re-check 1,000 services every 5 minutes. This means that we should use an inter-check delay of 0.005 minutes (0.3 seconds) when spacing out the initial service checks. By spacing each service check out by 0.3 seconds, we can somewhat guarantee that Icinga is scheduling and/or executing 3 new service checks every second. By spacing the checks out evenly over time like this, we can hope that the load on the local server that is running Icinga remains somewhat balanced.

Service Interleaving

As discussed above, the inter-check delay helps to equalize the load that Icinga imposes on the local host. What about remote hosts? Is it necessary to equalize load on remote hosts? Why? Yes, it is important and yes, Icinga can help out with this. If you monitor a large number of

services on a remote host and the checks were not spread out, the remote host might think that it was the victim of a SYN attack if there were a lot of open connections on the same port. Plus, attempting to equalize the load on hosts is just a nice thing to do...

By giving a value to the `service_interleave_factor` variable in the main config file, you can modify how the interleave factor is calculated. We will discuss how the "smart" calculation works, as this will probably be the setting you will want to use for normal operation. You can, however, use a pre-set interleave factor instead of having Icinga calculate one for you. Also of note, if you use an interleave factor of 1, service check interleaving is basically disabled.

When using the "smart" setting of the `service_interleave_factor` variable, Icinga will calculate an interleave factor by using the following calculation:

interleave factor = ceil (total number of services / total number of hosts)

Let's take an example. Say you have a total of 1,000 services and 150 hosts that you monitor. Icinga would calculate the interleave factor to be 7. This means that when Icinga schedules initial service checks it will schedule the first one it finds, skip the next 6, schedule the next one, and so on... This process will keep repeating until all service checks have been scheduled. Since services are sorted (and thus scheduled) by the name of the host they are associated with, this will help with minimizing/equalizing the load placed upon remote hosts.

The images below depict how service checks are scheduled when they are not interleaved (`service_interleave_factor=1`) and when they are interleaved with the `service_interleave_factor` variable equal to 4.

Notice how checks are scheduled consecutively, in the order they appear in the list. This is referred to as non-interleaved scheduling and is a

Notice how every 4th check in this list is consecutively scheduled. This is due to the fact that the automatically calculated service interval factor in this example was 4.

Notice how services are checked
recursively in the `getFirstCone`

Maximum Concurrent Service Checks

In order to prevent Icinga from consuming all of your CPU resources, you can restrict the maximum number of concurrent service checks that can be running at any given time. This is controlled by using the `max_concurrent_checks` option in the main config file.

The good thing about this setting is that you can regulate Icinga' CPU usage. The down side is that service checks may fall behind if this value is set too low. When it comes time to execute a service check, Icinga will make sure that no more than x service checks are either being executed or waiting to have their results processed (where x is the number of checks you specified for the `max_concurrent_checks` option). If that limit has been reached, Icinga will postpone the execution of any pending checks until some of the previous checks have completed. So how does one determine a reasonable value for the `max_concurrent_checks` option?

First off, you need to know the following things...

- The inter-check delay that Icinga uses to initially schedule service checks (use the `-s` command line argument to check this)
- The frequency (in seconds) of reaper events, as specified by the `check_result_reaper_frequency` variable in the main config file.
- A general idea of the average time that service checks actually take to execute (most plugins timeout after 10 seconds, so the average is probably going to be lower)

Next, use the following calculation to determine a reasonable value for the maximum number of concurrent checks that are allowed...

$$\text{max. concurrent checks} = \text{ceil}(\text{check result reaper frequency} , \text{average check execution time}) / \text{inter-check delay}$$

The calculated number should provide a reasonable starting point for the `max_concurrent_checks` variable. You may have to increase this value a bit if service checks are still falling behind schedule or decrease it if Icinga is hogging too much CPU time.

Let's say you are monitoring 875 services, each with an average check interval of 2 minutes. That means that your inter-check delay is going to be 0.137 seconds. If you set the check result reaper frequency to be 10 seconds, you can calculate a rough value for the max. number of concurrent checks as follows (we'll assume that the average execution time for service checks is less than 10 seconds) ...

$$\text{max. concurrent checks} = \text{ceil}(10 / 0.137)$$

In this case, the calculated value is going to be 73. This makes sense because (on average) Icinga are going to be executing just over 7 new service checks per second and it only processes service check results every 10 seconds. That means at given time there will be a just over 70 service checks that are either being executed or waiting to have their results processed. In this case, we would probably recommend bumping the max. concurrent checks value up to 80, since there will be delays when Icinga processes service check results and does its other work. Obviously, you're going to have test and tweak things a bit to get everything running smoothly on your system, but hopefully this provided some general guidelines...

Time Restraints

The `check_period` option determines the [time period](#) during which Icinga can run checks of the service. Regardless of what status a particular service is in, if the time that it is actually executed is not a valid time within the time period that has been specified, the check will *not* be executed. Instead, Icinga will reschedule the service check for the next valid time in the time period. If the check can be run (e.g. the time is valid within the time period), the service check is executed.



Note

Even though a service check may not be able to be executed at a given time, Icinga may still *schedule* it to be run at that time. This is most likely to happen during the initial scheduling of services, although it may happen in other instances as well. This does *not* mean that Icinga will execute the check! When it comes time to actually *execute* a service check, Icinga will verify that the check can be run at the current time. If it cannot, Icinga will not execute the service check, but will instead just reschedule it for a later time. Don't let this one throw you confuse you! The scheduling and execution of service checks are two distinctly different (although related) things.

Normal Scheduling

In an ideal world you wouldn't have network problems. But if that were the case, you wouldn't need a network monitoring tool. Anyway, when things are running smoothly and a service is in an OK state, we'll call that "normal". Service checks are normally scheduled at the frequency specified by the *check_interval* option. That's it. Simple, huh?

Scheduling During Problems

So what happens when there are problems with a service? Well, one of the things that happens is the service check scheduling changes. If you've configured the *max_attempts* option of the service definition to be something greater than 1, Icinga will recheck the service before deciding that a real problem exists. While the service is being rechecked (up to *max_attempts* times) it is considered to be in a "soft" state (as described [here](#)) and the service checks are rescheduled at a frequency determined by the *retry_interval* option.

If Icinga rechecks the service *max_attempts* times and it is still in a non-OK state, Icinga will put the service into a "hard" state, send out notifications to contacts (if applicable), and start rescheduling future checks of the service at a frequency determined by the *check_interval* option.

As always, there are exceptions to the rules. When a service check results in a non-OK state, Icinga will check the host that the service is associated with to determine whether or not it is up (see the note [below](#) for info on how this is done). If the host is not up (i.e. it is either down or unreachable), Icinga will immediately put the service into a hard non-OK state and it will reset the current attempt number to 1. Since the service is in a hard non-OK state, the service check will be rescheduled at the normal frequency specified by the *check_interval* option instead of the *retry_interval* option.

Host Checks

Unlike service checks, host checks are *not* scheduled on a regular basis. Instead they are run on demand, as Icinga sees a need. This is a common question asked by users, so it needs to be clarified.

One instance where Icinga checks the status of a host is when a service check results in a non-OK status. Icinga checks the host to decide whether or not the host is up, down, or unreachable. If the first host check returns a non-OK state, Icinga will keep pounding out checks of the host until either (a) the maximum number of host checks (specified by the *max_attempts* option in the host definition) is reached or (b) a host check results in an OK state.

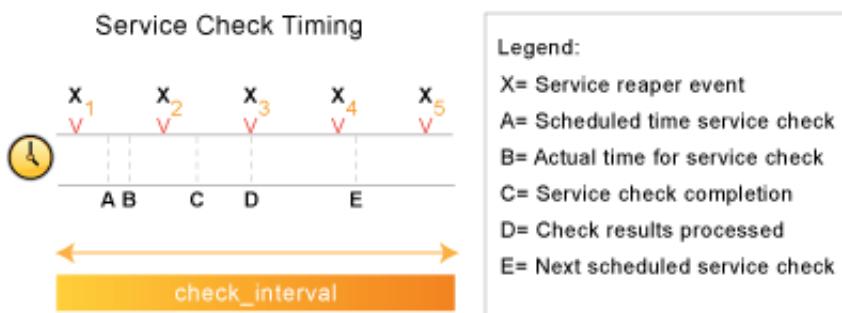
Also of note - when Icinga is checking the status of a host, it holds off on doing anything else (executing new service checks, processing other service check results, etc). This can slow things down a bit and cause pending service checks to be delayed for a while, but it is necessary to determine the status of the host before Icinga can take any further action on the service(s) that are having problems.

Scheduling Delays

It should be noted that service check scheduling and execution is done on a best effort basis. Individual service checks are considered to be low priority events in Icinga, so they can get delayed if high priority events need to be executed. Examples of high priority events include log file rotations, external command checks, and check results reaper events. Additionally, host checks will slow down the execution and processing of service checks.

Scheduling Example

The scheduling of service checks, their execution, and the processing of their results can be a bit difficult to understand, so let's look at a simple example. Look at the diagram below - we'll refer to it as we explain how things are done.



First off, the X_n events are check result reaper events that are scheduled at a frequency specified by the [check_result_reaper_frequency](#) option in the main config file. Check result reaper events do the work of gathering and processing service check results. They serve as the core logic for Icinga, kicking off host checks, event handlers and notifications as necessary.

For the example here, a service has been scheduled to be executed at time A. However, Icinga got behind in its event queue, so the check was not actually executed until time B. The service check finished executing at time C, so the difference between points C and B is the actual amount of time that the check was running.

The results of the service check are not processed immediately after the check is done executing. Instead, the results are saved for later processing by a check result reaper event. The next check result reaper event occurs at time D, so that is approximately the time that the results are processed (the actual time may be later than D since other service check results may be processed before this one).

At the time that the check result reaper event processes the service check results, it will reschedule the next service check and place it into Icinga' event queue. We'll assume that the service check resulted in an OK status, so the next check at time E is scheduled after the originally scheduled check time by a length of time specified by the *check_interval* option. Note that the service is *not* rescheduled based off the time that it was actually executed! There is one exception to this (isn't there always?) - if the time that the service check is actually executed (point B) occurs after the next service check time (point E), Icinga will compensate by adjusting the next check time. This is done to ensure that Icinga doesn't go nuts trying to keep up with service checks if it comes under heavy load. Besides, what's the point of scheduling something in the past...?

Service Definition Options That Affect Scheduling

Each service definition contains a *check_interval* and *retry_interval* option. Hopefully this will clarify what these two options do, how they relate to the *max_check_attempts* option in the service definition, and how they affect the scheduling of the service.

First off, the *check_interval* option is the interval at which the service is checked under "normal" circumstances. "Normal" circumstances mean whenever the service is in an OK state or when its in a **hard** non-OK state.

When a service first changes from an OK state to a non-OK state, Icinga gives you the ability to temporarily slow down or speed up the interval at which subsequent checks of that service will occur. When the service first changes state, Icinga will perform up to *max_check_attempts*-1 retries of the service check before it decides its a real problem. While the service is being retried, it is scheduled according to the *retry_interval* option, which might be faster or slower than the normal *check_interval* option. While the service is being rechecked (up to *max_check_attempts*-1 times), the service is in a **soft state**. If the service is rechecked *max_check_attempts*-1 times and it is still in a non-OK state, the service turns into a **hard state** and is subsequently rescheduled at the normal rate specified by the *check_interval* option.

On a side note, if you specify a value of 1 for the *max_check_attempts* option, the service will not ever be checked at the interval specified by the *retry_interval* option. Instead, it immediately turns into a **hard state** and is subsequently rescheduled at the rate specified by the *check_interval* option.

TODO

Host Check Directives

Most of the above applies to host checks as well.

This documentation is being rewritten. Stay tuned for more information in a later beta release...

[Prev](#)

[Up](#)

[Next](#)

[Passive Host State Translation](#)

[Home](#)

[Custom CGI Headers and Footers](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Custom CGI Headers and Footers

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

Custom CGI Headers and Footers

Introduction

If you're doing custom installs of Icinga for clients, you may want to have a custom header and/or footer displayed in the output of the [CGIs](#). This is particularly useful for displaying support contact information, etc. to the end user.

It is important to note that, unless they are executable, custom header and footer files are not pre-processed in any way before they are displayed. The contents of the header and footer include files are simply read and displayed in the CGI output. That means they can only contain information a web browser can understand (HTML, JavaScript, etc.).

If the custom header and footer files are executable, then the files are executed and their output returned to the user, so they should output valid HTML. Using this you can run your own custom designed CGI to insert data into the Icinga display. This has been used to insert graphs from rrdtool using ddraw and command menus into the Icinga display pane. The executable customer header and footer files are run with the same CGI environment as the main Icinga CGI, so your files can parse the query information, authenticated user information, etc. to produce appropriate output.

How Does It Work?

You can include custom headers and footers in the output of the CGIs by dropping some appropriately named HTML files in the `ssi/` subdirectory of the Icinga HTML directory (i.e. `/usr/local/icinga/share/ssi`).

Custom headers are included immediately after the `<BODY>` tag in the CGI output, while custom footers are included immediately before the closing `</BODY>` tag.

There are two types of customer headers and footers:

- Global headers/footers. These files should be named `common-header.ssi` and `common-footer.ssi`, respectively. If these files exist, they will be included in the output of all CGIs.
- CGI-specific headers/footers. These files should be named in the format `CGINAME-header.ssi` and `CGINAME-footer.ssi`, where `CGINAME` is the physical name of the CGI without the `.cgi` extension. For example, the header and footer files for the [alert summary CGI](#) (`summary.cgi`) would be named `summary-header.ssi` and `summary-footer.ssi`, respectively.

You are not required to use any custom headers or footers. You can use only a global header if you wish. You can use only CGI-specific headers and a global footer if you wish. Whatever you want. Really.

[Prev](#)[Up](#)[Next](#)[Service and Host Check
Scheduling](#)[Home](#)[Object Inheritance](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Object Inheritance

[Prev](#)

Chapter 7. Advanced Topics

[Next](#)

Object Inheritance

Introduction

This documentation attempts to explain object inheritance and how it can be used in your [object definitions](#).

If you are confused about how recursion and inheritance work after reading this, take a look at the sample object config files provided in the Icinga distribution. If that still doesn't help, drop an email message with a *detailed* description of your problem to the *icinga-users* mailing list.

Basics

There are three variables affecting recursion and inheritance that are present in all object definitions. They are "*indicated*" as follows...

```
define someobjecttype{
    object-specific variables ...
    name           template_name
    use            name_of_template_to_use
    register       [0/1]
}
```

The first variable is *name*. Its just a "template" name that can be referenced in other object definitions so they can inherit the objects properties/variables. Template names must be unique amongst objects of the same type, so you can't have two or more host definitions that have "hosttemplate" as their template name.

The second variable is *use*. This is where you specify the name of the template object that you want to inherit properties/variables from. The name you specify for this variable must be defined as another object's template named (using the *name* variable).

The third variable is *register*. This variable is used to indicate whether or not the object definition should be "registered" with Icinga. By default, all object definitions are registered. If you are using a partial object definition as a template, you would want to prevent it from being registered (an example of this is provided later). Values are as follows: 0 = do NOT register object definition, 1 = register object definition (this is the default). This variable is NOT inherited; every (partial) object definition used as a template must explicitly set the *register* directive to be 0. This prevents the need to override an inherited *register* directive with a value of 1 for every object that should be registered.

Local Variables vs. Inherited Variables

One important thing to understand with inheritance is that "local" object variables always take precedence over variables defined in the template object. Take a look at the following example of two host definitions (not all required variables have been supplied):

```
define host{
    host_name          bighost1
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 5

    name               hosttemplate1
}

define host{
    host_name          bighost2
    max_check_attempts 3

    use                hosttemplate1
}
```

You'll note that the definition for host *bighost1* has been defined as having *hosttemplate1* as its template name. The definition for host *bighost2* is using the definition of *bighost1* as its template object. Once Icinga processes this data, the resulting definition of host *bighost2* would be equivalent to this definition:

```
define host{
    host_name          bighost2
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 3
}
```

You can see that the *check_command* and *notification_options* variables were inherited from the template object (where host *bighost1* was defined). However, the *host_name* and *max_check_attempts* variables were not inherited from the template object because they were defined locally. Remember, locally defined variables override variables that would normally be inherited from a template object. That should be a fairly easy concept to understand.



Tip

If you would like local string variables to be appended to inherited string values, you can do so. Read more about how to accomplish this [below](#).

Inheritance Chaining

Objects can inherit properties/variables from multiple levels of template objects. Take the following example:

```
define host{
    host_name          bighost1
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 5

    name               hosttemplate1
}

define host{
    host_name          bighost2
    max_check_attempts 3
```

```

use          hosttemplate1

name         hosttemplate2
}

define host{
host_name    bighost3

use          hosttemplate2
}

```

You'll notice that the definition of host *bighost3* inherits variables from the definition of host *bighost2*, which in turn inherits variables from the definition of host *bighost1*. Once Icinga processes this configuration data, the resulting host definitions are equivalent to the following:

```

define host{
host_name    bighost1
check_command check-host-alive
notification_options d,u,r
max_check_attempts 5
}

define host{
host_name    bighost2
check_command check-host-alive
notification_options d,u,r
max_check_attempts 3
}

define host{
host_name    bighost3
check_command check-host-alive
notification_options d,u,r
max_check_attempts 3
}

```

There is no inherent limit on how "deep" inheritance can go, but you'll probably want to limit yourself to at most a few levels in order to maintain sanity.

Using Incomplete Object Definitions as Templates

It is possible to use incomplete object definitions as templates for use by other object definitions. By "incomplete" definition, I mean that all required variables in the object have not been supplied in the object definition. It may sound odd to use incomplete definitions as templates, but it is in fact recommended that you use them. Why? Well, they can serve as a set of defaults for use in all other object definitions. Take the following example:

```

define host{
check_command      check-host-alive
notification_options d,u,r
max_check_attempts 5

name               generichosttemplate

register           0
}

define host{
host_name          bighost1
address            192.168.1.3

use                generichosttemplate
}

```

```
define host{
    host_name          bighost2
    address            192.168.1.4

    use                generichosthosttemplate
}
```

Notice that the first host definition is incomplete because it is missing the required `host_name` variable. We don't need to supply a host name because we just want to use this definition as a generic host template. In order to prevent this definition from being registered with Icinga as a normal host, we set the `register` variable to 0.

The definitions of hosts `bighost1` and `bighost2` inherit their values from the generic host definition. The only variable we've chosen to override is the `address` variable. This means that both hosts will have the exact same properties, except for their `host_name` and `address` variables. Once Icinga processes the config data in the example, the resulting host definitions would be equivalent to specifying the following:

```
define host{
    host_name          bighost1
    address            192.168.1.3
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 5
}

define host{
    host_name          bighost2
    address            192.168.1.4
    check_command      check-host-alive
    notification_options d,u,r
    max_check_attempts 5
}
```

At the very least, using a template definition for default variables will save you a lot of typing. It'll also save you a lot of headaches later if you want to change the default values of variables for a large number of hosts.

Custom Object Variables

Any [custom object variables](#) that you define in your host, service, or contact definition templates will be inherited just like other standard variables. Take the following example:

```
define host{
    _customvar1        somevalue ; <-- Custom host variable
    _snmp_community   public  ; <-- Custom host variable

    name              generichosttemplate

    register          0
}

define host{
    host_name          bighost1
    address            192.168.1.3

    use                generichosthosttemplate
}
```

The host *bighost1* will inherit the custom host variables *_customvar1* and *_snmp_community*, as well as their respective values, from the *generichosttemplate* definition. The effective result is a definition for *bighost1* that looks like this:

```
define host{
    host_name          bighost1
    address            192.168.1.3
    _customvar1        somevalue
    _snmp_community   public
}
```

Cancelling Inheritance of String Values

In some cases you may not want your host, service, or contact definitions to inherit values of string variables from the templates they reference. If this is the case, you can specify "null" (without quotes) as the value of the variable that you do not want to inherit. Take the following example:

```
define host{
    event_handler      my-event-handler-command

    name               generichosttemplate

    register           0
}

define host{
    host_name          bighost1
    address            192.168.1.3
    event_handler      null

    use                generichosthosttemplate
}
```

In this case, the host *bighost1* will not inherit the value of the *event_handler* variable that is defined in the *generichosttemplate*. The resulting effective definition of *bighost1* is the following:

```
define host{
    host_name          bighost1
    address            192.168.1.3
}
```

Additive Inheritance of String Values

Icinga gives preference to local variables instead of values inherited from templates. In most cases local variable values override those that are defined in templates. In some cases it makes sense to allow Icinga to use the values of inherited *and* local variables together.

This "additive inheritance" can be accomplished by prepending the local variable value with a plus sign (+). This features is only available for standard (non-custom) variables that contain string values. Take the following example:

```
define host{
    hostgroups         all-servers

    name               generichosttemplate

    register           0
}

define host{
    host_name          linuxserver1
```

```

hostgroups          +linux-servers,web-servers
use                generichosthosttemplate
}

```

In this case, the host *linuxserver1* will append the value of its local *hostgroups* variable to that from *generichosthosttemplate*. The resulting effective definition of *linuxserver1* is the following:

```

define host{
    host_name          linuxserver1
    hostgroups         all-servers,linux-servers,web-servers
}

```

Implied Inheritance

Normally you have to either explicitly specify the value of a required variable in an object definition or inherit it from a template. There are a few exceptions to this rule, where Icinga will assume that you want to use a value that instead comes from a related object. For example, the values of some service variables will be copied from the host the service is associated with if you don't otherwise specify them.

The following table lists the object variables that will be implicitly inherited from related objects if you don't explicitly specify their value in your object definition or inherit them from a template.

Object Type	Object Variable	Implied Source
Services	<i>contact_groups</i>	<i>contact_groups</i> in the associated host definition
	<i>notification_interval</i>	<i>notification_interval</i> in the associated host definition
	<i>notification_period</i>	<i>notification_period</i> in the associated host definition
Host Escalations	<i>contact_groups</i>	<i>contact_groups</i> in the associated host definition
	<i>notification_interval</i>	<i>notification_interval</i> in the associated host definition
	<i>escalation_period</i>	<i>notification_period</i> in the associated host definition
Service Escalations	<i>contact_groups</i>	<i>contact_groups</i> in the associated service definition
	<i>notification_interval</i>	<i>notification_interval</i> in the associated service definition
	<i>escalation_period</i>	<i>notification_period</i> in the associated service definition

Implied/Additive Inheritance in Escalations

Service and host escalation definitions can make use of a special rule that combines the features of implied and additive inheritance. If escalations 1) do not inherit the values of their *contact_groups* or *contacts* directives from another escalation template and 2) their *contact_groups* or *contacts* directives begin with a plus sign (+), then the values of their corresponding host or service definition's *contact_groups* or *contacts* directives will be used in the additive inheritance logic.

Confused? Here's an example:

```

define host{
    name          linux-server
    contact_groups linux-admins
    ...
}

define hostescalation{
    host_name      linux-server
    contact_groups +management
    ...
}

```

This is a much simpler equivalent to:

```

define hostescalation{
    host_name      linux-server
    contact_groups linux-admins,management
    ...
}

```

Important values

Service templates can make use of a special rule which gives precedence to their check_command value. If the check_command is prefixed with an exclamation mark (!), then the template's check_command is marked as important and will be used over the check_command defined for the service (this is styled after CSS syntax, which uses ! as an important attribute).

Why is this useful? It is mainly useful when setting a different check_command for distributed systems. You may want to set a freshness threshold and a check_command that forces the service into a failed state, but this doesn't work with the normal templating system. Using this important flag allows the custom check_command to be written, but a general distributed template can be used to overrule the check_command when used on a central Icinga-erver.

For instance:

```

# On master
define service {
    name          service-distributed
    register      0
    active_checks_enabled 0
    check_freshness 1
    check_command   !set_to_stale
}
# On slave
define service {
    name          service-distributed
    register      0
    active_checks_enabled 1
}
# Service definition, used by master and slave
define service {
    host_name      host1
    service_description serviceA
    check_command   check_http...
    use           service-distributed
    ...
}

```



Note

Please note that only one level of inheritance is possible using important values. That means that you cannot inherit the `check_command` from one template to another and from the latter to a service.

```
Template1 => Service1           <== will work
Template1 => Template2 => Service1 <== will NOT work
```

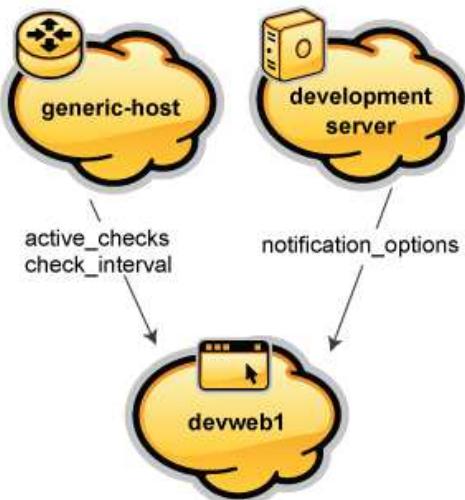
Multiple Inheritance Sources

Thus far, all examples of inheritance have shown object definitions inheriting variables/values from just a single source. You are also able to inherit variables/values from multiple sources for more complex configurations, as shown below.

```
# Generic host template
define host{
    name          generic-host
    active_checks_enabled 1
    check_interval 10
    ...
    register      0
}

# Development web server template
define host{
    name          development-server
    check_interval 15
    notification_options d,u,r
    ...
    register      0
}

# Development web server
define host{
    use           generic-host,development-server
    host_name     devweb1
    ...
}
```



In the example above, `devweb1` is inheriting variables/values from two sources: `generic-host` and `development-server`. You'll notice that a `check_interval` variable is defined in both sources. Since `generic-host` was the first template specified in `devweb1`'s `use` directive, its value for the `check_interval` variable is inherited by the `devweb1` host. After inheritance, the effective definition of `devweb1` would be as follows:

```
# Development web server
define host{
    host_name     devweb1
    active_checks_enabled 1
    check_interval 10
    notification_options d,u,r
    ...
}
```

Precedence With Multiple Inheritance Sources

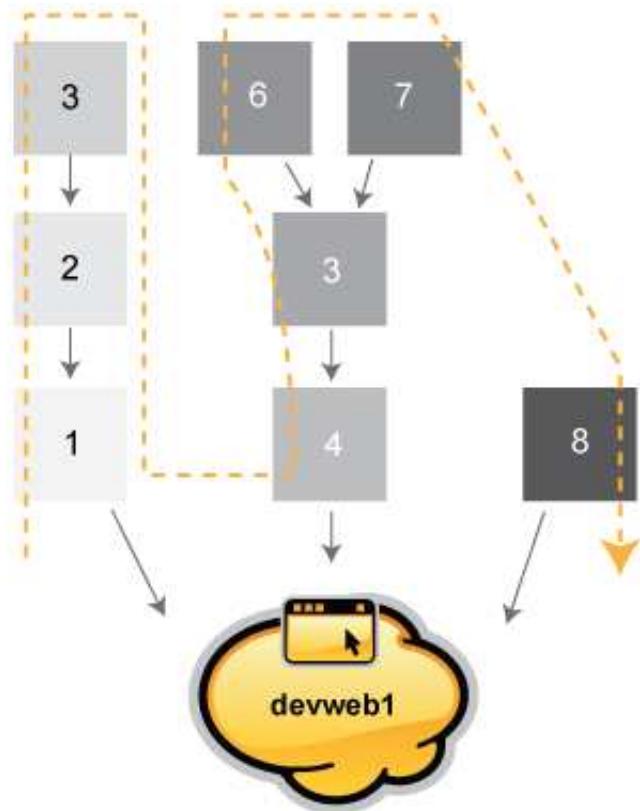
When you use multiple inheritance sources, it is important to know how Icinga handles variables that are defined in multiple sources. In these cases Icinga will use the variable/value from the first source that is specified in the `use` directive. Since inheritance sources can themselves inherit variables/values from one or more other sources, it can get tricky to figure out what variable/value pairs take precedence.

Consider the following host definition that references three templates:

```
# Development web server
define host{
    use      1, 4, 8
    host_name devweb1 ...
}
```

If some of those referenced templates themselves inherit variables/values from one or more other templates, the precedence rules are shown to the right.

Testing, trial, and error will help you better understand exactly how things work in complex inheritance situations like this. :-)


[Prev](#)
[Up](#)
[Next](#)
[Custom CGI Headers and Footers](#)
[Home](#)
[Time-Saving Tricks For Object Definitions](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Time-Saving Tricks For Object Definitions

[Prev](#)
[Chapter 7. Advanced Topics](#)
[Next](#)

Time-Saving Tricks For Object Definitions

Introduction

This documentation attempts to explain how you can exploit the (somewhat) hidden features of [template-based object definitions](#) to save your sanity. How so, you ask? Several types of objects allow you to specify multiple host names and/or hostgroup names in definitions, allowing you to "copy" the object definition to multiple hosts or services. We'll cover each type of object that supports these features separately. For starters, the object types which support this time-saving feature are as follows:

- [Services](#)
- [Service escalations](#)
- [Service dependencies](#)
- [Host escalations](#)
- [Host dependencies](#)
- [Hostgroups](#)

Object types that are not listed above (i.e. timeperiods, commands, etc.) do not support the features we're about to describe.

Regular Expression Matching

The examples we give below use "standard" matching of object names. If you wish, you can enable regular expression matching for object names by using the [use_regexp_matching](#) config option. By default, regular expression matching will only be used in object names that contain *, ?, +, or \.. If you want regular expression matching to be used on all object names, enable the [use_true_regexp_matching](#) config option. Regular expressions can be used in any of the fields used in the examples below (host names, hostgroup names, service names, and servicegroup names).



Note

Be careful when enabling regular expression matching - you may have to change your config file, since some directives that you might not want to be interpreted as a regular expression just might be! Any problems should become evident once you verify your configuration.

Service Definitions

Multiple Hosts:

If you want to create identical [services](#) that are assigned to multiple hosts, you can specify multiple hosts in the *host_name* directive. The definition below would create a service called *SOMESERVICE* on hosts *HOST1* through *HOSTN*. All the instances of the *SOMESERVICE* service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    host_name           HOST1,HOST2,HOST3,...,HOSTN
    service_description SOMESERVICE
    other service directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create identical services that are assigned to all hosts in one or more hostgroups, you can do so by creating a single service definition. How? The *hostgroup_name* directive allows you to specify the name of one or more hostgroups that the service should be created for. The definition below would create a service called *SOMESERVICE* on all hosts that are members of hostgroups *HOSTGROUP1* through *HOSTGROUPN*. All the instances of the *SOMESERVICE* service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description SOMESERVICE
    other service directives ...
}
```

All Hosts:

If you want to create identical services that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the *host_name* directive. The definition below would create a service called *SOMESERVICE* on **all hosts** that are defined in your configuration files. All the instances of the *SOMESERVICE* service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    host_name          *
    service_description SOMESERVICE
    other service directives ...
}
```

Excluding Hosts:

If you want to create identical services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define service{
    host_name           HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description SOMESERVICE
    other service directives ...
}
```

Service Escalation Definitions

Multiple Hosts:

If you want to create [service escalations](#) for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the *host_name* directive. The definition below would create a service escalation for services called *SOMESERVICE* on hosts *HOST1* through *HOSTN*. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name           HOST1,HOST2,HOST3,...,HOSTN
    service_description SOMESERVICE
    other escalation directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create service escalations for services of the same name/description that are assigned to all hosts in one or more hostgroups, you can do use the *hostgroup_name* directive. The definition below would create a service escalation for services called *SOMESERVICE* on all hosts that are members of hostgroups *HOSTGROUP1* through *HOSTGROUPN*. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description SOMESERVICE
    other escalation directives ...
}
```

All Hosts:

If you want to create identical service escalations for services of the same name/description that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the *host_name* directive. The definition below would create a service escalation for all services called *SOMESERVICE* on **all hosts** that are defined in your configuration files. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name          *
    service_description SOMESERVICE
    other escalation directives ...
}
```

Excluding Hosts:

If you want to create identical services escalations for services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define serviceescalation{
    host_name           HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name      HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description SOMESERVICE
    other escalation directives ...
}
```

All Services On Same Host:

If you want to create **service escalations** for all services assigned to a particular host, you can use a wildcard in the *service_description* directive. The definition below would create a service escalation for **all** services on host *HOST1*. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

If you feel like being particularly adventurous, you can specify a wildcard in both the *host_name* and *service_description* directives. Doing so would create a service escalation for **all services** that you've defined in your configuration files.

```
define serviceescalation{
    host_name           HOST1
    service_description *
    other escalation directives ...
}
```

Multiple Services On Same Host:

If you want to create **service escalations** for all multiple services assigned to a particular host, you can use a specify more than one service description in the *service_description* directive. The definition below would create a service escalation for services *SERVICE1* through *SERVICEN* on host *HOST1*. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name           HOST1
    service_description SERVICE1,SERVICE2,...,SERVICEN
    other escalation directives ...
}
```

All Services In Multiple Servicegroups:

If you want to create service escalations for all services that belong in one or more servicegroups, you can do use the *servicegroup_name* directive. The definition below would create service escalations for all services that are members of servicegroups *SERVICEGROUP1* through *SERVICEGROUPN*. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    servicegroup_name      SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    other escalation directives ...
}
```

Service Dependency Definitions

Multiple Hosts:

If you want to create **service dependencies** for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the *host_name* and or *dependent_host_name* directives. In the example below, service *SERVICE2* on hosts *HOST3* and *HOST4* would be dependent on service *SERVICE1* on hosts *HOST1* and *HOST2*. All the instances of the service dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    host_name           HOST1,HOST2
    service_description SERVICE1
    dependent_host_name HOST3,HOST4
    dependent_service_description SERVICE2
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create service dependencies for services of the same name/description that are assigned to all hosts in one or more hostgroups, you can do use the *hostgroup_name* and/or *dependent_hostgroup_name* directives. In the example below, service *SERVICE2* on all hosts in hostgroups *HOSTGROUP3* and *HOSTGROUP4* would be dependent on service *SERVICE1* on all hosts in hostgroups *HOSTGROUP1* and *HOSTGROUP2*. Assuming there were five hosts in each of the hostgroups, this definition would be equivalent to creating 100 single service dependency definitions! All the instances of the service dependency would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    hostgroup_name        HOSTGROUP1,HOSTGROUP2
    service_description   SERVICE1
    dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
    dependent_service_description SERVICE2
    other dependency directives ...
}
```

All Services On A Host:

If you want to create service dependencies for all services assigned to a particular host, you can use a wildcard in the *service_description* and/or *dependent_service_description* directives. In the example below, **all services** on host *HOST2* would be dependent on **all services** on host *HOST1*. All the instances of the service dependencies would be identical (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    host_name           HOST1
    service_description *
    dependent_host_name HOST2
    dependent_service_description *
    other dependency directives ...
}
```

Multiple Services On A Host:

If you want to create service dependencies for multiple services assigned to a particular host, you can specify more than one service description in the *service_description* and/or *dependent_service_description* directives as follows:

```
define servicedependency{
    host_name           HOST1
    service_description SERVICE1,SERVICE2,...,SERVICEN
    dependent_host_name HOST2
    dependent_service_description SERVICE1,SERVICE2,...,SERVICEN
    other dependency directives ...
}
```

All Services In Multiple Servicegroups:

If you want to create service dependencies for all services that belong in one or more servicegroups, you can do use the *servicegroup_name* and/or *dependent_servicegroup_name* directive as follows:

```
define servicedependency{
    servicegroup_name           SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    dependent_servicegroup_name SERVICEGROUP3,SERVICEGROUP4,...SERVICEGROUPN
    other dependency directives ...
}
```

Same Host Dependencies:

If you want to create service dependencies for multiple services that are dependent on services on the same host, leave the *dependent_host_name* and *dependent_hostgroup_name* directives empty. The example below assumes that hosts *HOST1* and *HOST2* have at least the following four services associated with them: *SERVICE1*, *SERVICE2*, *SERVICE3*, and *SERVICE4*. In this example, *SERVICE3* and *SERVICE4* on *HOST1* will be dependent on both *SERVICE1* and *SERVICE2* on *HOST1*. Similarly, *SERVICE3* and *SERVICE4* on *HOST2* will be dependent on both *SERVICE1* and *SERVICE2* on *HOST2*.

```
define servicedependency{
    host_name                  HOST1,HOST2
    service_description         SERVICE1,SERVICE2
    dependent_service_description SERVICE3,SERVICE4
    other dependency directives ...
}
```

Same Host Dependencies With Servicegroups:

If you want to create service dependencies for all services that belong to one or more servicegroups of a service on the same host running the dependent service, leave the *host_name* and *hostgroup_name* directives empty. The example below assumes that hosts running services belonging to *SERVICEGROUP1* and *SERVICEGROUP2* have the following service associated with them: *SERVICE1*. In this example, all services belonging to *SERVICEGROUP1* and *SERVICEGROUP2* will be dependent on *SERVICE1* on the same host running the dependent service.

```
define servicedependency{
    service_description          SERVICE1
    dependent_service_description SERVICEGROUP1,SERVICEGROUP2
    other dependency directives ...
}
```

Host Escalation Definitions

Multiple Hosts:

If you want to create [host escalations](#) for multiple hosts, you can specify multiple hosts in the *host_name* directive. The definition below would create a host escalation for hosts *HOST1* through *HOSTN*. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    host_name                  HOST1,HOST2,HOST3,...,HOSTN
    other escalation directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create host escalations for all hosts in one or more hostgroups, you can do use the *hostgroup_name* directive. The definition below would create a host escalation on all hosts that are members of hostgroups *HOSTGROUP1* through *HOSTGROUPN*. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    hostgroup_name           HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    other escalation directives ...
}
```

All Hosts:

If you want to create identical host escalations for all hosts that are defined in your configuration files, you can use a wildcard in the *host_name* directive. The definition below would create a hosts escalation for **all hosts** that are defined in your configuration files. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    host_name               *
    other escalation directives ...
}
```

Excluding Hosts:

If you want to create identical host escalations on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define hostescalation{
    host_name               HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name          HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    other escalation directives ...
}
```

Host Dependency Definitions

Multiple Hosts:

If you want to create **host dependencies** for multiple hosts, you can specify multiple hosts in the *host_name* and/or *dependent_host_name* directives. The definition below would be equivalent to creating six separate host dependencies. In the example above, hosts *HOST3*, *HOST4* and *HOST5* would be dependent upon both *HOST1* and *HOST2*. All the instances of the host dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define hostdependency{
    host_name               HOST1,HOST2
    dependent_host_name     HOST3,HOST4,HOST5
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create host escalations for all hosts in one or more hostgroups, you can do use the *hostgroup_name* and /or *dependent_hostgroup_name* directives. In the example below, all hosts in hostgroups *HOSTGROUP3* and *HOSTGROUP4* would be dependent on all hosts in hostgroups *HOSTGROUP1* and *HOSTGROUP2*. All the instances of the host dependencies would be identical except for host names (i.e. have the same notification failure criteria, etc.).

```
define hostdependency{
    hostgroup_name           HOSTGROUP1,HOSTGROUP2
    dependent_hostgroup_name HOSTGROUP3,HOSTGROUP4
    other dependency directives ...
}
```

Hostgroups

All Hosts:

If you want to create a hostgroup that has all hosts that are defined in your configuration files as members, you can use a wildcard in the *members* directive. The definition below would create a hostgroup called *HOSTGROUP1* that has all **all hosts** that are defined in your configuration files as members.

```
define hostgroup{
    hostgroup_name           HOSTGROUP1
    members                  *
    other hostgroup directives ...
}
```

[Prev](#)
[Up](#)
[Next](#)
[Object Inheritance](#)
[Home](#)
[Chapter 8. Security and Performance Tuning](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 8. Security and Performance Tuning

[Prev](#)

[Next](#)

Chapter 8. Security and Performance Tuning

Table of Contents

- [Security Considerations](#)
 - [Enhanced CGI Security and Authentication](#)
 - [Tuning Icinga For Maximum Performance](#)
 - [Fast Startup Options](#)
 - [Large Installation Tweaks](#)
 - [Using The Icingastats Utility](#)
 - [Graphing Performance Info With PNP4Nagios](#)
 - [Temporary Data](#)
-

[Prev](#)

[Next](#)

[Time-Saving Tricks For Object Definitions](#)

[Home](#)

[Security Considerations](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Security Considerations

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Security Considerations

Introduction



This is intended to be a brief overview of some things you should keep in mind when installing Icinga, so as set it up in a secure manner.

Your monitoring box should be viewed as a backdoor into your other systems. In many cases, the Icinga server might be allowed access through firewalls in order to monitor remote servers. In most all cases, it is allowed to query those remote servers for various information. Monitoring servers are always given a certain level of trust in order to query remote systems. This presents a potential attacker with an attractive backdoor to your systems. An attacker might have an easier time getting into your other systems if they compromise the monitoring server first. This is particularly true if you are making use of shared SSH keys in order to monitor remote systems.

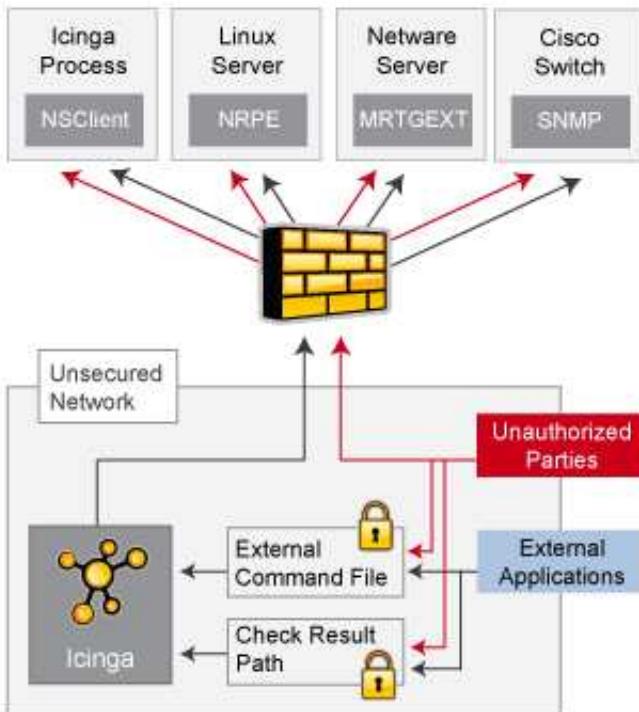
If an intruder has the ability to submit check results or external commands to the Icinga daemon, they have the potential to submit bogus monitoring data, drive you nuts with bogus notifications, or cause event handler scripts to be triggered. If you have event handler scripts that restart services, cycle power, etc. this could be particularly problematic.

Another area of concern is the ability for intruders to sniff monitoring data (status information) as it comes across the wire. If communication channels are not encrypted, attackers can gain valuable information by watching your monitoring information. Take as an example the following situation: An attacker captures monitoring data on the wire over a period of time and analyzes the typical CPU and disk load usage of your systems, along with the number of users that are typically logged into them. The attacker is then able to determine the best time to compromise a system and use its resources (CPU, etc.) without being noticed.

Here are some tips to help ensure that you keep your systems secure when implementing a Icinga-based monitoring solution...

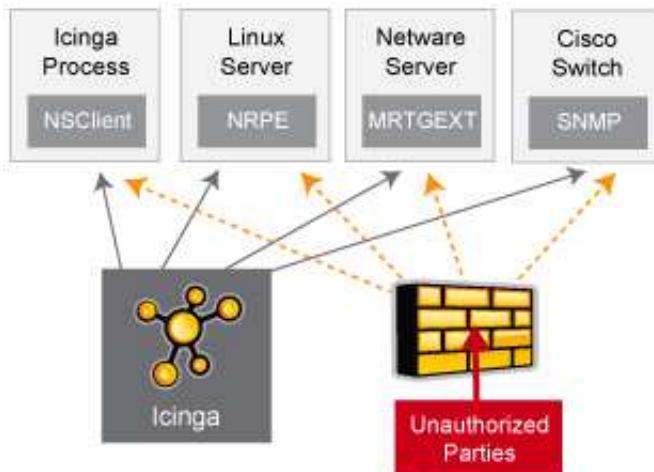
Best Practices

- 1. Use a Dedicated Monitoring Box**. We would recommend that you install Icinga on a server that is dedicated to monitoring (and possibly other admin tasks). Protect your monitoring server as if it were one of the most important servers on your network. Keep running services to a minimum and lock down access to it via TCP wrappers, firewalls, etc. Since the Icinga server is allowed to talk to your servers and may be able to poke through your firewalls, allowing users access to your monitoring server can be a security risk. Remember, it's always easier to gain root access through a system security hole if you have a local account on a box.



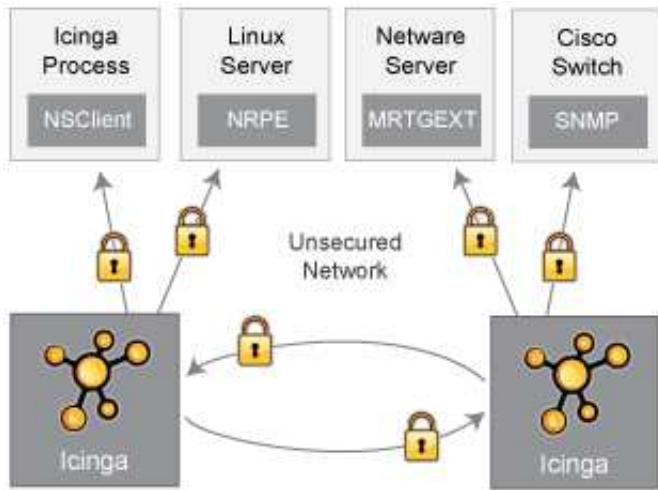
- 2. Don't Run Icinga As Root**. Icinga doesn't need to run as root, so don't do it. You can tell Icinga to drop privileges after startup and run as another user/group by using the `icinga_user` and `icinga_group` directives in the main config file. If you need to execute event handlers or plugins which require root access, you might want to try using `sudo`.
- 3. Lock Down The Check Result Directory**. Make sure that only the `icinga` user is able to read/write in the `check result path`. If users other than `icinga` (or `root`) are able to write to this directory, they could send fake host/service check results to the Icinga daemon. This could result in annoyances (bogus notifications) or security problems (event handlers being kicked off).
- 4. Lock Down The External Command File**. If you enable `external commands`, make sure you set proper permissions on the `/usr/local/icinga/var/rw` directory. You only want the Icinga user (usually `icinga`) and the web server user (usually `nobody`, `httpd`, `apache2`, or `www-data`) to have permissions to write to the command file. If you've installed Icinga on a machine that is dedicated to monitoring and admin tasks and is not used for public accounts, that should be fine. If you've installed it on a public or multi-user machine (not recommended), allowing the web server user to have write access to the command file can be a security problem. After all, you don't want just any user on your system controlling Icinga through the external command file. In this case, we would suggest only granting write access on the command file to the `icinga` user and using something like `CGIWrap` to run the CGIs as the `icinga` user instead of `nobody`.

5. **Require Authentication In The CGIs**. We would strongly suggest requiring authentication for accessing the CGIs. Once you do that, read the documentation on the default rights that authenticated contacts have, and only authorize specific contacts for additional rights as necessary. Instructions on setting up authentication and configuring authorization rights can be found [here](#). If you disable the CGI authentication features using the `use_authentication` directive in the CGI config file, the `command CGI` will refuse to write any commands to the `external command file`. After all, you don't want the world to be able to control Icinga do you?
6. **Implement Enhanced CGI Security Measures**. We would strongly suggest that you consider implementing enhanced security measures for the CGIs as described [here](#). These measures can help ensure that the username/password you use to access the Icinga web interface are not intercepted by third parties.
7. **Use Full Paths In Command Definitions**. When you define commands, make sure you specify the *full path* (not a relative one) to any scripts or binaries you're executing.
8. **Hide Sensitive Information With \$USERn\$ Macros**. The CGIs read the [main config file](#) and [object config file\(s\)](#), so you don't want to keep any sensitive information (usernames, passwords, etc) in there. If you need to specify a username and/or password in a command definition use a \$USERn\$ [macro](#) to hide it. \$USERn\$ macros are defined in one or more [resource files](#). The CGIs will not attempt to read the contents of resource files, so you can set more restrictive permissions (600 or 660) on them. See the sample `resource.cfg` file in the base of the Icinga distribution for an example of how to define \$USERn\$ macros.
9. **Strip Dangerous Characters From Macros**. Use the `illegal_macro_output_chars` directive to strip dangerous characters from the \$HOSTOUTPUT\$, \$SERVICEOUTPUT\$, \$HOSTPERFDATA\$, and \$SERVICEPERFDATA\$ macros before they're used in notifications, etc. Dangerous characters can be anything that might be interpreted by the shell, thereby opening a security hole. An example of this is the presence of backtick (`) characters in the \$HOSTOUTPUT\$, \$SERVICEOUTPUT\$, \$HOSTPERFDATA\$, and/or \$SERVICEPERFDATA\$ macros, which could allow an attacker to execute an arbitrary command as the icinga user (one good reason not to run Icinga as the root user).



10. **Secure Access to Remote Agents**. Make sure you lock down access to agents (NRPE, NSClient, SNMP, etc.) on remote systems using firewalls, access lists, etc. You don't want everyone to be able to query your systems for status information. This information could be used by an attacker to execute remote event handler scripts or to determine the best times to go unnoticed.

11. **Secure Communication Channels**. Make sure you encrypt communication channels between different Icinga installations and between your Icinga servers and your monitoring agents whenever possible. You don't want someone to be able to sniff status information going across your network. This information could be used by an attacker to determine the best times to go unnoticed.



[Prev](#)[Up](#)[Next](#)

Chapter 8. Security and
Performance Tuning

[Home](#)

Enhanced CGI Security and
Authentication

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Enhanced CGI Security and Authentication

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Enhanced CGI Security and Authentication

Introduction



This is intended to be an introduction for implementation of stronger authentication and server security focused around the CGI web interface.

There are many ways to enhance the security of your monitoring server and Icinga environment. This should not be taken as the end all approach to security. Instead, think of it as an introduction to some of the techniques you can use to tighten the security of your system. As always, you should do your research and use the best techniques available. Treat your monitoring server as it were the most important server in your network and you shall be rewarded.

Additional Techniques

- **Stronger Authentication using Digest Authentication** . If you have followed the [quickstart guides](#), chances are that you are using Apache's [Basic Authentication](#). Basic Authentication will send your username and password in "clear text" with every http request. Consider using a more secure method of authentication such as [Digest Authentication](#) which creates a MD5 Hash of your username and password to send with each request.
- **Forcing TLS/SSL for all Web Communication** . Apache provides [TLS/SSL](#) through the [mod_ssl](#) module. TLS/SSL provides a secure tunnel between the client and server that prevents eavesdropping and tampering using strong publickey/privatekey cryptography.
- **Locking Down Apache Using Access Controls** . Consider locking down access to the Icinga box to your IP address, IP address range, or IP subnet. If you require access outside your network you could use VPN or SSH Tunnels. This is a easy and strong to limit access to HTTP/HTTPS on your system.

Implementing Digest Authentication

The implementation of Digest Authentication is simple. You will have to create the new type of password file using the '[htdigest](#)' tool, then modify the Apache configuration for Icinga (typically /etc/httpd/conf.d/icinga.conf).

Create a new passwords file using the '[htdigest](#)' tool. The difference that you will notice if you are familiar with '[htpasswd](#)' tools is the requirement to supply a 'realm' argument. Where 'realm' in this case refers to the value of the 'AuthName' directive in the Apache configuration.

```
htdigest -c /usr/local/icinga/etc/.digest_pw "Icinga Access" icingaadmin
```

Next, edit the Apache configuration file for Icinga (typically /etc/httpd/conf.d/icinga.conf) using the following example.

```
## BEGIN APACHE CONFIG SNIPPET - ICINGA.CONF
ScriptAlias /icinga/cgi-bin "/usr/local/icinga/sbin"
<Directory "/usr/local/icinga/sbin">
    Options ExecCGI
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthType Digest
    AuthName "Icinga Access"
    AuthDigestFile /usr/local/icinga/etc/.digest_pw
    Require valid-user
</Directory>

Alias /icinga "/usr/local/icinga/share"
<Directory "/usr/local/icinga/share">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
    AuthType Digest
    AuthName "Icinga Access"
    AuthDigestFile /usr/local/icinga/etc/.digest_pw
    Require valid-user
</Directory>
## END APACHE CONFIG SNIPPETS
```

Then, restart the Apache service so the new settings can take effect.

```
/etc/init.d/httpd restart
```

Implementing Forced TLS/SSL

Make sure you've installed Apache and OpenSSL. By default you should have [mod_ssl](#) support if you are still having trouble you may find help reading Apache's [TLS/SSL Encryption Documentation](#).

Next, verify that TLS/SSL support is working by visiting your Icinga Web Interface using HTTPS (<https://your.domain/Icinga>). If it is working you can continue on to the next steps that will force using HTTPS and block all HTTP requests for the Icinga Web Interface. If you are having trouble visit Apache's [TLS/SSL Encryption Documentation](#) and [Google](#) for troubleshooting your specific Apache installation.

Next, edit the Apache configuration file for Icinga (typically /etc/httpd/conf.d/icinga.conf) by adding the 'SSLRequireSSL' directive to both the 'sbin' and 'share' directories.

```
## BEGIN APACHE CONFIG SNIPPET - ICINGA.CONF
ScriptAlias /icinga/cgi-bin "/usr/local/icinga/sbin"
<Directory "/usr/local/icinga/sbin">
    ...
    SSLRequireSSL
    ...
</Directory>

Alias /icinga "/usr/local/icinga/share"
```

```
<Directory "/usr/local/icinga/share">
...
SSLRequireSSL
...
</Directory>
## END APACHE CONFIG SNIPPETS
```

Restart the Apache service so the new settings can take effect.

```
/etc/init.d/httpd restart
```

Implementing IP subnet lockdown

The following example will show how to lock down Icinga CGIs to a specific IP address, IP address range, or IP subnet using Apache's [access controls](#).

Edit the Apache configuration file for Icinga (typically /etc/httpd/conf.d/icinga.conf) by using the 'Allow', 'Deny', and 'Order' directives using the following as an example.

```
## BEGIN APACHE CONFIG SNIPPET - ICINGA.CONF
ScriptAlias /icinga/cgi-bin "/usr/local/icinga/sbin"
<Directory "/usr/local/icinga/sbin">
...
AllowOverride None
Order deny,allow
Deny from all
Allow from 127.0.0.1 10.0.0.25          # Allow single IP addresses
Allow from 10.0.0.0/255.255.255.0       # Allow network/netmask pair
Allow from 10.0.0.0/24                  # Allow network/nnn CIDR spec
...
</Directory>

Alias /icinga "/usr/local/icinga/share"
<Directory "/usr/local/icinga/share">
...
AllowOverride None
Order deny,allow
Deny from all
Allow from 127.0.0.1 10.0.0.25          # Allow single IP addresses
Allow from 10.0.0.0/255.255.255.0       # Allow network/netmask pair
Allow from 10.0.0.0/24                  # Allow network/nnn CIDR spec
...
</Directory>
## END APACHE CONFIG SNIPPET
```

Important Notes

- **Digest Authentication sends data in the clear but not your username and password .**
- **Digest Authentication is not as universally supported as Basic Authentication .**
- **TLS/SSL has potential for "man-in-the-middle attacks" .** MITM attacks are vulnerable if an attacker is able to insert itself between the server and client such as in a Phishing attack, ISP monitoring, or corporate LAN firewall certificate resigning. So read up on certificate verification!
- **Apache access controls only protect the HTTP/HTTPS protocols .** Look into [IPtables](#) for strong system wide firewall control.
- **Most importantly, Security is a moving target so stay informed and do research !** Perhaps by listening to a Podcast such as "[Security Now!](#)".

[Prev](#)

[Up](#)

[Next](#)

[Security Considerations](#)

[Home](#)

[Tuning Icinga For Maximum Performance](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Tuning Icinga For Maximum Performance**[Prev](#)[Chapter 8. Security and Performance Tuning](#)[Next](#)

Tuning Icinga For Maximum Performance

Introduction



So you've finally got Icinga up and running and you want to know how you can tweak it a bit. Tuning Icinga to increase performance can be necessary when you start monitoring a large number (> 1,000) of hosts and services. Here are a few things to look at for optimizing Icinga...

Optimization Tips:

1. **Graph performance statistics with PNP4Nagios** . In order to keep track of how well your Icinga installation handles load over time and how your configuration changes affect it, you should be graphing several important statistics with PNP4Nagios. This is really, really, really useful when it comes to tuning the performance of a Icinga installation. Really. Information on how to do this can be found [here](#).
2. **Use large installation tweaks** . Enabling the `use_large_installation_tweaks` option may provide you with better performance. Read more about what this option does [here](#).
3. **Disable environment macros** . Macros are normally made available to check, notification, event handler, etc. commands as environment variables. This can be a problem in a large Icinga installation, as it consumes some additional memory and (more importantly) more CPU. If your scripts don't need to access the macros as environment variables (e.g. you pass all necessary macros on the command line), you don't need this feature. You can prevent macros from being made available as environment variables by using the `enable_environment_macros` option.
4. **Check Result Reaper Frequency** . The `check_result_reaper_frequency` variable determines how often Icinga should check for host and service check results that need to be processed. The maximum amount of time it can spend processing those results is determined by the max reaper time (see below). If your reaper frequency is too high (too infrequent), you

might see high latencies for host and service checks.

5. **Max Reaper Time** . The [max_check_result_reaper_time](#) variables determines the maximum amount of time the Icinga daemon can spend processing the results of host and service checks before moving on to other things - like executing new host and service checks. Too high of a value can result in large latencies for your host and service checks. Too low of a value can have the same effect. If you're experiencing high latencies, adjust this variable and see what effect it has. Again, you should be [graphing statistics](#) in order to make this determination.
6. **Adjust buffer slots** . You may need to adjust the value of the [external_command_buffer_slots](#) option. Graphing buffer slot statistics with [PNP4Nagios](#) (see above) is critical in determining what values you should use for this option.
7. **Check service latencies to determine best value for maximum concurrent checks** . Icinga can restrict the number of maximum concurrently executing service checks to the value you specify with the [max_concurrent_checks](#) option. This is good because it gives you some control over how much load Icinga will impose on your monitoring host, but it can also slow things down. If you are seeing high latency values (> 10 or 15 seconds) for the majority of your service checks (via the [extinfo CGI](#)), you are probably starving Icinga of the checks it needs. That's not Icinga's fault - its yours. Under ideal conditions, all service checks would have a latency of 0, meaning they were executed at the exact time that they were scheduled to be executed. However, it is normal for some checks to have small latency values. We would recommend taking the minimum number of maximum concurrent checks reported when running Icinga with the `-s` command line argument and doubling it. Keep increasing it until the average check latency for your services is fairly low. More information on service check scheduling can be found [here](#).
8. **Use passive checks when possible** . The overhead needed to process the results of [passive service checks](#) is much lower than that of "normal" active checks, so make use of that piece of info if you're monitoring a slew of services. It should be noted that passive service checks are only really useful if you have some external application doing some type of monitoring or reporting, so if you're having Icinga do all the work, this won't help things.
9. **Avoid using interpreted plugins** . One thing that will significantly reduce the load on your monitoring host is the use of compiled (C/C++, etc.) plugins rather than interpreted script (Perl, etc) plugins. While Perl scripts and such are easy to write and work well, the fact that they are compiled/interpreted at every execution instance can significantly increase the load on your monitoring host if you have a lot of service checks. If you want to use Perl plugins, consider compiling them into true executables using perlcc(1) (a utility which is part of the standard Perl distribution) or compiling Icinga with an embedded Perl interpreter (see below).
10. **Use the embedded Perl interpreter** . If you're using a lot of Perl scripts for service checks, etc., you will probably find that compiling the [embedded Perl interpreter](#) into the Icinga binary will speed things up.
11. **Optimize host check commands** . If you're checking host states using the `check_ping` plugin you'll find that host checks will be performed much faster if you break up the checks. Instead of specifying a `max_attempts` value of 1 in the host definition and having the `check_ping` plugin send 10 ICMP packets to the host, it would be much faster to set the `max_attempts` value to 10 and only send out 1 ICMP packet each time. This is due to the fact that Icinga can often determine the status of a host after executing the plugin once, so you want to make the first check as fast as possible. This method does have its pitfalls in some situations (i.e. hosts that are slow to respond may be assumed to be down), but you'll see faster host checks if you use it. Another option would be to use a faster plugin (i.e. `check_fping`) as the `host_check_command` instead of `check_ping`.

12. **Schedule regular host checks** . Scheduling regular checks of hosts can actually help performance in Icinga. This is due to the way the [cached check logic](#) works (see below). Host checks are run in parallel - just like service checks. To schedule regular checks of a host, set the `check_interval` directive in the [host definition](#) to something greater than 0.
13. **Enable cached host checks** . On-demand host checks can benefit from caching. On-demand host checks are performed whenever Icinga detects a service state change. These on-demand checks are executed because Icinga wants to know if the host associated with the service changed state. By enabling cached host checks, you can optimize performance. In some cases, Icinga may be able to use the old/cached state of the host, rather than actually executing a host check command. This can speed things up and reduce load on monitoring server. In order for cached checks to be effective, you need to schedule regular checks of your hosts (see above). More information on cached checks can be found [here](#).
14. **Don't use aggressive host checking** . Unless you're having problems with Icinga recognizing host recoveries, we would recommend not enabling the [use_aggressive_host_checking](#) option. With this option turned off host checks will execute much faster, resulting in speedier processing of service check results. However, host recoveries can be missed under certain circumstances when this is turned off. For example, if a host recovers and all of the services associated with that host stay in non-OK states (and don't "wobble" between different non-OK states), Icinga may miss the fact that the host has recovered. A few people may need to enable this option, but the majority don't and we would recommend not using it unless you find it necessary...
15. **External command optimizations** . If you're processing a lot of external commands (i.e. passive checks in a [distributed setup](#), you'll probably want to set the `command_check_interval` variable to -1. This will cause Icinga to check for external commands as often as possible. You should also consider increasing the number of available [external command buffer slots](#). Buffers slots are used to hold external commands that have been read from the [external command file](#) (by a separate thread) before they are processed by the Icinga daemon. If your Icinga daemon is receiving a lot of passive checks or external commands, you could end up in a situation where the buffers are always full. This results in child processes (external scripts, NSCA daemon, etc.) blocking when they attempt to write to the external command file. We would highly recommend that you graph external command buffer slot usage using PNP4Nagios and the nagiostats utility as described [here](#), so you understand the typical external command buffer usage of your Icinga installation.
16. **Optimize hardware for maximum performance** . NOTE: Hardware performance shouldn't be an issue unless: 1) you're monitoring thousands of services, 2) you're doing a lot of post-processing of performance data, etc. Your system configuration and your hardware setup are going to directly affect how your operating system performs, so they'll affect how Icinga performs. The most common hardware optimization you can make is with your hard drives. CPU and memory speed are obviously factors that affect performance, but disk access is going to be your biggest bottleneck. Don't store plugins, the status log, etc on slow drives (i.e. old IDE drives or NFS mounts). If you've got them, use UltraSCSI drives or fast IDE drives. An important note for IDE/Linux users is that many Linux installations do not attempt to optimize disk access. If you don't change the disk access parameters (by using a utility like `hdparam`), you'll lose out on a lot of the speedy features of the new IDE drives.
17. **Use a RAM disk for temporary data** . Several files are created and processed very often. That includes the current status stored in the [status file](#) and the configuration being cached in the [object cache file](#). To reduce physical I/O it is advisable to have this data on a RAM disk. Data loss due to a power failure or something alike is not critical because the two files are created every time Icinga is (re)started. Setting up the RAM disk and the changes to the main config file is described [here](#).

[Prev](#)

[Up](#)

[Next](#)

Enhanced CGI Security and
Authentication

[Home](#)

Fast Startup Options

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Fast Startup Options

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Fast Startup Options

Introduction

There are a few things you can do that can decrease the amount of time it takes Icinga to startup (or restart). These speedups involve easing some of the burden involved in processing your configuration files.

Using these techniques is particularly useful when you have one or more of the following:

- Large configurations
- Complex configurations (heavy use of template features)
- Installations where frequent restarts are necessary

Background

Whenever Icinga starts/restarts it has to process your configuration files before it can get down to the business of monitoring. This configuration startup process involves a number of steps:

- Reading the config files
- Resolving template definitions
- "Recombobulating" your objects (Ethan Galstads term for the various types of work that occurs)
- Duplicating object definitions
- Inheriting object properties
- Sorting your object definitions
- Verifying object relationship integrity
- Checking for circular paths
- and more...

Some of these steps can be quite time-consuming when you have large or complex configurations. Is there a way to speed any of these steps up? Yes!

Evaluating Startup Times

Before we get on to making things faster, we need to see what's possible and whether or not we should even bother with the whole thing. This is easy to do - simply start Icinga with the **-s** or **--test-scheduling** command line switch to get timing and scheduling information.

Starting with Icinga 1.0.2 there is an additional option **-S** or **--show-scheduling**. This will add information about the scheduling queue to the output.

An example of the output (abbreviated to only show relevant portions) is shown below. For this example, we're using an Icinga config that has 25 hosts defined and just over 10,000 services.

```
#> /usr/local/icinga/bin/icinga -s /usr/local/icinga/etc/icinga.cfg
```

```
Icinga 1.4
Copyright (c) 2009 Nagios Core Development Team and Community Contributors
Copyright (c) 1999-2009 Ethan Galstad
Last Modified: 12-16-2009
License: GPL
```

Timing information on object configuration processing is listed below. You can use this information to see if precaching your object configuration would be useful.

Object Config Source: Config files (uncached)

OBJECT CONFIG PROCESSING TIMES		(* = Potential for precache savings with -u option)

Read:	0.486780 sec	
Resolve:	0.004106 sec	*
Recomb Contactgroups:	0.000077 sec	*
Recomb Hostgroups:	0.000172 sec	*
Dup Services:	0.028801 sec	*
Recomb Servicegroups:	0.010358 sec	*
Duplicate:	5.666932 sec	*
Inherit:	0.003770 sec	*
Recomb Contacts:	0.030085 sec	*
Sort:	2.648863 sec	*
Register:	2.654628 sec	
Free:	0.021347 sec	
=====		
TOTAL:	11.555925 sec	* = 8.393170 sec (72.63%) estimated savings

Timing information on configuration verification is listed below.

CONFIG VERIFICATION TIMES		(* = Potential for speedup with -x option)

Object Relationships:	1.400807 sec	
Circular Paths:	54.676622 sec	*
Misc:	0.006924 sec	
=====		
TOTAL:	56.084353 sec	* = 54.676622 sec (97.5%) estimated savings

Okay, let's see what happened. Looking at the totals, it took roughly **11.6** seconds to process the configuration files and another **56** seconds to verify the config. That means that every time we start or restart Icinga with this configuration, it will take nearly **68 seconds** of startup work before it can monitor anything! That's not acceptable if we have to restart Icinga on a semi-regular basis.

What can we do about this? Take another look at the output and you'll see that Icinga estimates that we could save about **8.4** seconds off the configuration processing time and another **54.7** off the verification times. In total, Icinga thinks we could save **63 seconds** of the normal startup time if some optimizations were taken.

Whoa! From **68 seconds** to just **5 seconds**? Yep, read on for how to do it.

Pre-Caching Object Configuration

Icinga can spend quite a bit of time parsing your config files, especially if you make use of the template features such as inheritance, etc. In order to reduce the time it takes to parse your config, you can have Icinga pre-process and pre-cache your config files for future use.

When you run Icinga with the **-p** command line option, Icinga will read your config files in, process them, and save them to a pre-cached object config file (specified by the [precached_object_file](#) directive). This pre-cached config file will contain pre-processed configuration entries that are easier/faster for Icinga to process in the future.

You must use the **-p** command line option along with either the **-v** or **-s** command line options, as shown below. This ensures that your configuration is verified before the precached file is created.

```
/usr/local/icinga/bin/icinga -pv /usr/local/icinga/etc/icinga.cfg
```

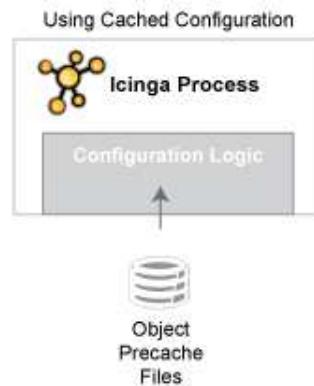
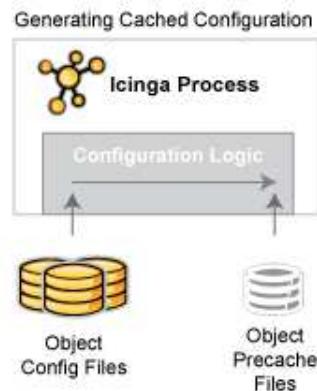
The size of your precached config file will most likely be significantly larger than the sum of the sizes of your object config files. This is normal and by design.

Once the precached object configuration file have been created, you can start Icinga and tell it to use the precached config file instead of your object config file(s) by using the **-u** command line option.

```
/usr/local/icinga/bin/icinga -ud /usr/local/icinga/etc/icinga.cfg
```



If you modify your configuration files, you will need to re-verify and re-cache your configuration files before restarting Icinga. If you don't re-generate the precached object file, Icinga will continue to use your old configuration because it is now reading from the precached file, rather than your source configuration files.



Skipping Circular Path Tests

The second (and most time-intensive) portion of the configuration startup phase is the circular path check. In the example above, it took nearly a minute to perform this step of the configuration verification.

What is the circular path check and why does it take so long? The circular patch check is designed to ensure that you don't define any circular paths in your host, host dependency, or service dependency definitions. If a circular path existed in your config files, Icinga could end up in a deadlock situation. The most likely reason for the check taking so long is that we're not using an efficient algorithm. A much more efficient algorithm for detecting circular paths would be most welcomed. Hint: That means all you CompSci graduate students who have been emailing me about doing your thesis on Icinga can contribute some code back. :-)

If you want to skip the circular path check when Icinga starts, you can add the **-x** command line option like this:

```
/usr/local/icinga/bin/icinga -xd /usr/local/icinga/etc/icinga.cfg
```



It is of utmost importance that you verify your configuration before starting/restarting Icinga when skipping circular path checks. Failure to do so could lead to deadlocks in the Icinga logic. You have been warned.

Putting It All Together

Follow these steps if you want to make use of potential speedups from pre-caching your configuration and skipping circular path checks.

1. Verify your configuration and create the precache file with the following command:

```
/usr/local/icinga/bin/icinga -vp /usr/local/icinga/etc/icinga.cfg
```

2. Stop Icinga if it is currently running.

3. Start Icinga like so to use the precached config file and skip circular path checks:

```
/usr/local/icinga/bin/icinga -uxd /usr/local/icinga/etc/icinga.cfg
```

4. When you modify your original configuration files in the future and need to restart Icinga to make those changes take place, repeat step 1 to re-verify your config and regenerate your cached config file. Once that is done you can restart Icinga through the web interface or by sending a SIGHUP signal. If you don't re-generate the precached object file, Icinga will continue to use your old configuration because it is now reading from the precached file, rather than your source configuration files.

5. That's it! Enjoy the increased startup speed.
-

[Prev](#)
[Up](#)
[Next](#)
[Tuning Icinga For Maximum Performance](#)
[Home](#)
[Large Installation Tweaks](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Large Installation Tweaks

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Large Installation Tweaks

Introduction

Users with large Icinga installations may benefit from the [use_large_installation_tweaks](#) configuration option. Enabling this option allows the Icinga daemon to take certain shortcuts which result in lower system load and better performance.

Effects

When you enable the [use_large_installation_tweaks](#) option in your main Icinga config file, several changes are made to the way the Icinga daemon operates:

1. **No Summary Macros In Environment Variables** - The [summary macros](#) will not be available to you as environment variables. Calculating the values of these macros can be quite time-intensive in large configurations, so they are not available as environment variables when use this option. Summary macros will still be available as regular macros if you pass them to to your scripts as arguments.
2. **Different Memory Cleanup** - Normally Icinga will free all allocated memory in child processes before they exit. This is probably best practice, but is likely unnecessary in most installations, as most OSes will take care of freeing allocated memory when processes exit. The OS tends to free allocated memory faster than can be done within Icinga itself, so Icinga won't attempt to free memory in child processes if you enable this option.
3. **Checks fork() Less** - Normally Icinga will fork() twice when it executes host and service checks. This is done to (1) ensure a high level of resistance against plugins that go awry and segfault and (2) make the OS deal with cleaning up the grandchild process once it exits. The extra fork() is not really necessary, so it is skipped when you enable this option. As a result, Icinga will itself clean up child processes that exit (instead of leaving that job to the OS). This feature should result in significant load savings on your Icinga installation.

[Prev](#)[Up](#)[Next](#)[Fast Startup Options](#)[Home](#)[Using The Icingastats Utility](#)



Using The Icingastats Utility

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Using The Icingastats Utility

Introduction

A utility called **icingastats** is included in the Icinga distribution. It is compiled and installed along with the main Icinga daemon. The icingastats utility allows you to obtain various information about a running Icinga process that can be very helpful in [tuning performance](#). You can obtain information either in human-readable or performance data compatible format.

Usage Information

You can run the *icingastats* utility with the **--help** option to get usage information.

Human-Readable Output

To obtain human-readable information on the performance of a running Icinga process, run the *icingastats* utility with the **-c** command line argument to specify your main configuration file location like such:

```
[icinga@monitoring ~]# /usr/local/icinga/bin/icingastats -c /usr/local/icinga/etc/icinga.cfg

Icinga Stats 1.4
Copyright (c) 2009 Nagios Core Development Team and Community Contributors
Copyright (c) 1999-2009 Ethan Galstad
Last Modified: 02-16-2011
License: GPL

CURRENT STATUS DATA
-----
Status File: /usr/local/icinga/var/status.dat
Status File Age: 0d 0h 0m 27s
Status File Version: 1.3.0

Program Running Time: 0d 14h 28m 16s
Icinga PID: 21182
Used/High/Total Command Buffers: 0 / 3 / 4096

Total Services: 1001
Services Checked: 945
Services Scheduled: 950
Services Actively Checked: 1000
Services Passively Checked: 1
Total Service State Change: 0.000 / 100.000 / 1.881 %
Active Service Latency: 0.000 / 285.165 / 25.045 sec
Active Service Execution Time: 0.000 / 304.925 / 0.834 sec
Active Service State Change: 0.000 / 100.000 / 1.883 %
Active Services Last 1/5/15/60 min: 20 / 191 / 471 / 926
Passive Service Latency: 0.862 / 0.862 / 0.862 sec
```

```

Passive Service State Change:          0.000 / 0.000 / 0.000 %
Passive Services Last 1/5/15/60 min: 1 / 1 / 1 / 1
Services Ok/Warn/Unk/Crit:           816 / 56 / 51 / 78
Services Flapping:                  39
Services In Downtime:               0

Total Hosts:                         111
Hosts Checked:                      104
Hosts Scheduled:                     104
Hosts Actively Checked:             111
Host Passively Checked:              0
Total Host State Change:            0.000 / 100.000 / 10.574 %
Active Host Latency:                0.000 / 279.257 / 21.700 sec
Active Host Execution Time:         0.000 / 6.405 / 0.432 sec
Active Host State Change:           0.000 / 100.000 / 10.574 %
Active Hosts Last 1/5/15/60 min:   17 / 50 / 74 / 104
Passive Host Latency:               0.000 / 0.000 / 0.000 sec
Passive Host State Change:          0.000 / 0.000 / 0.000 %
Passive Hosts Last 1/5/15/60 min:  0 / 0 / 0 / 0
Hosts Up/Down/Unreach:              89 / 7 / 15
Hosts Flapping:                     22
Hosts In Downtime:                 0

Active Host Checks Last 1/5/15 min: 73 / 97 / 246
  Scheduled:                        13 / 21 / 50
  On-demand:                        60 / 76 / 196
  Parallel:                          45 / 63 / 171
  Serial:                            0 / 0 / 0
  Cached:                            28 / 34 / 75
Passive Host Checks Last 1/5/15 min: 0 / 0 / 0
Active Service Checks Last 1/5/15 min: 142 / 192 / 501
  Scheduled:                        142 / 192 / 500
  On-demand:                         0 / 0 / 1
  Cached:                            0 / 0 / 0
Passive Service Checks Last 1/5/15 min: 6 / 6 / 15

External Commands Last 1/5/15 min:    6 / 6 / 15

[icinga@monitoring ~]#

```

As you can see, the utility displays a number of different metrics pertaining to the Icinga process. Metrics which have multiple values are (unless otherwise specified) min, max and average values for that particular metric.

PNP4Nagios Integration

You can use the *icingastats* utility to display various Icinga metrics using PNP4Nagios (or other compatible programmes). To do so, run the *icingastats* utility using the **--mrtg** and **--data** arguments. The **--data** argument is used to specify what statistics should be graphed. Possible values for the **--data** argument can be found by running the *icingastats* utility with the **--help** option.



Note

Information on using the *icingastats* utility to generate PNP4Nagios graphs for Icinga performance statistics can be found [here](#).

[Prev](#)
[Up](#)
[Next](#)
[Large Installation Tweaks](#)
[Home](#)
[Graphing Performance Info With
PNP4Nagios](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Graphing Performance Info With PNP4Nagios

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Graphing Performance Info With PNP4Nagios

Introduction

The [icingastats](#) utility allows you to graph various Icinga performance statistics over time using [PNP4Nagios](#). This is important because it can help you to:

- Ensure Icinga is operating efficiently
- Locate problem areas in the monitoring process
- Observe the performance impacts of changes in your Icinga configuration

Prerequisites

PNP4Nagios is one of the most popular addons due to easy installation and little maintenance during operation. The documentation along with further links to download the software can be found at <http://docs.pnp4nagios.org/pnp-0.6/start>.

[check_nagiostats](#) was published by [Jochen Bern](#) and will be called via crontab to submit passive check results. Despite its name it can be used for Icinga as well.

- After downloading the plugin and placing it into the plugin directory (e.g. `/usr/local/icinga/libexec` if you used the quickstart installation guides) you have to check the values of the configuration section within the script.
 - Most **important** is "EXEC=/path/to/icingastats" (e.g. `/usr/local/icinga/bin/icingastats`) pointing to the `icingastats` binary.
 - Depending on your likings you may want to change the value of CUMULATE from "AVG" to "MIN" or "MAX", respectively. The setting of TIMEFRAME influences the timeperiod which will be used for cumulated values in the output of the plugin.
 - Changing the values of PASSIVE_EMERGENCY_HOST and PASSIVE_EMERGENCY_SERVICE shouldn't be necessary because you pass the values as arguments to the script.
- Make sure that your object configuration files contain a matching service definition such as

```
define service{
    host_name          <the Icinga server>
    service_description icingastats # (or something appropriate)
    active_checks_enabled 0
    check_command      check_dummy!0
    ...
}
```

Don't forget to restart Icinga after this change.

- Create a symbolic link in the templates folder of PNP4Nagios

```
$> ln -s ../templates.dist/nagiostats.php icingastats.php
```

Make sure that *icingastats* (without the extension .php) matches the value you specified for the service description. Blanks in the service description have to be replaced by underscores in the filename (e.g. "Icinga Stats" --> "Icinga_Stats.php")

- Add a line to the crontab of the Icinga user which will call the *icingastats* binary and submit the results to the command pipe

```
* * * * * /usr/local/icinga/libexec/check_nagiostats -passive <host> <service> >> /usr/local/icinga/var/icinga.cmd
```

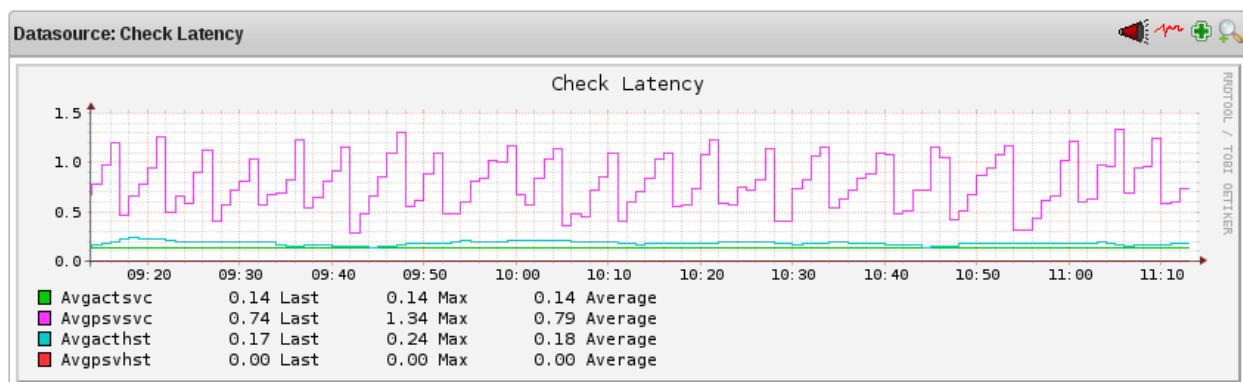
This way the values are updated in regular intervals.

Example Graphs

We'll describe what the graphs produced by *check_nagiostats* mean and what they can be used for...

Average Host / Service Check Latency

Figure 8.1. Average Host / Service Check Latency



This graph shows the average latency times of hosts and services over time for both active and passive checks, respectively. Useful for understanding:

- Host checks
- Service checks
- Active checks
- Passive checks

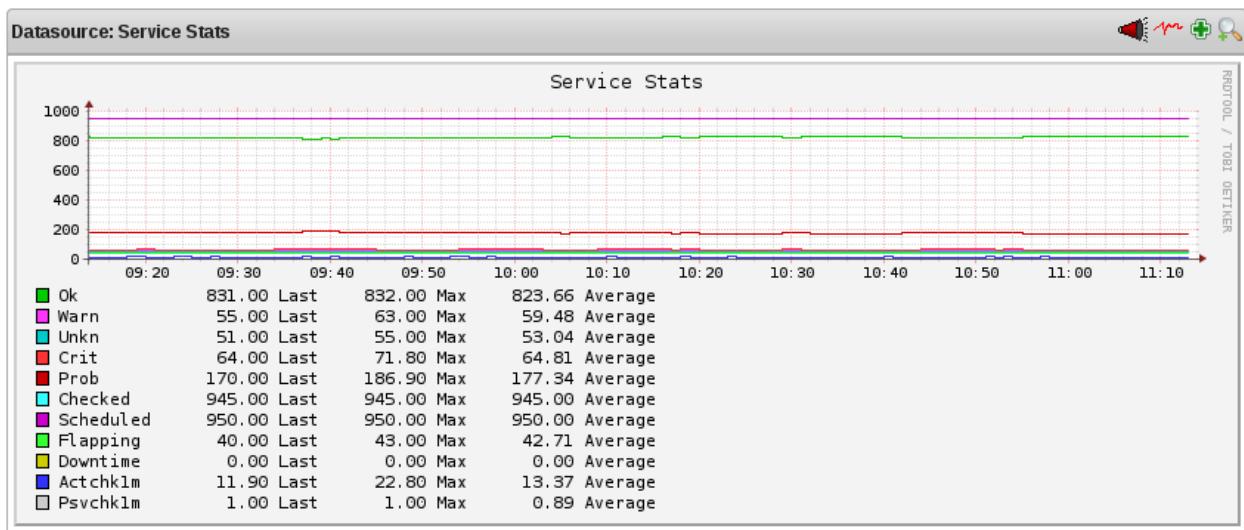
- [Performance tuning](#)

Consistently high latencies can be an indication that one or more of the following variables need tweaking:

- [max_concurrent_checks](#)
- [check_result_reaper_frequency](#)
- [max_check_result_reaper_time](#)

Service Statistics

Figure 8.2. Service Statistics

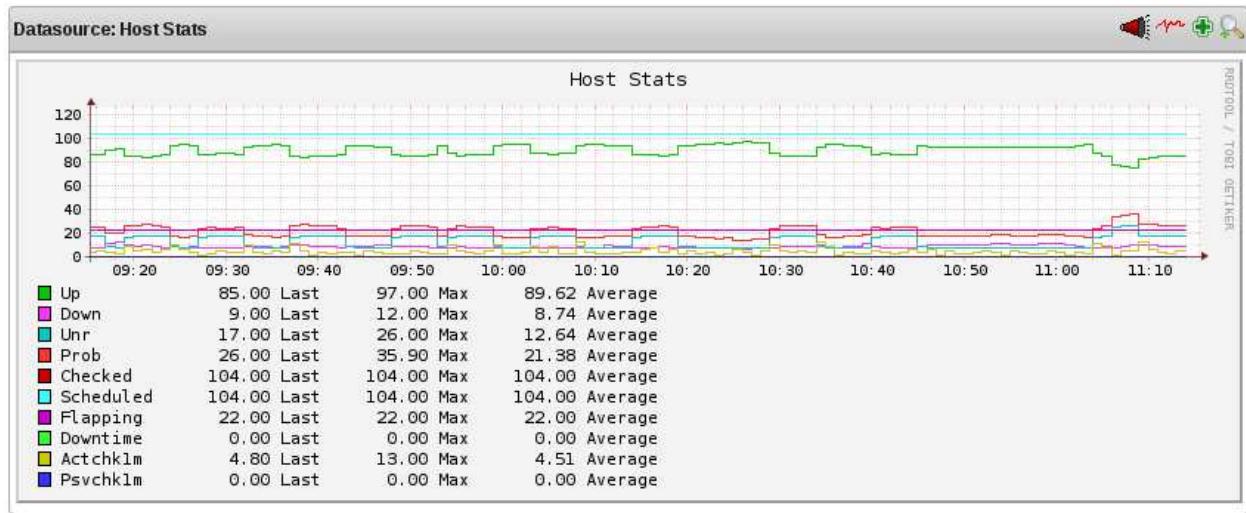


This graph shows the values for the several service states along with the average number of services being checked actively/passively within the timeperiod you specified. Useful for understanding:

- [Service checks](#)
- [Predictive service dependency checks](#)
- [Cached checks](#)
- [Flap detection](#)

Host Statistics

Figure 8.3. Host Statistics

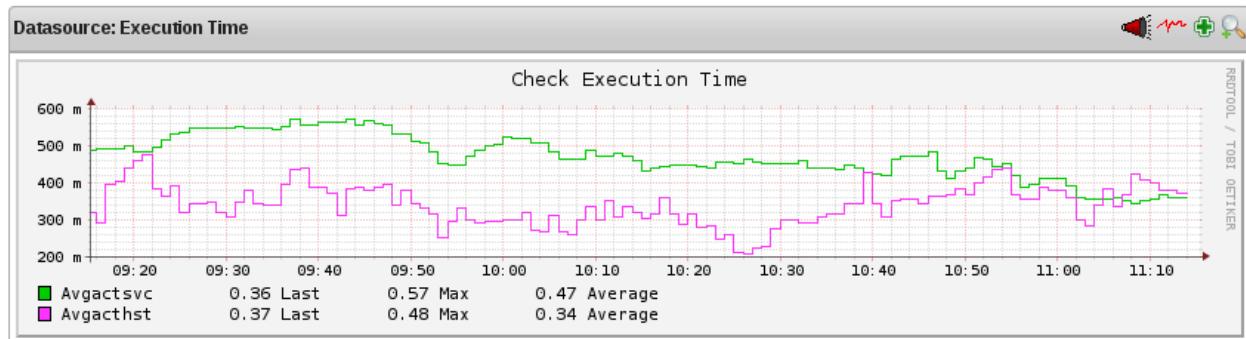


This graph shows the values for the several host states along with the average number of hosts being checked actively/passively within the timeperiod you specified. Useful for understanding:

- Host checks
- Predictive host dependency checks
- Cached checks
- Flap detection

Average Execution Times

Figure 8.4. Average Execution Times



This graph shows the average execution times of hosts and services over time. Useful for understanding:

- Host checks
- Service checks
- Performance tuning



Note

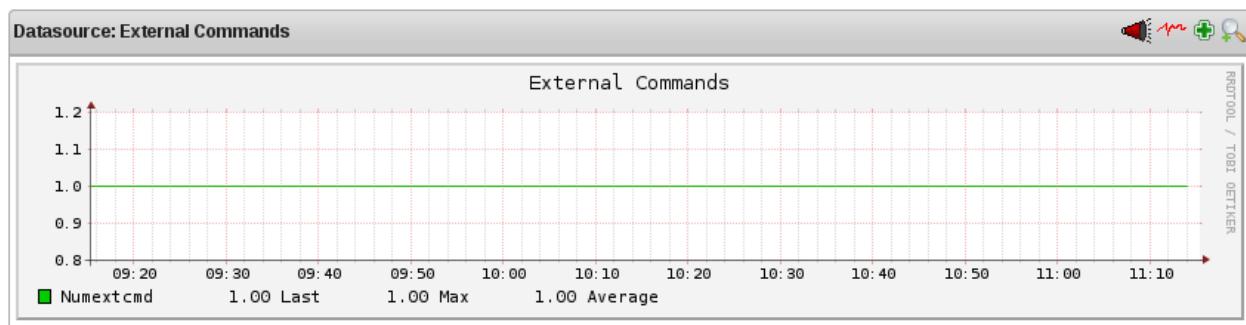
To be honest: We tweaked the graphs a bit, meaning the colours. Yellow is sometimes hard to distinguish from the background so we changed some lines in the PNP4Nagios template file `template.dist/nagiostats.php` from `$i=0;` to `$i=1;`.

Additional graphs

Well, we tweaked the template a bit more because the plugin delivers the data but there aren't appropriate graphs (but looking at the template file it is very easy to add the lines if you really need the following graphs).

External Commands

Figure 8.5. External Commands

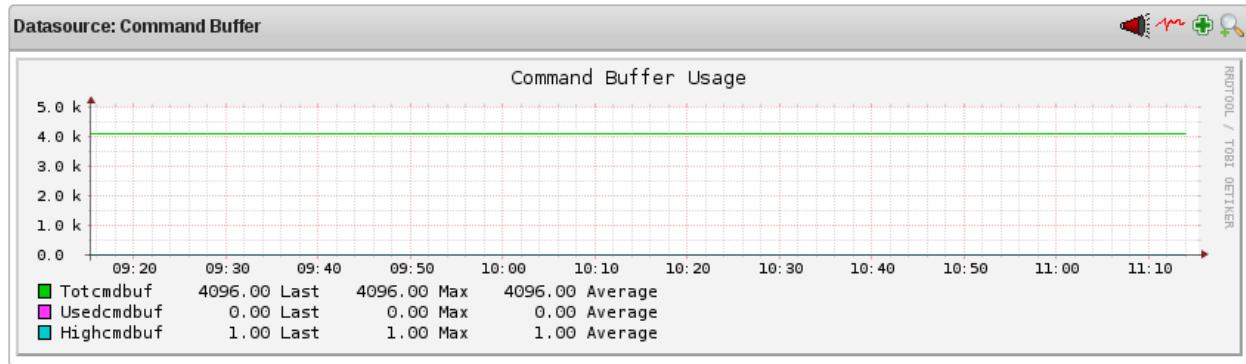


This graph shows how many external commands have been processed by the Icinga daemon over time. Unless you're processing a large number of external commands (as in the case with distributed monitoring setups), this graph may appear mostly empty. Monitoring external commands can be useful for understanding the impacts of:

- [Passive checks](#)
- [Distributed monitoring](#)
- [Redundant/failover monitoring](#)

External Command Buffers

Figure 8.6. External Command Buffers

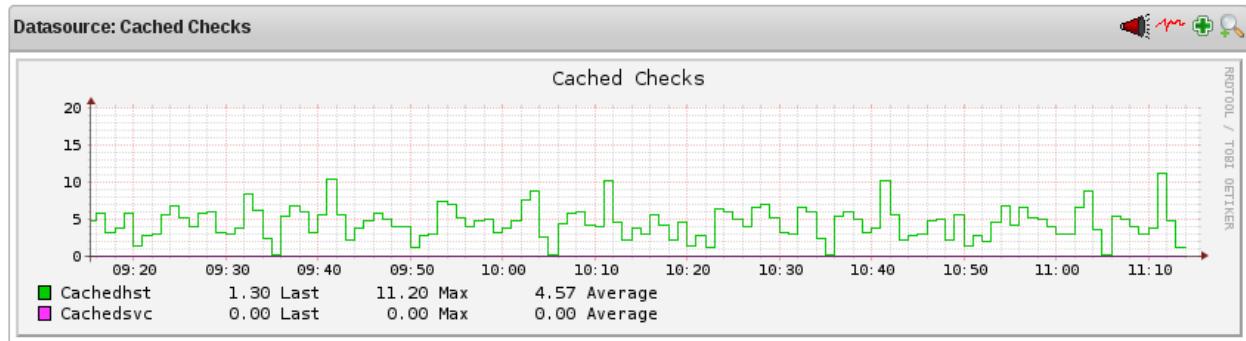


This graph shows how many external command buffer slots are in use over time. If the number of used buffers is near the number of available buffers on a regular basis, it is likely you need to increase the available [external command buffer slots](#). Each buffer slot can hold one external command. Buffers are used for temporarily holding external commands from the time they are read from the [external command file](#) to the time they are processed by the Icinga daemon.

As you can see just one buffer is used and that's the one for the graphs as mentioned above.

Cached Host and Service Checks

Figure 8.7. Cached Host and Service Checks

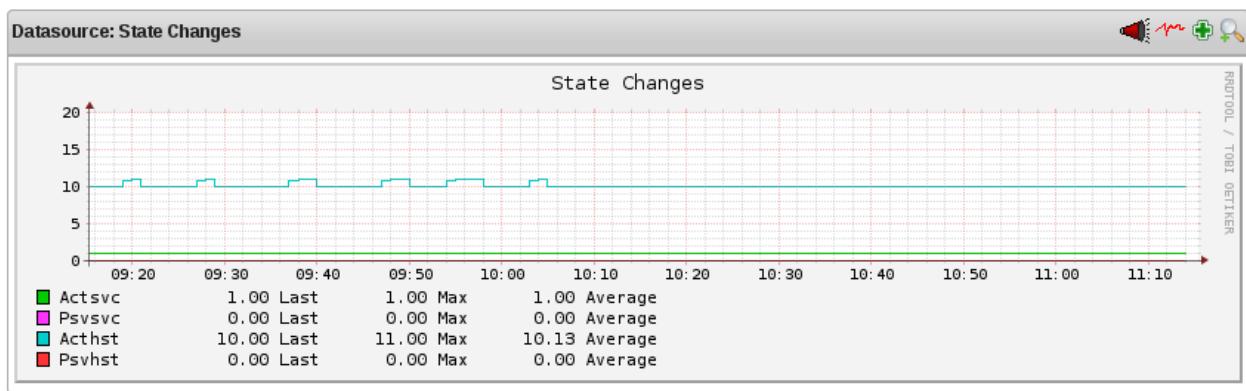


This graph shows how many cached host and service checks have occurred over time. Useful for understanding:

- [Cached checks](#)
- [Predictive host and service dependency checks](#)

Average State Changes

Figure 8.8. Average State Changes



This graph shows the average percent state change (a measure of volatility) over time, broken down by hosts and services that were last checked either actively or passively. Useful for understanding:

- [Flap detection](#)
-

[Prev](#)[Up](#)[Next](#)[Using The Icingastats Utility](#)[Home](#)[Temporary Data](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Temporary Data

[Prev](#)

Chapter 8. Security and Performance Tuning

[Next](#)

Temporary Data

Several files are created while starting Icinga and processed very often during runtime. Depending on the size of your configuration this may lead to high I/O and therefore degraded responsiveness. To reduce physical I/O operations it might be a good idea to place temporary files on a RAM disk. The following lines show the steps to setup a RAM disk and the changes to the main config file.



Note

Please keep in mind that these files will be lost if you reboot the system. Also note that sometimes it is difficult to determine the size of these files which may lead to a full RAM disk.

1. Have a look at the current location of the [status file](#) (e.g.

/usr/local/icinga/var/status.dat) and the [object cache file](#) (e.g. /usr/local/icinga/var/objects.cache) and determine the size of both files.

```
#> ls -la /usr/local/icinga/var/
-rw-rw-r-- 1 icinga icinga 8.2M Jun 10 11:57 status.dat
-rw-r--r-- 1 icinga icinga 5.9M Jun 10 11:58 objects.cache
```

2. Increase the number to a considerable value to allow for future growth (100 MB should be sufficient in this case) and create the RAM disk.



Caution

If the value is too high then this will throttle your system because it will start to swap resulting in physical I/O once again.

```
#> mkdir /var/icinga/ramdisk
#> mount -t tmpfs tmpfs /var/icinga/ramdisk -o size=100m
#> chown icinga:icinga /var/icinga/ramdisk
```

Adapt the values of user and group to the ones found in your configuration if necessary.

3. Add an entry to `/etc/fstab` to make the setting permanent so the RAM disk will be created automatically after the next reboot.

tmpfs	/var/icinga/ramdisk	tmpfs	size=100m	0	0
-------	---------------------	-------	-----------	---	---

4. Edit the Icinga main configuration file and change the setting of the two appropriate directives

```
#object_cache_file=/usr/local/icinga/var/objects.cache  
object_cache_file=/var/icinga/ramdisk/objects.cache
```

```
#status_file=/usr/local/icinga/var/status.dat  
status_file=/var/icinga/ramdisk/status.dat
```

5. Restart Icinga to put the changes into effect

```
#> /etc/init.d/icinga restart
```

You may want to use the RAM disk for other files like the check results as well. Please increase the size of the RAM disk if necessary and change the directive in the main config file

```
check_result_path=/var/icinga/ramdisk/checkresults
```

Please make sure that addons like check_mk are aware of this change.

[Prev](#)

[Up](#)

[Next](#)

Graphing Performance Info With
PNP4Nagios

[Home](#)

Chapter 9. Integration With Other
Software

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 9. Integration With Other Software

[Prev](#)

[Next](#)

Chapter 9. Integration With Other Software

Table of Contents

- [Integration Overview](#)
 - [SNMP Trap Integration](#)
 - [TCP Wrapper Integration](#)
 - [MKLiveStatus Integration](#)
 - [Installation of the Icinga-Reporting with JasperServer](#)
-

[Prev](#)

[Next](#)

[Temporary Data](#)

[Home](#)

[Integration Overview](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Integration Overview

[Prev](#)

Chapter 9. Integration With Other Software

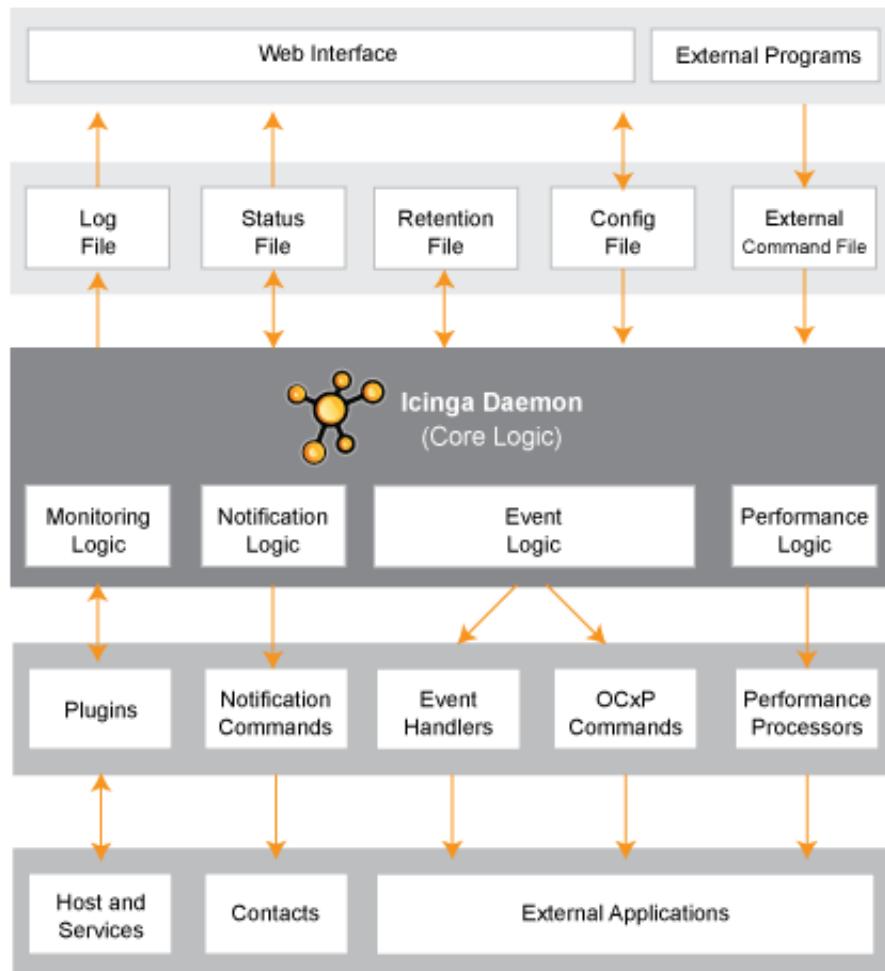
[Next](#)

Integration Overview

Introduction

Icinga can be easily integrated in your existing infrastructure. There are several methods of integrating Icinga with the management software you're already using and you can monitor almost any type of new or custom hardware, service, or application that you might have.

Integration Points



To monitor new hardware, services, or applications, check out the docs on:

- [Plugins](#)
- [Plugin API](#)
- [Passive Checks](#)
- [Event Handlers](#)

To get data into Icinga from external applications, check out the docs on:

- [Passive Checks](#)
- [External Commands](#)

To send status, performance, or notification information from Icinga to external applications, check out the docs on:

- [Event Handlers](#)
- [OCSP and OCHP Commands](#)
- [Performance Data](#)
- [Notifications](#)

Integration Examples

The following documents show some examples on how to integrate Icinga with external applications:

- [TCP Wrappers](#) (security alerts)
 - [SNMP Traps](#) (Arcserve backup job status)
 - [mklivestatus](#) (interface from Icinga to several addons like [NagVis](#) and [Thruk](#))
-

[Prev](#)

[Up](#)

[Next](#)

Chapter 9. Integration With Other
Software

[Home](#)

[SNMP Trap Integration](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**SNMP Trap Integration**[Prev](#)**Chapter 9. Integration With Other Software**[Next](#)

SNMP Trap Integration

Introduction

**Note**

Icinga is not designed to be a replacement for a full-blown SNMP management application like HP OpenView or [OpenNMS](#). However, you can set things up so that SNMP traps received by a host on your network can generate alerts in Icinga.

As if designed to make the Gods of Hypocrisy die of laughter, SNMP is anything but simple. Translating SNMP traps and getting them into Icinga (as passive check results) can be a bit tedious. To make this task easier, we suggest you check out Alex Burger's SNMP Trap Translator project located at <http://www.snmptt.org>. When combined with Net-SNMP, SNMPTT provides an enhanced trap handling system that can be integrated with Icinga.

Yep, that's all.

[Prev](#)[Up](#)[Next](#)[Integration Overview](#)[Home](#)[TCP Wrapper Integration](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



TCP Wrapper Integration

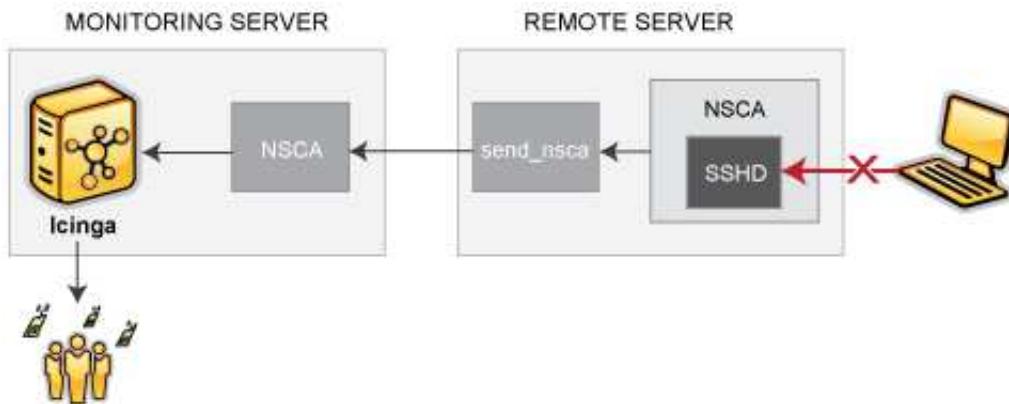
[Prev](#)

Chapter 9. Integration With Other Software

[Next](#)

TCP Wrapper Integration

Introduction



This document explains how to easily generate alerts in Icinga for connection attempts that are rejected by TCP wrappers. For example, if an unauthorized host attempts to connect to your SSH server, you can receive an alert in Icinga that contains the name of the host that was rejected. If you implement this on your Linux/Unix boxes, you'll be surprised how many port scans you can detect across your network.

These directions assume:

1. You are already familiar with [passive checks](#) and how they work.
2. You are already familiar with [volatile services](#) and how they work.
3. The host which you are generating alerts for (i.e. the host you are using TCP wrappers on) is a remote host (called *firestorm* in this example). If you want to generate alerts on the same host that Icinga is running you will need to make a few modifications to the examples we provide.
4. You have installed the [NSCA daemon](#) on your monitoring server and the NSCA client ([send_nsca](#)) on the remote machine that you are generating TCP wrapper alerts from.

Defining A Service

If you haven't done so already, create a [host definition](#) for the remote host (*firestorm*).

Next, define a service in one of your [object configuration files](#) for the TCP wrapper alerts on host *firestorm*. The service definition might look something like this:

```
define service{
    host_name          firestorm
    service_description TCP Wrappers
    is_volatile        1
    active_checks_enabled 0
    passive_checks_enabled 1
    max_check_attempts 1
    check_command      check_none
    ...
}
```

There are some important things to note about the above service definition:

1. The *volatile* option is enabled. We want this option enabled because we want a notification to be generated for every alert that comes in.
2. Active checks of the service are disabled, while passive checks are enabled. This means that the service will never be actively checked by Icinga - all alert information will have to be received passively from an external source.
3. The *max_check_attempts* value is set to 1. This guarantees you will get a notification when the first alert is generated.

Configuring TCP Wrappers

Now you're going to have to modify the */etc/hosts.deny* file on *firestorm*. In order to have the TCP wrappers send an alert to the monitoring host whenever a connection attempt is denied, you'll have to add a line similar to the following:

```
ALL: ALL: RFC931: twist (/usr/local/icinga/libexec/eventhandlers/handle_tcp_wrapper %h %d) &
```

This line assumes that there is a script called *handle_tcp_wrapper* in the */usr/local/icinga/libexec/eventhandlers/* directory on *firestorm*. We'll write that script next.

Writing The Script

The last thing you need to do is write the *handle_tcp_wrapper* script on *firestorm* that will send the alert back to the Icinga server. It might look something like this:

```
#!/bin/sh
/usr/local/icinga/libexec/eventhandlers/submit_check_result firestorm "TCP Wrappers" 2 "Denied $2-$1" > /dev/null 2> /dev/null
```

Notice that the *handle_tcp_wrapper* script calls the *submit_check_result* script to actually send the alert back to the monitoring host. Assuming your Icinga server is called *monitor*, the *submit_check_result* script might look like this:

```
#!/bin/sh
# Arguments
#   $1 = name of host in service definition
#   $2 = name/description of service in service definition
#   $3 = return code
#   $4 = output

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/icinga/bin/send_nsca monitor -c /usr/local/nagios/etc/send_nsca.cfg
```

Finishing Up

You've now configured everything you need to, so all you have to do is restart the *inetd* process on *firestorm* and restart Icinga on your monitoring server. That's it! When the TCP wrappers on *firestorm* deny a connection attempt, you should be getting alerts in Icinga. The plugin output for the alert will look something like the following:

```
Denied sshd2-sdn-ar-002mnminnP321.dialsprint.net
```

[Prev](#)[Up](#)[Next](#)[SNMP Trap Integration](#)[Home](#)[MKLiveStatus Integration](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



MKLiveStatus Integration

[Prev](#)
[Chapter 9. Integration With Other Software](#)
[Next](#)

MKLiveStatus Integration

Introduction

MKLiveStatus is a module written by Mathias Kettner interfacing Icinga (or Nagios) with several addons like NagVis or Thruk. Unless you need a database for storing historical data this might be a good choice because it's quite small and easy to install. Please have a look at the [official website](#) for a complete documentation as the following lines are just a very short guide on how to install and configure MKLiveStatus for Icinga. We assume that you have installed Icinga in /usr/local/icinga.

1. Download the software and compile it (please check the website for the latest version)

```
wget http://mathias-kettner.de/download/mk-livestatus-1.1.10p1.tar.gz
tar xzvf mk-livestatus-1.1.10p1.tar.gz
cd mk-livestatus-1.1.10p1
./configure --prefix=/usr/local/icinga --exec-prefix=/usr/local/icinga
make
cp src/livestatus.o /usr/local/icinga/bin
```

2. Edit *icinga.cfg* to integrate the module. Please make sure that the directory */usr/local/icinga/var/rw* exists and is writable for the Icinga user. It should be the same directory used for the command file (mostly *icinga.cmd*). "live" is a socket and it will only be present during runtime of the module.

```
broker_module=/usr/local/icinga/bin/livestatus.o
/usr/local/icinga/var/rw/live
```

3. Restart Icinga

```
service icinga restart
```

or

```
/etc/init.d/icinga restart
```

4. Check operation

```
ps -ef | grep livestatus
ls -la /usr/local/icinga/var/rw/live
```

If there is no running process and/or no socket then please check the Icinga log file and resolve any errors.

[Prev](#)

[Up](#)

[Next](#)

[TCP Wrapper Integration](#)

[Home](#)

Installation of the
Icinga-Reporting with
JasperServer

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Installation of the Icinga-Reporting with JasperServer

[Prev](#)

Chapter 9. Integration With Other Software

[Next](#)

Installation of the Icinga-Reporting with JasperServer

The Icinga-Reporting is based on the IDOUtils backend and just like the Icinga-Core, it will support all major database platforms.

Here we'll give you a little instruction how to install the Icinga-Reporting with JasperServer.

Prerequisites

You need Icinga-Core and IDOUtils installed and running. Icinga-Reporting also requires a system to run the [JasperServer](#) based on [Tomcat](#).



Note

If you don't have Icinga yet please follow the instructions given in the "[quickstart-idoutils](#)" documentation.

Install the JasperServer

You can use the JasperServer installation binary or the specific WAR-Archive to deploy the software in an exisiting engine.

- Install from binary

You can use the JasperServer-ce-linux-installer provided on [Sourceforge](#) with included the MySQL-Database and the Tomcat Server. Just execute the installer-binary which leads you through the install process.

- Install WAR-Archive

You can deploy the JasperServer-CE WAR into an existing Tomcat6 Server and use the existing Icinga Database for the repository. Here we describe the WAR-installation into a Tomcat6 Server. Also checkout the official JasperServer-Installation Guide [here](#).

Install Tomcat using your OS repositories

Fedora/RHEL/CentOS/openSuSE/SLES

```
#> yum install tomcat6
```

Debian/Ubuntu

```
#> apt-get install tomcat6
```

Install JasperServer CE into your Tomcat Server

- Download jasperserver-ce-x.x.x-bin.zip from [Sourceforge/Jasper](#)
- Follow the installation documentation published on [Sourceforge/Jasper](#) (JasperServer-CE-Install-Guide.pdf)

After sucessfull JasperServer installation the interface is reachable on <http://<yourhost>:8080/jasperserver> and you are able to login with jasperadmin/jasperadmin go the next step. Please change the default password as soon as possible.

In case of any error consult the JasperServer Troubleshooting Guide

Download the Templates

- Download the Icinga-Reporting package from [Sourceforge/Icinga](#)
- **Or** using GIT: you can also download all templates and the actual servlet implementation in our <git://git.icinga.org/icinga-reports.git/>.

Take your clone from the icinga-reports.git to get a fresh branch

```
#> git clone git://git.icinga.org/icinga-reports.git
```

- **Or** download the software using <https://git.icinga.org/index?p=icinga-reports.git;a=snapshot;h=refs/heads/master;sf=tgz.>

Install the Icinga-Reporting- Package

Unpack the downloaded file and copy it to your JasperServer-Directory.

```
#> tar xzvf icinga-reports-xxx.tar.gz
#> cp icinga-reports/ReportPackage/icinga_report_package.zip /opt/jasperserver/scripts/
#> cd /opt/jasperserver/scripts/
```

Importing the whole reporting package requires just one command:

```
#> ./js-import.sh --input-zip icinga_report_package.zip
```

If you want to update an existing repository please use the following command:

```
#> ./js-import.sh --input-zip icinga_report_package.zip --update
```

You can export the whole Icinga part of the repository with the following command:

```
#> ./js-export.sh --uris /Icinga --output-zip icinga_report_package.zip
```

**Note**

If the import scripts fails, please check your jasperserver.xml for the current user and password. You can change the default user and password in <jasperserver-ce-dir>/scripts/config/js.jdbc.properties

**Note**

The export-/import-process is explained in detail at chapter 5.12 of the JasperServer CE-Install-Guide

Install the JAVA classes for SLA- Reports

Generating automatic reports for last week, last month or last year requires to calc the dates automatically for the specific reports. To use that feature you have to install the icinga-reporting.jar archive into the Jasper lib directory. It should be in the WEB-INF folder of your installation. Please don't forget to restart your jasper-server after installation.

Configure quartz the scheduler

The reporting distribution is done via quartz scheduler. To configure the sender mail and your local gateways please edit the following file

<tomcat_home>/webapps/jasperserver/WEB-INF/js.quartz.properties. Please restart the tomcat server to activate the new settings.

```
service tomcat6 restart or /etc/init.d/tomcat restart
```

Configure your database connection

Login to <http://localhost:8080/jasperserver> with jasperadmin/jasperadmin.

After a successfull package installation you will find the datasource here:

/root/Icinga/datasource (be sure that **Refine** contains "changed by anyone").

- Edit the existing datasource and configure your values.
- Test the configuration and save the connection.
- All reports in our package point to this datasource and should be able to run.

**Note**

Remember to clear the searchfield and set the 4 dropdowns to following posistions to find the datasource:

- "Changed by anyone" (already mentioned above)
- "All"
- "Any time"
- "Any schedule"

After this is done, hit search to "apply" the filters. Now you should be able to see the data sources. Default for dropdown filter #2 is Visualization type, which hides data sources.

Different table prefix

If you changed your table prefix during installation you can replace the existing prefixes using the following script.

```
grep -l -r "icinga_" . | xargs sed -i.BAK -e 's/ icinga_/_/g'
grep -l -r "icinga_" . | xargs sed -i.BAK -e 's/^icinga_/_/g'
find . -iname "*BAK" -exec rm -f {} \;
```

Known Bugs: If you don't see any graph labels in a pdf export please switch from OpenJDK to SUNJava.

Figure 9.1. Icinga-Reporting in Icinga-Web

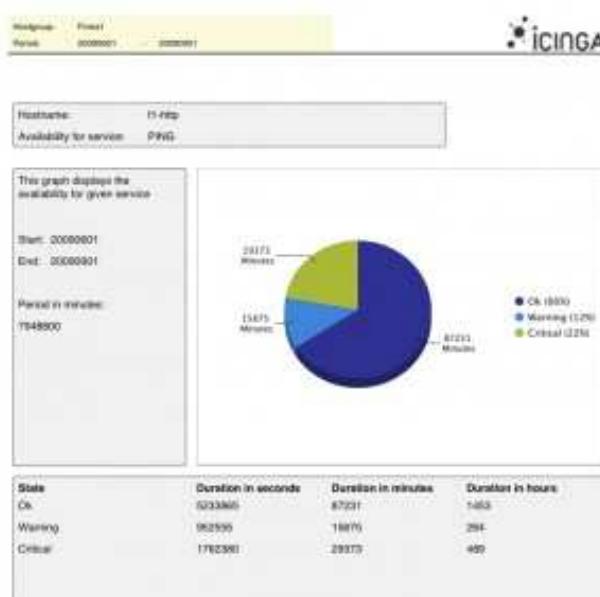
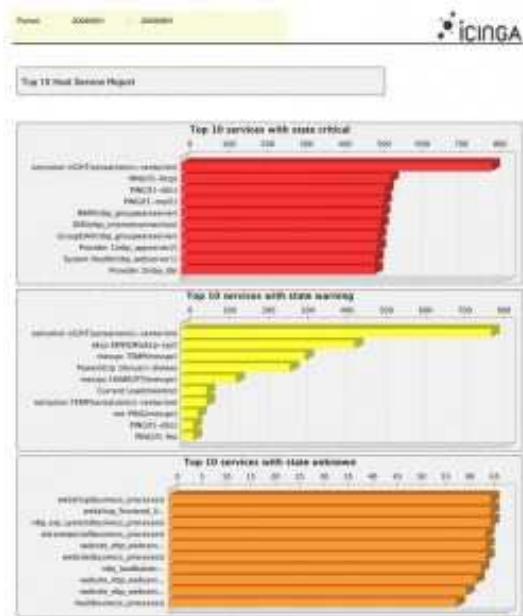


Figure 9.2. Icinga-Reporting TOP10 in Icinga-Web



Note

Integration to Icinga-Web not implemented yet!

Feel free to edit our sample reports with iReport and change these to your needs :)

That's all, you're done!

[Prev](#)

[Up](#)

[Next](#)

MKLiveStatus Integration

[Home](#)

[Chapter 10. Additional software](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 10. Additional software

[Prev](#)

[Next](#)

Chapter 10. Additional software

Table of Contents

[Icinga Addons](#)

[NRPE](#)

[NSCA](#)

[Prev](#)

[Next](#)

Installation of the Icinga-Reporting
with JasperServer

[Home](#)

[Icinga Addons](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Addons

[Prev](#)

Chapter 10. Additional software

[Next](#)

Icinga Addons

Introduction

There are a lot of "addon" software packages that are available for Icinga. Addons can be used to extend Icinga's functionality or integrate Icinga with other applications.

Addons are available for:

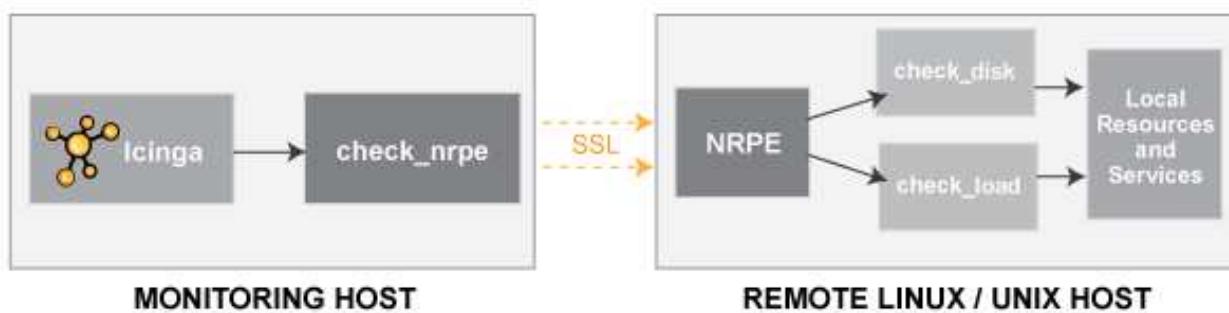
- Managing the config files through a web interface
 - [NConf](#), [NagiosQL](#), [LConf](#), [Lilac](#), ...
- Monitoring remote hosts (*NIX, Windows, etc.)
 - [NSCA](#), [NRPE](#), [check_mk](#), ...
 - [NSClient++](#), ...
- Submitting passive checks from remote hosts
 - [NSClient++](#), [check_mk](#), ...
- Simplifying/extending the notification logic
 - [Business Process Addon](#) ...
- Visualizing the information
 - [PNP4Nagios](#)
 - [NagVis](#)
- Alternative web interfaces
 - [Thruk](#), [MultiSite](#)
- ...and much more

You can find many addons for Icinga by visiting:

- Nagios.org
- SourceForge.net
- <http://www.monitoringexchange.org>

First we'll give a brief introduction to a few of the addons that Ethan Galstad developed for Nagios...

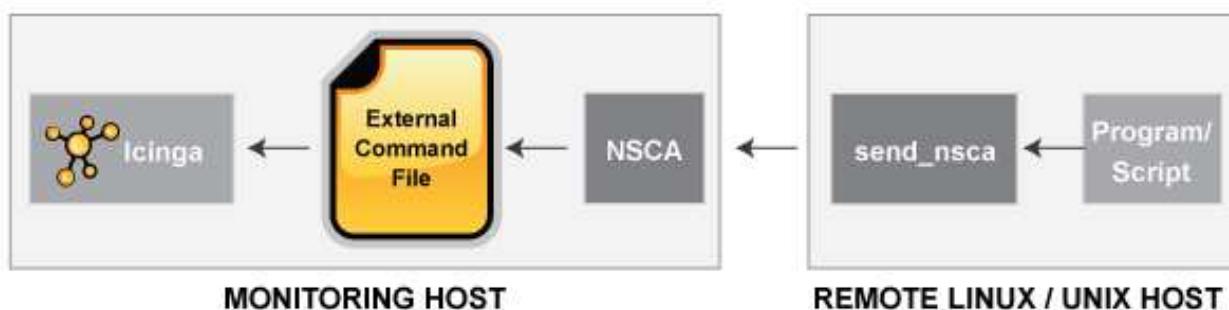
NRPE



NRPE is an addon that allows you to execute [plugins](#) on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host. Similar functionality can be accomplished by using the `check_by_ssh` plugin, although it can impose a higher CPU load on the monitoring machine - especially if you are monitoring hundreds or thousands of hosts.

The NRPE addon can be found at <https://git.icinga.org/?p=icinga-nrpe.git>. You will find the [documentation](#) in the next section.

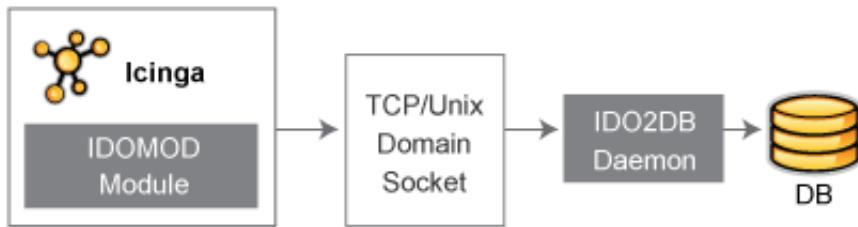
NSCA



NSCA is an addon that allows you to send [passive check](#) results from remote Linux/Unix hosts to the Icinga daemon running on the monitoring server. This is very useful in [distributed](#) and [redundant/failover](#) monitoring setups.

The NSCA addon can be found at <https://git.icinga.org/?p=icinga-nsca.git>. You will find the [documentation](#) in the next section.

IDOUtils



IDOUtils is an addon that allows you to store all status information from Icinga in a database. Multiple instances of Icinga can all store their information in a central database for centralized reporting. In the future it will probably be used as a basis for a PHP based web interface.

The IDOUtils addon install instructions can be found in the [Quickstart-Idoutils](#) and more documentation can be found at <http://docs.icinga.org/>.

[Prev](#)[Up](#)[Next](#)[Chapter 10. Additional software](#)[Home](#)[NRPE](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**NRPE**[Prev](#)**Chapter 10. Additional software**[Next](#)

NRPE

Introduction

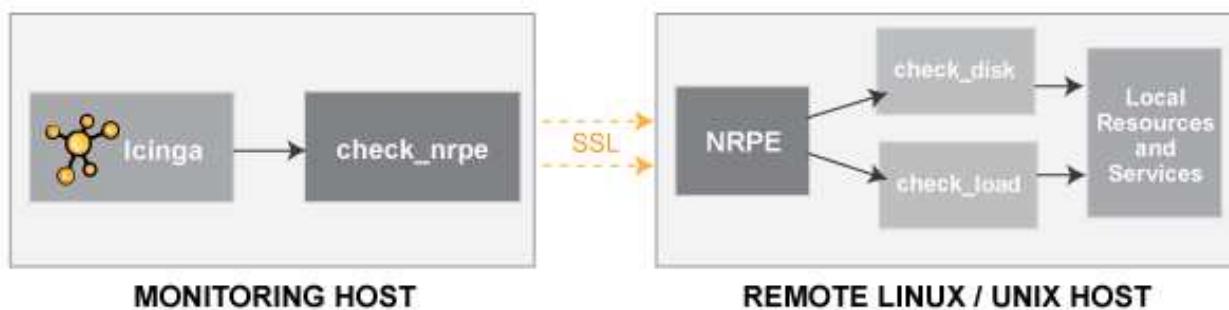
Nagios Remote Plugin Executor (or NRPE for short) is an addon used to execute plugins to monitor "local" resources on remote (Linux/Unix) systems. Some resources cannot (or should not) be monitored via SNMP or using other agents across the network so you have to check them using programs installed locally on the machines to be monitored and transmit the results back to the Icinga server. In contrast to NSCA this is done actively, i.e. initiated by the Icinga server.

**Note**

Using [NSClient++](#) instead of NRPE on the remote host you can execute checks on Windows machines as well.

You can use `check_by_ssh` to execute plugins on remote machines but there is a drawback to this approach. Setting up an SSH session consumes CPU resources on both the local and the remote machine which may become a performance issue if you are monitoring a lot of hosts and/or services this way. Using NRPE is a bit less secure than SSH but in many cases the performance may outweigh the security difference. SSL can be activated though if you need a more secure connection.

Figure 10.1. NRPE



`check_nrpe` is a plugin executed by the local Icinga server like any other plugin. It calls the NRPE process which is running as a daemon on the remote machine. The daemon itself executes the plugin on the same machine and transmits the information gathered back to the `check_nrpe` plugin which in turn delivers it to Icinga.

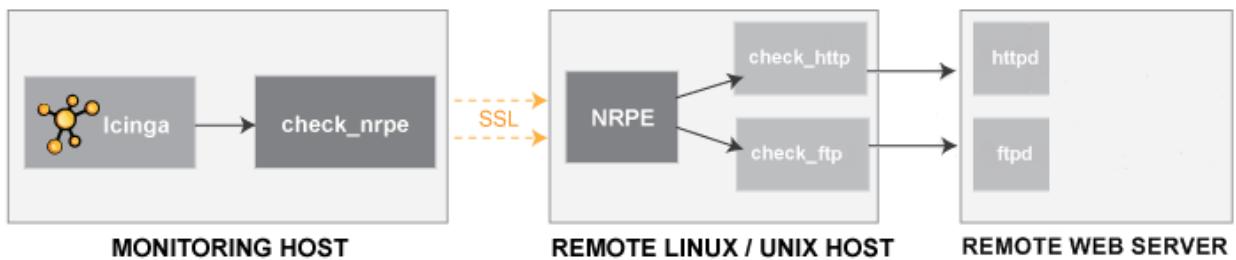


Note

Depending on the CPU / OS of the remote machine you may have to compile NRPE and the plugins on several platforms.

Using NRPE you will mostly monitor resources located on the same machine like CPU load, disk space, memory usage, processes running etc. but it can also be used to check resources which may not be reachable directly from the monitoring server itself. The machine running the NRPE daemon is acting as a relay in this case.

Figure 10.2. NRPE remote



The following instructions are partially based on documentation found in the original NRPE package by Ethan Galstad.

Prerequisites

- Icinga should be up and running on the monitoring server
- a C-compiler (like gcc) is installed on the local host. If not:

```
#> yum install gcc      # RHEL / Fedora / CentOS
#> apt-get install gcc  # Debian / Ubuntu
#> zypper install gcc   # SLES / openSuSE (or use YaST instead)
```

- openssl is (optionally) installed on both the local host. If not:

```
#> yum install openssl openssl-devel      # RHEL / Fedora / CentOS
#> apt-get install openssl openssl-devel  # Debian / Ubuntu
#> zypper install openssl openssl-devel    # SLES / openSuSE (or use YaST instead)
```

Download

Download the software from the [Icinga git repository](https://git.icinga.org/?p=icinga-nrpe.git;a=snapshot;h=HEAD;sf=tgz) by clicking on "tar.gz" or use

```
#> cd /usr/src
#> wget "https://git.icinga.org/?p=icinga-nrpe.git;a=snapshot;h=HEAD;sf=tgz" -O nrpe.tgz
#> tar xzf nrpe.tgz
```

or use the current version from the git repository

```
#> git clone git://git.icinga.org/icinga-nrpe.git
```

Optional changes

The maximum length of data to be transmitted is set to 2,048 bytes, the maximal length of plugin output is set to 512 bytes. If that is not sufficient then you have to alter the appropriate value in `icinga-nrpe/include/common.h` (and recompile Icinga!)

```
#define MAX_INPUT_BUFFER      2048    /* max size of most buffers we use */
#define MAX_PLUGINOUTPUT_LENGTH 512
```

Please keep in mind that you have to recompile the programs if you change these values at a later stage.

Due to the setting of the following define in `include/common.h` (in Icinga core) the max. value cannot exceed 8,192 bytes.

```
#define MAX_EXTERNAL_COMMAND_LENGTH     8192    /* max length of an external command */
```

Compile on the Icinga server

Change to the newly created directory and call configure and make

```
#> cd icinga-nrpe
#> ./configure
#> make all
#> make install-plugin
```



Note

If you want to use SSL at a later stage then you have to use "`./configure --enable-ssl`" instead. There are other options to specify the location of SSL files if they are not found automatically.

If the user or group running the daemon deviate from "icinga" or the port to be used is not the default 5666 you can use several options to specify different values (`--with-nrpe-user=<user>`, `--with-nrpe-group=<group>`, `--with-nrpe-port=<port>`). For a complete list of the options available call "`./configure -h`". "make install-plugin" will copy `check_nrpe` to the plugin directory.



Note

You may want to check if SSL is included using "`ldd src/check_nrpe`" and "`ldd src/nrpe`".

First test

Startup the daemon and call the plugin

```
#> /usr/src/icinga-nrpe/src/nrpe -n \
  -c /usr/src/icinga-nrpe/sample-config/nrpe.cfg -d
#> /usr/local/icinga/libexec/check_nrpe -H 127.0.0.1 -n
```

This should return the version of NRPE. If you receive the message "CHECK_NRPE: Error receiving data from daemon" the monitoring server was not found in `nrpe.cfg` (directive `allowed_hosts`). Multiple IP addresses are separated by commas.

Stop the daemon

```
#> kill `ps -ef | grep "sample-config/nrpe.cfg" | grep -v grep | awk '{print $2}'`
```

Remote system(s)

The configuration and installation on the Icinga server is finished so far. The second part has to be done on the remote host(s) running the NRPE daemon which listens for incoming requests, executing them and returning the results to the Icinga server.

Prerequisites on the remote host(s)

- Make sure that the necessary plugins are available. Take a look at the [quickstart guide](#) if you are unsure on how to install them.
- You can copy the folder `icinga-nrpe` including the sub-directories from the Icinga server. One way would be

```
#> cd /usr/src/
#> scp -pr <Icinga-server>:/$PWD/icinga-nrpe .
```



Note

If the architecture of your remote host differs from your Icinga server you will have to recompile the sources. This is true if you are using different CPUs (i386/Itanium/PA-RISC ...) and/or different OS versions (32-Bit/64-Bit). If this is the case then you have to install a C-Compiler and OpenSSL (if you want to use SSL) before you can start to compile.

```
#> cd icinga-nrpe
#> make distclean
#> ./configure      # please use the same options as on the Icinga server
#> make all
```

Edit the config file `sample-config/nrpe.cfg` and change the setting of "allowed_hosts=<IP address>" to the IP address of your Icinga server. Multiple IP addresses are separated by commas.

Second test

Startup the daemon on the remote host

```
#> /usr/src/icinga-nrpe/src/nrpe -n \
-c /usr/src/icinga-nrpe/sample-config/nrpe.cfg -d
```

and execute the plugin on the Icinga server once more, this time using the IP address of the remote host

```
#> /usr/local/icinga/libexec/check_nrpe -H <IP remote host> -n
```

This should return the version of NRPE. If you receive the message "CHECK_NRPE: Error receiving data from daemon" the specified host was not found in `nrpe.cfg` (directive `allowed_hosts`) on the remote host.

Stop the daemon on the remote host

```
#> kill `ps -ef | grep "sample-config/nrpe.cfg" | grep -v grep | awk '{print $2}'`
```

Installation on the remote host

Independent from the method the NRPE process is running on the remote host you need a config file containing the commands to be called. You install it issuing

```
#> make install-daemon-config
```

There are two ways to run the `nrpe` process, one as a standalone daemon, the other using `xinetd` (which is recommended).

- **nrpe daemon**

First install the daemon

```
#> make install-daemon
```

If you choose to use xinetd the daemon will be started automatically. Otherwise you have to start the daemon manually

```
#> /usr/local/icinga/bin/nrpe -c /usr/local/icinga/etc/nrpe.cfg
```

- **inetd/xinetd**

If you want the daemon to be started by (x)inetd you have to extend /etc/services, alter/copy another file and restart (x)inetd. If the package is not installed then please do the following

```
#> yum install xinetd      # RHEL / Fedora / CentOS
#> apt-get install xinetd  # Debian / Ubuntu
#> zypper install xinetd   # SLES / openSuSE (or use YaST instead)
```



Note

The setting of "server_port" specified in nrpe.cfg is ignored when you use inetd/xinetd.

```
#> echo "nrpe 5666/tcp # nrpe" >> /etc/services
```

Depending on the superserver which is installed on your system there are three alternatives

- **inetd WITH tcpwrappers**

Add entries to your /etc/hosts.allow and /etc/hosts.deny file to enable TCP wrapper protection for the nrpe service. This is optional, although highly recommended. Add "nrpe stream tcp nowait <user> /usr/sbin/tcpd <nrpe-binary> -c <nrpe-cfg> --inetd" to /etc/inetd.conf, e.g.

```
#> echo "nrpe stream tcp nowait icinga /usr/sbin/tcpd /usr/local/icinga/bin/nrpe \
-c /usr/local/icinga/etc/nrpe.cfg --inetd" >> /etc/inetd.conf
#> /etc/init.d/inetd restart
```

- **inetd WITHOUT tcpwrappers**

Add "nrpe stream tcp nowait <user> <nrpe-binary> -c <nrpe-cfg> --inetd" to /etc/inetd.conf, e.g.

```
#> echo "nrpe stream tcp nowait icinga /usr/local/icinga/bin/nrpe \
-c /usr/local/icinga/etc/nrpe.cfg --inetd" >> /etc/inetd.conf
#> /etc/init.d/inetd restart
```

- **xinetd (recommended)**

Consider editing the config file nrpe.xinetd in the sample-config folder and replacing the address following <only_from> by the IP address of the Icinga server (where check_nrpe will be running). Multiple IP addresses are separated by spaces.

Add entries to your `/etc/hosts.allow` and `/etc/hosts.deny` file to enable TCP wrapper protection for the nrpe service. This is optional, although highly recommended. Copy the file to your xinetd folder and restart the xinetd process

```
#> make install-xinetd
#> /etc/init.d/xinetd restart
```

Third test

Switch to the Icinga server, change to the Icinga user and run another test

```
#> su - icinga
$> /usr/local/icinga/libexec/check_nrpe -H <IP remote server>
```

This should return the version of NRPE another time. If this test fails then there is no sense in continuing. Instead verify the settings in `nrpe.cfg/nrpe.xinet` on the remote server. Check for messages in your syslog (e.g. `/var/log/messages`) on the remote server as well.

Troubleshooting

Check that the nrpe process is running on the remote server

- when installed as standalone daemon

```
#> ps -ef | grep -v grep | grep nrpe
```

If not, then

- start it as mentioned above
- check that the file `/usr/local/icinga/etc/nrpe.cfg` is present
- the `allowed_hosts` directive in the file `/usr/local/icinga/etc/nrpe.cfg` contains an entry for the IP address of the Icinga server. Multiple IP addresses are separated by commas.

- when installed using xinetd

```
#> netstat -at | grep -v grep | grep nrpe
```

The output should be showing something like

```
tcp 0 0 *:nrpe *:* LISTEN
```

If not then verify that

- you added the nrpe entry to your `/etc/services` file
- the file `/etc/xinetd.d/nrpe` is present
- the `only_from` directive in the file `/etc/xinetd.d/nrpe` contains an entry for the IP address of the Icinga server. Multiple IP addresses are separated by spaces.
- xinetd is installed and started
- the system log files don't show any errors about xinetd and/or nrpe and fix any problems that are reported

Activate "debug=1" in `nrpe.cfg`, restart the daemon (if applicable) and look for messages in the syslog / `nrpe.log`.

Security

Read the `SECURITY` file for more information on the security risks of running NRPE, along with an explanation of what kind of protection the encryption provides you.

Definition of local checks

Some things have been predefined in `etc/nrpe.cfg` on the remote host

```
# command[<command_name>]=<command_line>
command[check_users]=/usr/local/icinga/libexec/check_users -w 5 -c 10
command[check_load]=/usr/local/icinga/libexec/check_load -w 1.5,1.1,0.9 -c 3.0,2.2,1.9
command[check_hda1]=/usr/local/icinga/libexec/check_disk -w 20% -c 10% -p /dev/hda1
command[check_zombie_procs]=/usr/local/icinga/libexec/check_procs -w 5 -c 10 -s Z
command[check_total_procs]=/usr/local/icinga/libexec/check_procs -w 150 -c 200
```

The first line shows the general format

String	Description
command	tag showing that the following is a definition for a command
<command_name>	link between the command definition on the Icinga server and the command on the remote host
<command_line>	call of the plugin including all necessary arguments

Definitions on the Icinga server

Now we switch over to the Icinga server to create some object definitions. First add a command definition to your configuration (unless you already have it). As usual the name of the config file is up to you but most people have a file called `commands.cfg`.

```
define command{
  command_name      check_nrpe
  command_line      $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$}
```

We assume that you already have a host definition like the following

```
define host{
  use           generic-host      ; Inherit default values from a template
  host_name     remotehost        ; The name we're giving to this server
  alias         Linux Host        ; A longer name for the server
  address       192.168.0.1       ; IP address of the server
}
```

These example service definitions will use the sample commands shown above.

The following service will monitor the number of currently logged in users on the remote host

```
define service{
  use           generic-service
  host_name     remotehost
  service_description Current Users
  check_command  check_nrpe!check_users
}
```

"check_nrpe" is the link between the service directive "check_command" and the directive "command_name" in the command definition on the Icinga server. The "command_line" in the command definition shows that "check_nrpe" is called. "check_users" is passed as the first argument. The nrpe process on the remote host takes this argument and searches for an appropriate definition in nrpe.cfg. The command is executed and the result is transferred back to the check_nrpe-plugin.

The following service will monitor the CPU load on the remote host

```
define service{
    use           generic-service
    host_name     remotehost
    service_description CPU Load
    check_command  check_nrpe!check_load
}
```

The following service will monitor the free drive space on /dev/hda1 on the remote host

```
define service{
    use           generic-service
    host_name     remotehost
    service_description /dev/hda1 Free Space
    check_command  check_nrpe!check_hda1
}
```

The following service will monitor the total number of processes on the remote host

```
define service{
    use           generic-service
    host_name     remotehost
    service_description Total Processes
    check_command  check_nrpe!check_total_procs
}
```

The following service will monitor the number of zombie processes on the remote host

```
define service{
    use           generic-service
    host_name     remotehost
    service_description Zombie Processes
    check_command  check_nrpe!check_zombie_procs
}
```

Restart Icinga to include the definitions in your running configuration

```
#> /etc/init.d/icinga restart
```

After some time your plugins should have been called.

More Troubleshooting

Some errors during the initial setup have been mentioned already. Unfortunately you may encounter others errors. Below you'll find hints for some of the more common errors with the NRPE addon.

- "NRPE: Command timed out after x seconds"

The command that was run by the NRPE daemon did not finish executing within the specified time. You can increase the timeout for commands by editing the NRPE configuration file and changing the value of the command_timeout variable. Use the -t command line option to specify a longer timeout for the check_nrpe plugin. The following example will increase the timeout to 30 seconds:

```
/usr/local/icinga/libexec/check_nrpe -H localhost -c somecommand -t 30
```

If you're running the NRPE daemon as a standalone daemon (and not under inetd or xinetd), you'll need to restart it in order for the new timeout to be recognised.

- "Connection refused or timed out"

This error can indicate several things:

- There is a firewall that is blocking the communication between the monitoring host (which runs the check_nrpe plugin) and the remote host (which runs the NRPE daemon). Verify that the firewall rules (e.g. iptables) that are running on the remote host allow for communication and make sure there isn't a physical firewall that is located between the monitoring host and the remote host.
- If you are using the standalone daemon: The IP address specified in `nrpe.cfg` (`allowed_hosts=...`) on the remote server doesn't match the IP address of the monitoring server. If it does then you might have forgotten to restart the daemon after the last change.
- If you are using the xinetd version: The IP address specified in `/etc/xinetd.d/nrpe` (`only_from=...`) on the remote server doesn't match the IP address of the monitoring server. If it does then you might have forgotten to restart the xinetd process after the last change.
- The NRPE daemon is not installed or running on the remote host. Verify that the NRPE daemon is running as a standalone daemon or under inetd/xinetd with one of the following commands:

```
ps axuw | grep nrpe      # if run as standalone daemon
netstat -at | grep nrpe  # if run via xinetd
```

- "CHECK_NRPE: Received 0 bytes from daemon. Check the remote server logs for an error message."

First thing you should do is check the remote server logs for an error message. Seriously. :-) This error could be due to the following problem:

- The check_nrpe plugin was unable to complete an SSL handshake with the NRPE daemon. An error message in the logs should indicate whether or not this was the case. Check the versions of OpenSSL that are installed on the monitoring host and remote host. If you're running a commercial version of SSL on the remote host, there might be some compatibility problems.

- "NRPE: Unable to read output"

This error indicates that the command that was run by the NRPE daemon did not return any character output. This could be an indication of the following problems:

- The path of the plugin to be run is incorrect on the remote host. If you change the definition in `nrpe.cfg`, remember to restart the daemon.
- The plugin that is specified in the command line is malfunctioning. Run the command line manually to make sure the plugin returns some kind of text output. DON'T run the command as root!

- "NRPE: Command 'x' not defined"

The command 'x' was not defined in the NRPE configuration file on the remote host. Please add the command definition for x. See the existing command definitions in the NRPE configuration file for more information on doing this. If you're running the NRPE daemon as a standalone daemon (and not under inetd or xinetd), you'll need to restart it in order for the new command to be recognised.

If you still have problems then set "debug=1" in nrpe.cfg on the remote host. Remember to restart the NRPE process if it is running as a standalone daemon. Execute the check on the monitoring server. Afterwards you should see debugging information in the syslog (e.g. /var/log/messages) which might help resolving the problem.

You might as well get help using one of the mailing lists or forums (<http://www.icinga.org/community/get-help/>).

Upgrading

- Upgrading the Icinga server

Download the software

```
#> cd /usr/src
#> wget "https://git.icinga.org/?p=icinga-nrpe.git;a=snapshot;h=HEAD;sf=tgz" -O nrpe.tgz
#> tar xzf nrpe.tgz
```

or use the current version from the git repository

```
#> git clone git://git.icinga.org/icinga-nrpe.git
```

Then compile the software and install the plugin

```
#> cd icinga-nrpe
#> make distclean
#> ./configure      # use the same options as during the first run
#> make all
#> make install-plugin
```

- Upgrading the remote host

Download the software

```
#> cd /usr/src
#> wget "https://git.icinga.org/?p=icinga-nrpe.git;a=snapshot;h=HEAD;sf=tgz" -O nrpe.tgz
#> tar xzf nrpe.tgz
```

or use the current version from the git repository

```
#> git clone git://git.icinga.org/icinga-nrpe.git
```

Then compile the software and install the daemon process

```
#> cd icinga-nrpe
#> make distclean
#> ./configure      # use the same options as during the first run
#> make all
### kill the process if running as standalone daemon
#> kill `ps -ef | grep "sample-config/nrpe.cfg" | grep -v grep | awk '{print $2}'`'
#> make install-daemon
### start the daemon if running as standalone daemon
#> /usr/src/icinga-nrpe/src/nrpe -n \
    -c /usr/src/icinga-nrpe/sample-config/nrpe.cfg -d
```

[Prev](#)[Up](#)[Next](#)[Icinga Addons](#)[Home](#)[NSCA](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



NSCA

[Prev](#)

Chapter 10. Additional software

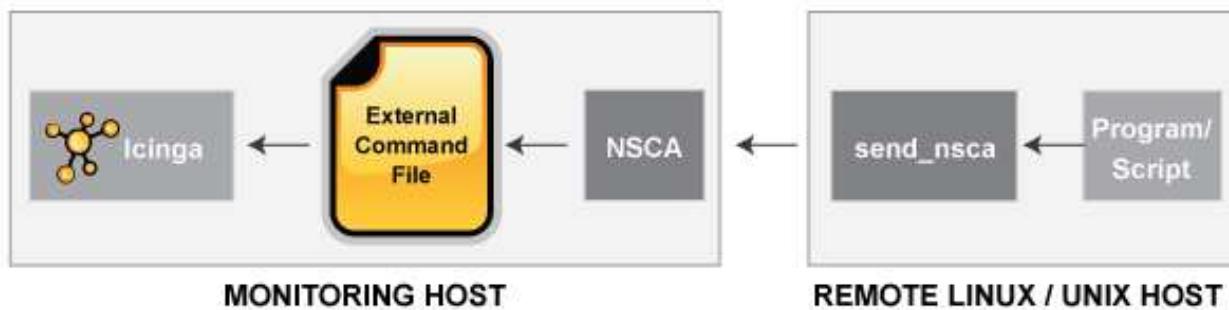
[Next](#)

NSCA

Introduction

Nagios Service Check Acceptor (or NSCA for short) is an addon to transmit check results from one system to another. It consists of two modules: the sender (`send_nsca`) and the receiver (`nsca`). The data is sent in a secure manner. Despite its name host check results are transmitted as well.

Figure 10.3. NSCA



NSCA is running as a daemon on the Icinga server. It listens for information sent from remote machines using the `send_nsca` program (on Unix/Linux machines) or NSClient++ (often used on Windows machines). The data will be encrypted using the method specified in `send_nsca.cfg` (or `nsc.ini` in the case of NSClient++). The daemon will validate the data in a *very* basic manner by decrypting the information using the password which is stored in the local `nsca.cfg` file. If the data looks like it was encrypted using the same password then the daemon will try to output the data as external commands into the local Icinga command pipe.

The following instructions are mainly based on the README included in the NSCA package.

Prerequisites

- Icinga should be up and running
- `check_external_commands = 1` should be set in `icinga.cfg`
- `command_check_interval = <n>[s]` should be set in `icinga.cfg`

- `log_passive_checks = 1` should be set in `icinga.cfg` while testing, otherwise there will be no messages about incoming passive checks
- libmcrypt and libmcrypt-devel packages are installed (which may be named similar depending on the distribution used), otherwise use one of the following commands to install the packages:

```
#> apt-get install libmcrypt libmcrypt-devel # Debian / Ubuntu
#> yum install libmcrypt libmcrypt-devel      # RHEL / Fedora / CentOS
#> zypper install libmcrypt libmcrypt-devel    # SLES / OpenSuSE (or use Yast instead)
```

Download and compile

Download the software from the [Icinga git repository](https://git.icinga.org/?p=icinga-nsca.git;a=snapshot;h=HEAD;sf=tgz) by clicking on "tar.gz" or use

```
#> wget "https://git.icinga.org/?p=icinga-nsca.git;a=snapshot;h=HEAD;sf=tgz" -O nsca.tgz
#> tar xzf nsca.tgz
```

or use the current version from the git repository

```
#> git clone git://git.icinga.org/icinga-nsca.git
```

The maximum length of data to be transmitted is set to 2,048 bytes, the maximal length of plugin output is set to 512 bytes. If that is not sufficient then you have to alter the appropriate value in `icinga-nsca/include/common.h`.

```
#define MAX_INPUT_BUFFER          2048     /* max size of most buffers we use */
#define MAX_PLUGINOUTPUT_LENGTH   512
```

Please keep in mind that you have to recompile the programs if you change this values at a later stage.

Due to the setting of the following define in `include/common.h` (in Icinga core) the max. value cannot exceed 8,192 bytes.

```
#define MAX_EXTERNAL_COMMAND_LENGTH      8192     /* max length of an external command */
```

After setting the owner change to the newly created directory and call configure and make

```
#> chown -R icinga icinga-nsca
#> cd icinga-nsca
#> ./configure
#> make all
```

Afterwards two programs (`send` and `send_nsca`) are created in the `src` directory.



Caution

If the libmcrypt packages are not found `./configure` will complain but will NOT end with a non-zero return code so please check `config.log` using

```
#> grep mcrypt.h: config.log
```

This command should return no lines.

If the libmcrypt modules are not found, the user or group deviate from "icinga" or the port to be used is not the default 5667 you can use several options to specify different values. Call `./configure -h` to see the options available.

After changing to "nsca_tests" you can try to execute "./runtests". Please note that these tests require several Perl modules described in the README file.

Customise

The sample-config directory contains nsca.cfg and send_nsca.cfg. At least the settings of the directives "password" and "encryption_method" / "decryption_method" should be reviewed/ altered before copying the files. Remember to set the same password in all copies of these config files. If you want to have different passwords on different remote servers you have to run multiple nsca daemons on the Icinga server listening on different ports. This doesn't work if you start the daemon using inetd/xinetd.

First test

Switch to the Icinga user and run a first test

```
#> su - icinga
$> cd /usr/src/icinga-nsca/src
$> ./nsca -c ../sample-config/nsca.cfg
$> echo -e "A\tB\tC\tD\n" | ./send_nsca -H localhost -c ../sample-config/send_nsca.cfg
$> exit
```

This should return the message "1 data packet(s) sent to host successfully.". In the first place it only means that send_nsca and nsca are able to communicate with each other on the local host using the sample config files as it works perfectly without a running Icinga instance. But it is important nevertheless: If this test fails then there is no sense in continuing. Instead verify the settings in nsca.cfg and send_nsca.cfg. Check for messages in your syslog (e.g. /var/log/messages) as well.

If the prerequisites are fulfilled then you should see some warnings in icinga.log complaining that host "A" and service "B" are not found in the Icinga configuration. This means that nsca has sufficient permissions to write to the Icinga command file. Check if the nsca daemon and Icinga are running with different users if there are no messages in icinga.log. If there are no messages check the setting of [log_passive_checks](#) in icinga.cfg.

Install

"make install" will do nothing (yet) so you have to copy some files to new locations. The following commands will copy the nsca module to the directory where the Icinga binary is located and the config file to the Icinga config folder. We assume that you installed Icinga using one of the quickstart guides.

```
#> cp -p nsca /usr/local/icinga/bin/
#> cp ../sample-config/nsca.cfg /usr/local/icinga/etc/
```

- **nsca daemon**

If you choose to use xinetd the daemon will be started automatically. Otherwise you have to start the daemon manually after switching to the Icinga user (which you might have done already during "First test")

```
#> su - icinga
$> /usr/local/icinga/bin/nsca -c /usr/local/icinga/etc/nsca.cfg
```

- **inetd/xinetd**

If you want the daemon to be started by (x)inetd you have to extend /etc/services, alter/copy another file and restart (x)inetd. Don't forget to stop the nsca daemon you started during "First test".

**Note**

The setting of "server_port" specified in nsca.cfg is ignored when you use inetd/xinetd.

```
#> kill < /var/run/nsca.pid
#> echo "nsca 5667/tcp # NSCA" >> /etc/services
```

Depending on the superserver which is installed on your system there are three alternatives

- inetd WITH tcpwrappers

Add entries to your /etc/hosts.allow and /etc/hosts.deny file to enable TCP wrapper protection for the nsca service. This is optional, although highly recommended. Add "nsca stream tcp nowait <user> /usr/sbin/tcpd <nsca-binary> -c <nsca-cfg> --inetd" to /etc/inetd.conf, e.g.

```
#> echo "nsca stream tcp nowait icinga /usr/sbin/tcpd /usr/local/icinga/bin/nsca -c /usr/local/icinga/etc/nsca.cfg --inetd" >> /etc/inetd.conf
#> /etc/init.d/inetd restart
```

- inetd WITHOUT tcpwrappers

Add "nsca stream tcp nowait <user> <nsca-binary> -c <nsca-cfg> --inetd" to /etc/inetd.conf, e.g.

```
#> echo "nsca stream tcp nowait icinga /usr/local/icinga/bin/nsca -c /usr/local/icinga/etc/nsca.cfg --inetd" >> /etc/inetd.conf
#> /etc/init.d/inetd restart
```

- xinetd

Consider editing the config file nsca.xinetd in the sample-config folder and replacing <ipaddress1>... by the IP addresses of your client machines (where send_nsca will be running). This only works if xinetd was compiled with support for tcpwrappers. If you are running DHCP then this will not work and you should delete this line.

Add entries to your /etc/hosts.allow and /etc/hosts.deny file to enable TCP wrapper protection for the nsca service. This is optional, although highly recommended. Copy the file to your xinetd folder

```
#> cp -p ../sample-config/nsca.xinetd /etc/xinetd.d/
#> /etc/init.d/xinetd restart
```

Remote system(s)

You're done on the local system but apparently send_nsca has to be copied to the remote system(s).

Please note that you have to compile send_nsca for the platform it should run upon so you might have to install the libmcrypt packages and configure/make as described above on multiple servers.

Copy files

You're free where to place binary and config file. We assume that you have a directory structure similar to the Icinga server

```
#> scp -p <Icinga server>:/usr/local/icinga-nsca/src/send_nsca /usr/local/icinga/bin/
#> scp -p <Icinga server>:/usr/local/icinga-nsca/sample-config/send_nsca.cfg /usr/local/icinga/etc/
```

Second test

Now you can rerun the test on the remote system(s)

```
#> su - icinga
$> echo -e "A\tB\tC\tD\n" | /usr/local/icinga/bin/send_nsca -H <Icinga server> -c /usr/local/icinga/etc/send_nsca.cfg
```

This as well should return the message "1 data packet(s) sent to host successfully." and there should be warnings in icinga.log on the Icinga server similar to the ones mentioned above. If there are no messages check the setting of [log_passive_checks](#) in icinga.cfg.

Troubleshooting

If the daemon is not permitted to write to the command pipe then the data will be lost! You should run the daemon using the same user you specified for Icinga.

If the object (host and/or service) is not included in the running configuration then the data will be rejected.

Host name (and service description, if applicable) are case sensitive and have to match the definition in Icinga.

Check if you specified the same password in nsca.cfg and send_nsca.cfg. Otherwise the transmission will fail.

Check if you used the same encryption/decryption method. Otherwise the transmission will fail.

Check if your firewall setting allow the communication on the port specified (default is 5667)

If you are using xinetd check if the IP address(es) specified after "only_from=" match to the remote system(s) or remove this line (and restart xinetd)

Activate "debug=1" in nsca.cfg, restart the daemon (if applicable) and look for messages in the syslog / nsca.log.

Security

There are some security implications with allowing remote clients to provide check results to Icinga. Because of this, you have the option of encrypting the packets that the NSCA client sends to the NSCA daemon. Read the SECURITY file for more information on the security risks of running NSCA, along with an explanation of what kind of protection the encryption provides you.

Operation

send_nsca is used to send the check results from the remote host to the Icinga server. The syntax differs depending on the object type. To submit service check information use

```
<host_name>[tab]<svc_description>[tab]<return_code>[tab]<plugin_output>[newline]
```

where:

<host_name>=short name of the host that the service is associated with (as defined in the host_name directive of the service definition)

<svc_description>=description of the service (as defined in the service_description directive of the service definition)

<return_code>=numeric return code (0,1,2,3 as explained [here](#))

<plugin_output>=output from host/service check

Host check information is submitted in a similiar fashion - just leave out the service description:

<host_name>[tab]<return_code>[tab]<plugin_output>[newline]

Integration into Icinga

So far you only created some prerequisites for running passive checks but didn't define any host or service actually using the functionality.

Although you only receive check results you still need to specify the "check_command" directive in your definitions. There is a plugin called "check_dummy" which can be used for this purpose. You may have to add a command definition if it's not already included. The second argument is optional and might contain an explanatory text.

```
define command{
    command_name check_dummy
    command_line $USER1$/check_dummy $ARG1$ $ARG2$
}
```

You may want to create a service template. The host template might look similiar to that (just replace "service" by "host")

```
define service{
    use           generic-service ; template to inherit from
    name          passive-service ; name of this template
    active_checks_enabled 0       ; no active checks
    passive_checks_enabled 1       ; allow passive checks
    check_command   check_dummy!0 ; use "check_dummy", RC=0 (OK)
    check_period    24x7          ; check active all the time
    check_freshness 0             ; don't check if check result is "stale"
    register       0             ; this is a template, not a real service
}
```

Using the template above the service definition might look like

```
define service{
    use           passive-service ; template to inherit from
    host_name     remotehost      ; host where send_nsca is located
    service_description Diskspace   ; service to be checked
}
```

Restart Icinga to include the definitions in your running configuration

```
#> /etc/init.d/icinga restart
```

Change to your remote host, switch to the Icinga user and execute send_nsca replacing <Icinga server> by the IP address of the server running Icinga

```
#> su - icinga
$> echo -e "remotehost\tDiskspace\t0\t/var=78%\n" | /usr/local/icinga/bin/send_nsca -H <Icinga server> -c /usr/local/icinga/etc/send_nsca.cfg
```

Please keep in mind to specify the host name and service description exactly as defined in your Icinga definition (because the processing is case sensitive). Otherwise you'll get messages in icinga.log that the object could not be found. If there are no messages check the setting of [log_passive_checks](#) in icinga.cfg.

After a short while you should see messages in icinga.log showing that the information submitted was processed. You should see the information in the classic UI as well turning the service state from "Pending" to "OK" and containing the data you specified.

[Prev](#)[Up](#)[Next](#)[NRPE](#)[Home](#)[Chapter 11. Development](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 11. Development

[Prev](#)

[Next](#)

Chapter 11. Development

Table of Contents

- [Icinga Plugin API](#)
 - [Developing Plugins For Use With Embedded Perl](#)
 - [List of External Commands](#)
 - [Installation and use of the Icinga API](#)
 - [The Icinga-Web REST API](#)
-

[Prev](#)

[Next](#)

[NSCA](#)

[Home](#)

[Icinga Plugin API](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Icinga Plugin API

[Prev](#)

Chapter 11. Development

[Next](#)

Icinga Plugin API

Other Resources

If you're looking at writing your own plugins for Icinga, please make sure to visit these other resources:

- The official [Nagios plugin project website](#)
- The official [Nagios plugin development guidelines](#)

Plugin Overview

Scripts and executables must do two things (at a minimum) in order to function as Icinga plugins:

- Exit with one of several possible return values
- Return at least one line of text output to STDOUT

The inner workings of your plugin are unimportant to Icinga. Your plugin could check the status of a TCP port, run a database query, check disk free space, or do whatever else it needs to check something. The details will depend on what needs to be checked - that's up to you.

Return Code

Icinga determines the status of a host or service by evaluating the return code from plugins. The following tables shows a list of valid return codes, along with their corresponding service or host states.

Plugin Return Code	Service State	Host State
0	OK	UP
1	WARNING	UP or DOWN/UNREACHABLE*
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

Plugin Output Spec

At a minimum, plugins should return at least one of text output but they can optionally return multiple lines of output. Plugins may also return optional performance data that can be processed by external applications. The basic format for plugin output is shown below:

TEXT OUTPUT | OPTIONAL PERFDATA

LONG TEXT LINE 1 LONG TEXT LINE 2 ... LONG TEXT LINE N | **PERFDATA LINE 2**

PERFDATA LINE 3 ... PERFDATA LINE N

The performance data (shown in orange) is optional. If a plugin returns performance data in its output, it must separate the performance data from the other text output using a pipe (|) symbol. Additional lines of long text output (shown in blue) are also optional.

Plugin Output Examples

Let's see some examples of possible plugin output...

Case 1: One line of output (text only)

Assume we have a plugin that returns one line of output that looks like this:

DISK OK - free space: / 3326 MB (56%); If this plugin was used to perform a service check, the entire line of output will be stored in the **\$SERVICEOUTPUT\$** macro.

Case 2: One line of output (text and perfdta)

A plugin can return optional performance data for use by external applications. To do this, the performance data must be separated from the text output with a pipe (|) symbol like such:

DISK OK - free space: / 3326 MB (56%); | /=2643MB;5948;5958;0;5968

If this plugin was used to perform a service check, the red portion of output (left of the pipe separator) will be stored in the **\$SERVICEOUTPUT\$** macro and the orange portion of output (right of the pipe separator) will be stored in the **\$SERVICEPERFDATA\$** macro.

Case 3: Multiple lines of output (text and perfdta)

A plugin optionally returns multiple lines of both text output and perfdta, like such:

DISK OK - free space: / 3326 MB (56%);

| /=2643MB;5948;5958;0;5968

/ 15272 MB (77%);

/boot 68 MB (69%);

/home 69357 MB (27%);

/var/log 819 MB (84%); | /boot=68MB;88;93;0;98

/home=69357MB;253404;253409;0;253414

/var/log=818MB;970;975;0;980

If this plugin was used to perform a service check, the **red** portion of first line of output (left of the pipe separator) will be stored in the **\$SERVICEOUTPUT\$** macro. The **orange** portions of the first and subsequent lines are concatenated (with spaces) are stored in the **\$SERVICEPERFDATA\$** macro. The **blue** portions of the 2nd - 5th lines of output will be concatenated (with escaped newlines) and stored in the **\$LONGSERVICEOUTPUT\$** macro.

The final contents of each macro are listed below:

Macro	Value
\$SERVICEOUTPUT\$	DISK OK - free space: / 3326 MB (56%);
\$SERVICEPERFDATA\$	/=2643MB;5948;5958;0;5968 /boot=68MB;88;93;0;98 /home=69357MB;253404;253409;0;253414 /var=log=818MB;970;975;0;980
\$LONGSERVICEOUTPUT\$	/ 15272 MB (77%);\n/boot 68 MB (69%);\n/var/log 819 MB (84%);

With regards to multiple lines of output, you have the following options for returning performance data:

- You can choose to return no performance data whatsoever
- You can return performance data on the first line only
- You can return performance data only in subsequent lines (after the first)
- You can return performance data in both the first line and subsequent lines (as shown above)

Plugin Output Length Restrictions

Icinga will only read the first 8 KB of data that a plugin returns. This is done in order to prevent runaway plugins from dumping megs or gigs of data back to Icinga. This 8 KB output limit is fairly easy to change if you need. Simply edit the value of the `MAX_PLUGIN_OUTPUT_LENGTH` definition in the `include/icinga.h.in` file of the source code distribution and recompile Icinga. If you increase the 8k cap by modifying this value make sure that you also increase the value of `MAX_EXTERNAL_COMMAND_LENGTH` in `include/common.h` before you compile to allow for passive checks results of this length to be received through the external command file.

Examples

If you're looking for some example plugins to study, we would recommend that you download the official Nagios plugins and look through the code for various C, Perl, and shell script plugins. Information on obtaining the official Nagios plugins can be found [here](#).

Perl Plugins

Icinga features an optional [embedded Perl interpreter](#) which can speed up the execution of Perl plugins. More information on developing Perl plugins for use with the embedded Perl interpreter can be found [here](#).

[Prev](#)

[Up](#)

[Next](#)

Chapter 11. Development

[Home](#)

Developing Plugins For Use With
Embedded Perl

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Developing Plugins For Use With Embedded Perl

[Prev](#)

Chapter 11. Development

[Next](#)

Developing Plugins For Use With Embedded Perl

Introduction

Stanley Hopcroft has worked with the embedded Perl interpreter quite a bit and has commented on the advantages/disadvantages of using it. He has also given several helpful hints on creating Perl plugins that work properly with the embedded interpreter. The majority of this documentation comes from his comments.

It should be noted that "ePN", as used in this documentation, refers to embedded Perl Nagios, or if you prefer, Icinga compiled with an embedded Perl interpreter.

Target Audience

- Average Perl developers; those with an appreciation of the language's powerful features without knowledge of internals or an in-depth knowledge of those features.
- Those with a utilitarian appreciation rather than a great depth of understanding.
- If you are happy with Perl objects, name management, data structures, and the debugger, that's probably sufficient.

Things you should do when developing a Perl Plugin (ePN or not)

- Always always generate some output
- Use 'use utils' and import the stuff it exports (\$TIMEOUT %ERRORS &print_revision &support)
- Have a look at how the standard Perl plugins do their stuff e.g.
 - Always exit with \$ERRORS{CRITICAL}, \$ERRORS{OK}, etc.
 - Use getopt to read command line arguments
 - Manage timeouts
 - Call print_usage (supplied by you) when there are no command line arguments
 - Use standard switch names (e.g. H 'host', V 'version')

Things you must do to develop a Perl plugin for ePN

1. <DATA> can not be used; use here documents instead e.g.

```
my $data = <<DATA;
portmapper 100000
portmap 100000
sunrpc 100000
rpcbind 100000
rstatd 100001
rstat 100001
rup 100001
..
DATA

%proignum = map { my($a, $b) = split; ($a, $b) } split(/\n/, $data) ;
```

2. BEGIN blocks will not work as you expect. May be best to avoid.
3. Ensure that it is squeaky clean at compile time i.e.

- use strict
- use perl -w (other switches [T notably] may not help)
- use perl -c

4. Avoid lexical variables (my) with global scope as a means of passing __variable__ data into subroutines. In fact this is __fatal__ if the subroutine is called by the plugin more than once when the check is run. Such subroutines act as 'closures' that lock the global lexicals first value into subsequent calls of the subroutine. If however, your global is read-only (a complicated structure for example) this is not a problem. What Bekman [recommends you do instead](#), is any of the following:

- make the subroutine anonymous and call it via a code ref e.g.

turn this	into
<pre>my \$x = 1 ; sub a { .. Process \$x ... } . . a ; \$x = 2 a ;</pre>	<pre>my \$x = 1 ; \$a_cr = sub { ... Process \$x ... }; . . &\$a_cr ; \$x = 2 ; &\$a_cr ;</pre>
# anon closures __always__ rebind the current lexical value	

- put the global lexical and the subroutine using it in their own package (as an object or a module)
- pass info to subs as references or aliases (\\$lex_var or \$_[n])
- replace lexicals with package globals and exclude them from 'use strict' objections with 'use vars qw(global1 global2 ..)'

5. Be aware of where you can get more information.

Useful information can be had from the usual suspects (the O'Reilly books, plus Damien Conways "Object Oriented Perl") but for the really useful stuff in the right context start at Stas Bekman's mod_perl guide at <http://perl.apache.org/guide/>.

This wonderful book sized document has nothing whatsoever about Icinga, but all about writing Perl programs for the embedded Perl interpreter in Apache (i.e. Doug MacEachern's mod_perl).

The perlembed manpage is essential for context and encouragement.

On the basis that Lincoln Stein and Doug MacEachern know a thing or two about Perl and embedding Perl, their book 'Writing Apache Modules with Perl and C' is almost certainly worth looking at.

6. Be aware that your plugin may return strange values with an ePN and that this is likely to be caused by the problem in item #4 above

7. Be prepared to debug via:

- having a test ePN and
- adding print statements to your plugin to display variable values to STDERR (can't use STDOUT)
- adding print statements to p1.pl to display what ePN thinks your plugin is before it tries to run it (vi)
- running the ePN in foreground mode (probably in conjunction with the former recommendations)
- use the 'Deparse' module on your plugin to see how the parser has optimised it and what the interpreter will actually get. (see 'Constants in Perl' by Sean M. Burke, The Perl Journal, Fall 2001)

```
perl -MO::Deparse <your_program>
```

8. Be aware of what ePN is transforming your plugin too, and if all else fails try and debug the transformed version.

As you can see below p1.pl rewrites your plugin as a subroutine called 'hndlr' in the package named 'Embed::<something_related_to_your_plugin_file_name>'.

Your plugin may be expecting command line arguments in @ARGV so pl.pl also assigns @_ to @ARGV.

This in turn gets 'eval' ed and if the eval raises an error (any parse error and run error), the plugin gets chucked out.

The following output shows how a test ePN transformed the *check_rpc* plugin before attempting to execute it. Most of the code from the actual plugin is not shown, as we are interested in only the transformations that the ePN has made to the plugin). For clarity, transformations are shown in red:

```
package main;
use subs 'CORE::GLOBAL::exit';
sub CORE::GLOBAL::exit { die "ExitTrap: $_[0]
(Embed::check_5frpc)"; }

package Embed::check_5frpc; sub hndlr { shift(@_);
@ARGV=@_;
#! /usr/bin/perl -w
#
# check_rpc plugin for Nagios
#
```

```

# usage:
#   check_rpc host service
#
# Check if an rpc serice is registered and running
# using rpcinfo - $proto $host $proignum 2>&1 | ";
#
# Use these hosts.cfg entries as examples
#
# command[check_nfs]=/some/path/libexec/check_rpc $HOSTADDRESS$ nfs
# service[check_nfs]=NFS;24x7;3;5;5;unix-admin;60;24x7;1;1;1;;check_rpc
#
# initial version: 3 May 2000 by Truongchinh Nguyen and Karl DeBisschop
# current status: $Revision: 1.20 $
#
# Copyright Notice: GPL
#
... rest of plugin code goes here (it was removed for brevity) ...
}

```

9. Don't use 'use diagnostics' in a plugin run by your production ePN. We think it causes all the Perl plugins to return CRITICAL.
 10. Consider using a mini embedded Perl C program to check your plugin. This is not sufficient to guarantee your plugin will perform Ok with an ePN but if the plugin fails this test it will certainly fail with your ePN. [A sample mini ePN is included in the *contrib*/ directory of the Icinga distribution for use in testing Perl plugins. Change to the contrib/ directory and type 'make mini_epn' to compile it. It must be executed from the same directory that the p1.pl file resides in (this file is distributed with Icinga).]
-

[Prev](#)[Up](#)[Next](#)[Icinga Plugin API](#)[Home](#)[List of External Commands](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



List of External Commands

[Prev](#)

[Chapter 11. Development](#)

[Next](#)

List of External Commands

Below you will find descriptions of each external command. It is quite easy to send external commands to Icinga and you only need to change the last line of the example code to use a different command (so we omitted these lines in the description).

Example:

```
#!/bin/sh
# Adjust variables to fit your environment as necessary.

now=`date +%s`
commandfile='/usr/local/icinga/var/rw/icinga.cmd'

/bin/printf "[%lu] ACKNOWLEDGE_HOST_PROBLEM;Host1;1;1;1;John Doe;Some comment\n" $now > $commandfile
```

ACKNOWLEDGE_HOST_PROBLEM

ACKNOWLEDGE_HOST_PROBLEM; <host_name>;<sticky>;<notify>;<persistent>;<author>;<comment>

Allows you to acknowledge the current problem for the specified host. By acknowledging the current problem, future notifications (for the same host state) are disabled. If the "sticky" option is set to two (2), the acknowledgement will remain until the host returns to an UP state. Otherwise the acknowledgement will automatically be removed when the host changes state. If the "notify" option is set to one (1), a notification will be sent out to contacts indicating that the current host problem has been acknowledged. If the "persistent" option is set to one (1), the comment associated with the acknowledgement will survive across restarts of the Icinga process. If not, the comment will be deleted the next time Icinga restarts.

ACKNOWLEDGE_SVC_PROBLEM

ACKNOWLEDGE_SVC_PROBLEM; <host_name>;<service_description>;<sticky>;<notify>;<persistent>;<author>;<comment>

Allows you to acknowledge the current problem for the specified service. By acknowledging the current problem, future notifications (for the same servicestate) are disabled. If the "sticky" option is set to two (2), the acknowledgement will remain until the service returns to an OK state. Otherwise the acknowledgement will automatically be removed when the service changes state. If the "notify" option is set to one (1), a notification will be sent out to contacts indicating that the current service problem has been acknowledged. If the "persistent" option is set to one (1), the comment associated with the acknowledgement will survive across restarts of the Icinga process. If not, the comment will be deleted the next time Icinga restarts.

ADD_HOST_COMMENT

```
ADD_HOST_COMMENT ;<host_name>;<persistent>;<author>;<comment>
```

Adds a comment to a particular host. If the "persistent" field is set to zero (0), the comment will be deleted the next time Icinga is restarted. Otherwise, the comment will persist across program restarts until it is deleted manually.

ADD_SVC_COMMENT

```
ADD_SVC_COMMENT ;<host_name>;<service_description>;<persistent>;<author>;<comment>
```

Adds a comment to a particular service. If the "persistent" field is set to zero (0), the comment will be deleted the next time Icinga is restarted. Otherwise, the comment will persist across program restarts until it is deleted manually.

CHANGE_CONTACT_HOST_NOTIFICATION_TIMEPERIOD

```
CHANGE_CONTACT_HOST_NOTIFICATION_TIMEPERIOD ;<contact_name>;<notification_timeperiod>
```

Changes the host notification timeperiod for a particular contact to what is specified by the "notification_timeperiod" option. The "notification_timeperiod" option should be the short name of the timeperiod that is to be used as the contact's host notification timeperiod. The timeperiod must have been configured in Icinga before it was last (re)started.

CHANGE_CONTACT_MODATTR

```
CHANGE_CONTACT_MODATTR ;<contact_name>;<value>
```

This command changes the modified attributes value for the specified contact. Modified attributes values are used by Icinga to determine which object properties should be retained across program restarts. Thus, modifying the value of the attributes can affect data retention. This is an advanced option and should only be used by people who are intimately familiar with the data retention logic in Icinga.

CHANGE_CONTACT_MODHATTR

```
CHANGE_CONTACT_MODHATTR ;<contact_name>;<value>
```

This command changes the modified host attributes value for the specified contact. Modified attributes values are used by Icinga to determine which object properties should be retained across program restarts. Thus, modifying the value of the attributes can affect data retention. This is an advanced option and should only be used by people who are intimately familiar with the data retention logic in Icinga.

CHANGE_CONTACT_MODSATTR

```
CHANGE_CONTACT_MODSATTR ;<contact_name>;<value>
```

This command changes the modified service attributes value for the specified contact. Modified attributes values are used by Icinga to determine which object properties should be retained across program restarts. Thus, modifying the value of the attributes can affect data retention. This is an advanced option and should only be used by people who are intimately familiar with the data retention logic in Icinga.

CHANGE_CONTACT_SVC_NOTIFICATION_TIMEPERIOD

```
CHANGE_CONTACT_SVC_NOTIFICATION_TIMEPERIOD ; <contact_name> ; <notification_timeperiod>
```

Changes the service notification timeperiod for a particular contact to what is specified by the "notification_timeperiod" option. The "notification_timeperiod" option should be the short name of the timeperiod that is to be used as the contact's service notification timeperiod. The timeperiod must have been configured in Icinga before it was last (re)started.

CHANGE_CUSTOM_CONTACT_VAR

```
CHANGE_CUSTOM_CONTACT_VAR ; <contact_name> ; <varname> ; <varvalue>
```

Changes the value of a custom contact variable.

CHANGE_CUSTOM_HOST_VAR

```
CHANGE_CUSTOM_HOST_VAR ; <host_name> ; <varname> ; <varvalue>
```

Changes the value of a custom host variable.

CHANGE_CUSTOM_SVC_VAR

```
CHANGE_CUSTOM_SVC_VAR ; <host_name> ; <service_description> ; <varname> ; <varvalue>
```

Changes the value of a custom service variable.

CHANGE_GLOBAL_HOST_EVENT_HANDLER

```
CHANGE_GLOBAL_HOST_EVENT_HANDLER ; <event_handler_command>
```

Changes the global host event handler command to be that specified by the "event_handler_command" option. The "event_handler_command" option specifies the short name of the command that should be used as the new host event handler. The command must have been configured in Icinga before it was last (re)started.

CHANGE_GLOBAL_SVC_EVENT_HANDLER

```
CHANGE_GLOBAL_SVC_EVENT_HANDLER ; <event_handler_command>
```

Changes the global service event handler command to be that specified by the "event_handler_command" option. The "event_handler_command" option specifies the short name of the command that should be used as the new service event handler. The command must have been configured in Icinga before it was last (re)started.

CHANGE_HOST_CHECK_COMMAND

```
CHANGE_HOST_CHECK_COMMAND ; <host_name> ; <check_command>
```

Changes the check command for a particular host to be that specified by the "check_command" option. The "check_command" option specifies the short name of the command that should be used as the new host check command. The command must have been configured in Icinga before it was last (re)started.

CHANGE_HOST_CHECK_TIMEPERIOD

CHANGE_HOST_CHECK_TIMEPERIOD ; <host_name>;<timeperiod>

Changes the valid check period for the specified host.

CHANGE_HOST_EVENT_HANDLER

CHANGE_HOST_EVENT_HANDLER ; <host_name>;<event_handler_command>

Changes the event handler command for a particular host to be that specified by the "event_handler_command" option. The "event_handler_command" option specifies the short name of the command that should be used as the new host event handler. The command must have been configured in Icinga before it was last (re)started.

CHANGE_HOST_MODATTR

CHANGE_HOST_MODATTR ; <host_name>;<value>

This command changes the modified attributes value for the specified host. Modified attributes values are used by Icinga to determine which object properties should be retained across program restarts. Thus, modifying the value of the attributes can affect data retention. This is an advanced option and should only be used by people who are intimately familiar with the data retention logic in Icinga.

CHANGE_HOST_NOTIFICATION_TIMEPERIOD

CHANGE_HOST_NOTIFICATION_TIMEPERIOD ; <host_name>;<notification_timeperiod>

Changes the notification timeperiod for a particular host to what is specified by the "notification_timeperiod" option. The "notification_timeperiod" option should be the short name of the timeperiod that is to be used as the service notification timeperiod. The timeperiod must have been configured in Icinga before it was last (re)started.

CHANGE_MAX_HOST_CHECK_ATTEMPTS

CHANGE_MAX_HOST_CHECK_ATTEMPTS ; <host_name>;<check_attempts>

Changes the maximum number of check attempts (retries) for a particular host.

CHANGE_MAX_SVC_CHECK_ATTEMPTS

CHANGE_MAX_SVC_CHECK_ATTEMPTS ; <host_name>;<service_description>;<check_attempts>

Changes the maximum number of check attempts (retries) for a particular service.

CHANGE_NORMAL_HOST_CHECK_INTERVAL

CHANGE_NORMAL_HOST_CHECK_INTERVAL ; <host_name>;<check_interval>

Changes the normal (regularly scheduled) check interval for a particular host.

CHANGE_NORMAL_SVC_CHECK_INTERVAL

CHANGE_NORMAL_SVC_CHECK_INTERVAL ; <host_name>;<service_description>;<check_interval>

Changes the normal (regularly scheduled) check interval for a particular service

CHANGE_RETRY_HOST_CHECK_INTERVAL

CHANGE_RETRY_HOST_CHECK_INTERVAL ; <host_name>;<service_description>;<check_interval>

Changes the retry check interval for a particular host.

CHANGE_RETRY_SVC_CHECK_INTERVAL

CHANGE_RETRY_SVC_CHECK_INTERVAL ; <host_name>;<service_description>;<check_interval>

Changes the retry check interval for a particular service.

CHANGE_SVC_CHECK_COMMAND

CHANGE_SVC_CHECK_COMMAND ; <host_name>;<service_description>;<check_command>

Changes the check command for a particular service to be that specified by the "check_command" option. The "check_command" option specifies the short name of the command that should be used as the new service check command. The command must have been configured in Icinga before it was last (re)started.

CHANGE_SVC_CHECK_TIMEPERIOD

CHANGE_SVC_CHECK_TIMEPERIOD ; <host_name>;<service_description>;<check_timeperiod>

Changes the check timeperiod for a particular service to what is specified by the "check_timeperiod" option. The "check_timeperiod" option should be the short name of the timeperiod that is to be used as the service check timeperiod. The timeperiod must have been configured in Icinga before it was last (re)started.

CHANGE_SVC_EVENT_HANDLER

CHANGE_SVC_EVENT_HANDLER ; <host_name>;<service_description>;<event_handler_command>

Changes the event handler command for a particular service to be that specified by the "event_handler_command" option. The "event_handler_command" option specifies the short name of the command that should be used as the new service event handler. The command must have been configured in Icinga before it was last (re)started.

CHANGE_SVC_MODATTR

CHANGE_SVC_MODATTR ; <host_name>;<service_description>;<value>

This command changes the modified attributes value for the specified service. Modified attributes values are used by Icinga to determine which object properties should be retained across program restarts. Thus, modifying the value of the attributes can affect data retention. This is an advanced option and should only be used by people who are intimately familiar with the data retention logic in Icinga.

CHANGE_SVC_NOTIFICATION_TIMEPERIOD

CHANGE_SVC_NOTIFICATION_TIMEPERIOD ; <host_name>;<service_description>;<notification_timeperiod>

Changes the notification timeperiod for a particular service to what is specified by the "notification_timeperiod" option. The "notification_timeperiod" option should be the short name of the timeperiod that is to be used as the service notification timeperiod. The timeperiod must have been configured in Icinga before it was last (re)started.

DEL_ALL_HOST_COMMENTS

```
DEL_ALL_HOST_COMMENTS ;<host_name>
```

Deletes all comments associated with a particular host.

DEL_ALL_SVC_COMMENTS

```
DEL_ALL_SVC_COMMENTS ;<host_name>;<service_description>
```

Deletes all comments associated with a particular service.

DEL_HOST_COMMENT

```
DEL_HOST_COMMENT ;<comment_id>
```

Deletes a host comment. The id number of the comment that is to be deleted must be specified.

DEL_DOWNTIME_BY_HOST_NAME

```
DEL_DOWNTIME_BY_HOST_NAME ;<host_name>
```

Deletes the host downtime entry for the host whose `host_name` matches the "host_name" argument. If the downtime is currently in effect, the host will come out of scheduled downtime (as long as there are no other overlapping active downtime entries).

**Note**

This command is available starting with Icinga 1.4. Changes provided by the [Opsview](#) team.

DEL_DOWNTIME_BY_HOSTGROUP_NAME

```
DEL_DOWNTIME_BY_HOSTGROUP_NAME ;<hostgroup_name>
```

Deletes the host downtime entries of all hosts of the host group matching the "hostgroup_name" argument. If the downtime is currently in effect, the host will come out of scheduled downtime (as long as there are no other overlapping active downtime entries).

**Note**

This command is available starting with Icinga 1.4. Changes provided by the [Opsview](#) team.

DEL_DOWNTIME_BY_START_TIME_COMMENT

```
DEL_DOWNTIME_BY_START_TIME_COMMENT ;<start time[;comment_id]>
```

Deletes downtimes with start times matching the timestamp specified by the "start time" argument and an optional comment ID.

**Note**

This command is available starting with Icinga 1.4. Changes provided by the [Opsview](#) team.

DEL_HOST_DOWNTIME

```
DEL_HOST_DOWNTIME ;<downtime_id>
```

Deletes the host downtime entry that has an ID number matching the "downtime_id" argument. If the downtime is currently in effect, the host will come out of scheduled downtime (as long as there are no other overlapping active downtime entries).

DEL_SVC_COMMENT

```
DEL_SVC_COMMENT ;<comment_id>
```

Deletes a service comment. The id number of the comment that is to be deleted must be specified.

DEL_SVC_DOWNTIME

```
DEL_SVC_DOWNTIME ;<downtime_id>
```

Deletes the service downtime entry that has an ID number matching the "downtime_id" argument. If the downtime is currently in effect, the service will come out of scheduled downtime (as long as there are no other overlapping active downtime entries).

DELAY_HOST_NOTIFICATION

```
DELAY_HOST_NOTIFICATION ;<host_name>;<notification_time>
```

Delays the next notification for a particular host until "notification_time". The "notification_time" argument is specified in time_t format (seconds since the UNIX epoch). Note that this will only have an affect if the host stays in the same problem state that it is currently in. If the host changes to another state, a new notification may go out before the time you specify in the "notification_time" argument.

DELAY_SVC_NOTIFICATION

```
DELAY_SVC_NOTIFICATION ;<host_name>;<service_description>;<notification_time>
```

Delays the next notification for a parciular service until "notification_time". The "notification_time" argument is specified in time_t format (seconds since the UNIX epoch). Note that this will only have an affect if the service stays in the same problem state that it is currently in. If the service changes to another state, a new notification may go out before the time you specify in the "notification_time" argument.

DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST

```
DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST ;<host_name>
```

Disables notifications for all hosts and services "beyond" (e.g. on all child hosts of) the specified host. The current notification setting for the specified host is not affected.

DISABLE_CONTACT_HOST_NOTIFICATIONS

```
DISABLE_CONTACT_HOST_NOTIFICATIONS ;<contact_name>
```

Disables host notifications for a particular contact.

DISABLE_CONTACT_SVC_NOTIFICATIONS

DISABLE_CONTACT_SVC_NOTIFICATIONS ; <contact_name>

Disables service notifications for a particular contact.

DISABLE_CONTACTGROUP_HOST_NOTIFICATIONS

DISABLE_CONTACTGROUP_HOST_NOTIFICATIONS ; <contactgroup_name>

Disables host notifications for all contacts in a particular contactgroup.

DISABLE_CONTACTGROUP_SVC_NOTIFICATIONS

DISABLE_CONTACTGROUP_SVC_NOTIFICATIONS ; <contactgroup_name>

Disables service notifications for all contacts in a particular contactgroup.

DISABLE_EVENT_HANDLERS

DISABLE_EVENT_HANDLERS

Disables host and service event handlers on a program-wide basis.

DISABLE_FAILURE_PREDICTION

DISABLE_FAILURE_PREDICTION

Disables failure prediction on a program-wide basis. This feature is not currently implemented in Icinga.

DISABLE_FLAP_DETECTION

DISABLE_FLAP_DETECTION

Disables host and service flap detection on a program-wide basis.

DISABLE_HOST_AND_CHILD_NOTIFICATIONS

DISABLE_HOST_AND_CHILD_NOTIFICATIONS ; <host_name>

Disables notifications for the specified host, as well as all hosts "beyond" (e.g. on all child hosts of) the specified host.

DISABLE_HOST_CHECK

DISABLE_HOST_CHECK ; <host_name>

Disables (regularly scheduled and on-demand) active checks of the specified host.

DISABLE_HOST_EVENT_HANDLER

DISABLE_HOST_EVENT_HANDLER ; <host_name>

Disables the event handler for the specified host.

DISABLE_HOST_FLAP_DETECTION

DISABLE_HOST_FLAP_DETECTION ;<host_name>

Disables flap detection for the specified host.

DISABLE_HOST_FRESHNESS_CHECKS

DISABLE_HOST_FRESHNESS_CHECKS

Disables freshness checks of all hosts on a program-wide basis.

DISABLE_HOST_NOTIFICATIONS

DISABLE_HOST_NOTIFICATIONS ;<host_name>

Disables notifications for a particular host.

DISABLE_HOST_SVC_CHECKS

DISABLE_HOST_SVC_CHECKS ;<host_name>

Disables active checks of all services on the specified host.

DISABLE_HOST_SVC_NOTIFICATIONS

DISABLE_HOST_SVC_NOTIFICATIONS ;<host_name>

Disables notifications for all services on the specified host.

DISABLE_HOSTGROUP_HOST_CHECKS

DISABLE_HOSTGROUP_HOST_CHECKS ;<hostgroup_name>

Disables active checks for all hosts in a particular hostgroup.

DISABLE_HOSTGROUP_HOST_NOTIFICATIONS

DISABLE_HOSTGROUP_HOST_NOTIFICATIONS ;<hostgroup_name>

Disables notifications for all hosts in a particular hostgroup. This does not disable notifications for the services associated with the hosts in the hostgroup - see the DISABLE_HOSTGROUP_SVC_NOTIFICATIONS command for that.

DISABLE_HOSTGROUP_PASSIVE_HOST_CHECKS

DISABLE_HOSTGROUP_PASSIVE_HOST_CHECKS ;<hostgroup_name>

Disables passive checks for all hosts in a particular hostgroup.

DISABLE_HOSTGROUP_PASSIVE_SVC_CHECKS

DISABLE_HOSTGROUP_PASSIVE_SVC_CHECKS ;<hostgroup_name>

Disables passive checks for all services associated with hosts in a particular hostgroup.

DISABLE_HOSTGROUP_SVC_CHECKS

DISABLE_HOSTGROUP_SVC_CHECKS ;<hostgroup_name>

Disables active checks for all services associated with hosts in a particular hostgroup.

DISABLE_HOSTGROUP_SVC_NOTIFICATIONS

DISABLE_HOSTGROUP_SVC_NOTIFICATIONS ;<hostgroup_name>

Disables notifications for all services associated with hosts in a particular hostgroup. This does not disable notifications for the hosts in the hostgroup - see the **DISABLE_HOSTGROUP_HOST_NOTIFICATIONS** command for that.

DISABLE_NOTIFICATIONS

DISABLE_NOTIFICATIONS

Disables host and service notifications on a program-wide basis.

DISABLE_PASSIVE_HOST_CHECKS

DISABLE_PASSIVE_HOST_CHECKS ;<host_name>

Disables acceptance and processing of passive host checks for the specified host.

DISABLE_PASSIVE_SVC_CHECKS

DISABLE_PASSIVE_SVC_CHECKS ;<host_name>;<service_description>

Disables passive checks for the specified service.

DISABLE_PERFORMANCE_DATA

DISABLE_PERFORMANCE_DATA

Disables the processing of host and service performance data on a program-wide basis.

DISABLE_SERVICE_FLAP_DETECTION

DISABLE_SERVICE_FLAP_DETECTION ;<host_name>;<service_description>

Disables flap detection for the specified service.

DISABLE_SERVICE_FRESHNESS_CHECKS

DISABLE_SERVICE_FRESHNESS_CHECKS

Disables freshness checks of all services on a program-wide basis.

DISABLE_SERVICEGROUP_HOST_CHECKS

DISABLE_SERVICEGROUP_HOST_CHECKS ;<servicegroup_name>

Disables active checks for all hosts that have services that are members of a particular servicegroup.

DISABLE_SERVICEGROUP_HOST_NOTIFICATIONS

DISABLE_SERVICEGROUP_HOST_NOTIFICATIONS ;<servicegroup_name>

Disables notifications for all hosts that have services that are members of a particular servicegroup.

DISABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS

DISABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS ;<servicegroup_name>

Disables the acceptance and processing of passive checks for all hosts that have services that are members of a particular service group.

DISABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS

DISABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS ;<servicegroup_name>

Disables the acceptance and processing of passive checks for all services in a particular servicegroup.

DISABLE_SERVICEGROUP_SVC_CHECKS

DISABLE_SERVICEGROUP_SVC_CHECKS ;<servicegroup_name>

Disables active checks for all services in a particular servicegroup.

DISABLE_SERVICEGROUP_SVC_NOTIFICATIONS

DISABLE_SERVICEGROUP_SVC_NOTIFICATIONS ;<servicegroup_name>

Disables notifications for all services that are members of a particular servicegroup.

DISABLE_SVC_CHECK

DISABLE_SVC_CHECK ;<host_name>;<service_description>

Disables active checks for a particular service.

DISABLE_SVC_EVENT_HANDLER

DISABLE_SVC_EVENT_HANDLER ;<host_name>;<service_description>

Disables the event handler for the specified service.

DISABLE_SVC_FLAP_DETECTION

DISABLE_SVC_FLAP_DETECTION ;<host_name>;<service_description>

Disables flap detection for the specified service.

DISABLE_SVC_NOTIFICATIONS

DISABLE_SVC_NOTIFICATIONS ;<host_name>;<service_description>

Disables notifications for a particular service.

ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST

ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST ;<host_name>

Enables notifications for all hosts and services "beyond" (e.g. on all child hosts of) the specified host. The current notification setting for the specified host is not affected. Notifications will only be sent out for these hosts and services if notifications are also enabled on a program-wide basis.

ENABLE_CONTACT_HOST_NOTIFICATIONS

`ENABLE_CONTACT_HOST_NOTIFICATIONS ; <contact_name>`

Enables host notifications for a particular contact.

ENABLE_CONTACT_SVC_NOTIFICATIONS

`ENABLE_CONTACT_SVC_NOTIFICATIONS ; <contact_name>`

Disables service notifications for a particular contact.

ENABLE_CONTACTGROUP_HOST_NOTIFICATIONS

`ENABLE_CONTACTGROUP_HOST_NOTIFICATIONS ; <contactgroup_name>`

Enables host notifications for all contacts in a particular contactgroup.

ENABLE_CONTACTGROUP_SVC_NOTIFICATIONS

`ENABLE_CONTACTGROUP_SVC_NOTIFICATIONS ; <contactgroup_name>`

Enables service notifications for all contacts in a particular contactgroup.

ENABLE_EVENT_HANDLERS

`ENABLE_EVENT_HANDLERS`

Enables host and service event handlers on a program-wide basis.

ENABLE_FAILURE_PREDICTION

`ENABLE_FAILURE_PREDICTION`

Enables failure prediction on a program-wide basis. This feature is not currently implemented in Icinga.

ENABLE_FLAP_DETECTION

`ENABLE_FLAP_DETECTION`

Enables host and service flap detection on a program-wide basis.

ENABLE_HOST_AND_CHILD_NOTIFICATIONS

`ENABLE_HOST_AND_CHILD_NOTIFICATIONS ; <host_name>`

Enables notifications for the specified host, as well as all hosts "beyond" (e.g. on all child hosts of) the specified host. Notifications will only be sent out for these hosts if notifications are also enabled on a program-wide basis.

ENABLE_HOST_CHECK

ENABLE_HOST_CHECK ;<host_name>

Enables (regularly scheduled and on-demand) active checks of the specified host.

ENABLE_HOST_EVENT_HANDLER

ENABLE_HOST_EVENT_HANDLER ;<host_name>

Enables the event handler for the specified host.

ENABLE_HOST_FLAP_DETECTION

ENABLE_HOST_FLAP_DETECTION ;<host_name>

Enables flap detection for the specified host. In order for the flap detection algorithms to be run for the host, flap detection must be enabled on a program-wide basis as well.

ENABLE_HOST_FRESHNESS_CHECKS

ENABLE_HOST_FRESHNESS_CHECKS

Enables freshness checks of all hosts on a program-wide basis. Individual hosts that have freshness checks disabled will not be checked for freshness.

ENABLE_HOST_NOTIFICATIONS

ENABLE_HOST_NOTIFICATIONS ;<host_name>

Enables notifications for a particular host. Notifications will be sent out for the host only if notifications are enabled on a program-wide basis as well.

ENABLE_HOST_SVC_CHECKS

ENABLE_HOST_SVC_CHECKS ;<host_name>

Enables active checks of all services on the specified host.

ENABLE_HOST_SVC_NOTIFICATIONS

ENABLE_HOST_SVC_NOTIFICATIONS ;<host_name>

Enables notifications for all services on the specified host. Note that notifications will not be sent out if notifications are disabled on a program-wide basis.

ENABLE_HOSTGROUP_HOST_CHECKS

ENABLE_HOSTGROUP_HOST_CHECKS ;<hostgroup_name>

Enables active checks for all hosts in a particular hostgroup.

ENABLE_HOSTGROUP_HOST_NOTIFICATIONS

ENABLE_HOSTGROUP_HOST_NOTIFICATIONS ;<hostgroup_name>

Enables notifications for all hosts in a particular hostgroup. This does not enable notifications for the services associated with the hosts in the hostgroup - see the **ENABLE_HOSTGROUP_SVC_NOTIFICATIONS** command for that. In order for notifications to be sent out for these hosts, notifications must be enabled on a program-wide basis as well.

ENABLE_HOSTGROUP_PASSIVE_HOST_CHECKS

```
ENABLE_HOSTGROUP_PASSIVE_HOST_CHECKS ; <hostgroup_name>
```

Enables passive checks for all hosts in a particular hostgroup.

ENABLE_HOSTGROUP_PASSIVE_SVC_CHECKS

```
ENABLE_HOSTGROUP_PASSIVE_SVC_CHECKS ; <hostgroup_name>
```

Enables passive checks for all services associated with hosts in a particular hostgroup.

ENABLE_HOSTGROUP_SVC_CHECKS

```
ENABLE_HOSTGROUP_SVC_CHECKS ; <hostgroup_name>
```

Enables active checks for all services associated with hosts in a particular hostgroup.

ENABLE_HOSTGROUP_SVC_NOTIFICATIONS

```
ENABLE_HOSTGROUP_SVC_NOTIFICATIONS ; <hostgroup_name>
```

Enables notifications for all services that are associated with hosts in a particular hostgroup. This does not enable notifications for the hosts in the hostgroup - see the ENABLE_HOSTGROUP_HOST_NOTIFICATIONS command for that. In order for notifications to be sent out for these services, notifications must be enabled on a program-wide basis as well.

ENABLE_NOTIFICATIONS

```
ENABLE_NOTIFICATIONS
```

Enables host and service notifications on a program-wide basis.

ENABLE_PASSIVE_HOST_CHECKS

```
ENABLE_PASSIVE_HOST_CHECKS ; <host_name>
```

Enables acceptance and processing of passive host checks for the specified host.

ENABLE_PASSIVE_SVC_CHECKS

```
ENABLE_PASSIVE_SVC_CHECKS ; <host_name>;<service_description>
```

Enables passive checks for the specified service.

ENABLE_PERFORMANCE_DATA

```
ENABLE_PERFORMANCE_DATA
```

Enables the processing of host and service performance data on a program-wide basis.

ENABLE_SERVICE_FRESHNESS_CHECKS

```
ENABLE_SERVICE_FRESHNESS_CHECKS
```

Enables freshness checks of all services on a program-wide basis. Individual services that have freshness checks disabled will not be checked for freshness.

ENABLE_SERVICEGROUP_HOST_CHECKS

`ENABLE_SERVICEGROUP_HOST_CHECKS ;<servicegroup_name>`

Enables active checks for all hosts that have services that are members of a particular servicegroup.

ENABLE_SERVICEGROUP_HOST_NOTIFICATIONS

`ENABLE_SERVICEGROUP_HOST_NOTIFICATIONS ;<servicegroup_name>`

Enables notifications for all hosts that have services that are members of a particular servicegroup. In order for notifications to be sent out for these hosts, notifications must also be enabled on a program-wide basis.

ENABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS

`ENABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS ;<servicegroup_name>`

Enables the acceptance and processing of passive checks for all hosts that have services that are members of a particular service group.

ENABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS

`ENABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS ;<servicegroup_name>`

Enables the acceptance and processing of passive checks for all services in a particular servicegroup.

ENABLE_SERVICEGROUP_SVC_CHECKS

`ENABLE_SERVICEGROUP_SVC_CHECKS ;<servicegroup_name>`

Enables active checks for all services in a particular servicegroup.

ENABLE_SERVICEGROUP_SVC_NOTIFICATIONS

`ENABLE_SERVICEGROUP_SVC_NOTIFICATIONS ;<servicegroup_name>`

Enables notifications for all services that are members of a particular servicegroup. In order for notifications to be sent out for these services, notifications must also be enabled on a program-wide basis.

ENABLE_SVC_CHECK

`ENABLE_SVC_CHECK ;<host_name>;<service_description>`

Enables active checks for a particular service.

ENABLE_SVC_EVENT_HANDLER

`ENABLE_SVC_EVENT_HANDLER ;<host_name>;<service_description>`

Enables the event handler for the specified service.

ENABLE_SVC_FLAP_DETECTION

ENABLE_SVC_FLAP_DETECTION ;<host_name>;<service_description>

Enables flap detection for the specified service. In order for the flap detection algorithms to be run for the service, flap detection must be enabled on a program-wide basis as well.

ENABLE_SVC_NOTIFICATIONS

ENABLE_SVC_NOTIFICATIONS ;<host_name>;<service_description>

Enables notifications for a particular service. Notifications will be sent out for the service only if notifications are enabled on a program-wide basis as well.

PROCESS_FILE

PROCESS_FILE ;<file_name>;<delete>

Directs Icinga to process all external commands that are found in the file specified by the <file_name> argument. If the <delete> option is non-zero, the file will be deleted once it has been processes. If the <delete> option is set to zero, the file is left untouched.

PROCESS_HOST_CHECK_RESULT

PROCESS_HOST_CHECK_RESULT ;<host_name>;<status_code>;<plugin_output>

This is used to submit a passive check result for a particular host. The "status_code" indicates the state of the host check and should be one of the following: 0=UP, 1=DOWN, 2=UNREACHABLE. The "plugin_output" argument contains the text returned from the host check, along with optional performance data.

PROCESS_SERVICE_CHECK_RESULT

PROCESS_SERVICE_CHECK_RESULT ;<host_name>;<service_description>;<return_code>;<plugin_output>

This is used to submit a passive check result for a particular service. The "return_code" field should be one of the following: 0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN. The "plugin_output" field contains text output from the service check, along with optional performance data.

READ_STATE_INFORMATION

READ_STATE_INFORMATION

Causes Icinga to load all current monitoring status information from the state retention file. Normally, state retention information is loaded when the Icinga process starts up and before it starts monitoring. WARNING: This command will cause Icinga to discard all current monitoring status information and use the information stored in state retention file! Use with care.

REMOVE_HOST_ACKNOWLEDGEMENT

REMOVE_HOST_ACKNOWLEDGEMENT ;<host_name>

This removes the problem acknowledgement for a particular host. Once the acknowledgement has been removed, notifications can once again be sent out for the given host.

REMOVE_SVC_ACKNOWLEDGEMENT

REMOVE_SVC_ACKNOWLEDGEMENT ; <host_name>;<service_description>

This removes the problem acknowledgement for a particular service. Once the acknowledgement has been removed, notifications can once again be sent out for the given service.

RESTART_PROGRAM

RESTART_PROGRAM

Restarts the Icinga process.

SAVE_STATE_INFORMATION

SAVE_STATE_INFORMATION

Causes Icinga to save all current monitoring status information to the state retention file. Normally, state retention information is saved before the Icinga process shuts down and (potentially) at regularly scheduled intervals. This command allows you to force Icinga to save this information to the state retention file immediately. This does not affect the current status information in the Icinga process.

SCHEDULE_AND_PROPAGATE_HOST_DOWNTIME

SCHEDULE_AND_PROPAGATE_HOST_DOWNTIME ; <host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for a specified host and all of its children (hosts). If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The specified (parent) host downtime can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the specified (parent) host should not be triggered by another downtime entry.

SCHEDULE_AND_PROPAGATE_TRIGGERED_HOST_DOWNTIME

SCHEDULE_AND_PROPAGATE_TRIGGERED_HOST_DOWNTIME ; <host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for a specified host and all of its children (hosts). If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). Downtime for child hosts are all set to be triggered by the downtime for the specified (parent) host. The specified (parent) host downtime can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the specified (parent) host should not be triggered by another downtime entry.

SCHEDULE_FORCED_HOST_CHECK

SCHEDULE_FORCED_HOST_CHECK ; <host_name>;<check_time>

Schedules a forced active check of a particular host at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Forced checks are performed regardless of what time it is (e.g. timeperiod restrictions are ignored) and whether or not active checks are enabled on a host-specific or program-wide basis.

SCHEDULE_FORCED_HOST_SVC_CHECKS

```
SCHEDULE_FORCED_HOST_SVC_CHECKS ;<host_name>;<check_time>
```

Schedules a forced active check of all services associated with a particular host at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Forced checks are performed regardless of what time it is (e.g. timeperiod restrictions are ignored) and whether or not active checks are enabled on a service-specific or program-wide basis.

SCHEDULE_FORCED_SVC_CHECK

```
SCHEDULE_FORCED_SVC_CHECK ;<host_name>;<service_description>;<check_time>
```

Schedules a forced active check of a particular service at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Forced checks are performed regardless of what time it is (e.g. timeperiod restrictions are ignored) and whether or not active checks are enabled on a service-specific or program-wide basis.

SCHEDULE_HOST_CHECK

```
SCHEDULE_HOST_CHECK ;<host_name>;<check_time>
```

Schedules the next active check of a particular host at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Note that the host may not actually be checked at the time you specify. This could occur for a number of reasons: active checks are disabled on a program-wide or host-specific basis, the host is already scheduled to be checked at an earlier time, etc. If you want to force the host check to occur at the time you specify, look at the SCHEDULE_FORCED_HOST_CHECK command.

SCHEDULE_HOST_DOWNTIME

```
SCHEDULE_HOST_DOWNTIME ;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>
```

Schedules downtime for a specified host. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The specified host downtime can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the specified host should not be triggered by another downtime entry.

SCHEDULE_HOST_SVC_CHECKS

```
SCHEDULE_HOST_SVC_CHECKS ;<host_name>;<check_time>
```

Schedules the next active check of all services on a particular host at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Note that the services may not actually be checked at the time you specify. This could occur for a number of reasons: active checks are disabled on a program-wide or service-specific basis, the services are already scheduled to be checked at an earlier time, etc. If you want to force the service checks to occur at the time you specify, look at the SCHEDULE_FORCED_HOST_SVC_CHECKS command.

SCHEDULE_HOST_SVC_DOWNTIME

SCHEDULE_HOST_SVC_DOWNTIME ;<host_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for all services associated with a particular host. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The service downtime entries can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the services should not be triggered by another downtime entry.

SCHEDULE_HOSTGROUP_HOST_DOWNTIME

SCHEDULE_HOSTGROUP_HOST_DOWNTIME ;<hostgroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for all hosts in a specified hostgroup. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The host downtime entries can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the hosts should not be triggered by another downtime entry.

SCHEDULE_HOSTGROUP_SVC_DOWNTIME

SCHEDULE_HOSTGROUP_SVC_DOWNTIME ;<hostgroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for all services associated with hosts in a specified hostgroup. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The service downtime entries can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the services should not be triggered by another downtime entry.

SCHEDULE_SERVICEGROUP_HOST_DOWNTIME

SCHEDULE_SERVICEGROUP_HOST_DOWNTIME ;<servicegroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for all hosts that have services in a specified servicegroup. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The host downtime entries can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the hosts should not be triggered by another downtime entry.

SCHEDULE_SERVICEGROUP_SVC_DOWNTIME

SCHEDULE_SERVICEGROUP_SVC_DOWNTIME ;<servicegroup_name>;<start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for all services in a specified servicegroup. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The service downtime entries can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument

to zero (0) if the downtime for the services should not be triggered by another downtime entry.

SCHEDULE_SVC_CHECK

SCHEDULE_SVC_CHECK ;<host_name>;<service_description>;<check_time>

Schedules the next active check of a specified service at "check_time". The "check_time" argument is specified in time_t format (seconds since the UNIX epoch). Note that the service may not actually be checked at the time you specify. This could occur for a number of reasons: active checks are disabled on a program-wide or service-specific basis, the service is already scheduled to be checked at an earlier time, etc. If you want to force the service check to occur at the time you specify, look at the SCHEDULE_FORCED_SVC_CHECK command.

SCHEDULE_SVC_DOWNTIME

SCHEDULE_SVC_DOWNTIME ;<host_name>;<service_desription><start_time>;<end_time>;<fixed>;<trigger_id>;<duration>;<author>;<comment>

Schedules downtime for a specified service. If the "fixed" argument is set to one (1), downtime will start and end at the times specified by the "start" and "end" arguments. Otherwise, downtime will begin between the "start" and "end" times and last for "duration" seconds. The "start" and "end" arguments are specified in time_t format (seconds since the UNIX epoch). The specified service downtime can be triggered by another downtime entry if the "trigger_id" is set to the ID of another scheduled downtime entry. Set the "trigger_id" argument to zero (0) if the downtime for the specified service should not be triggered by another downtime entry.

SEND_CUSTOM_HOST_NOTIFICATION

SEND_CUSTOM_HOST_NOTIFICATION ;<host_name>;<options>;<author>;<comment>

Allows you to send a custom host notification. Very useful in dire situations, emergencies or to communicate with all admins that are responsible for a particular host. When the host notification is sent out, the \$NOTIFICATIONTYPE\$ macro will be set to "CUSTOM". The <options> field is a logical OR of the following integer values that affect aspects of the notification that are sent out: 0 = No option (default), 1 = Broadcast (send notification to all normal and all escalated contacts for the host), 2 = Forced (notification is sent out regardless of current time, whether or not notifications are enabled, etc.), 4 = Increment current notification # for the host (this is not done by default for custom notifications). The comment field can be used with the

SEND_CUSTOM_SVC_NOTIFICATION

SEND_CUSTOM_SVC_NOTIFICATION ;<host_name>;<service_description>;<options>;<author>;<comment>

Allows you to send a custom service notification. Very useful in dire situations, emergencies or to communicate with all admins that are responsible for a particular service. When the service notification is sent out, the \$NOTIFICATIONTYPE\$ macro will be set to "CUSTOM". The <options> field is a logical OR of the following integer values that affect aspects of the notification that are sent out: 0 = No option (default), 1 = Broadcast (send notification to all normal and all escalated contacts for the service), 2 = Forced (notification is sent out regardless of current time, whether or not notifications are enabled, etc.), 4 = Increment current notification # for the service(this is not done by default for custom notifications). The comment field can be used with the

SET_HOST_NOTIFICATION_NUMBER

SET_HOST_NOTIFICATION_NUMBER ; <host_name>;<notification_number>

Sets the current notification number for a particular host. A value of 0 indicates that no notification has yet been sent for the current host problem. Useful for forcing an escalation (based on notification number) or replicating notification information in redundant monitoring environments. Notification numbers greater than zero have no noticeable affect on the notification process if the host is currently in an UP state.

SET_SVC_NOTIFICATION_NUMBER

SET_SVC_NOTIFICATION_NUMBER ; <host_name>;<service_description>;<notification_number>

Sets the current notification number for a particular service. A value of 0 indicates that no notification has yet been sent for the current service problem. Useful for forcing an escalation (based on notification number) or replicating notification information in redundant monitoring environments. Notification numbers greater than zero have no noticeable affect on the notification process if the service is currently in an OK state.

SHUTDOWN_PROGRAM

SHUTDOWN_PROGRAM

Shuts down the Icinga process.

START_ACCEPTING_PASSIVE_HOST_CHECKS

START_ACCEPTING_PASSIVE_HOST_CHECKS

Enables acceptance and processing of passive host checks on a program-wide basis.

START_ACCEPTING_PASSIVE_SVC_CHECKS

START_ACCEPTING_PASSIVE_SVC_CHECKS

Enables passive service checks on a program-wide basis.

START_EXECUTING_HOST_CHECKS

START_EXECUTING_HOST_CHECKS

Enables active host checks on a program-wide basis.

START_EXECUTING_SVC_CHECKS

START_EXECUTING_SVC_CHECKS

Enables active checks of services on a program-wide basis.

START_OBSESSING_OVER_HOST

START_OBSESSING_OVER_HOST ; <host_name>

Enables processing of host checks via the OCHP command for the specified host.

START_OBSESSING_OVER_HOST_CHECKS

START_OBSESSING_OVER_HOST_CHECKS

Enables processing of host checks via the OCHP command on a program-wide basis.

START_OBSESSING_OVER_SVC

START_OBSESSING_OVER_SVC ;<host_name>;<service_description>

Enables processing of service checks via the OCSP command for the specified service.

START_OBSESSING_OVER_SVC_CHECKS

START_OBSESSING_OVER_SVC_CHECKS

Enables processing of service checks via the OCSP command on a program-wide basis.

STOP_ACCEPTING_PASSIVE_HOST_CHECKS

STOP_ACCEPTING_PASSIVE_HOST_CHECKS

Disables acceptance and processing of passive host checks on a program-wide basis.

STOP_ACCEPTING_PASSIVE_SVC_CHECKS

STOP_ACCEPTING_PASSIVE_SVC_CHECKS

Disables passive service checks on a program-wide basis.

STOP_EXECUTING_HOST_CHECKS

STOP_EXECUTING_HOST_CHECKS

Disables active host checks on a program-wide basis.

STOP_EXECUTING_SVC_CHECKS

STOP_EXECUTING_SVC_CHECKS

Disables active checks of services on a program-wide basis.

STOP_OBSESSING_OVER_HOST

STOP_OBSESSING_OVER_HOST ;<host_name>

Disables processing of host checks via the OCHP command for the specified host.

STOP_OBSESSING_OVER_HOST_CHECKS

STOP_OBSESSING_OVER_HOST_CHECKS

Disables processing of host checks via the OCHP command on a program-wide basis.

STOP_OBSESSING_OVER_SVC

STOP_OBSESSING_OVER_SVC ;<host_name>;<service_description>

Disables processing of service checks via the OCSP command for the specified service.

STOP_OBSESSING_OVER_SVC_CHECKS

STOP_OBSESSING_OVER_SVC_CHECKS

Disables processing of service checks via the OCSP command on a program-wide basis.

[Prev](#)

[Up](#)

[Next](#)

Developing Plugins For Use With
Embedded Perl

[Home](#)

Installation and use of the Icinga
API

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Installation and use of the Icinga API

[Prev](#)

Chapter 11. Development

[Next](#)

Installation and use of the Icinga API

Prerequisites

You need Icinga Core and IDOUtils or [MKLiveStatus](#) installed and running in order to use the API.



Note

If you don't have Icinga yet please follow the instructions given in the "[quickstart-idoutils](#)" documentation.

If you are using IDOUtils database as data source, install PHP-PDO.

- **RHEL/Fedora/CentOS**

Make sure you have a repository/packages for PHP 5.2.x - RHEL/CentOS only support 5.1.6 out of the box.

```
# yum install php-pdo php-mysql|pgsql
```

- **Debian/Ubuntu**

```
# apt-get install php5 php5-mysql|pgsql
```

- **openSuSE**

Please use yast to install the packages php5, php5-pdo, and php5-mysql|php5-pgsql.

Installation and Configuration



Note

Icinga-API is already included in the package with Core, IDOUtils and docs and also installed during 'make install'. If you already installed this package, there's no need to install Icinga-API. It is located by default in /usr/local/icinga/share/icinga-api/ and you can skip this section!



Note

If you just require Icinga-API for Icinga-Web, and already installed the Core with IDOUtils, skip this Howto and refer directly to [installing Icinga Web](#).

1. Download

Take your clone from the icinga-api.git to get a fresh branch

```
# git clone git://git.icinga.org/icinga-api.git
```

or if you just need an update:

```
# cd icinga-api && git pull origin master
```

or download the software using

<https://git.icinga.org/index?p=icinga-api.git;a=snapshot;h=refs/heads/master;sf=tgz>.

2. Installation

Unpack Icinga-API run configure and install it.

```
# tar xzvf icinga-api-(version).tar.gz
# ./configure
```

You can set the prefix where it will be installed, and point Icinga-API where your Icinga and IDOUtils config is located and which users are required to run (those settings are directly applied when installing the API through Icinga Core Installation).

```
# ./configure --datarootdir=/usr/local/icinga/share \
--sysconfdir=/usr/local/icinga/etc \
--with-command-user=icinga-cmd \
--with-command-group=icinga-cmd \
--with-icinga-user=icinga \
--with-icinga-group=icinga \
--with-web-user=www-data \
--with-web-group=www-data
```



Note

The --with-web... directives have to be set. Otherwise the web logs will not be written correctly. This may also lead to an empty main cronk. Please note that the values of user and group differ across distributions.

```
# make install
```

Configuration

If you are developing your own Addon based on the Icinga-API, you need the following associative array.

```
$idoConfig = array (
    'type'          => '<Type of database>',
    'host'          => '<Database hostname>',
    'database'      => '<Databasename>',
    'user'          => '<Username>',
    'password'      => '<password>',
    'persistent'    => <true | false>,
    'table_prefix'  => '<table prefix>',
);
```

Example:

```
$idoConfig = array (
    'type'          => 'mysql',
    'host'          => 'localhost',
    'database'      => 'ido',
    'user'          => 'idouser',
    'password'      => 'idopassword',
    'persistent'    => true,
    'table_prefix'  => 'icinga_',
);

```

Supported Backends

Currently the following backend types are available. More information about that can be found in doc/icinga-api-types.txt.

- IDOUtils DB - OK
- Livestatus Module - experimental, not for productive usage
- Filebased, status.dat - experimental, not for productive usage

Use of the API

Examples can be found in doc/examples

1. Fetching data

hostnames and corresponding states

Create an instance of class IcingaApi:

```
$api = IcingaApi::getConnection(IcingaApi::CONNECTION_IDO, $idoConfig);
```

Create your search:

```
$apiRes = $api->createSearch()
->setSearchTarget(IcingaApi::TARGET_HOST)
->setResultColumns(array('HOST_NAME', 'HOST_CURRENT_STATE'))
->fetch();
```

By using setSearchFilter() you can define filters to narrow down the result set:

```
$apiRes = $api->createSearch()
->setSearchTarget(IcingaApi::TARGET_HOST)
->setResultColumns(array('HOST_NAME', 'HOST_CURRENT_STATE'))
->setSearchFilter(HOST_NAME, 'Swith%', IcingaApi::MATCH_LIKE)
->fetch();
```

2. Processing results

```
foreach($apiRes as $apiHandle){
    echo 'Host '.$apiHandle->HOST_NAME.' has state '.$apiHandle->HOST_CURRENT_STATE.'  
>';
}
```

Output without filter:

```
Host localhost has state 0
Host MySql has state 0
Host router-01 has state 0
Host windows100 has state 0
Host Apache_01 has state 0
```

Output with filter:

```
Host switch70 has the current state 0
Host switch71 has the current state 0
Host switch72 has the current state 0
Host switch73 has the current state 0
Host switch74 has the current state 0
Host switch75 has the current state 0
Host switch76 has the current state 0
Host switch77 has the current state 0
```

3. Complete code without use of filters

```
<?
// Path to icinga api file
$apiFile = 'icinga-api/IcingaApi.php';

// Database connection
$idoConfig = array (
    'type'          => 'mysql',
    'host'          => 'localhost',
    'database'      => 'ido',
    'user'          => 'idouser',
    'password'      => 'idopassword',
    'persistent'    => true,
    'table_prefix'  => 'icinga_',
);

// Include required files
require_once($apiFile);

// Instance the class
$api = IcingaApi::getConnection(IcingaApi::CONNECTION_IDO, $idoConfig);

// Create search
$apiRes = $api->createSearch()
->setSearchTarget(IcingaApi::TARGET_HOST)
->setResultColumns(array('HOST_NAME', 'HOST_CURRENT_STATE'))
->fetch();

// Create output
foreach($apiRes as $apiHandle){
    echo 'Host '.$apiHandle->HOST_NAME.' has the current state '.$apiHandle->HOST_CURRENT_STATE.'  
>';
}
?>
```

Please have a look at the [git repository](#) for further information or consult the examples in the doc/examples folder.

[Prev](#)

[Up](#)

[Next](#)

[List of External Commands](#)

[Home](#)

[The Icinga-Web REST API](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



The Icinga-Web REST API

[Prev](#)

[Chapter 11. Development](#)

[Next](#)

The Icinga-Web REST API

In this document we'll describe the Icinga-Web REST API (yes, we know that the term is not yet completed) which allows you to request your monitoring information via GET or POST requests (in the future (>1.2), you will also be able to send commands via PUT).

Why should I use the API?

For most people, the combination Icinga/Icinga-Web will fit most needs. You can watch your monitoring status, act on problems and extend it to suit your needs (Modules/Cronks).

But sometimes, you have another piece of software that is interested in some monitoring data (for example: [Icinga-Chromed-Status](#)). You could parse the cgi output of Icinga (many programs do that at this time), but that's not really a high performance solution - and certainly no fun for the developer. The goal of the REST API is to return the data you want (and only the data you want) in a standardized, machine-readable format like JSON or XML.

Features of the Icinga-Web REST API

Currently supported (v1.2):

- Availability of almost all monitoring fields via GET or POST.
- Return data as xml or json.
- AND & OR search filtergroups with unlimited nesting levels (AND)).
- You choose which columns you want returned, not the API (less overhead).
- Support of limit, offset, order, group by.
- Return an additional total count field.
- Authorization via auth_key in request or cookies.
- Respects Icinga-web principals (for example, limit to specific hostgroups).

Planned in future(> 1.2):

- Send commands via PUT

What's the difference between the Icinga-API and the Icinga-Web REST API?

The Icinga-API can at this point considered as an internal toolkit to access the database informations. In fact, the REST API acts on top of this API and extends it via the HTTP protocol. In the future, the Icinga-API will be merged into Icinga-Web.

Prerequisites

In order to use the API, you first have to enable the Auth provider for it. This can be done under icinga-web/app/modules/AppKit/config/auth.xml.

Change "auth_enabled" to 'true' in this section:

```
<ae:parameter name="auth_key">
    <ae:parameter name="auth_module">AppKit</ae:parameter>
    <ae:parameter name="auth_provider">Auth.Provider.AuthKey</ae:parameter>
    <ae:parameter name="auth_enable">true</ae:parameter>
    <ae:parameter name="auth_authoritative">true</ae:parameter>
</ae:parameter>
```



Note

If you edit any *.xml file you have to clear the cache afterwards!

```
rm -f app/cache/config/*.php
```

or

```
icinga-web/bin/clearcache.sh
```

Now, in Icinga-Web, you have to add a user with API access:

- Create a new user
- Choose auth_key in the Auth_via field
- Insert an API key to use
- Under principals, add the appkit.api.access principal

That's it, now you can start.

Reference

So, here are the goodies. In the next few points we'll explain how the API can be accessed:

GET

Advantages:

- Easy to use, it's just an URL!
- You always see what parameters have been requested.

Disadvantages:

- If you request it in a browser your API key could be in the browser history.

- In a browser, you cannot add URLs with unlimited size (2,083 characters for Internet explorer, for example).
- Especially when parameters are escaped, the parameter list lacks a bit of clarity.

The structure of the URL:

To access the api, the URL should look as in the following (italics are optional, bold ones are required) host.com/icinga-web/web/api/ **TARGET** / **COLUMNS** / **FILTER** / **ORDER** / **GROUPING** / **LIMIT** / **COUNTFIELD** / **OUTPUT_TYPE**

The Parameters en detail:

- TARGET: Which field to request, is a simple string like host.
- COLUMNS: A listing of columns to return, must look like this: columns[COL1|COL2|COL3|...]
- FILTER: Defines which filters to use in the request. Must always be nested in AND or OR groups.

The filter itself looks like this:

```
filters[AND/OR(COLUMN|OPERATOR|VALUE;COLUMN2|OPERATOR2|VALUE2;OR(...),AND)]
```

Example: Select all services with smtp in the name, but only if they're ok or unknown

Wrong:

```
filters[SERVICE_NAME|like|*smtp*;OR(SERVICE_CURRENT_STATE|=|0;SERVICE_CURRENT_STATE|=|3)]
```

You always need a nesting level at the beginning, see:

Correct:

```
filters[AND( SERVICE_NAME|like|*smtp*;OR( SERVICE_CURRENT_STATE|=|0;SERVICE_CURRENT_STATE|=|3) ) ]
```

- ORDER: Defines which field to use for ordering and if ascending or descending ordering should be used. Example: order[COLUMN| ASC or DESC]
- GROUPING: Defines a field to group by: group[COL]
- LIMIT: Defines a starting offset and/or a limit: limit[START;END (if needed)]
- COUNTFIELD: Adds a total field to the result which counts by this field (in most cases, the id): countColumn=COL
- OUTPUT: At this time either json or xml

Example for GET

Get all services that are critical or warning, but have a host that is ok. Sort descending by the service state and count the services. Authentication is done via authkey (here APITEST123456). The request is broken into pieces for better readability, XML:

```
http://localhost/icinga-web/web/api/service/filter[AND(HOST_CURRENT_STATE|=|0;OR(SERVICE_CURRENT_STATE|=|1;SERVICE_CURRENT_STATE|=|2))]/columns(SERVICE_NAME|HOST_NAME|SERVICE_CURRENT_STATE|HOST_NAME|HOST_CURRENT_STATE|HOSTGROUP_NAME)/order(SERVICE_CURRENT_STATE;DESC)/countColumn=SERVICE_ID/authkey=APITEST123456/xml
```

This would return something like:

```
<results>
  <result>
    <column name="SERVICE_ID">295</column>
    <column name="SERVICE_OBJECT_ID">139</column>
    <column name="SERVICE_IS_ACTIVE">1</column>
    <column name="SERVICE_INSTANCE_ID">1</column>
    <column name="SERVICE_NAME">MailQ</column>
    <column name="SERVICE_DISPLAY_NAME">MailQ</column>
    <column name="SERVICE_OUTPUT">Error occured:error=1:0:0</column>
    <column name="SERVICE_PERFDATA"></column>
  </result>
  <result>
    <column name="SERVICE_ID">311</column>
    <column name="SERVICE_OBJECT_ID">155</column>
    <column name="SERVICE_IS_ACTIVE">1</column>
    <column name="SERVICE_INSTANCE_ID">1</column>
    <column name="SERVICE_NAME">POP3</column>
    <column name="SERVICE_DISPLAY_NAME">POP3</column>
    <column name="SERVICE_OUTPUT">Verbindungsauftbau abgelehnt</column>
    <column name="SERVICE_PERFDATA"></column>
  </result>
  <total>2</total>
</results>
```

If you change the xml to json you get the same information (plus additional infos for ExtJS, which you can ignore if you're not using it) in json format:

```
{
  "metaData": {
    "paramNames": {
      "start": "limit_start",
      "limit": "limit"
    },
    "totalProperty": "total",
    "root": "result",
    "fields": null
  },
  "result": [
    {
      "SERVICE_ID": "295",
      "SERVICE_OBJECT_ID": "139",
      "SERVICE_IS_ACTIVE": "1",
      "SERVICE_INSTANCE_ID": "1",
      "SERVICE_NAME": "MailQ",
      "SERVICE_DISPLAY_NAME": "MailQ",
      "SERVICE_OUTPUT": "Error occured:error=1:0:0",
      "SERVICE_PERFDATA": ""
    },
    {
      "SERVICE_ID": "311",
      "SERVICE_OBJECT_ID": "155",
      "SERVICE_IS_ACTIVE": "1",
      "SERVICE_INSTANCE_ID": "1",
      "SERVICE_NAME": "POP3",
      "SERVICE_DISPLAY_NAME": "POP3",
      "SERVICE_OUTPUT": "Connection refused",
      "SERVICE_PERFDATA": ""
    }
  ],
  "success": "true",
  "total": "2"
}
*CAUTION*: If you're not using the countField parameter, you'll get a flat json with the result.
```

POST

Advantages:

- Unlimited parameter size, as it's made for big requests.
- Your parameters don't appear in the browser history, only the base url.

- It's easier to implement in applications (ok, that's my opinion :))

Disadvantages:

- POST will be send via the header, so you can't request it easily from the browser's address field.

Parameters en detail

The link is almost the same like the GET baselink, but with the output type in it: For example, host.com/icinga-web/web/api/json. The following parameters are supported:

- 'target': The search target, like host
- 'columns[]': An array of columns

Example:

columns [0] = SERVICE_NAME

columns [1] = SERVICE_ID

- 'groups[]': Group by this field
- 'filters_json': A json describing how to filter

Example:

```
[ {
    "type": "AND",
    "field": [
        {
            "type": "atom",
            "field": ['SERVICE_NAME'],
            "method": ['like'],
            "value": ['*pop*']
        },
        {
            "type": "OR",
            "field": [
                {
                    "type": "atom",
                    "field": ['SERVICE_CURRENT_STATE'],
                    "method": ['>'],
                    "value": [0]
                },
                {
                    "type": "atom",
                    "field": ['SERVICE_IS_FLAPPING'],
                    "method": ['='],
                    "value": [1]
                }
            ]
        }
    ]
}]
```

- 'order_col': Column to order by
- 'order_dir': Order direction (asc oder desc)
- 'limit_start': The offset of the records to start
- 'limit': Limits the result to x responses

- ‘countColumn’ : Add a total field with this column

Example for POST

Lets take the example from Example for GET and use a post request this time. We’re going to use curl, so the example can be repeated from the console:

```
curl
-d 'target=service
-d 'filter[json][0][type]:'AND', 'field':[['type':'atom','field':["HOST_CURRENT_STATE"]],'method':{'+':[]}, 'value':[{}]],[['type':'OR','field':[['type':'atom','field':["SERVICE_CURRENT_STATE"]],'method':{'+':[]}, 'value':[{}]]],[['type':'atom','field':["SERVICE_CURRENT_STATE"]],'method':{'+':[]}, 'value':[{}]]]]
-d columns[0]=SERVICE_NAME
-d columns[1]=HOSTNAME
-d columns[2]=HOSTCURRENTSTATE
-d columns[3]=HOSTGROUPNAME
-d columns[4]=HOSTGROUPCURRENTSTATE
-d countColumn=SERVICE_ID
-d 'authKey:AF2123456'
http://localhost:8000/icinga-web/web/api/xml'
```

This would return the same result as the GET request shown before.

[Prev](#)
[Up](#)
[Next](#)
[Installation and use of the Icinga API](#)
[Home](#)
[Chapter 12. IDOUtils](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Chapter 12. IDOUtils

[Prev](#)

[Next](#)

Chapter 12. IDOUtils

Table of Contents

[Introduction](#)

[Purpose](#)

[Design Overview](#)

[Instances](#)

[Installation](#)

[Components](#)

[Overview](#)

[IDOMOD](#)

[LOG2IDO](#)

[FILE2SOCK](#)

[IDO2DB, IDO2DB](#)

[Example Configurations](#)

[Single Server, Single Instance Setup](#)

[Single Server, Multiple Instance Setup](#)

[Single Server, Single Instance Log File Import](#)

[IDOUtils Database Model](#)

[Central Tables](#)

[Debugging Tables](#)

[Historical Tables](#)

[Current Status Tables](#)

[Configuration Tables](#)

[Database changes/alterations](#)

[Prev](#)

[Next](#)

[The Icinga-Web REST API](#)

[Home](#)

[Introduction](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Introduction

[Prev](#)

Chapter 12. IDOUtils

[Next](#)

Introduction

The IDOUtils addon is based on NDOUtils originally written by Ethan Galstad, creator of Nagios(R), so the fundamentals apply to Nagios as well as to Icinga.

Purpose

The IDOUtils addon is designed to store all configuration and event data from Icinga in a relational database. Storing information from Icinga in an RDBMS will allow for quicker retrieval and processing of that data. The Icinga-API relies on that data.

Until now MySQL, Oracle and PostgreSQL are supported by the addon. Support for other database servers may be added if there is sufficient user interest and even more user who are interested in testing.

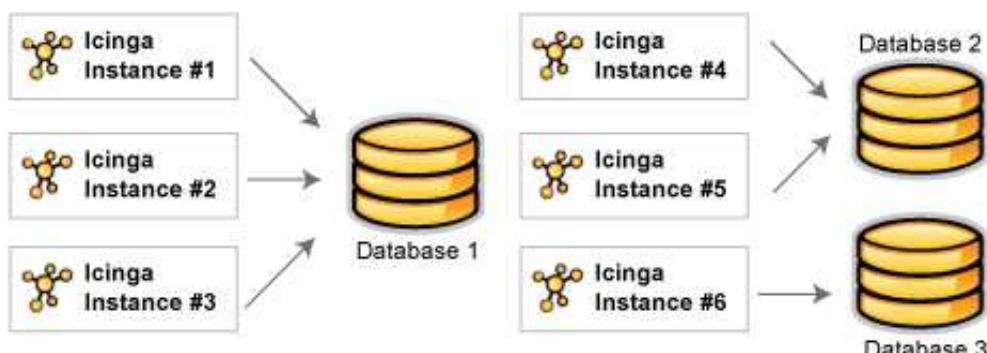
Design Overview

The IDOUtils addon was designed to work for users who have:

- Single Icinga installations
- Multiple standalone or "vanilla" Icinga installations
- Multiple Icinga installations in distributed, redundant, and/or fail over environments

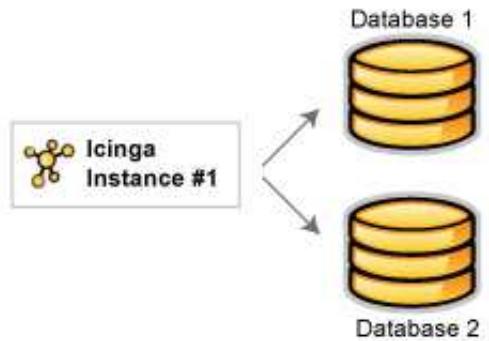
Data from each Icinga process (hereafter referred to as "instances") can be stored either in the same database or different databases than data from other Icinga instances.

Figure 12.1. Multiple instances



Although not yet supported, future development should allow for data from any given Icinga instance to be stored in multiple databases if desired.

Figure 12.2. Future development: One Instance, multiple databases

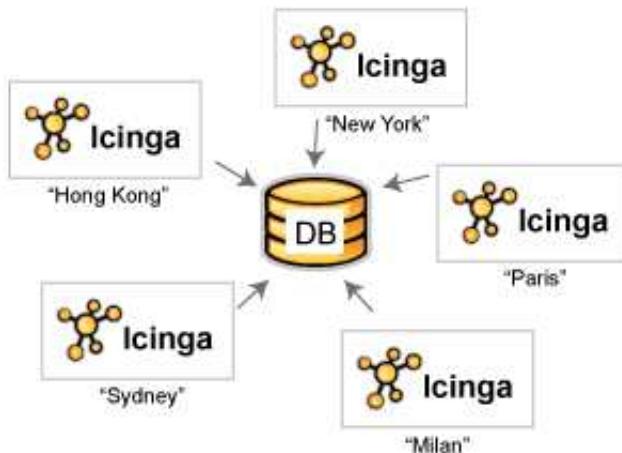


Instances

Each Icinga process, whether it is a standalone monitoring server, or part of a distributed, redundant, or fail over monitoring setup, is referred to as an "instance". In order to maintain the integrity of stored data, each Icinga instance must be labeled with a unique identifier or name.

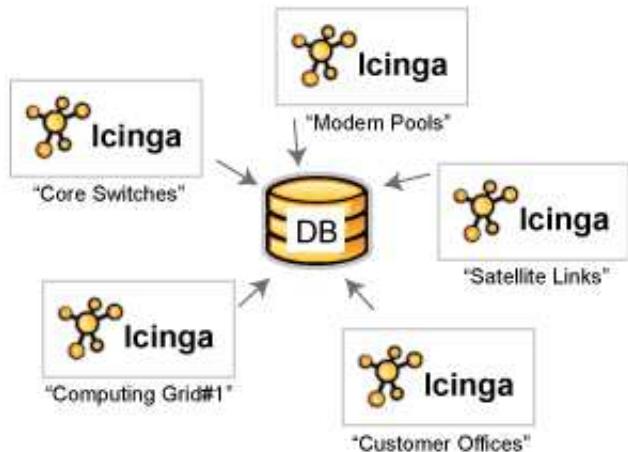
You can choose the name of each Icinga instance to suit your needs. For instance, you could choose to name Icinga instances based on their geographical location...

Figure 12.3. Instance names based on geographical locations



Or you could name Icinga instances based on their purpose...

Figure 12.4. Instance names based on their purpose



How you name Icinga instances is up to you. The key point to remember is that each and every Icinga process must have its own unique instance name.

More information on how instance names come into play will be discussed in the next sections.

Installation

The installation of the IDOUtils is described in the [Quickstart IDOUtils](#)

[Prev](#)

[Up](#)

[Next](#)

[Chapter 12. IDOUtils](#)

[Home](#)

[Components](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Components

[Prev](#)

Chapter 12. IDOUtils

[Next](#)

Components

Overview

There are four main components that make up the IDO utilities:

1. IDOMOD Event Broker Module
2. LOG2IDO Utility
3. FILE2SOCK Utility
4. IDO2DB Daemon

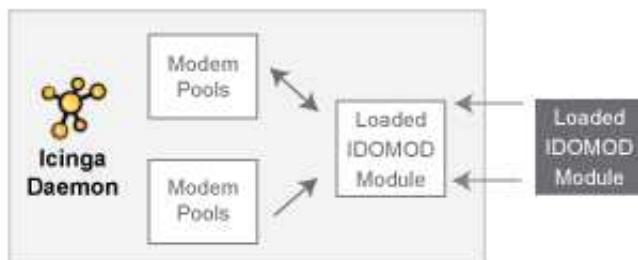
Each component is described in more detail on the following pages.

IDOMOD

The IDO utilities includes an Icinga event broker module (IDOMOD.O) that exports data from the Icinga daemon.

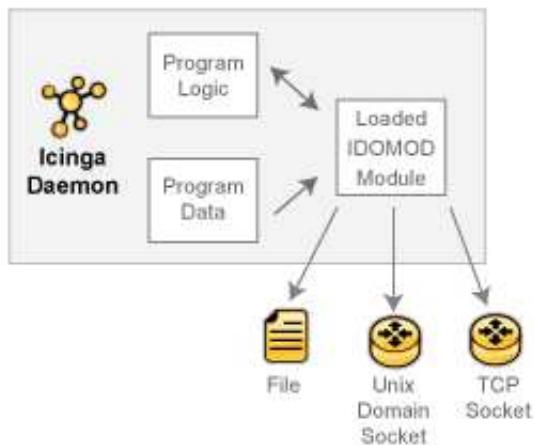
Assuming that Icinga has been compiled with the event broker enabled (this is the default), you can configure Icinga to load the IDOMOD module during runtime. Once the module has been loaded by the Icinga daemon, it can access all of the data and logic present in the running Icinga process.

Figure 12.5. Loaded IDOMOD Event broker Module



The IDOMOD module has been designed to export configuration data, as well as information about various runtime events that occur in the monitoring process, from the Icinga daemon. The module can send this data to a standard file, a Unix domain socket, or a TCP socket.

Figure 12.6. IDOMOD Capabilities



The IDOMOD module writes data in a format that the IDO2DB daemon (described later) can understand.

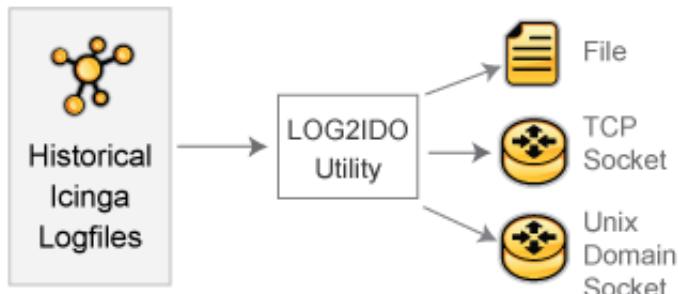
If the IDOMOD module is writing its output to a file, you can configure it to periodically rotate and/or process the output file using a predefined Icinga command. This can be useful if you want to transfer the output file to another physical machine (using SSH, etc.) and send its contents to the IDO2DB daemon using the FILE2SOCK utility (described later).

If the IDOMOD module is writing its output to a TCP or Unix domain socket, it has some resistance to connection dropouts. The module will attempt to cache its output until it can (re)connect to the socket for writing. This is helpful if the process that creates and listens on the socket needs to be restarted, etc.

LOG2IDO

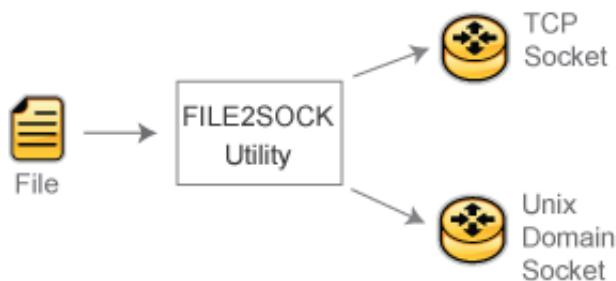
The LOG2IDO utility has been designed to allow you to import historical Icinga, Nagios and NetSaint log files into a database via the IDO2DB daemon ([described later](#)). The utility works by sending historical log file data to a standard file, a Unix domain socket, or a TCP socket in a format the IDO2DB daemon understands. The IDO2DB daemon can then be used to process that output and store the historical logfile information in a database.

Figure 12.7. LOG2IDO Utility



FILE2SOCK

The FILE2SOCK utility is quite simple. It reads input from a standard file (or STDIN) and writes all of that data to either a Unix domain socket or TCP socket. The data that is read is not processed in any way before it is sent to the socket.

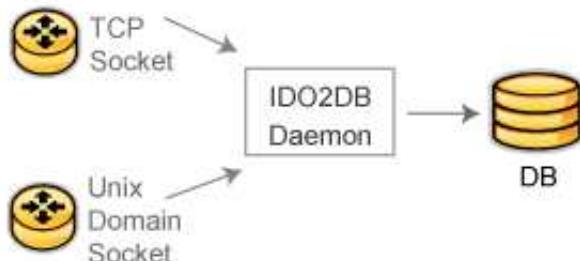
Figure 12.8. FILE2SOCK Utility

This utility is useful if you are directing the output of the IDOMOD event broker module and/or LOG2IDO utility to a standard file. Once these components finish writing their output to a file, you can use the FILE2SOCK utility to send the contents of the file to the IDO2DB daemon's TCP or Unix domain socket.

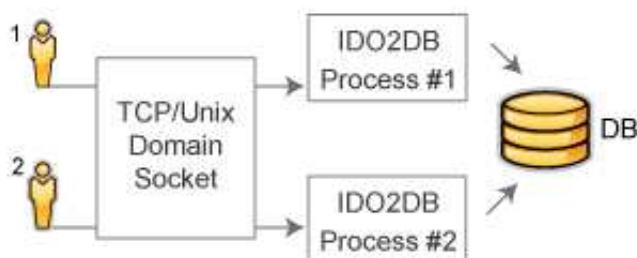
IDO2DB

The IDO2DB utility is designed to take the data output from the IDOMOD and LOG2IDO components and store it in a MySQL, Oracle, or PostgreSQL database.

When it starts, the IDO2DB daemon creates either a TCP or Unix domain socket and waits for clients to connect. IDO2DB can run either as a standalone, multi-process daemon or under INETD (if using a TCP socket).

Figure 12.9. IDO2DB Daemon

Multiple clients can connect to the IDO2DB daemon's socket and transmit data simultaneously. A separate IDO2DB process is spawned to handle each new client that connects. Data is read from each client and stored in a user-specified database for later retrieval and processing.

Figure 12.10. IDO2DB with multiple Clients

The IDO2DB daemon currently supports MySQL, Oracle, and PostgreSQL databases.

[Prev](#)

[Up](#)

[Next](#)

[Introduction](#)

[Home](#)

[Example Configurations](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Example Configurations

[Prev](#)

Chapter 12. IDOUtils

[Next](#)

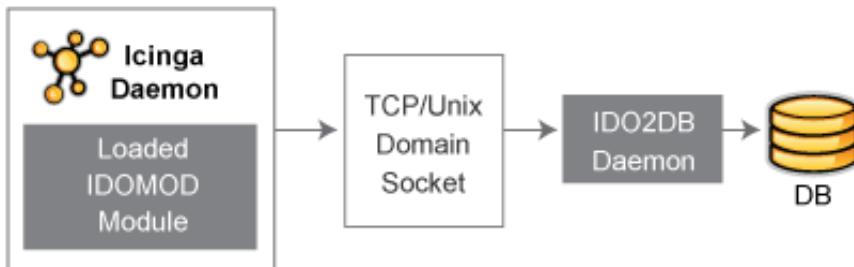
Example Configurations

Single Server, Single Instance Setup

The easiest configuration occurs when you have a single Icinga instance running on your network. In this case, installing and configuring the various components of the IDOUtils addon is fairly straightforward.

The following diagram illustrates how the various components can fit together in a single server, single Icinga instance setup...

Figure 12.11. Single Server, Single Instance Setup



Here's a description of what's happening at each point in the diagram:

1. The IDOMOD module is configured with an instance name of "default" since there is only one instance of Icinga that is running on the network.
2. While the Icinga daemon is running and performing its usual business of monitoring the network, the IDOMOD module is sending configuration data and event information to the TCP or Unix domain socket that was created by the IDO2DB daemon.
3. The IDO2DB daemon reads data that is coming into the socket from the IDOMOD module.
4. The IDO2DB daemon processes and transforms data that has been received from the IDOMOD module.
5. The processed data is stored in a database for later retrieval and processing.

This example assumes that:

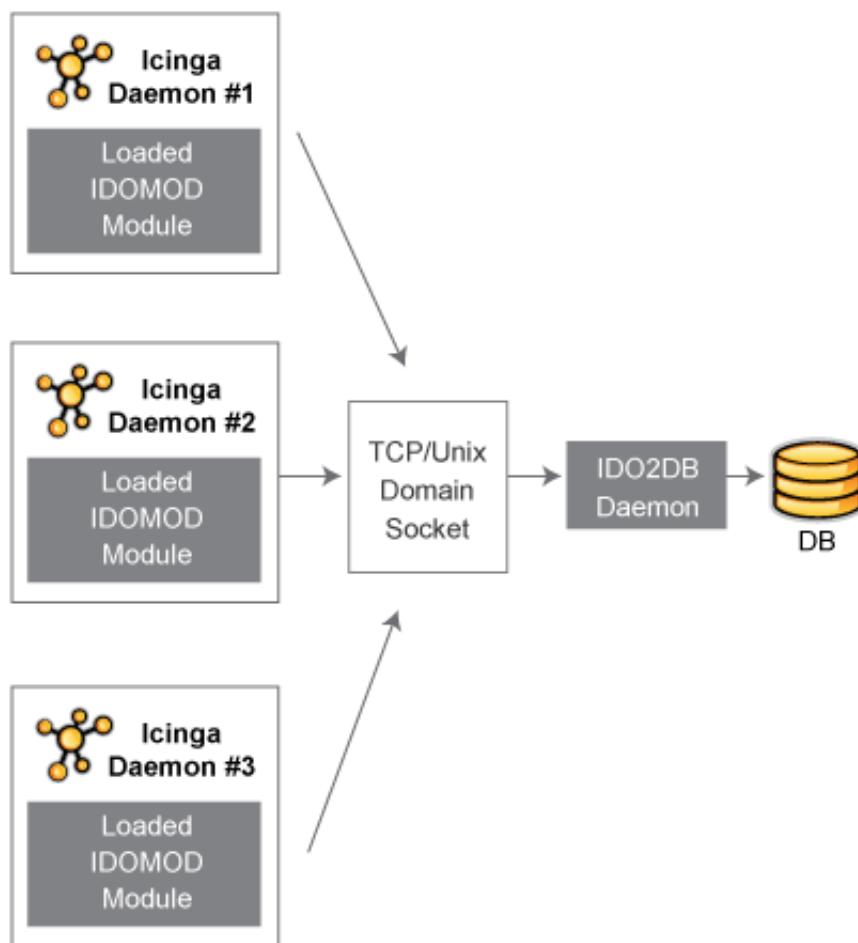
1. Icinga is configured to load the IDOMOD module at startup.
2. The IDO2DB daemon (which is a separate process from the Icinga daemon) is running.

Single Server, Multiple Instance Setup

Another simple configuration can be used when you have multiple Icinga instances running on a single server. Installing and configuring the various components of the IDOUtils addon is similiar as to what was shown in the previous example.

The following diagram illustrates how the various components can fit together in a single server, multiple Icinga instance setup...

Figure 12.12. Single Server, Multiple Instance Setup



You'll notice that the diagram above is similiar to the one for the single-server, single instance configuration. The main difference is that there are now three (3) different Icinga daemons instead of just one.

1. Each Icinga daemon loads the IDOMOD module at startup with a unique instance name. In this example the instances are simply named "Icinga1", "Icinga2" and "Icinga3".
2. Each IDOMOD module sends configuration data and event information for its specific instance of the Icinga daemon to the TCP or Unix domain socket that was created by the IDO2DB daemon.

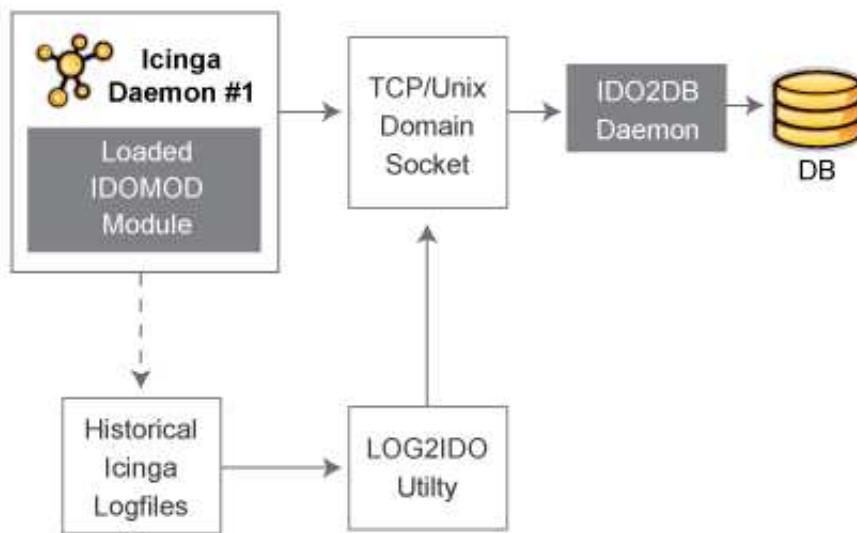
3. The IDO2DB daemon reads data that is coming into the socket from the three IDOMOD modules.
4. The IDO2DB daemon processes and transforms data that has been received from the IDOMOD modules.
5. The processed data is stored in a database for later retrieval and processing. Data from each instance of Icinga is kept separate (using the instance names as unique identifiers) in the database.

Single Server, Single Instance Log File Import

There are two reasons you'll probably want to import your Icinga log files into the same database that Icinga configuration and event data is stored in:

1. Historical log file data isn't imported into the database by default and having a record of events that occurred before you implemented the IDOUtils addon is probably desireable.
2. The IDOMOD module is not able to process realtime log entries from the time right after the Icinga daemon starts to the time that the IDOMOD module is loaded by the Icinga daemon. This "blackout period" is unavoidable and causing log entries such as "Icinga 1.0 starting..." to be missed by the IDOMOD module. Thus, importing day-old Icinga log files on a daily basis (via a cron job) is recommended.

Figure 12.13. Single Server, Single Instance Log File Import



Here's a description of what's happening at each point in the diagram:

1. Historical Icinga log files are read by the LOG2IDO utility.
2. The LOG2IDO utility processes the contents of the log files and tags them with an instance name of "default". This instance name must match the same instance name used by the IDOMOD module in the Icinga daemon.
3. Historical log file data is sent to the TCP or Unix domain socket in a format that the IDO2DB daemon can understand.

4. The IDO2DB daemon reads the log file data from the Unix domain socket.
5. The IDO2DB daemon processes the log file data.
6. Historical log file data is stored in a database for later retrieval and processing. The IDO2DB daemon will perform some checks to make sure it doesn't re-import duplicate historical log entries, so running the LOG2IDO utility on the same historical log file multiple times shouldn't have any negative side effects.

That's it! Pretty simple.

[Prev](#)

[Up](#)

[Next](#)

[Components](#)

[Home](#)

[IDOUtils Database Model](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>



IDOUtils Database Model

[Prev](#)

Chapter 12. IDOUtils

[Next](#)

IDOUtils Database Model

This documentation is based on the NDOUtils database model documentation by Ethan Galstad.

Introduction

This documentation is still in flux, and there are undoubtably errors present, so take everything you find here with a grain of salt. If you have suggestions, changes, etc. for the documentation, please let us know.

Table Names

The IDOUtils addon allows users to specify a custom prefix to each table name in the database. By default, this prefix is set to "icinga_" in ido2db.cfg. The tables documented here are listed without any prefix.



Note

Due to limitations in Oracle the length of table names cannot exceed 30 characters so

- The name of one table has been shortened: serviceescalation_contactgroups -> serviceescalationcontactgroups
- The table prefix is ignored

Keys

Every table has a primary key (designated as "PK"). Most tables have a unique key consisting of one ("UK") or more columns ("UKn" whereas n shows the position in the key). Some tables have a non-unique key ("NK") which may be composed of several columns as well ("NKn").

There are a lot of tables containing different information so the description is divided into five parts:

- [Central Tables](#)
- [Debugging Tables](#)
- [Historical Tables](#)

- Current Status Tables
- Configuration Tables

Central Tables

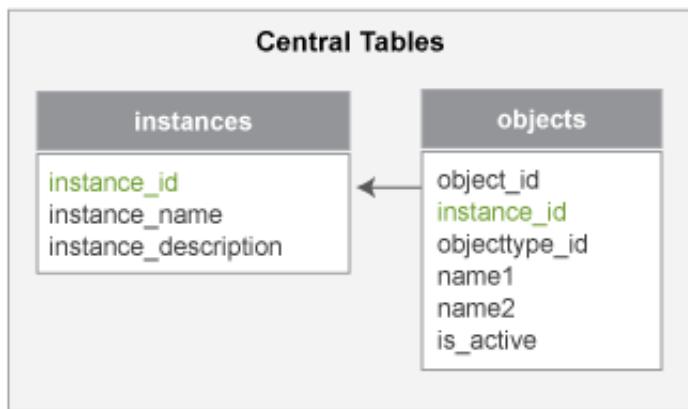
There are two "core" or "central" tables, described below, that are referenced by nearly every table in the database. Read below for more information.

Table List

- [instances](#)
- [objects](#)

Relationship Diagram

Figure 12.14. Relationship of Central Tables



Instance Table

Description: This table is needed to ensure that multiple instances of Icinga can store their configuration and status information in the same database. Each instance represents a different Icinga installation/process. A new instance will automatically be created when the user specifies a new instance name (when running one of the IDOUtils components) that does not already exist in the database.

Structure:

Field	Type	Notes	Key
instance_id	SMALLINT	Unique number identifying a distinct instance of Icinga	PK
instance_name	VARCHAR(64)	Instance name, as passed to and used by IDOUtils components	
instance_description	VARCHAR(128)	Optional text describing the instance in more detail	

Objects Table

Description: This table is used to store all current (and past) objects that are (and have been) defined in your Icinga configuration files. Why are the names of the objects stored in this table and not elsewhere? Well, when you delete an object definition from your Icinga configuration, that object will no longer appear in the object tables of the database. Since you're still going to want to be able to run reports for old hosts, service, etc., we store the name of the object here so you're not completely baffled by the reports you get. :-)

Structure:

Field	Type	Notes	Values	Key
object_id	INT	A unique number identifying the object		PK
instance_id	SMALLINT	A number indicating the instance of Icinga to which the object belongs		
objecttype_id	SMALLINT	A number indicating what type of object this is	1 = Host; 2 = Service; 3 = Host group; 4 = Service group; 5 = Host escalation; 6 = Service escalation; 7 = Host dependency; 8 = Service dependency; 9 = Timeperiod; 10 = Contact; 11 = Contact group; 12 = Command; 13 = Extended host info (deprecated); 14 = Extended service info (deprecated)	NK1
name1	VARCHAR(128)	The first name associated with the object definition, as used in your Icinga configuration files		NK2
name2	VARCHAR(128)	The second name (if any) associated with the object definition, as used in your Icinga configuration files. This field is only used for service definitions which have a host name (name1 field) and service description (name2 field)		NK3
is_active	SMALLINT	A number indicating whether or not the object is currently defined in your Icinga configuration files. If an object definition is removed from your Icinga configuration files, it will remain in this table, but will be marked as inactive	0 = Inactive; 1 = Active	

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Debugging Tables

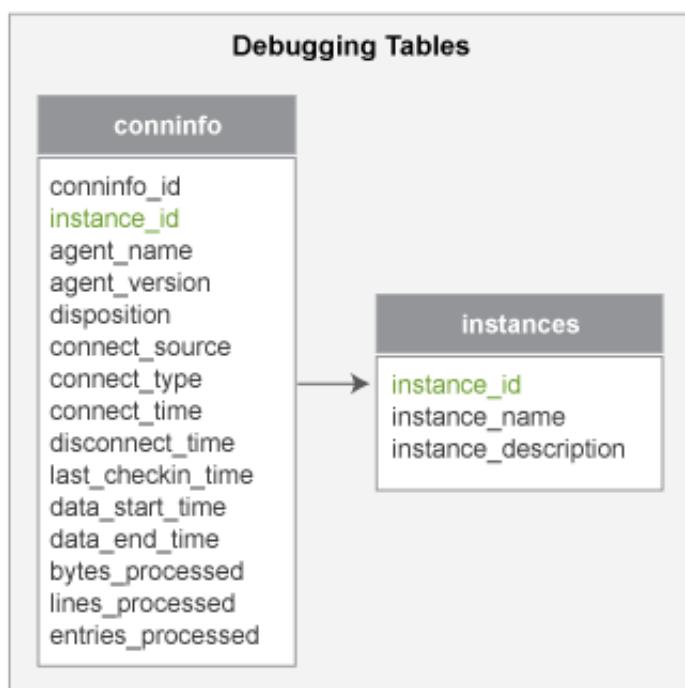
There is currently only one table in the database that is used to hold information that might be useful for debugging purposes. Read below for more information.

Table List

- [conninfo](#)

Relationship Diagram

Figure 12.15. Relationship of Debugging Tables



Conninfo Table

Description: This table is used to store debugging information regarding the IDO2DB daemon and the user agents (e.g. LOG2DB, IDOMOD NEB module, etc.) that connect to it. This information is probably only interesting if you are attempting to debug connection problems.

Structure:

Field	Type	Notes	Values	Key
conninfo_id	INT	Unique number identifying the connection info record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga for which data is being transmitted/processed		

agent_name	VARCHAR(32)	Text string identifying the user agent that is sending data to the IDO2DB daemon	Typically "IDOMOD" or "LOG2IDO"	
agent_version	VARCHAR(8)	Text string identifying the version of the user agent that is sending data		
disposition	VARCHAR(16)	Text string identifying the disposition or type of data that is being sent to the IDO2DB daemon	"REALTIME" if being sent directly from a running Icinga process or "ARCHIVED" if being sent from a flat file	
connect_source	VARCHAR(16)	Text string identifying the method that the user agent is using to connect to the IDO2DB daemon	"TCPSOCKET" or "UNIXSOCKET"	
connect_type	VARCHAR(16)	Text string identifying whether this connect was a new connection, or if it was a reconnect due to an earlier communications failure between the user agent and the IDO2DB daemon	"INITIAL" or "RECONNECT"	
connect_time	DATETIME	The initial time the user agent connected to the daemon		
disconnect_time	DATETIME	The time (if any) the user agent disconnect from the daemon		
last_checkin_time	DATETIME	The time that the user agent last checked in with the daemon to indicate that it was still alive and sending data		
data_start_time	DATETIME	The timestamp of the first data that the user agent sent to the daemon		
data_end_time	DATETIME	The timestamp of the last (or latest) data that the user agent sent to the daemon		
bytes_processed	INT	The number of bytes of data that have been sent by the user agent and processed by the daemon		

lines_processed	INT	The number of lines of data that have been sent by the user agent and processed by the daemon		
entries_processed	INT	The number of data entries that have been sent by the user agent and processed by the daemon		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Historical Tables

Historical Data Tables

There are several tables in the database which are used to hold "historical" information about Icinga and the hosts/services it is monitoring or was monitoring at some point in the past. Keep in mind that historical items may not necessarily be "old" - they could have occurred 5 seconds ago, so the information used within these tables could/should be used when reporting current status information. Links to hosts/services which no longer exist in the Icinga configuration are maintained due to references for these previous objects existing in the objects table - this is by design.

Table List

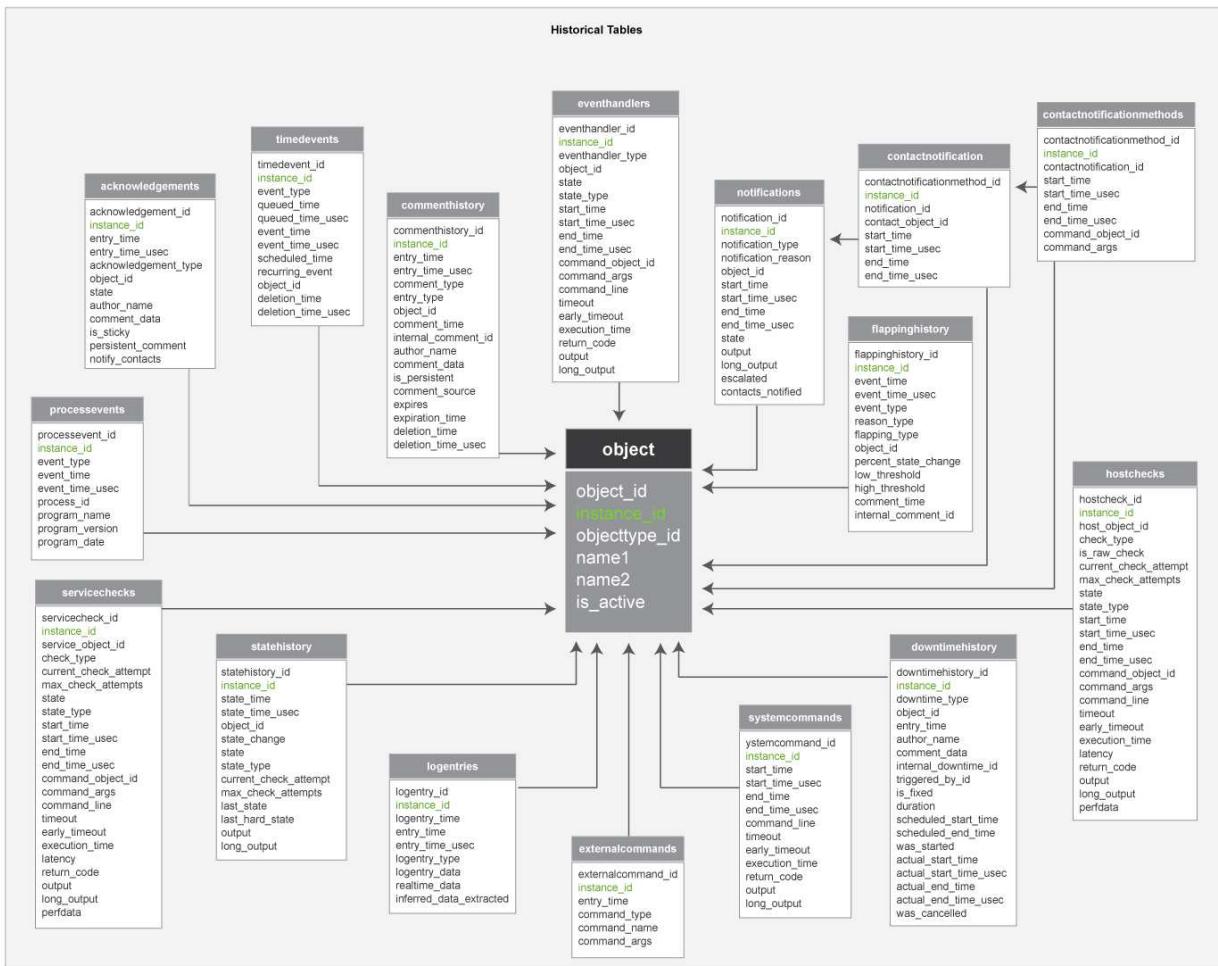
- [acknowledgements](#)
- [commenthistory](#)
- [contactnotifications](#)
- [downtimehistory](#)
- [eventhandlers](#)
- [externalcommands](#)
- [flappinghistory](#)
- [hostchecks](#)
- [logentries](#)
- [notifications](#)
- [processevents](#)
- [servicechecks](#)

- [statehistory](#)
- [systemcommands](#)
- [timedevents](#)

Relationship Diagram

Notes: For clarity, the instances table (to which all these tables are related) is not shown. There are 17 historical tables, so please excuse the mess. :-)

Figure 12.16. Relationship of Historical Tables



Acknowledgements Table

Table Description: This table is used to store host and service acknowledgements for historical purposes.

Structure:

Field	Type	Notes	Values	Key
acknowledgement_id	INT	Unique number identifying the acknowledgement record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
entry_time	DATETIME	Date and time the acknowledgement was entered		

entry_time_usec	INT	Microsecond portion of acknowledgement entry time		
acknowledgement_type	SMALLINT	Indicates whether this is a host or service acknowledgement	0 = Host ack; 1 = Service ack	
object_id	INT	The object id of the host or service this acknowledgement applies to		
state	SMALLINT	Integer indicating the state the host or service was in when the acknowledgement was made	Host acks: 0 = UP; 1 = DOWN; 2 = UNREACHABLE; Service acks: 0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
author_name	VARCHAR(64)	Text field containing the name of the person who made the acknowledgement		
comment_data	VARCHAR(255)	Text field containing notes on the acknowledgement		
is_sticky	SMALLINT	Indicates whether or not the acknowledgement is considered "sticky"	0 = Not sticky; 1 = Sticky	
persistent_comment	SMALLINT	Indicates whether or not the comment associated with the acknowledgement is persistent	0 = Not persistent; 1 = Persistent	
notify_contacts	SMALLINT	Indicates whether or not contacts are to be notified of the acknowledgement	0 = Don't notify; 1 = Notify	

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Commenthistory Table

Table Description: This table is used to store historical host and service comments. Current comments will also appear in this table, but it is recommended to use the comments table to retrieve a list of current host and service comments.

Structure:

Field	Type	Notes	Values	Key
commenthistory_id	INT	Unique number identifying the comment record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
entry_time	DATETIME	Date and time the comment was entered		
entry_time_usec	INT	Microsecond portion of comment entry time		
comment_type	SMALLINT	Indicates whether this is a host or service comment	1 = Host comment; 2 = Service comment	
entry_type	SMALLINT	Indicates how this comment came to be entered	1 = User; 2 = Scheduled downtime; 3 = Flapping; 4 = Acknowledgement	
object_id	INT	The object id of the host or service this acknowledgement applies to		
comment_time	DATETIME	Date and time associated with the comment		UK2
Internal_comment_id	INT	The comment ID internal to the Icinga daemon, which may no longer be valid or present		UK3
author_name	VARCHAR(64)	Text field containing the name of the person who made the comment		
comment_data	VARCHAR(255)	Text field containing the comment		

is_persistent	SMALLINT	Indicates whether or not the comment is persistent	0 = Not persistent; 1 = Persistent	
comment_source	SMALLINT	Indicates the source of the comment	0 = Internal (Icinga); 1 = External (user)	
expires	SMALLINT	Indicates whether or not the comment expires	0 = Doesn't expire; 1 = Expires	
expiration_time	DATETIME	Date and time at which the comment expires		
deletion_time	DATETIME	Date and time (if any) when the comment was deleted		
deletion_time_usec	INT	Microsecond time (if any) when the comment was deleted		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Contactnotifications Table

Description: This table is used to store a historical record of host and service notifications that have been sent out to individual contacts.

Structure:

Field	Type	Notes	Values	Key
contactnotification_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
notification_id	INT	The id of the notification this record is associated with		
contact_object_id	INT	The object id of the contact this notification was send to		UK2
start_time	DATETIME	The date/time the notification to this contact was started		UK3
start_time_usec	INT	The microsecond portion of the time the notification started		UK4
end_time	DATETIME	The date/time the notification to this contact ended		
end_time_usec	INT	The microsecond portion of the time the notification ended		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
notification_id	notifications.notification_id
contact_object_id	objects.object_id

Contactnotificationmethods Table

Description: This table is used to store a historical record of commands (methods) that were used to contact individuals about host and service problems and recoveries.

Structure:

Field	Type	Notes	Values	Key
contactnotificationmethod_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
contactnotification_id	INT	The id of the contact notification this record is associated with		UK2
start_time	DATETIME	The date/time the notification command started		UK3
start_time_usec	INT	The microsecond portion of the time the notification command started		UK4
end_time	DATETIME	The date/time the notification command ended		
end_time_usec	INT	The microsecond portion of the time the notification command ended		
command_object_id	INT	The id of the command that was used for the notification command		
command_args	VARCHAR	The arguments that were passed to the notification command		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
contactnotification_id	contactnotifications.contactnotification_id
command_object_id	objects.object_id

Downtimehistory Table

Description: This table is used to store a historical record of scheduled host and service downtime

Structure:

Field	Type	Notes	Values	Key
downtimehistory_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1

downtime_type	SMALLINT	A number identifying what type of scheduled downtime this is 1 = Service downtime; 2 = Host downtime		
object_id	INT	The object id of the host or service this scheduled downtime is associated with		UK2
entry_time	DATETIME	The date/time the scheduled downtime was entered/submitted		UK3
author_name	VARCHAR	The name of the person who scheduled this downtime		
comment_data	VARCHAR	A comment, as entered by the author, associated with the scheduled downtime		
internal_downtime_id	INT	A number (internal to the Icinga daemon) associated with the scheduled downtime		UK4
triggered_by_id	INT	The id of another scheduled downtime entry that scheduled downtime is optionally triggered by. Non-triggered downtimes will have a value of 0 in this field		
is_fixed	SMALLINT	A number indicating whether or not this scheduled downtime is fixed (i.e. its start and end times are exactly what they are listed below as) or if it is flexible	0 = Flexible (Not fixed); 1 = Fixed	
duration	SMALLINT	The number of seconds that the scheduled downtime should last. This is only used by Icinga if the downtime is flexible. If the downtime is fixed, this value should reflect the difference between the start and end times		
scheduled_start_time	DATETIME	The date/time the scheduled downtime is supposed to start. If this is a flexible (non-fixed) downtime, this refers to the earliest possible time that the downtime can start		
scheduled_end_time	DATETIME	The date/time the scheduled downtime is supposed to end. If this is a flexible (non-fixed) downtime, this refers to the last possible time that the downtime can start		

was_started	SMALLINT	Number indicated whether or not the scheduled downtime was started. Some flexible downtimes may never actually start if the host/service they are associated with never enter a problem state	0 = Was not started; 1 = Was started	
actual_start_time	DATETIME	The date/time the scheduled downtime was actually started (if applicable)		
actual_start_time_usec	INT	Microsecond portion of the actual start time		
actual_end_time	DATETIME	The date/time the scheduled downtime actually ended		
actual_end_time_usec	INT	Microsecond portion of the actual end time		
was_cancelled	SMALLINT	Number indicating whether or not the scheduled downtime was cancelled before it ended normally	0 = Not cancelled; 1 = Cancelled early	

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id
triggered_by_id	[downtimehistory.]downtimehistory_id

Eventhandlers Table

Description: This table is used to store a historical record of host and service event handlers that have been run. NOTE: This table is usually trimmed periodically by the IDO2DB daemon, as it would otherwise grow to an enormous size.

Structure:

Field	Type	Notes	Values	Key
eventhandler_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1

eventhandler_type	SMALLINT	A number indicating what type of event handler this is	0 = Host event handler; 1 = Service event handler; 2 = Global host event handler; 3 = Global service event handler	
object_id	INT	The object id of the host or service associated with this event handler		UK2
state	SMALLINT	Number indicating the state of host or service when the event handler was run.	For host event handlers: 0 = UP; 1 = DOWN; 2 = UNREACHABLE; For service event handlers: 0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
state_type	SMALLINT	Number indicating the state type of the host or service when the event handler was run	0 = SOFT state; 1 = HARD state	
start_time	DATETIME	The date/time the event handler started		UK3
start_time_usec	INT	The microsecond portion of the time the event handler started		UK4
end_time	DATETIME	The date/time the event handler ended		
end_time_usec	INT	The microsecond portion of the time the event handler ended		
command_object_id	INT	The id of the command that was run		
command_args	ARGS	Arguments to the event handler command that was run		
command_line	ARGS	Fully expanded command line of the event handler that was run		
timeout	SMALLINT	Timeout value in seconds for the event handler		

early_timeout	SMALLINT	Number indicating whether or not the event handler command timed out	0 = Did NOT time out. 1 = Timed out	
execution_time	DOUBLE	Time in seconds that the event handler command was running		
return_code	SMALLINT	The return code value from the event handler command		
output	VARCHAR	The first line of text output (if any) from the event handler command		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id
command_object_id	objects.object_id

Externalcommands Table

Description: This table is used to store a historical record of external commands that have been processed by the Icinga daemon.

Structure:

Field	Type	Notes	Values	Key
externalcommand_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
entry_time	DATETIME	The date/time the external command was processed		
command_type	SMALLINT	A number indicating what type of external command this is. Each external command has its own type or "id"	See Icinga source code	
command_name	VARCHAR	The name of the command that was processed		
command_args	VARCHAR	Optional arguments that were specified with the command.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Flappinghistory Table

Table Description: This table is used to store a historical record of host and service flapping events.

Structure:

Field	Type	Notes	Values	Key
flappinghistory_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
event_time	DATETIME	The date/time of the flapping event		
event_time_usecs	INT	The microsecond portion of the time of the flapping event		
event_type	SMALLINT	The type of flapping event indicated by this record	1000 = Flapping started; 1001 = Flapping stopped	
reason	SMALLINT	Number indicating the reason (if any) that the host or service stopped flapping. This is only valid if this record is a flapping stopped event (see event_type field)	1 = Flapping stopped normally 2 = Flapping was disabled	
flapping_type	SMALLINT	Number indicating whether this flapping event relates to a host or service	0 = Host 1 = Service	
object_id	INT	The id of the host or service associated with the flapping event		
percent_state_change	DOUBLE	The percent state change of the host or service at the time of the event		
low_threshold	DOUBLE	The low flapping percent state change threshold (as configured in Icinga) of the host or service		
high_threshold	DOUBLE	The high flapping percent state change threshold (as configured in Icinga) of the host or service		
comment_time	DATETIME	The date/time of the comment associated with the flapping event		
internal_comment_id	INT	The number (internal to the Icinga daemon) of the comment associated with the flapping event.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Hostchecks Table

Description: This table is used to store a historical record of "raw" and "processed" host checks. What's the difference between raw and processed host checks? Raw checks are the raw results from a host check command that gets executed. Icinga must do some processing on the raw host check results before it can determine the real state of the host. Host checks (plugins) cannot directly determine whether a host is DOWN or UNREACHABLE - only Icinga can do that. In fact, host checks return the same status codes as service checks (OK, WARNING, UNKNOWN, or CRITICAL). Icinga processes the raw host check result to determine the true state of the host (UP, DOWN, or UNREACHABLE). These "processed" checks contain the the true state of the host. NOTE: This table is usually trimmed periodically by the IDO2DB daemon, as it would otherwise grow to an enormous size.

Structure:

Field	Type	Notes	Values	Key
hostcheck_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
host_object_id	INT	The id of the host this check applies to		UK2
check_type	SMALLINT	Number indicating whether this is an active or passive check	0 = Active check 1 = Passive check	
is_raw_check	SMALLINT	Number indicating whether this is a "raw" or "processed" host check	0 = Processed check; 1 = Raw check	
current_check_attempt	SMALLINT	Current check attempt of the host		
max_check_attempts	SMALLINT	Max check attempts (as defined in Icinga) for the host		

state	SMALLINT	Current state of the host	For raw checks: 0 = UP 1 = DOWN/UNREACHABLE; For processed checks: 0 = UP 1 = DOWN 2 = UNREACHABLE	
state_type	SMALLINT	Number indicating whether the host is in a soft or hard state	0 = SOFT state 1 = HARD state	
start_time	DATETIME	The date/time the host check was started		UK3
start_time_usec	INT	Microsecond portion of the time the host check was started		UK4
end_time	DATETIME	The date/time the host check was completed		
end_time_usec	INT	Microsecond portion of the time the host check was completed		
command_object_id	INT	The id of the command that was used to perform the host check		
command_args	VARCHAR	The arguments that were passed to the host check command		
command_line	VARCHAR	The fully expanded command line that was used to check the host		
timeout	SMALLINT	Number of seconds before the host check command would time out		
early_timeout	SMALLINT	Number indicating whether or not the host check timed out early	0 = Did NOT timeout 1 = Timed out	
execution_time	DOUBLE	Number of seconds it took to execute the host check		

latency	DOUBLE	Number of seconds the host check was "late" in being executed. Scheduled host checks can have a latency, but on-demand checks will have a latency of 0. Latency is the difference between the time the check was scheduled to be executed and the time it was actually executed. For passive checks it is the difference between the timestamp on the passive host check result (submitted through the external command file) and the time the passive check result was processed by Icinga		
return_code	SMALLINT	The return code from the host check command		
output	VARCHAR	Status text output from the host check command		
perfdata	VARCHAR	Optional performance data returned from the host check command.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
host_object_id	objects.object_id
command_object_id	objects.object_id

Logentries Table

Description: This table is used to store a historical record of entries from the Icinga log.

Structure:

Field	Type	Notes	Values	Key
logentry_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
logentry_time	DATETIME	The date/time associated with the log entry. This is NOT necessarily the same as the date/time that Icinga wrote the log entry to the log file (see below)		
entry_time	DATETIME	The date/time that Icinga wrote this log entry to the log file		
entry_time_usec	INT	The microsecond portion of the time that Icinga wrote this log entry		
logentry_type	INT	A number indicating what general type of log entry this is	See Icinga source code	
logentry_data	VARCHAR	The log entry that was written out to the log file		
realtime_data	SMALLINT	A number used internally by the IDO2DB daemon		
inferred_data_extracted	SMALLINT	A number used internally by the IDO2DB daemon.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Notifications Table

Description: This table is used to store a historical record of host and service notifications that have been sent out. For each notification, one or more contacts receive notification messages. These contact notifications are stored in the contactnotifications table.

Structure:

Field	Type	Notes	Values	Key

notification_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
notification_type	SMALLINT	Number indicating whether this is a host or service notification	0 = Host notification 1 = Service notification	
notification_reason	SMALLINT	Number indicating the type of or reason for the notification	0 = Normal notification; 1 = Problem acknowledgement; 2 = Flapping started; 3 = Flapping stopped; 4 = Flapping was disabled; 5 = Downtime started; 6 = Downtime ended; 7 = Downtime was cancelled; 99 = Custom notification	
object_id	INT	The id of the host or service this notification applies to		UK2
start_time	DATETIME	The date/time the notification was started		UK3
start_time_usec	INT	Microsecond portion of the time the notification was started		UK4
end_time	DATETIME	The date/time the notification ended		
end_time_usec	INT	Microsecond portion of the time the notification ended		
state	SMALLINT	Number indicating the state of the host or service when the notification was sent out.	For Host Notifications: 0 = UP; 1 = DOWN; 2 = CRITICAL; For Service Notifications: 0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	

output	VARCHAR	The current plugin (text) output of the host or service when the notification was sent out		
escalated	SMALLINT	Number indicating whether or not this notification was escalated or not	0 = NOT escalated; 1 = Escalated	
contacts_notified	SMALLINT	Number of contacts that were notified about the host or service as part of this notification.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Processevents Table

Description: This table is used to store a historical record of Icinga process events (program starts, restarts, shutdowns, etc.).

Structure:

Field	Type	Notes	Values	Key
processevent_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
event_type	SMALLINT	Number indicating the type of process event that occurred.	100 = Process start; 101 = Process daemonized; 102 = Process restart; 103 = Process shutdown; 104 = Prelaunch; 105 = Event loop start; 106 = Event loop end	
event_time	DATETIME	The date/time that the event occurred		
event_time_usecs	INT	The microsecond portion of the time the event occurred		
process_id	INT	The current process ID (PID) of the Icinga daemon		
program_name	VARCHAR	"Icinga"		
program_version	VARCHAR	Version of Icinga that is running (e.g. "1.0")		
program_date	VARCHAR	Release date of Icinga		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Servicechecks Table

Description: This table is used to store a historical record of service checks that have been performed. NOTE: This table is usually trimmed periodically by the IDO2DB daemon, as it would otherwise grow to an enormous size.

Structure:

Field	Type	Notes	Values	Key
servicecheck_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1

service_object_id	INT	The id of the service this record refers to		UK2
check_type	SMALLINT	Number indicating whether this was an active or a passive service check	0 = Active check; 1 = Passive check	
current_check_attempt	SMALLINT	Number indicating the current check attempt for the service		
max_check_attempts	SMALLINT	Number indicating the max number of check attempts for the service		
state	SMALLINT	Number indicating the current state of the service	0 = OK 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
state_type	SMALLINT	Number indicating the current state type of the service	0 = SOFT state; 1 = HARD state	
start_time	DATETIME	The date/time the service check was started		UK3
start_time_usec	INT	Microsecond portion of the time the service check was started		UK4
end_time	DATETIME	The date/time the service check ended		
end_time_usec	INT	Microsecond portion of the time the service check ended		
command_object_id	INT	The id of the command that was run to perform the service check		
command_args	VARCHAR	The arguments passed to the command that was run to perform the service check		
command_line	VARCHAR	The fully expanded command line that was executed to perform the service check		
timeout	SMALLINT	Number of seconds before the service check command was scheduled to timeout		
early_timeout	SMALLINT	Number indicating whether or not the service check timed out	0 = Did NOT timeout 1 = Timed out	
execution_time	DOUBLE	Number of seconds it took to execute the service check command		

latency	DOUBLE	Number of seconds the service check was "late" in being executed. For active checks this is the difference between the scheduled service check time and the time the check actually occurred. For passive checks this is the difference between the timestamp on the passive check result (submitted through the external command file) and the time the passive check result was picked up by the Icinga daemon for processing		
return_code	SMALLINT	The return code from the service check command		
output	VARCHAR	The status output that was returned from the service check command		
perfdata	VARCHAR	Optional performance data that was returned from the service check command		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
service_object_id	objects.object_id
command_object_id	objects.object_id

Statehistory Table

Description: This table is used to store a historical record of host and service state changes.

Structure:

Field	Type	Notes	Values	Key
statehistory_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		

state_time	DATETIME	The date/time that the state change occurred		
state_time_usec	INT	The microsecond portion of the time the state change occurred		
object_id	INT	The id of the host or service object this state change applies to		
state_change	SMALLINT	Number indicating whether or not a state change occurred for the host or service	0 = No state change; 1 = State change	
state	SMALLINT	Number indicating the current state of the host or service	For Hosts: 0 = UP; 1 = DOWN; 2 = UNREACHABLE; For Services: 0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
state_type	SMALLINT	Number indicating whether the service is in a soft or hard state	0 = SOFT state; 1 = HARD state	
current_check_attempt	SMALLINT	Number indicating the current check attempt for the host or service		
max_check_attempts	SMALLINT	Number indicating the max check attempts (as configured in Icinga) for the host or service		
last_state	SMALLINT	Number indicating the last state (whether hard or soft) of the host or service (if available)	For Hosts: -1 = unavailable; 0 = UP; 1 = DOWN; 2 = UNREACHABLE; For Services: -1 = unavailable; 0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
last_hard_state	SMALLINT	Number indicating the last hard state of the host or service (if available)	For Hosts: -1 = unavailable; 0 = UP; 1 = DOWN; 2 = UNREACHABLE; For Services: -1 unavailable; 0 = OK; 1 = WARNING; 2 = CRITICAL	

output	VARCHAR	The current plugin/status output of the host or service		
--------	---------	---	--	--

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Systemcommands Table

Description: This table is used to store a historical record of system commands that are run by the Icinga daemon. Note that each event handler, notification, OCSP command, etc. requires that Icinga execute a system command. NOTE: This table is usually trimmed periodically by the IDO2DB daemon, as it would otherwise grow to an enormous size.

Structure:

Field	Type	Notes	Values	Key
systemcommand_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
start_time	DATETIME	The date/time the command was executed		UK2
start_time_usec	INT	The microsecond portion of the time the command was executed		UK3
end_time	DATETIME	The date/time the command finished executing		
end_time_usec	INT	The microsecond portion of the time the command finished executing		
command_line	VARCHAR	Fully expanded command line that was executed		
timeout	SMALLINT	Number of seconds before the command should timeout		
early_timeout	SMALLINT	Number indicating whether or not the command timed out early	0 = Did NOT timeout; 1 = Timed out	
execution_time	DOUBLE	Number of seconds it took to execute the command		
return_code	SMALLINT	Return code of the command		
output	VARCHAR	First line of text output (if available) that was returned from the command		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Timedevents Table

Description: This table is used to store a historical record of timed events that the Icinga process handled. Timed events are internal to the Icinga daemon and used to initiate service checks, host checks, status file updates, etc. They are at the heart of what Icinga does and how it operates. NOTE: This table is usually trimmed periodically by the IDO2DB daemon, as it would otherwise grow to an enormous size.

Structure:

Field	Type	Notes	Values	Key
systemcommand_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
event_type	SMALLINT	Number indicating the type of event that was run	See Icinga source code	UK2
queued_time	DATETIME	The date/time the event was added to the event queue		
queued_time_usec	INT	Microsecond portion of the time the event was added to the event queue		
event_time	DATETIME	The date/time the event was handled		
event_time_usec	INT	Microsecond portion of the time the event was handled		
scheduled_time	DATETIME	The date/time the event was scheduled to be handled/run		UK3
recurring_event	SMALLINT	Number indicating whether or not the event is a recurring one or a one-time event	0 = One-time event; 1 = Recurring event	
object_id	INT	The id of the host or service that the event applies to. Not all events apply to hosts or services - in these cases the value of this field will be 0. deletion_time DATETIME The date/time the event was deleted/removed from the event queue		UK4
deletion_time_usec	INT	Microsecond portion of the time the event was removed from the event queue		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Current Status Tables

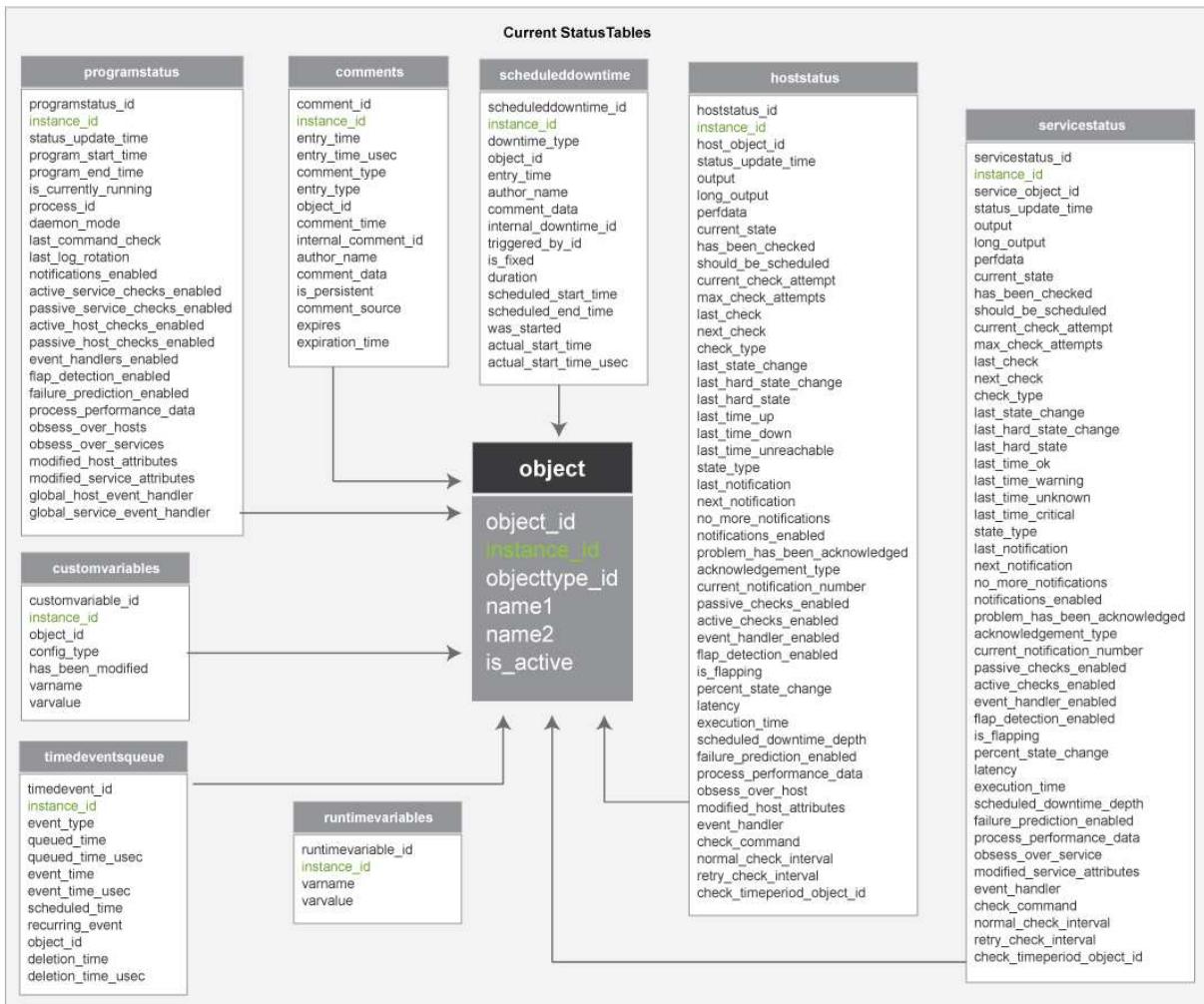
There are several tables in the database which are used to hold current status information on the Icinga process and all hosts and services that it is monitoring. Entries in these tables are cleared whenever the Icinga daemon (belonging to the same instance) (re)starts

Table List

- [comments](#)
- [customvariablestatus](#)
- [hoststatus](#)
- [programstatus](#)
- [runtimevariables](#)
- [scheduleddowntime](#)
- [servicestatus](#)
- [timedeventqueue](#)

*Relationship Diagram*Notes: To reduce clutter, the links to the instances table (to which all these tables are related) is not shown.

Figure 12.17. Relationship of Current Status Tables



Comments Table

Description: This table is used to store current host and service comments. Historical comments can be found in the commenthistory table.

Structure:

Field	Type	Notes	Values	Key
comment_id	INT	Unique number identifying the comment record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
entry_time	DATETIME	Date and time the comment was entered		
entry_time_usec	INT	Microsecond portion of comment entry time		

comment_type	SMALLINT	Indicates whether this is a host or service comment	1 = Host comment; 2 = Service comment	
entry_type	SMALLINT	Indicates how this comment came to be entered	1 = User; 2 = Scheduled downtime; 3 = Flapping; 4 = Acknowledgement	
object_id	INT	The object id of the host or service this acknowledgement applies to		
comment_time	DATETIME	Date and time associated with the comment		UK2
internal_comment_id	INT	The comment ID internal to the Icinga daemon		UK3
author_name	VARCHAR(64)	Text field containing the name of the person who made the comment		
comment_data	VARCHAR(255)	Text field containing the comment		
is_persistent	SMALLINT	Indicates whether or not the comment is persistent	0 = Not persistent; 1 = Persistent	
comment_source	SMALLINT	Indicates the source of the comment	0 = internal (Icinga); 1 = External (user)	
expires	SMALLINT	Indicates whether or not the comment expires	0 = Doesn't expire; 1 = Expires	
expiration_time	DATETIME	Date and time at which the comment expires.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Customvariablestatus Table

Description: This table is used to store the current state/values of all custom host, service, and contact variables. Custom variables are only support in Icinga or Nagios 3.x and higher, so this table will be empty for Nagios 2.x.

Structure:

Field	Type	Notes	Values	Key
customvariablestatus_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
object_id	INT	The object id of the host or service this acknowledgement applies to		UK1
status_update_time	DATETIME	Date and time the status of the custom variable was last updated		
has_been_modified	INT	Indicates whether the value of the custom variable has been modified (during runtime) from its original value in the config files	0 = Has not been modified; 1 = Has been modified	
varname	VARCHAR(255)	Text field containing the name of the custom variable		UK2, NK
varvalue	VARCHAR(255)	Text field containing the value of the custom variable		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Hoststatus Table

Description: This table is used to store the current status of hosts that are being monitored.

Structure:

Field	Type	Notes	Values	Key
hoststatus_id	INT	Unique number identifying the record		PK

instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
host_object_id	INT	The object id of the host this status entry is associated with		U1
status_update_time	DATETIME	Date and time the status data was updated		
output	VARCHAR	Plugin output from the latest host check		
perfdata	VARCHAR	Performance data from the latest host check		
current_state	SMALLINT	Number indicating the current state of the host	0 = UP; 1 = DOWN; 2 = UNREACHABLE	
has Been checked	SMALLINT	Number indicating whether or not the host has been checked yet	0 = Not checked; 1 = Checked	
should_be_scheduled	SMALLINT	Number indicating whether or not checks should be regularly scheduled for this host	0 = Not scheduled; 1 = Scheduled	
current_check_attempt	SMALLINT	Number indicating the current check attempt of the host. This is only interesting during soft host states		
max_check_attempts	SMALLINT	Number indicating how many maximum check attempts will be made to determine the hard state of the host		
last_check	DATETIME	Time the host was last checked		

next_check	DATETIME	The host is scheduled to be checked next. Will be set to the epoch if the host is not scheduled for another check		
check_type	SMALLINT	Number indicating if the last host check was an active or passive check	0 = Active; 1 = Passive	
last_state_change	DATETIME	Time the host last had a hard or soft state change. Will be set to the epoch if the host has not changed state		
last_hard_state_change	DATETIME	The host last had a hard state change. Will be setup to the epoch if the host has not changed state		
last_time_up	DATETIME	Time the host was last in an UP state (if ever)		
last_time_down	DATETIME	Time the host was last in a DOWN state (if ever)		
last_time_unreachable	DATETIME	Time the host was last in an UNREACHABLE state (if ever)		
state_type	SMALLINT	Number indicating the type of state the host is in	0 = SOFT state; 1 = HARD state	
last_notification	DATETIME	Time a notification was last sent out for the host (if ever)		
next_notification	DATETIME	Next possible time that a notification can be sent out for the host		

no_more_notifications	SMALLINT	Number indicating whether or not more notifications can be sent out about the current host problem	0 = Send notifications; 1 = Do not send notifications	
notifications_enabled	SMALLINT	Number indicating whether or not notifications are enabled for this host	0 = Notifications disabled; 1 = Notifications enabled	
problem_has_been_acknowledged	SMALLINT	Number indicating whether or not the current host problem has been acknowledged	0 = Not acknowledged; 1 = Acknowledged	
acknowledgement_type	SMALLINT	Number indicating the type of acknowledgement associated with the host	0 = None; 1 = Normal; 2 = Sticky	
current_notification_number	SMALLINT	Number indicating the current notification number for the current host problem. This number gets reset to 0 when the host recovers		
passive_checks_enabled	SMALLINT	Number indicating whether or not passive checks are enabled for this host	0 = Disabled; 1 = Enabled	
active_checks_enabled	SMALLINT	Number indicating whether or not active checks are enabled for this host	0 = Disabled; 1 = Enabled	
event_handler_enabled	SMALLINT	Number indicating whether or not the host's event handler is enabled	0 = Disabled; 1 = Enabled	
flap_detection_enabled	SMALLINT	Number indicating whether or not flap detection is enabled for this host	0 = Disabled; 1 = Enabled	

is_flapping	SMALLINT	Number indicating whether or not the host is currently flapping	0 = Not flapping; 1 = Flapping	
percent_state_change	DOUBLE	Number indicating the current percent state change (a measure of stability/volatility) for the host		
latency	DOUBLE	Number of seconds that the host check was "late" in being executed. The difference between the checks scheduled time and the time it was actually checked		
execution_time	DOUBLE	Number of seconds it took to perform the last check of the host		
scheduled_downtime_depth	SMALLINT	Number indicating how many periods of scheduled downtime are currently active for this host; >0 = In scheduled downtime	0 = Not in scheduled downtime downtime are currently active for this host; >0 = In scheduled downtime	
failure_prediction_enabled	SMALLINT	Number indicating whether or not failure prediction (not yet implemented) is enabled for this host	0 = Disabled; 1 = Enabled	
process_performance_data	SMALLINT	Number indicating whether or not performance data should be processed for this host	0 = Disabled; 1 = Enabled	
obsess_over_host	SMALLINT	Number indicating whether or not this host should be obsessed over	0 = Do not obsess; 1 = Obsess	

modified_host_attributes	INT	Number indicating which attributes of the host have been modified during runtime. Used by the retention data routines		
event_handler	VARCHAR	The current event handler command associated with the host		
check_command	VARCHAR	The current check command associated with the host		
check_interval	DOUBLE	Number of seconds between normal checks of the host		
retry_interval	DOUBLE	Number of seconds between retry checks of the host		
check_timeperiod_object_id	INT	Unique number of the timeperiod object currently used for determining times the host can be checked		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
host_object_id	objects.object_id
timeperiod_object_id	objects.object_id

Programstatus Table

Description: This table stored status information on the currently (or previously) running Icinga process/daemon.

Structure:

Field	Type	Notes	Values	Key

programstatus_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		U1
status_update_time	DATETIME	Date and time the status of the process was last updated		
program_start_time	DATETIME	Date and time the Icinga process was started		
program_end_time	DATETIME	Date and time the Icinga process was stopped (if currently not running)		
is_currently_running	SMALLINT	Indicates whether or not the Icinga process is currently running	0 = Process is not running; 1 = Process is running	
process_id	INT	The processed ID (PID) of the Icinga process		
daemon_mode	SMALLINT	Indicates whether Icinga is running as a foreground process or a daemon	0 = Foreground process; 1 = Daemon	
last_command_check	DATETIME	Date and time the Icinga process last checked external commands		
last_log_rotation	DATETIME	Date and time the log file was last rotated (if at all)		
notifications_enabled	SMALLINT	Indicates whether or not notifications are enabled	0 = Disabled; 1 = Enabled	
active_service_checks_enabled	SMALLINT	Indicates whether or not active service checks are enabled	0 = Disabled; 1 = Enabled	
passive_service_checks_enabled	SMALLINT	Indicates whether or not passive service checks are enabled	0 = Disabled; 1 = Enabled	

active_host_checks_enabled	SMALLINT	Indicates whether or not active host checks are enabled	0 = Disabled; 1 = Enabled	
passive_host_checks_enabled	SMALLINT	Indicates whether or not passive host checks are enabled	0 = Disabled; 1 = Enabled	
event_handlers_enabled	SMALLINT	Indicates whether or not event handlers are enabled	0 = Disabled; 1 = Enabled	
flap_detection_enabled	SMALLINT	Indicates whether or not flap detection is enabled	0 = Disabled; 1 = Enabled	
failure_prediction_enabled	SMALLINT	Indicates whether or not failure prediction is enabled	0 = Disabled; 1 = Enabled	
process_performance_data	SMALLINT	Indicates whether or not performance data is enabled/being processed	0 = Disabled; 1 = Enabled	
obsess_over_hosts	SMALLINT	Indicates whether or not hosts are being obsessed over	0 = Disabled; 1 = Enabled	
obsess_over_services	SMALLINT	Indicates whether or not services are being obsessed over	0 = Disabled; 1 = Enabled	
modified_host_attributes	INT	Indicates what (if any) host-related program status variables have been modified during runtime	See Icinga source code for values	
modified_service_attributes	INT	Indicates what (if any) service-related program status variables have been modified during runtime	See Icinga source code for values	
global_host_event_handler	VARCHAR(255)	Text field indicating the current global host event handler command that is being used.		

global_service_event_handlers	VARCHAR(255)	Text field indicating the current global service event handler command that is being used		
-------------------------------	--------------	---	--	--

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Runtimevariables Table

Table Description: This table is used to store some runtime variables from the Icinga process that may be useful to you. The only variables currently stored in this table are some initial variables calculated at startup, but more variables may be stored here in future versions.

Structure:

Field	Type	Notes	Values	Key
runtimevariable_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
varname	VARCHAR(64)	Text field containing the name of the variable		UK2
varvalue	VARCHAR(255)	Text field containing the value of the variable		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id

Scheduledowntime Table

Description: This table is used to store current host and service downtime, which may either be current in effect or scheduled to begin at a future time. Historical scheduled downtime information can be found in the downtimehistory table.

Structure:

Field	Type	Notes	Values	Key
scheduledowntime_id	INT	Unique number identifying the record		PK

instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
downtime_type	SMALLINT	Indicates whether this is a host or service downtime entry	1 = Service downtime; 2 = Host downtime	
object_id	INT	The object id of the host or service this downtime applies to		UK2
entry_time	DATETIME	Date and time this downtime was entered		UK3
author_name	VARCHAR(64)	Text field containing the name of the person who created this scheduled downtime		
comment_data	VARCHAR(255)	Text field containing information about this scheduled downtime (as entered by the user)		
internal_downtime_id	INT	The ID number (internal to the Icinga daemon) associated with this scheduled downtime entry		UK4
triggered_by_id	INT	The internal Icinga ID number (if any) of another scheduled downtime entry that this downtime is "triggered" (started) by. If this field is nonzero, this is a triggered downtime entry, otherwise it is not		
is_fixed	SMALLINT	Indicates whether this is a "fixed" scheduled downtime entry (that should start and end at the start and end times indicated) or a "flexible" entry that can start at a variable time	0 = Flexible (not fixed) 1 = Fixed	
duration	SMALLINT	Indicates the number of seconds that the scheduled downtime should last. This is usually only needed if this is "flexible" downtime, which can start at a variable time, but lasts for the specified duration		

scheduled_start_time	DATETIME	Date and time that the downtime is scheduled to start if it is "fixed" downtime. If this is a "flexible" downtime entry, this is the first possible time the downtime can start		
scheduled_end_time	DATETIME	Date and time the downtime is scheduled to end if it is "fixed" downtime. If this is a "flexible" downtime entry, this is the last possible time the downtime can start		
was_started	SMALLINT	Indicates whether or not the downtime was started (is currently #FIXME)	0 = Not started (inactive) 1 = Started (active)	
actual_start_time	DATETIME	Date and time the scheduled downtime was actually started		
actual_start_time_usec	INT	Microsecond portion of time the scheduled downtime was actually started		

Relationships:

Field	Foreign Key	
instance_id	instances.instance_id	
object_id	objects.object_id	

Servicestatus Table

Description: This table is used to store current status information for all services that are being monitored.

Structure:

Field	Type	Notes	Values	Key
servicestatus_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		

service_object_id	INT	The id of the service this record is associated with		U1
status_update_time	DATETIME	The date/time the status record was updated		
output	VARCHAR	The text output from the most current service check		
perfdata	VARCHAR	Optional performance data from the most current service check		
current_state	SMALLINT	Number indicating the current state of the service	0 = OK; 1 = WARNING; 2 = CRITICAL; 3 = UNKNOWN	
has_been_checked	SMALLINT	Number indicating whether or not the service has been checked yet	0 = Has NOT been checked; 1 = Has been checked	
should_be_scheduled	SMALLINT	Number indicating whether or not the service should be scheduled for periodic checks on a regular basis	0 = Not scheduled; 1 = Scheduled	
current_check_attempt	SMALLINT	The current check attempt for the service		
max_check_attempts	SMALLINT	The max check attempts (as configured in Icinga) for the service		
last_check	DATETIME	The date/time the service was last checked. Set to the epoch if the service has not been checked yet		

next_check	DATETIME	The date/time the service is scheduled to be checked next		
check_type	SMALLINT	Number indicating whether or not the last service check was active or passive	0 = Active; 1 = Passive	
last_state_change	DATETIME	The date/time the service last changed state (if at all). This gets updated for both HARD and SOFT state changes		
last_hard_state_change	DATETIME	The date/time the service last changed HARD states (if at all)		
last_time_ok	DATETIME	The date/time the service was last in an OK state (if at all)		
last_time_warning	DATETIME	The date/time the service was last in a WARNING state (if at all)		
last_time_unknown	DATETIME	The date/time the service was last in an UNKNOWN state (if at all)		
last_time_critical	DATETIME	The date/time the service was last in a CRITICAL state (if at all).		
state_type	SMALLINT	Number indicating whether the service is in a hard or soft state	0 = SOFT state; 1 = HARD	
state last_notification	DATETIME	The date/time that a notification was last sent out for the current service problem (if applicable)		

next_notification	DATETIME	The earliest date/time that the next notification can be sent out for the current service problem (if applicable)		
no_more_notifications	SMALLINT	Number indicating whether or not future notifications can be sent out for the current service problem	0 = Do not send more notifications; 1 = Keep sending notifications	
notifications_enabled	SMALLINT	Number indicating whether notifications are enabled for the service	0 = Disabled; 1 = Enabled	
problem_has_been_acknowledged	SMALLINT	Number indicating whether or not the current status problem has been acknowledged	0 = Not acknowledged; 1 = Acknowledged	
acknowledgement_type	SMALLINT	Number indicating the type of acknowledgement (if any)	0 = No acknowledgement; 1 = Normal acknowledgement; 2 = Sticky acknowledgement	
current_notification_number	SMALLINT	Number indicating how many notifications have been sent out about the current service problem (if applicable)		
passive_checks_enabled	SMALLINT	Number indicating whether or not passive checks are enabled for the service	0 = Disabled; 1 = Enabled	

active_checks_enabled	SMALLINT	Number indicating whether or not active checks are enabled for the service	0 = Disabled; 1 = Enabled	
event_handler_enabled	SMALLINT	Number indicating whether or not the service event handler is enabled	0 = Disabled; 1 = Enabled	
flap_detection_enabled	SMALLINT	Number indicating whether or not flap detection is enabled for the service	0 = Disabled; 1 = Enabled	
is_flapping	SMALLINT	Number indicating whether or not the service is currently flapping	0 = Not flapping; 1 = Flapping	
percent_state_change	DOUBLE	Number indicating the current percent state change (a measure of volatility) for the service		

latency	DOUBLE	<p>Number indicating how "late" the last service check was in being run. For active checks, this is the difference between the time the service was scheduled to be checked and the time it was actually checked. For passive checks, this is the difference between the timestamp on the passive check (submitted via an external command) and the time Icinga processed the check result.</p> <p>execution_time DOUBLE Number of seconds it took to run the last service check</p>		
scheduled_downtime_depth	SMALLINT	<p>Number indicating how many periods of scheduled downtime are currently in effect for the service. A value of 0 indicates the service is not in a period of downtime</p>		
failure_prediction_enabled	SMALLINT	<p>Number indicating whether or not failure prediction is enabled for the service. This feature has not yet been implemented</p>	0 = Disabled; 1 = Enabled	

process_performance_data	SMALLINT	Number indicating whether or not performance data should be processed for the service	0 = Do NOT process perftdata; 1 = Process perftdata	
obsess_over_service	SMALLINT	Number indicating whether or not Icinga should obsess of check results of the service	0 = Do NOT obsess; 1 = Obsess	
modified_service_attributes	INT	Number indicating what service attributes have been modified during runtime	See Icinga source code	
event_handler	VARCHAR	The current event handler command that is associated with the service		
check_command	VARCHAR	The current check command that is used to check the status of the service		
check_interval	DOUBLE	The current normal check interval for the service (in seconds)		
retry_interval	DOUBLE	The current retry check interval for the service (in seconds)		
check_timeperiod_object_id	INT	The currently timeperiod that is used to determine when the service can be checked.		

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
service_object_id	objects.object_id
check_timeperiod_object_id	objects.object_id

Timedequeue Table

Description: This table is used to store all timed events that are in the Icinga event queue, scheduled to be executed at a future time. Historical timed events can be found in the timedequeue table.

Structure:

Field	Type	Notes	Values	Key
timedequeue_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
event_type	SMALLINT	Value indicating the type of event		UK2
queued_time	DATETIME	Date and time the event was originally placed into the timed event queue		
queued_time_usec	INT	Microsecond portion of time the event was queued		
scheduled_time	INT	Date and time the event is scheduled to be executed		UK3
recurring_event	SMALLINT	Indicates whether or not this is a recurring event	0 = Not recurring; 1 = Recurring	
object_id	INT	The object id of the host, service, contact, etc. that this scheduled event applies to (if applicable). If the event is not associated with any particular object, this field will have a value of zero (0)		UK4

Relationships:

Field	Foreign Key
instance_id	instances.instance_id
object_id	objects.object_id

Configuration Tables



Note

The tables that contain configuration data have not yet been fully documented.

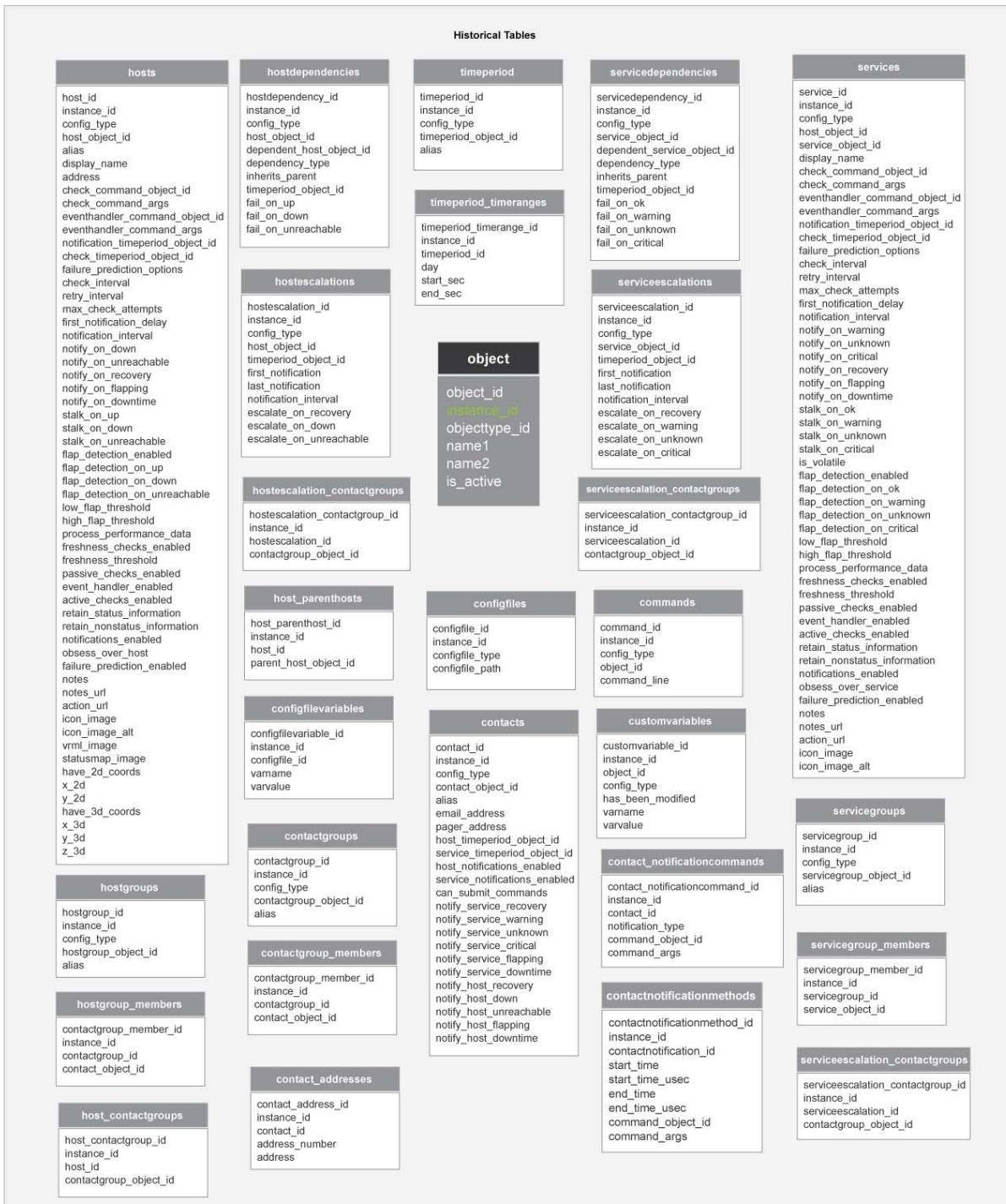
There are many tables in the database that are used to store Icinga configuration. Note that the data in these tables represents a read-only output view of the configuration that Icinga was using during its last (or current) run. Configuration information from these tables is NOT read by the Icinga daemon in any way, and thus cannot be used to configure Icinga.

Table List

- [commands](#)
- [configfiles](#)
- [configfilevariables](#)
- [contact_addresses](#)
- [contact_notificationcommands](#)
- [contactgroup_members](#)
- [contactgroups](#)
- [contactnotificationmethods](#)
- [contacts](#)
- [customvariables](#)
- [host_contactgroups](#)
- [host_parenthosts](#)
- [hostdependencies](#)
- [hostescalation_contactgroups](#)
- [hostescalations](#)
- [hostgroup_members](#)
- [hostgroups](#)
- [hosts](#)
- [service_contactgroups](#)
- [servicedependencies](#)
- [serviceescalation_contactgroups](#)
- [serviceescalations](#)

- [servicegroup_members](#)
- [servicegroups](#)
- [services](#)
- [timeperiod_timeranges](#)
- [timeperiods](#)

Figure 12.18. Relationship of Configuration Tables



Commands Table

Description: .

Structure:

Field	Type	Notes	Values	Key
command_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK3
object_id	INT			UK2
command_line	VARCHAR			

Configfiles Table

Description: .

Structure:

Field	Type	Notes	Values	Key
configfile_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
configfile_type	SMALLINT			UK2
configfile_path	VARCHAR			UK3

Configfilevariables Table

Description: .

Structure:

Field	Type	Notes	Values	Key
configfilevariable_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
configfile_id	INT			
varname	VARCHAR			
varvalue	VARCHAR			

Contact_addresses Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contact_address_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
contact_id	INT			UK1
address_number	SMALLINT			UK2
address	VARCHAR			

Contact_notificationcommands Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contact_notificationcommand_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
contact_id	INT			UK1
notification_type	SMALLINT			UK2
command_object_id	INT			UK3
command_args	VARCHAR			UK4

Contactgroup_members Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contactgroup_member_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
contactgroup_id	INT			UK1
contact_object_id	INT			UK2

Contactgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contactgroup_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
contactgroup_object_id	INT			UK3
alias	VARCHAR			

Contactnotificationmethods Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contactnotificationmethod_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
contactnotification_id	INT			UK2
start_time	DATETIME			UK3
start_time_usec	INT			UK4
end_time	DATETIME			
end_time_usec	INT			
command_object_id	INT			
command_args	VARCHAR			

Contacts Table

Description: .

Structure:

Field	Type	Notes	Values	Key
contact_id	INT	Unique number identifying the record		PK

instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
contact_object_id	INT			UK3
alias	VARCHAR	String describing the contact		
email_address	VARCHAR	String containing the e-mail address of the contact		
pager_address	VARCHAR	String containing the pager address of the contact		
host_timeperiod_object_id	INT			
service_timeperiod_object_id	INT			
host_notifications_enabled	SMALLINT	Indicates whether or not the contact will receive host notifications	0 = Disabled; 1 = Enabled	
service_notifications_enabled	SMALLINT	Indicates whether or not the contact will receive service notifications	0 = Disabled; 1 = Enabled	
can_submit_commands	SMALLINT	Indicates whether or not the contact can submit external commands via the web interface	0 = Disabled; 1 = Enabled	
notify_service_recovery	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "RECOVERY"	0 = Disabled; 1 = Enabled	
notify_service_warning	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "WARNING"	0 = Disabled; 1 = Enabled	
notify_service_unknown	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "UNKNOWN"	0 = Disabled; 1 = Enabled	
notify_service_critical	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "CRITICAL"	0 = Disabled; 1 = Enabled	
notify_service_flapping	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "FLAPPING"	0 = Disabled; 1 = Enabled	

notify_service_downtime	SMALLINT	Indicates whether or not the contact will receive notifications if a service enters the state "DOWNTIME"	0 = Disabled; 1 = Enabled	
notify_host_recovery	SMALLINT	Indicates whether or not the contact will receive notifications if a host enters the state "RECOVERY"	0 = Disabled; 1 = Enabled	
notify_host_down	SMALLINT	Indicates whether or not the contact will receive notifications if a host enters the state "DOWN"	0 = Disabled; 1 = Enabled	
notify_host_unreachable	SMALLINT	Indicates whether or not the contact will receive notifications if a host enters the state "UNREACHABLE"	0 = Disabled; 1 = Enabled	
notify_host_flapping	SMALLINT	Indicates whether or not the contact will receive notifications if a host enters the state "FLAPPING"	0 = Disabled; 1 = Enabled	
notify_host_downtime	SMALLINT	Indicates whether or not the contact will receive notifications if a host enters the state "DOWNTIME"	0 = Disabled; 1 = Enabled	

customvariables Table

Description: .

Structure:

Field	Type	Notes	Values	Key
customvariable_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
object_id	INT			UK1
config_type	SMALLINT			UK2
has_been_modified	SMALLINT			
varname	VARCHAR	String containing the name of the custom variable		UK3,NK
varvalue	VARCHAR	String containing the value of the custom variable		

Host_contactgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
host_contactgroup_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
host_id	INT			UK1
contactgroup_object_id	INT			UK2

Hostescalation_contactgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
hostescalation_contact_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
hostescalation_id	INT			UK1
contactgroup_object_id	INT			UK2

Host_parenthosts Table

Description: .

Structure:

Field	Type	Notes	Values	Key
host_parenthost_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
host_id	INT			UK1
parent_host_object_id	INT			UK2

Hostdependencies Table

Description: .

Structure:

Field	Type	Notes	Values	Key
hostdependency_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
host_object_id	INT			UK3
dependent_host_object_id	INT			UK4
dependency_type	SMALLINT	Indicates the type of the dependency	1 = Notification dependency, 2 = Execution dependency	UK5
inherits_parent	SMALLINT	Indicates whether or not the host will inherit dependencies from parent hosts	0 = do not inherit dependencies, 1 = inherit dependencies	UK6
timeperiod_object_id	INT			
fail_on_up	SMALLINT	Indicates whether or not the host will be checked if the master host is UP	0 = check host, 1 = do not check host	UK7
fail_on_down	SMALLINT	Indicates whether or not the host will be checked if the master host is DOWN	0 = check host, 1 = do not check host	UK8
fail_on_unreachable	SMALLINT	Indicates whether or not the host will be checked if the master host is UNREACHABLE	0 = check host, 1 = do not check host	UK9

Hostescalations Table

Description: .

Structure:

Field	Type	Notes	Values	Key
hostescalation_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
host_object_id	INT			UK3
timeperiod_object_id	INT			UK4
first_notification	SMALLINT			UK5
last_notification	SMALLINT			UK6
notification_interval	DOUBLE			
escalate_on_recovery	SMALLINT			
escalate_on_down	SMALLINT			
escalate_on_unreachable	SMALLINT			

Hostgroup_members Table

Description: .

Structure:

Field	Type	Notes	Values	Key
hostgroup_member_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
hostgroup_id	INT			UK1
host_object_id	INT			UK2

Hostgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
hostgroup_id	INT			PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			
hostgroup_object_id	INT			UK2
alias	VARCHAR	String describing the hostgroup		

Hosts Table

Description: .

Structure:

Field	Type	Notes	Values	Key
host_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
host_object_id	INT			UK3
alias	VARCHAR	String describing the host		
display_name	VARCHAR			
address	VARCHAR			
check_command_object_id	INT			
check_command_args	VARCHAR			
eventhandler_command_object_id	INT			
eventhandler_command_args	VARCHAR			
notification_timeperiod_object_id	INT			
check_timeperiod_object_id	INT			
failure_prediction_options	VARCHAR			
check_interval	DOUBLE			
retry_interval	DOUBLE			
max_check_attempts	SMALLINT			
first_notification_delay	DOUBLE			

notification_interval	DOUBLE			
notify_on_down	SMALLINT			
notify_on_unreachable	SMALLINT			
notify_on_recovery	SMALLINT			
notify_on_flapping	SMALLINT			
notify_on_downtime	SMALLINT			
stalk_on_up	SMALLINT			
stalk_on_down	SMALLINT			
stalk_on_unreachable	SMALLINT			
flap_detection_enabled	SMALLINT			
flap_detection_on_up	SMALLINT			
flap_detection_on_down	SMALLINT			
flap_detection_on_unreachable	SMALLINT			
low_flap_threshold	DOUBLE			
high_flap_threshold	DOUBLE			
process_performance_data	SMALLINT			
freshness_checks_enabled	SMALLINT			
freshness_threshold	SMALLINT			
passive_checks_enabled	SMALLINT			
eventhandler_enabled	SMALLINT			
active_checks_enabled	SMALLINT			
retain_status_information	SMALLINT			
retain_nonstatus_information	SMALLINT			
notifications_enabled	SMALLINT			
obsess_over_host	SMALLINT			
failure_prediction_enabled	SMALLINT			
notes	VARCHAR			
notes_url	VARCHAR			
action_url	VARCHAR			
icon_image	VARCHAR			
icon_image_alt	VARCHAR			
vrmImage	VARCHAR			
statusmap_image	VARCHAR			

have_2d_ccords	SMALLINT			
x_2d	SMALLINT			
y_2d	SMALLINT			
have_3d_coords	SMALLINT			
x_3d	DOUBLE			
y_3d	DOUBLE			
z_3d	DOUBLE			

Service_contactgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
service_contactgroup_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
service_id	INT			UK2
contactgroup_object_id	INT			UK3

Servicedependencies Table

Description: .

Structure:

Field	Type	Notes	Values	Key
servicedependency_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
service_object_id	INT			UK3
dependent_service_object_id	INT			UK4
dependency_type	SMALLINT			UK5
inherits_parent	SMALLINT			UK6
timeperiod_object_id	INT			
fail_on_ok	SMALLINT			UK7
fail_on_warning	SMALLINT			UK8
fail_on_unknown	SMALLINT			UK9
fail_on_critical	SMALLINT			UK10

Serviceescalation_contactgroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
serviceescalation_contactgroup_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
serviceescalation_id	INT			UK1
contactgroup_object_id	INT			UK2

Serviceescalations Table

Description: .

Structure:

Field	Type	Notes	Values	Key
serviceescalation_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
service_object_id	INT			UK3
timeperiod_object_id	INT			UK4
first_notification	SMALLINT			UK5
last_notification	SMALLINT			UK6
notification_interval	DOUBLE			
escalate_on_recovery	SMALLINT			
escalate_on_warning	SMALLINT			
escalate_on_unknown	SMALLINT			
escalate_on_critical	SMALLINT			

Servicegroup_members Table

Description: .

Structure:

Field	Type	Notes	Values	Key
servicegroup_member_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
servicegroup_id	INT			UK1
service_object_id	INT			UK2

Servicegroups Table

Description: .

Structure:

Field	Type	Notes	Values	Key
servicegroup_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
servicegroup_object_id	INT			UK3
alias	VARCHAR	String describing the servicegroup		

Services Table

Description: .

Structure:

Field	Type	Notes	Values	Key
service_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
host_object_id	INT			
service_object_id	INT			UK3
display_name	VARCHAR			
check_command_object_id	INT			
check_command_args	VARCHAR			
eventhandler_command_object_id	INT			
eventhandler_command_args	VARCHAR			
notification_timeperiod_object_id	INT			
check_timeperiod_object_id	INT			
failure_prediction_options	VARCHAR			
check_interval	DOUBLE			
retry_interval	DOUBLE			
max_check_attempts	SMALLINT			
first_notification_delay	DOUBLE			
notification_interval	DOUBLE			

notify_on_warning	SMALLINT			
notify_on_unknown	SMALLINT			
notify_on_critical	SMALLINT			
notify_on_recovery	SMALLINT			
notify_on_flapping	SMALLINT			
notify_on_downtime	SMALLINT			
stalk_on_ok	SMALLINT			
stalk_on_warning	SMALLINT			
stalk_on_unknown	SMALLINT			
stalk_on_critical	SMALLINT			
flap_detection_enabled	SMALLINT			
flap_detection_on_ok	SMALLINT			
flap_detection_on_warning	SMALLINT			
flap_detection_on_unknown	SMALLINT			
flap_detection_on_critical	SMALLINT			
low_flap_threshold	DOUBLE			
high_flap_threshold	DOUBLE			
process_performance_data	SMALLINT			
freshness_checks_enabled	SMALLINT			
freshness_threshold	SMALLINT			
passive_checks_enabled	SMALLINT			
eventhandler_enabled	SMALLINT			
active_checks_enabled	SMALLINT			
retain_status_information	SMALLINT			
retain_nonstatus_information	SMALLINT			
notifications_enabled	SMALLINT			
obsess_over_service	SMALLINT			
failure_prediction_enabled	SMALLINT			
notes	VARCHAR			
notes_url	VARCHAR			
action_url	VARCHAR			
icon_image	VARCHAR			
icon_image_alt	VARCHAR			

Timeperiod_timeranges Table

Description: .

Structure:

Field	Type	Notes	Values	Key
timeperiod_timerange_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		
timeperiod_id	INT			UK1
day	SMALLINT			UK2
start_sec	INT			UK3
end_sec	INT			UK4

Timeperiods Table

Description: .

Structure:

Field	Type	Notes	Values	Key
timeperiod_id	INT	Unique number identifying the record		PK
instance_id	SMALLINT	Unique number identifying the distinct instance of Icinga which this entry is associated with		UK1
config_type	SMALLINT			UK2
timeperiod_object_id	INT			UK3
alias	VARCHAR	String describing the timeperiod		

[Prev](#)[Up](#)[Next](#)[Example Configurations](#)[Home](#)[Database changes/alterations](#)© 2009-2011 Icinga Development Team, <http://www.icinga.org>



Database changes/alterations

[Prev](#)

Chapter 12. IDOUtils

[Next](#)

Database changes/alterations

Change the instance name

You may have the need to change the instance name. There are some steps to be done which are described in this section. Thanks to [ralfk](#) who provided us with these instructions.

- Stop Icinga and ido2db daemons (since otherwise a new instance would automatically be added to DB instead of renaming it)

```
#> /etc/init.d/icinga stop
#> /etc/init.d/ido2db stop
```

- Change the instance name in `/usr/local/icinga/etc/idomod.cfg`

```
instance_name=newinstance
```

- Change the instance name in the database table "icinga_instances" or "instances"

MySQL/PostgreSQL

```
SQL> UPDATE icinga_instances SET instance_name='NEWNAME' WHERE instance_name='OLDNAME';
```

Oracle

```
SQL> UPDATE instances SET instance_name='NEWNAME' WHERE instance_name='OLDNAME';
```

- Change the instance name in the command pipe configuration in one of these files (site file in first place)

- `/usr/local/icinga-web/app/modules/Web/config/icinga-io.xml`
- `/usr/local/icinga-web/app/modules/Web/config/icinga-io.site.xml`

- Clear the Web Cache

```
#> /usr/local/icinga-web/bin/clearcache.sh
```

- Start Icinga and ido2db Daemons

```
#> /etc/init.d/ido2db start
#> /etc/init.d/icinga start
```

[Prev](#)

[Up](#)

[Next](#)

[IDOUtils Database Model](#)

[Home](#)

[Index](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>

**Index**[Prev](#)**Index****A**

accept_passive_host_checks=
 Passive Host Check Acceptance, [Main Configuration File Options](#)

accept_passive_service_checks=
 Passive Service Check Acceptance, [Main Configuration File Options](#)

Acknowledgements, [About Icinga](#)

action_url_target=
 Action URL Target, [CGI Configuration File Options](#)

Active Checks
 at regular intervals, [Active Checks](#)
 on demand, [Active Checks](#)

Adaptive Monitoring, [Adaptive Monitoring](#)

additional_freshness_latency=
 Additional Freshness Threshold Latency, [Main Configuration File Options](#)

Addons
 Business Process Addon, [Icinga Addons](#)
 check_mk, [Icinga Addons](#)
 IDOUtils, [Icinga Addons](#)
 LConf, [Icinga Addons](#)
 Lilac, [Icinga Addons](#)
 MKLiveStatus-Integration, [MKLiveStatus Integration](#)
 MultiSite, [Icinga Addons](#)
 NagiosQL, [Icinga Addons](#)
 NagVis, [Icinga Addons](#)
 NConf, [Icinga Addons](#)
 NRPE, [Icinga Addons](#)
 NSCA, [Icinga Addons](#)
 NSClient++, [Icinga Addons](#)
 PNP4Nagios, [Icinga Addons](#)
 Thruk, [Icinga Addons](#)

add_notif_num_hard=
 Show service state and notification number (hard), [CGI Configuration File Options](#)

add_notif_num_soft=
 Show service state and notification number (soft), [CGI Configuration File Options](#)

admin_email=

Administrator Email Address, [Main Configuration File Options](#)
admin_pager=
 Administrator Pager, [Main Configuration File Options](#)
allow_empty_hostgroup_assignment=
 Allow Empty Hostgroup Assignment, [Main Configuration File Options](#)
API/Icinga, Installation and use of the Icinga API
audio_alerts=
 Audio Alerts, [CGI Configuration File Options](#)
authorized_for_all_hosts=
 Global Host Information, [CGI Configuration File Options](#)
authorized_for_all_services=
 Global Service Information, [CGI Configuration File Options](#)
authorized_for_all_service_commands=
 Global Service Command, [CGI Configuration File Options](#)
authorized_for_configuration_information=
 Configuration Information, [CGI Configuration File Options](#)
authorized_for_host_commands=
 Global Host Command, [CGI Configuration File Options](#)
authorized_for_system_commands=
 System/Process Command, [CGI Configuration File Options](#)
authorized_for_system_information=
 System/Process Information, [CGI Configuration File Options](#)
auto_reschedule_checks=
 Auto-Rescheduling Option, [Main Configuration File Options](#)
auto_rescheduling_interval=
 Auto-Rescheduling, [Main Configuration File Options](#)
auto_rescheduling_windows=
 Auto-Rescheduling Window, [Main Configuration File Options](#)

B

Beginners, Advice for, [Advice for beginners](#)
broker_module=
 Event Broker Modules, [Main Configuration File Options](#)
 Business Process Addon, [Icinga Addons](#)

C

Cached Checks, [Cached Checks](#)
cached_host_check_horizon=
 Cached Host Check Horizon, [Main Configuration File Options](#)
cached_service_check_horizon=
 Cached Service Check, [Main Configuration File Options](#)
cfg_dir=
 Object Configuration Directory, [Main Configuration File Options](#)
cfg_file=
 Object Configuration File, [Main Configuration File Options](#)
CGI Authentication
 Authenticated Contact, [Authentication And Authorization In The CGIs](#)
 Authenticated User, [Authentication And Authorization In The CGIs](#)
 Authentication On Secured Web Servers, [Authentication And Authorization In The CGIs](#)
 Default Permissions To CGI Information, [Authentication And Authorization In The CGIs](#)
 Enabling Authentication/Authorization Functionality In The CGIs, [Authentication And Authorization In The CGIs](#)

[Granting Additional Permissions To CGI Information, Authentication And Authorization In The CGIs](#)

[Setting Up Authenticated Users, Authentication And Authorization In The CGIs](#)

[CGI Authorization, Authentication And Authorization In The CGIs](#)

[CGI Configuration File Options, CGI Configuration File Options](#)

CGI parameter

ahas, [Information On CGI parameters](#)

alerttypes, [Information On CGI parameters](#)

archive, [Information On CGI parameters](#)

assumeinitialstates, [Information On CGI parameters](#)

assumestateretention, [Information On CGI parameters](#)

assumestatesduringnotrunning, [Information On CGI parameters](#)

backtrack, [Information On CGI parameters](#)

breakdown, [Information On CGI parameters](#)

broadcast_notification, [Information On CGI parameters](#)

childoptions, [Information On CGI parameters](#)

cmd_mod, [Information On CGI parameters](#)

cmd_typ, [Information On CGI parameters](#)

columns, [Information On CGI parameters](#)

com_author, [Information On CGI parameters](#)

com_data, [Information On CGI parameters](#)

com_id, [Information On CGI parameters](#)

contact, [Information On CGI parameters](#)

createimage, [Information On CGI parameters](#)

csvoutput, [Information On CGI parameters](#)

displaytype, [Information On CGI parameters](#)

down_id, [Information On CGI parameters](#)

eday, [Information On CGI parameters](#)

ehour, [Information On CGI parameters](#)

embedded, [Information On CGI parameters](#)

emin, [Information On CGI parameters](#)

emon, [Information On CGI parameters](#)

end_time, [Information On CGI parameters](#)

esec, [Information On CGI parameters](#)

eyear, [Information On CGI parameters](#)

fixed, [Information On CGI parameters](#)

force_check, [Information On CGI parameters](#)

force_notification, [Information On CGI parameters](#)

full_log_entries, [Information On CGI parameters](#)

get_date_parts, [Information On CGI parameters](#)

graphevents, [Information On CGI parameters](#)

graphstatetypes, [Information On CGI parameters](#)

host, [Information On CGI parameters](#)

hostgroup, [Information On CGI parameters](#)

hostprops, [Information On CGI parameters](#)

hoststates, [Information On CGI parameters](#)

hoststatustypes, [Information On CGI parameters](#)

hours, [Information On CGI parameters](#)

includesoftstates, [Information On CGI parameters](#)

initialassumedhoststate, [Information On CGI parameters](#)

initialassumedservicestate, [Information On CGI parameters](#)

initialstateslogged, [Information On CGI parameters](#)

input, [Information On CGI parameters](#)

jsonoutput, [Information On CGI parameters](#)
 limit, [Information On CGI parameters](#)
 minutes, [Information On CGI parameters](#)
 navbarsearch, [Information On CGI parameters](#)
 newstatesonly, [Information On CGI parameters](#)
 nodowntime, [Information On CGI parameters](#)
 noflapping, [Information On CGI parameters](#)
 nofrills, [Information On CGI parameters](#)
 noheader, [Information On CGI parameters](#)
 nosystem, [Information On CGI parameters](#)
 notimebreaks, [Information On CGI parameters](#)
 not_dly, [Information On CGI parameters](#)
 oldestfirst, [Information On CGI parameters](#)
 performance_data, [Information On CGI parameters](#)
 persistent, [Information On CGI parameters](#)
 plugin_output, [Information On CGI parameters](#)
 plugin_state, [Information On CGI parameters](#)
 ptc, [Information On CGI parameters](#)
 report, [Information On CGI parameters](#)
 report_type, [Information On CGI parameters](#)
 rpttimeperiod, [Information On CGI parameters](#)
 sched_dly, [Information On CGI parameters](#)
 sday, [Information On CGI parameters](#)
 send_notification, [Information On CGI parameters](#)
 service, [Information On CGI parameters](#)
 servicefilter, [Information On CGI parameters](#)
 servicegroup, [Information On CGI parameters](#)
 serviceprops, [Information On CGI parameters](#)
 servicestates,
 servicestatustypes, [Information On CGI parameters](#)
 service_divisor, [Information On CGI parameters](#)
 shour, [Information On CGI parameters](#)
 showscheduleddowntime, [Information On CGI parameters](#)
 show_log_entries, [Information On CGI parameters](#)
 smin, [Information On CGI parameters](#)
 smon, [Information On CGI parameters](#)
 sortoption, [Information On CGI parameters](#)
 sorttype, [Information On CGI parameters](#)
 ssec, [Information On CGI parameters](#)
 standardreport, [Information On CGI parameters](#)
 start_time, [Information On CGI parameters](#)
 statetype, [Information On CGI parameters](#)
 statetypes, [Information On CGI parameters](#)
 sticky_ack, [Information On CGI parameters](#)
 style, [Information On CGI parameters](#)
 syear, [Information On CGI parameters](#)
 t1, [Information On CGI parameters](#)
 t2, [Information On CGI parameters](#)
 timeperiod, [Information On CGI parameters](#)
 trigger, [Information On CGI parameters](#)
 type, [Information On CGI parameters](#)

CGI security

Additional Techniques, Enhanced CGI Security and Authentication

Enhanced CGI Security and Authentication, [Enhanced CGI Security and Authentication](#)
 Implementing Digest Authentication, [Enhanced CGI Security and Authentication](#)
 Implementing Forced TLS/SSL, [Enhanced CGI Security and Authentication](#)
 Implementing IP subnet lockdown, [Enhanced CGI Security and Authentication](#)

CGIs

- Alert Histogram CGI, [Icinga Classic UI: Information On The CGIs](#)
- Alert History CGI, [Icinga Classic UI: Information On The CGIs](#)
- Alert Summary CGI, [Icinga Classic UI: Information On The CGIs](#)
- Availability Reporting CGI, [Icinga Classic UI: Information On The CGIs](#)
- Changes in the Classic UI, [Icinga Classic UI: Information On The CGIs](#)
- Command CGI, [Icinga Classic UI: Information On The CGIs](#)
- Configuration CGI, [Icinga Classic UI: Information On The CGIs](#)
- Event log CGI, [Icinga Classic UI: Information On The CGIs](#)
- Executing CGIs on the command line, [Executing CGIs on the command line](#)
- Extended Information CGI, [Icinga Classic UI: Information On The CGIs](#)
- Filter properties, [Information On CGI parameters](#)
- Network Outages CGI, [Icinga Classic UI: Information On The CGIs](#)
- Notification CGI, [Icinga Classic UI: Information On The CGIs](#)
- Parameters, [Information On CGI parameters](#)
- Status CGI, [Icinga Classic UI: Information On The CGIs](#)
- Status Map CGI, [Icinga Classic UI: Information On The CGIs](#)
- Tactical Overview CGI, [Icinga Classic UI: Information On The CGIs](#)
- Trends CGI, [Icinga Classic UI: Information On The CGIs](#)
- WAP Interface CGI, [Icinga Classic UI: Information On The CGIs](#)

cgi_log_archive_path=

- Set location of CGI log archive directory, [CGI Configuration File Options](#)

cgi_log_file=

- Set location of CGI log file, [CGI Configuration File Options](#)

cgi_log_rotation_method=

- Set rotation method of CGI log file, [CGI Configuration File Options](#)

check_external_commands=

- External Command Check, [Main Configuration File Options](#)

check_for_orphaned_hosts=

- Orphaned Host Check Option, [Main Configuration File Options](#)

check_for_orphaned_services=

- Orphaned Service Check, [Main Configuration File Options](#)

check_host_freshness=

- Host Freshness Checking Option, [Main Configuration File Options](#)

check_mk, [Icinga Addons](#)

check_result_path=

- Check Result Path, [Main Configuration File Options](#)

check_result_reaper_frequency=

- Check Result Reaper, [Main Configuration File Options](#)

check_service_freshness=

- Service Freshness Checking, [Main Configuration File Options](#)

child_processes_fork_twice=

- Child Processes Fork, [Main Configuration File Options](#)

Cluster

- Monitoring Service and Host Clusters, [Monitoring Service and Host Clusters](#)

Command Definition, [Command Definition](#)

command_check_interval=

- External Command Check, [Main Configuration File Options](#)

command_file=

External Command File, [Main Configuration File Options](#)
 Compatibility, [About Icinga](#)
 Configuration
 check-config, [Verifying Your Configuration](#)
 show-errors, [Verifying Your Configuration](#)
 Verifying Your Configuration, [Verifying Your Configuration](#)
 Configuration Overview, [Configuration Overview](#)
 Contact Definition, [Contact Definition](#)
 Contactgroup Definition, [Contactgroup Definition](#)
 Custom CGI Headers and Footers, [Custom CGI Headers and Footers](#)
 Custom Object Variables, [Custom Object Variables](#)

D

date_format=
 Date Format, [Main Configuration File Options](#)
 debug_file=
 Debug File, [Main Configuration File Options](#)
 debug_level=
 Debug Level, [Main Configuration File Options](#)
 debug_verbosity=
 Debug Verbosity, [Main Configuration File Options](#)
 default_statusmap_layout=
 Default Statusmap Layout, [CGI Configuration File Options](#)
 default_user_name=
 Default User Name, [CGI Configuration File Options](#)
 Dependencies
 hard dependencies, [Host and Service Dependencies](#)
 Host and Service Dependencies, [Host and Service Dependencies](#)
 Predictive Dependency Checks, [Predictive Dependency Checks](#)
 Same Host Dependencies, [Time-Saving Tricks For Object Definitions](#)
 Same Host Dependencies With Servicegroups, [Time-Saving Tricks For Object Definitions](#)
 Distributed Monitoring, [Distributed Monitoring](#)
 Downloading The Latest Version, [About Icinga](#)
 Downtime
 Scheduled Downtime, [Scheduled Downtime](#)

E

Embedded Perl
 Developing Plugins For Use With Embedded Perl, [Developing Plugins For Use With Embedded Perl](#)
 Embedded Perl Interpreter
 Using ..., [Using The Embedded Perl Interpreter](#)
 enabled_embedded_perl=
 Embedded Perl Interpreter, [Main Configuration File Options](#)
 enable_environment_macros=
 Environment Macros Option, [Main Configuration File Options](#)
 enable_event_handlers=
 Event Handler Option, [Main Configuration File Options](#)
 enable_flap_detection=
 Flap Detection Option, [Main Configuration File Options](#)
 enable_notifications=

Notifications Option, [Main Configuration File Options](#)
 enable_predictive_host_dependency_checks=
 Predictive Host Dependency, [Main Configuration File Options](#)
 enable_predictive_service_dependency_checks=
 Predictive Service, [Main Configuration File Options](#)
 enable_splunk_integration=
 Splunk Integration Option, [CGI Configuration File Options](#)
 enforce_comments_on_actions=
 Enforce comments on actions, [CGI Configuration File Options](#)
 Escalations
 Escalation condition, [Escalation Condition](#)
 Notification Escalations, [Notification Escalations](#)
 escape_html_tags=
 Escape HTML Tags Option, [CGI Configuration File Options](#)
 Event Handlers, [Event Handlers](#)
 Example, [Event Handlers](#)
 event_broker_options=
 Event Broker Options, [Main Configuration File Options](#)
 event_handler_timeout=
 Event Handler Timeout, [Main Configuration File Options](#)
 execute_host_checks=
 Host Check Execution Option, [Main Configuration File Options](#)
 execute_service_checks=
 Service Check Execution, [Main Configuration File Options](#)
 Extended service information definition, [Serviceextinfo Definition](#)
 External Commands, [Adaptive Monitoring](#)
 External commands
 ACKNOWLEDGE_HOST_PROBLEM, [List of External Commands](#)
 ACKNOWLEDGE_SVC_PROBLEM, [List of External Commands](#)
 ADD_HOST_COMMENT, [List of External Commands](#)
 ADD_SVC_COMMENT, [List of External Commands](#)
 CHANGE_CONTACT_HOST_NOTIFICATION_TIMEPERIOD, [List of External Commands](#)
 CHANGE_CONTACT_MODATTR, [List of External Commands](#)
 CHANGE_CONTACT_MODHATTR, [List of External Commands](#)
 CHANGE_CONTACT_MODSATTR, [List of External Commands](#)
 CHANGE_CONTACT_SVC_NOTIFICATION_TIMEPERIOD, [List of External Commands](#)
 CHANGE_CUSTOM_CONTACT_VAR, [List of External Commands](#)
 CHANGE_CUSTOM_HOST_VAR, [List of External Commands](#)
 CHANGE_CUSTOM_SVC_VAR, [List of External Commands](#)
 CHANGE_GLOBAL_HOST_EVENT_HANDLER, [List of External Commands](#)
 CHANGE_GLOBAL_SVC_EVENT_HANDLER, [List of External Commands](#)
 CHANGE_HOST_CHECK_COMMAND, [List of External Commands](#)
 CHANGE_HOST_CHECK_TIMEPERIOD, [List of External Commands](#)
 CHANGE_HOST_EVENT_HANDLER, [List of External Commands](#)
 CHANGE_HOST_MODATTR, [List of External Commands](#)
 CHANGE_HOST_NOTIFICATION_TIMEPERIOD, [List of External Commands](#)
 CHANGE_MAX_HOST_CHECK_ATTEMPTS, [List of External Commands](#)
 CHANGE_MAX_SVC_CHECK_ATTEMPTS, [List of External Commands](#)
 CHANGE_NORMAL_HOST_CHECK_INTERVAL, [List of External Commands](#)
 CHANGE_NORMAL_SVC_CHECK_INTERVAL, [List of External Commands](#)
 CHANGE_RETRY_HOST_CHECK_INTERVAL, [List of External Commands](#)
 CHANGE_RETRY_SVC_CHECK_INTERVAL, [List of External Commands](#)

CHANGE_SVC_CHECK_COMMAND, [List of External Commands](#)
 CHANGE_SVC_CHECK_TIMEPERIOD, [List of External Commands](#)
 CHANGE_SVC_EVENT_HANDLER, [List of External Commands](#)
 CHANGE_SVC_MODATTR, [List of External Commands](#)
 CHANGE_SVC_NOTIFICATION_TIMEPERIOD, [List of External Commands](#)
 DELAY_HOST_NOTIFICATION, [List of External Commands](#)
 DELAY_SVC_NOTIFICATION, [List of External Commands](#)
 DEL_ALL_HOST_COMMENTS, [List of External Commands](#)
 DEL_ALL_SVC_COMMENTS, [List of External Commands](#)
 DEL_DOWNTIME_BY_HOSTGROUP_NAME, [List of External Commands](#)
 DEL_DOWNTIME_BY_HOST_NAME, [List of External Commands](#)
 DEL_DOWNTIME_BY_START_TIME_COMMENT, [List of External Commands](#)
 DEL_HOST_COMMENT, [List of External Commands](#)
 DEL_HOST_DOWNTIME, [List of External Commands](#)
 DEL_SVC_COMMENT, [List of External Commands](#)
 DEL_SVC_DOWNTIME, [List of External Commands](#)
 DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST, [List of External Commands](#)
 DISABLE_CONTACTGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_CONTACTGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_CONTACT_HOST_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_CONTACT_SVC_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_EVENT_HANDLERS, [List of External Commands](#)
 DISABLE_FAILURE_PREDICTION, [List of External Commands](#)
 DISABLE_FLAP_DETECTION, [List of External Commands](#)
 DISABLE_HOSTGROUP_HOST_CHECKS, [List of External Commands](#)
 DISABLE_HOSTGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_HOSTGROUP_PASSIVE_HOST_CHECKS, [List of External Commands](#)
 DISABLE_HOSTGROUP_PASSIVE_SVC_CHECKS, [List of External Commands](#)
 DISABLE_HOSTGROUP_SVC_CHECKS, [List of External Commands](#)
 DISABLE_HOSTGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_HOST_AND_CHILD_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_HOST_CHECK, [List of External Commands](#)
 DISABLE_HOST_EVENT_HANDLER, [List of External Commands](#)
 DISABLE_HOST_FLAP_DETECTION, [List of External Commands](#)
 DISABLE_HOST_FRESHNESS_CHECKS, [List of External Commands](#)
 DISABLE_HOST_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_HOST_SVC_CHECKS, [List of External Commands](#)
 DISABLE_HOST_SVC_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_PASSIVE_HOST_CHECKS, [List of External Commands](#)
 DISABLE_PASSIVE_SVC_CHECKS, [List of External Commands](#)
 DISABLE_PERFORMANCE_DATA, [List of External Commands](#)
 DISABLE_SERVICEGROUP_HOST_CHECKS, [List of External Commands](#)
 DISABLE_SERVICEGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS, [List of External Commands](#)
 DISABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS, [List of External Commands](#)
 DISABLE_SERVICEGROUP_SVC_CHECKS, [List of External Commands](#)
 DISABLE_SERVICEGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
 DISABLE_SERVICE_FLAP_DETECTION, [List of External Commands](#)
 DISABLE_SERVICE_FRESHNESS_CHECKS, [List of External Commands](#)
 DISABLE_SVC_CHECK, [List of External Commands](#)
 DISABLE_SVC_EVENT_HANDLER, [List of External Commands](#)
 DISABLE_SVC_FLAP_DETECTION, [List of External Commands](#)

DISABLE_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST, [List of External Commands](#)
ENABLE_CONTACTGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
ENABLE_CONTACTGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_CONTACT_HOST_NOTIFICATIONS, [List of External Commands](#)
ENABLE_CONTACT_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_EVENT_HANDLERS, [List of External Commands](#)
ENABLE_FAILURE_PREDICTION, [List of External Commands](#)
ENABLE_FLAP_DETECTION, [List of External Commands](#)
ENABLE_HOSTGROUP_HOST_CHECKS, [List of External Commands](#)
ENABLE_HOSTGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
ENABLE_HOSTGROUP_PASSIVE_HOST_CHECKS, [List of External Commands](#)
ENABLE_HOSTGROUP_PASSIVE_SVC_CHECKS, [List of External Commands](#)
ENABLE_HOSTGROUP_SVC_CHECKS, [List of External Commands](#)
ENABLE_HOSTGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_HOST_AND_CHILD_NOTIFICATIONS, [List of External Commands](#)
ENABLE_HOST_CHECK, [List of External Commands](#)
ENABLE_HOST_EVENT_HANDLER, [List of External Commands](#)
ENABLE_HOST_FLAP_DETECTION, [List of External Commands](#)
ENABLE_HOST_FRESHNESS_CHECKS, [List of External Commands](#)
ENABLE_HOST_NOTIFICATIONS, [List of External Commands](#)
ENABLE_HOST_SVC_CHECKS, [List of External Commands](#)
ENABLE_HOST_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_NOTIFICATIONS, [List of External Commands](#)
ENABLE_PASSIVE_HOST_CHECKS, [List of External Commands](#)
ENABLE_PASSIVE_SVC_CHECKS, [List of External Commands](#)
ENABLE_PERFORMANCE_DATA, [List of External Commands](#)
ENABLE_SERVICEGROUP_HOST_CHECKS, [List of External Commands](#)
ENABLE_SERVICEGROUP_HOST_NOTIFICATIONS, [List of External Commands](#)
ENABLE_SERVICEGROUP_PASSIVE_HOST_CHECKS, [List of External Commands](#)
ENABLE_SERVICEGROUP_PASSIVE_SVC_CHECKS, [List of External Commands](#)
ENABLE_SERVICEGROUP_SVC_CHECKS, [List of External Commands](#)
ENABLE_SERVICEGROUP_SVC_NOTIFICATIONS, [List of External Commands](#)
ENABLE_SERVICE_FRESHNESS_CHECKS, [List of External Commands](#)
ENABLE_SVC_CHECK, [List of External Commands](#)
ENABLE_SVC_EVENT_HANDLER, [List of External Commands](#)
ENABLE_SVC_FLAP_DETECTION, [List of External Commands](#)
ENABLE_SVC_NOTIFICATIONS, [List of External Commands](#)
PROCESS_FILE, [List of External Commands](#)
PROCESS_HOST_CHECK_RESULT, [List of External Commands](#)
PROCESS_SERVICE_CHECK_RESULT, [List of External Commands](#)
READ_STATE_INFORMATION, [List of External Commands](#)
REMOVE_HOST_ACKNOWLEDGEMENT, [List of External Commands](#)
REMOVE_SVC_ACKNOWLEDGEMENT, [List of External Commands](#)
RESTART_PROGRAM, [List of External Commands](#)
SAVE_STATE_INFORMATION, [List of External Commands](#)
SCHEDULE_AND_PROPAGATE_HOST_DOWNTIME, [List of External Commands](#)
SCHEDULE_AND_PROPAGATE_TRIGGERED_HOST_DOWNTIME, [List of External Commands](#)
SCHEDULE_FORCED_HOST_CHECK, [List of External Commands](#)
SCHEDULE_FORCED_HOST_SVC_CHECKS, [List of External Commands](#)
SCHEDULE_FORCED_SVC_CHECK, [List of External Commands](#)
SCHEDULE_HOSTGROUP_HOST_DOWNTIME, [List of External Commands](#)

SCHEDULE_HOSTGROUP_SVC_DOWNTIME, [List of External Commands](#)
 SCHEDULE_HOST_CHECK, [List of External Commands](#)
 SCHEDULE_HOST_DOWNTIME, [List of External Commands](#)
 SCHEDULE_HOST_SVC_CHECKS, [List of External Commands](#)
 SCHEDULE_HOST_SVC_DOWNTIME, [List of External Commands](#)
 SCHEDULE_SERVICEGROUP_HOST_DOWNTIME, [List of External Commands](#)
 SCHEDULE_SERVICEGROUP_SVC_DOWNTIME, [List of External Commands](#)
 SCHEDULE_SVC_CHECK, [List of External Commands](#)
 SCHEDULE_SVC_DOWNTIME, [List of External Commands](#)
 SEND_CUSTOM_HOST_NOTIFICATION, [List of External Commands](#)
 SEND_CUSTOM_SVC_NOTIFICATION, [List of External Commands](#)
 SET_HOST_NOTIFICATION_NUMBER, [List of External Commands](#)
 SET_SVC_NOTIFICATION_NUMBER, [List of External Commands](#)
 SHUTDOWN_PROGRAM, [List of External Commands](#)
 START_ACCEPTING_PASSIVE_HOST_CHECKS, [List of External Commands](#)
 START_ACCEPTING_PASSIVE_SVC_CHECKS, [List of External Commands](#)
 START_EXECUTING_HOST_CHECKS, [List of External Commands](#)
 START_EXECUTING_SVC_CHECKS, [List of External Commands](#)
 START_OBSESSING_OVER_HOST, [List of External Commands](#)
 START_OBSESSING_OVER_HOST_CHECKS, [List of External Commands](#)
 START_OBSESSING_OVER_SVC, [List of External Commands](#)
 START_OBSESSING_OVER_SVC_CHECKS, [List of External Commands](#)
 STOP_ACCEPTING_PASSIVE_HOST_CHECKS, [List of External Commands](#)
 STOP_ACCEPTING_PASSIVE_SVC_CHECKS, [List of External Commands](#)
 STOP_EXECUTING_HOST_CHECKS, [List of External Commands](#)
 STOP_EXECUTING_SVC_CHECKS, [List of External Commands](#)
 STOP_OBSESSING_OVER_HOST, [List of External Commands](#)
 STOP_OBSESSING_OVER_HOST_CHECKS, [List of External Commands](#)
 STOP_OBSESSING_OVER_SVC, [List of External Commands](#)
 STOP_OBSESSING_OVER_SVC_CHECKS, [List of External Commands](#)
 external_command_buffer_slots=
 External Command Buffer, [Main Configuration File Options](#)

F

Failover
 Redundant and Failover Network Monitoring, [Redundant and Failover Network Monitoring](#)
 Fast Startup Options, [Fast Startup Options](#)
 first_day_of_week=
 Set first day of week, [CGI Configuration File Options](#)
 Flapping
 Detection and Handling of State Flapping, [Detection and Handling of State Flapping](#)
 free_child_process_memory=
 Child Process Memory, [Main Configuration File Options](#)
 Freshness
 Service and Host Freshness Checks, [Service and Host Freshness Checks](#)

G

global_host_event_handler=
 Global Host Event Handler, [Main Configuration File Options](#)
 global_service_event_handler=

Global Service Event Handler, [Main Configuration File Options](#)
 Gnokii, [Notifications](#)
 Graphing Performance Info With PNP4Nagios, [Performance Data](#)

H

high_host_flap_threshold=
 High Host Flap Threshold, [Main Configuration File Options](#)
 high_service_flap_threshold=
 High Service Flap, [Main Configuration File Options](#)
 Host Checks, [Host Checks](#)
 Host Definition, [Host definition](#)
 Host Dependency Definitions, [Time-Saving Tricks For Object Definitions](#)
 Host Escalation Definitions, [Time-Saving Tricks For Object Definitions](#)
 Hostdependency Definition, [Hostdependency Definition](#)
 Hostescalation Definition, [Hostescalation Definition](#)
 Hostextinfo Definition, [Hostextinfo Definition](#)
 Hostgroup Definition, [Hostgroup Definition](#)
 Hostgroups, [Time-Saving Tricks For Object Definitions](#)
 host_check_timeout=
 Host Check Timeout, [Main Configuration File Options](#)
 host_freshness_check_interval=
 Host Freshness Check, [Main Configuration File Options](#)
 host_inter_check_delay_method=
 Host Inter-Check Delay, [Main Configuration File Options](#)
 host_perfdata_command=
 Host Performance Data Processing, [Main Configuration File Options](#)
 host_perfdata_file=
 Host Performance Data File, [Main Configuration File Options](#)
 host_perfdata_file_mode=
 Host Performance Data File, [Main Configuration File Options](#)
 host_perfdata_file_processing_command=
 Host Performance Data File, [Main Configuration File Options](#)
 host_perfdata_file_processing_interval=
 Host Performance Data File, [Main Configuration File Options](#)
 host_perfdata_file_template=
 Host Performance Data File, [Main Configuration File Options](#)
 host_perfdata_process_empty_results
 Process Empty Host Performance Data Results, [Main Configuration File Options](#)
 How Do I Use Plugin X?, [Icinga Plugins](#)
 Howtos
 User Howtos, [Links to other published Howtos](#)
 http_charset=, [CGI Configuration File Options](#)

I

Icinga command line options
 option -d (daemon mode), [Starting and Stopping Icinga](#)
 option -p (precache objects), [Fast Startup Options](#)
 option -S (timing and scheduling info including scheduling queue), [Fast Startup Options](#)
 option -s (timing and scheduling info), [Fast Startup Options](#)
 option -u (use precached objects), [Fast Startup Options](#)
 option -v (verify configuration), [Verifying Your Configuration](#)

option -x (don't check for circular paths), [Fast Startup Options](#)
Icinga logging options, [Starting and Stopping Icinga](#)
[Icinga-API, Installation and use of the Icinga API](#)
Icinga-Web REST API; [The Icinga-Web REST API](#)
Icinga-Reporting with JasperServer
[Icinga-Reporting, Installation of the Icinga-Reporting with JasperServer](#)
Icinga-Web
Authentication, [Configuration Overview of Icinga-Web](#)
Change default timezone, [Configuration Overview of Icinga-Web](#)
Configuration Options, [Configuration Overview of Icinga-Web](#)
Customised Configuration, [Configuration Overview of Icinga-Web](#)
Icinga-API connection, [Configuration Overview of Icinga-Web](#)
Module configuration, [Configuration Overview of Icinga-Web](#)
Overview (<= 1.2.x), [Introduction to Icinga-Web \(up to 1.2.x\)](#)
Overview (>= 1.3), [Introduction to Icinga-Web \(>= 1.3.x\)](#)
Session Cookie Lifetime, [Configuration Overview of Icinga-Web](#)
Update/upgrade Icinga-Web database, [Upgrading Icinga-Web and Icinga-Web Database](#)
Update/upgrade of Icinga-Web, [Upgrading Icinga-Web and Icinga-Web Database](#)
Icingastats
Using The Icingastats Utility, [Using The Icingastats Utility](#)
icinga_group=
Icinga Group, [Main Configuration File Options](#)
icinga_user=
Icinga User, [Main Configuration File Options](#)
IDOUtils
Central Tables, [Central Tables](#)
Change instance name, [Database changes/alterations](#)
Configuration Tables, [Configuration Tables](#)
Current Status Tables, [Current Status Tables](#)
Debugging Tables, [Debugging Tables](#)
Historical Tables, [Historical Tables](#)
illegal_macro_output_chars=
Illegal Macro Output, [Main Configuration File Options](#)
illegal_object_name_chars=
Illegal Object Name, [Main Configuration File Options](#)
Installation of the Webinterface, [Installation of the Icinga-Web Frontend](#)
Integration Overview, [Integration Overview](#)
interval_length=
Timing Interval Length, [Main Configuration File Options](#)

L

Large Installation Tweaks, [Large Installation Tweaks](#)
LConf, [Icinga Addons](#)
Licensing, [About Icinga](#)
Lilac, [Icinga Addons](#)
lock_author=
Lock Author Names, [CGI Configuration File Options](#)
lock_file=
Lock File, [Main Configuration File Options](#)
log_archive_path=
Log Archive Path, [Main Configuration File Options](#)
log_event_handlers=

Event Handler Logging Option, [Main Configuration File Options](#)
log_external_commands=
 External Command Logging, [Main Configuration File Options](#)
log_file=
 Log File, [Main Configuration File Options](#)
log_host_retries=
 Host Check Retry Logging Option, [Main Configuration File Options](#)
log_initial_states=
 Initial States Logging Option, [Main Configuration File Options](#)
log_notifications=
 Notification Logging Option, [Main Configuration File Options](#)
log_passive_checks=
 Passive Check Logging Option, [Main Configuration File Options](#)
log_rotation_method=
 Log Rotation Method, [Main Configuration File Options](#)
log_service_retries=
 Service Check Retry Logging, [Main Configuration File Options](#)
low_host_flap_threshold=
 Low Host Flap Threshold, [Main Configuration File Options](#)
low_service_flap_threshold=
 Low Service Flap, [Main Configuration File Options](#)

M

Macros

[\\$ADMINEMAIL\\$, Standard Macros in Icinga](#)
[\\$ADMINPAGER\\$, Standard Macros in Icinga](#)
[\\$ARGn\\$, Standard Macros in Icinga](#)
[\\$COMMANDFILE\\$, Standard Macros in Icinga](#)
[\\$COMMENTDATAFILE\\$, Standard Macros in Icinga](#)
[\\$CONTACTADDRESSn\\$, Standard Macros in Icinga](#)
[\\$CONTACTALIAS\\$, Standard Macros in Icinga](#)
[\\$CONTACTEMAIL\\$, Standard Macros in Icinga](#)
[\\$CONTACTGROUPALIAS\\$, Standard Macros in Icinga](#)
[\\$CONTACTGROUPMEMBERS\\$, Standard Macros in Icinga](#)
[\\$CONTACTGROUPNAME\\$, Standard Macros in Icinga](#)
[\\$CONTACTGROUPNAMES\\$, Standard Macros in Icinga](#)
[\\$CONTACTNAME\\$, Standard Macros in Icinga](#)
[\\$CONTACTPAGER\\$, Standard Macros in Icinga](#)
[\\$DATE\\$, Standard Macros in Icinga](#)
[\\$DOWNTIMEDATAFILE\\$, Standard Macros in Icinga](#)
[\\$EVENTSTARTTIME\\$, Standard Macros in Icinga](#)
[\\$HOSTACKAUTHOR\\$, Standard Macros in Icinga](#)
[\\$HOSTACKAUTHORALIAS\\$, Standard Macros in Icinga](#)
[\\$HOSTACKAUTHORNAME\\$, Standard Macros in Icinga](#)
[\\$HOSTACKCOMMENT\\$, Standard Macros in Icinga](#)
[\\$HOSTACTIONURL\\$, Standard Macros in Icinga](#)
[\\$HOSTADDRESS\\$, Standard Macros in Icinga](#)
[\\$HOSTADDRESS6\\$, Standard Macros in Icinga](#)
[\\$HOSTALIAS\\$, Standard Macros in Icinga](#)
[\\$HOSTATTEMPT\\$, Standard Macros in Icinga](#)
[\\$HOSTCHECKCOMMAND\\$, Standard Macros in Icinga](#)
[\\$HOSTDISPLAYNAME\\$, Standard Macros in Icinga](#)

\$HOSTDOWNTIME\$, Standard Macros in Icinga
 \$HOSTDURATION\$, Standard Macros in Icinga
 \$HOSTDURATIONSEC\$, Standard Macros in Icinga
 \$HOSTEVENTID\$, Standard Macros in Icinga
 \$HOSTEXECUTIONTIME\$, Standard Macros in Icinga
 \$HOSTGROUPACTIONURL\$, Standard Macros in Icinga
 \$HOSTGROUPALIAS\$, Standard Macros in Icinga
 \$HOSTGROUPMEMBERS\$, Standard Macros in Icinga
 \$HOSTGROUPNAME\$, Standard Macros in Icinga
 \$HOSTGROUPNAMES\$, Standard Macros in Icinga
 \$HOSTGROUPNOTES\$, Standard Macros in Icinga
 \$HOSTGROUPNOTESURL\$, Standard Macros in Icinga
 \$HOSTLATENCY\$, Standard Macros in Icinga
 \$HOSTNAME\$, Standard Macros in Icinga
 \$HOSTNOTESS\$, Standard Macros in Icinga
 \$HOSTNOTESURL\$, Standard Macros in Icinga
 \$HOSTNOTIFICATIONID\$, Standard Macros in Icinga
 \$HOSTNOTIFICATIONNUMBER\$, Standard Macros in Icinga
 \$HOSTOUTPUT\$, Standard Macros in Icinga
 \$HOSTPERCENTCHANGE\$, Standard Macros in Icinga
 \$HOSTPERFDATA\$, Standard Macros in Icinga
 \$HOSTPERFDATAFILE\$, Standard Macros in Icinga
 \$HOSTPROBLEMD\$, Standard Macros in Icinga
 \$HOSTSTATE\$, Standard Macros in Icinga
 \$HOSTSTATEID\$, Standard Macros in Icinga
 \$HOSTSTATETYPE\$, Standard Macros in Icinga
 \$LASTHOSTCHECK\$, Standard Macros in Icinga
 \$LASTHOSTDOWN\$, Standard Macros in Icinga
 \$LASTHOSTEVENTID\$, Standard Macros in Icinga
 \$LASTHOSTPROBLEMD\$, Standard Macros in Icinga
 \$LASTHOSTSTATE\$, Standard Macros in Icinga
 \$LASTHOSTSTATECHANGE\$, Standard Macros in Icinga
 \$LASTHOSTSTATEID\$, Standard Macros in Icinga
 \$LASTHOSTUNREACHABLE\$, Standard Macros in Icinga
 \$LASTHOSTUP\$, Standard Macros in Icinga
 \$LASTSERVICECHECK\$, Standard Macros in Icinga
 \$LASTSERVICECRITICAL\$, Standard Macros in Icinga
 \$LASTSERVICEEVENTID\$, Standard Macros in Icinga
 \$LASTSERVICEOK\$, Standard Macros in Icinga
 \$LASTSERVICEPROBLEMD\$, Standard Macros in Icinga
 \$LASTSERVICESTATE\$, Standard Macros in Icinga
 \$LASTSERVICESTATECHANGE\$, Standard Macros in Icinga
 \$LASTSERVICESTATEID\$, Standard Macros in Icinga
 \$LASTSERVICEUNKNOWN\$, Standard Macros in Icinga
 \$LASTSERVICEWARNING\$, Standard Macros in Icinga
 \$LOGFILE\$, Standard Macros in Icinga
 \$LONGDATETIME\$, Standard Macros in Icinga
 \$LONGHOSTOUTPUT\$, Standard Macros in Icinga
 \$LONGSERVICEOUTPUT\$, Standard Macros in Icinga
 \$MAINCONFIGFILE\$, Standard Macros in Icinga
 \$MAXHOSTATTEMPTS\$, Standard Macros in Icinga
 \$MAXSERVICEATTEMPTS\$, Standard Macros in Icinga
 \$NOTIFICATIONAUTHOR\$, Standard Macros in Icinga

\$NOTIFICATIONAUTHORALIAS\$, Standard Macros in Icinga
 \$NOTIFICATIONAUTHORNAME\$, Standard Macros in Icinga
 \$NOTIFICATIONCOMMENT\$, Standard Macros in Icinga
 \$NOTIFICATIONISESCALATED\$, Standard Macros in Icinga
 \$NOTIFICATIONRECIPIENTS\$, Standard Macros in Icinga
 \$NOTIFICATIONTYPE\$, Standard Macros in Icinga
 \$OBJECTCACHEFILE\$, Standard Macros in Icinga
 \$PROCESSSTARTTIME\$, Standard Macros in Icinga
 \$RESOURCEFILE\$, Standard Macros in Icinga
 \$RETENTIONDATAFILE\$, Standard Macros in Icinga
 \$SERVICEACKAUTHOR\$, Standard Macros in Icinga
 \$SERVICEACKAUTHORALIAS\$, Standard Macros in Icinga
 \$SERVICEACKAUTHORNAME\$, Standard Macros in Icinga
 \$SERVICEACKCOMMENT\$, Standard Macros in Icinga
 \$SERVICEACTIONURL\$, Standard Macros in Icinga
 \$SERVICEATTEMPT\$, Standard Macros in Icinga
 \$SERVICECHECKCOMMAND\$, Standard Macros in Icinga
 \$SERVICEDESC\$, Standard Macros in Icinga
 \$SERVICEDIPLAYNAME\$, Standard Macros in Icinga
 \$SERVICEDOWNTIME\$, Standard Macros in Icinga
 \$SERVICEDURATION\$, Standard Macros in Icinga
 \$SERVICEDURATIONSEC\$, Standard Macros in Icinga
 \$SERVICEEVENTID\$, Standard Macros in Icinga
 \$SERVICEEXECUTIONTIME\$, Standard Macros in Icinga
 \$SERVICEGROUPACTIONURL\$, Standard Macros in Icinga
 \$SERVICEGROUPALIAS\$, Standard Macros in Icinga
 \$SERVICEGROUPMEMBERS\$, Standard Macros in Icinga
 \$SERVICEGROUPNAME\$, Standard Macros in Icinga
 \$SERVICEGROUPNAMES\$, Standard Macros in Icinga
 \$SERVICEGROUPNOTES\$, Standard Macros in Icinga
 \$SERVICEGROUPNOTESURL\$, Standard Macros in Icinga
 \$SERVICEISVOLATILE\$, Standard Macros in Icinga
 \$SERVICELATENCY\$, Standard Macros in Icinga
 \$SERVICENOTES\$, Standard Macros in Icinga
 \$SERVICENOTESURL\$, Standard Macros in Icinga
 \$SERVICENOTIFICATIONID\$, Standard Macros in Icinga
 \$SERVICENOTIFICATIONNUMBER\$, Standard Macros in Icinga
 \$SERVICEOUTPUT\$, Standard Macros in Icinga
 \$SERVICEPERCENTCHANGE\$, Standard Macros in Icinga
 \$SERVICEPERFDATA\$, Standard Macros in Icinga
 \$SERVICEPERFDATAFILE\$, Standard Macros in Icinga
 \$SERVICEPROBLEMD\$, Standard Macros in Icinga
 \$SERVICESTATE\$, Standard Macros in Icinga
 \$SERVICESTATEID\$, Standard Macros in Icinga
 \$SERVICESTATETYPE\$, Standard Macros in Icinga
 \$SHORTDATETIME\$, Standard Macros in Icinga
 \$STATUSDATAFILE\$, Standard Macros in Icinga
 \$TEMPFILE\$, Standard Macros in Icinga
 \$TEMPPATH\$, Standard Macros in Icinga
 \$TIME\$, Standard Macros in Icinga
 \$TIMET\$, Standard Macros in Icinga
 \$TOTALHOSTPROBLEMS\$, Standard Macros in Icinga
 \$TOTALHOSTPROBLEMSUNHANDLED\$, Standard Macros in Icinga

\$TOTALHOSTSDOWN\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSDOWNUNHANDLED\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSERVICES\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSERVICESCRITICAL\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSERVICESOK\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSERVICESUNKNOWN\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSERVICESWARNING\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSUNREACHABLE\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSUNREACHABLEUNHANDLED\$, [Standard Macros in Icinga](#)
 \$TOTALHOSTSUP\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICEPROBLEMS\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICEPROBLEMSUNHANDLED\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESCRITICAL\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESCRITICALUNHANDLED\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESOK\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESUNKNOWN\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESUNKNOWNUNHANDLED\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESWARNING\$, [Standard Macros in Icinga](#)
 \$TOTALSERVICESWARNINGUNHANDLED\$, [Standard Macros in Icinga](#)
 \$USERn\$, [Standard Macros in Icinga](#)
 Standard Macros in Icinga, [Standard Macros in Icinga](#)
 Summary Macros, [Standard Macros in Icinga](#)
 Understanding Macros and How They Work, [Understanding Macros and How They Work](#)
 Main Configuration File Options, [Main Configuration File Options](#)
 main_cfg_file=
 Main Configuration File Location, [CGI Configuration File Options](#)
 max_check_result_file_age=
 Max Check Result File Age, [Main Configuration File Options](#)
 max_check_result_reaper_time=
 Maximum Check Result Reaper, [Main Configuration File Options](#)
 max_concurrent_checks=
 Maximum Concurrent Service, [Main Configuration File Options](#)
 max_debug_file_size=
 Maximum Debug File Size, [Main Configuration File Options](#)
 max_host_check_spread=
 Maximum Host Check Spread, [Main Configuration File Options](#)
 max_service_check_spread=
 Maximum Service Check, [Main Configuration File Options](#)
 Migration
 from Nagios to Icinga, [Upgrading \(to\) Icinga](#)
 MKLiveStatus-Integration, [MKLiveStatus Integration](#)
 Monitoring
 Monitoring Linux/Unix Machines, [Monitoring Linux/Unix Machines](#)
 Monitoring Netware Servers, [Monitoring Netware Servers](#)
 Monitoring Network Printers, [Monitoring Network Printers](#)
 Monitoring Publicly Available Services, [Monitoring Publicly Available Services](#)
 Monitoring Routers and Switches, [Monitoring Routers and Switches](#)
 Monitoring Windows Machines, [Monitoring Windows Machines](#)
 Redundant and Failover Network Monitoring, [Redundant and Failover Network Monitoring](#)
 MultiSite, [Icinga Addons](#)

N

Nagios

migrating from Nagios to Icinga, [Upgrading \(to\) Icinga](#)

NagiosQL, [Icinga Addons](#)

NagVis, [Icinga Addons](#)

NConf, [Icinga Addons](#)

Network Hosts

Determining Status and Reachability of Network Hosts, [Determining Status and Reachability of Network Hosts](#)

notes_url_target=

Notes URL Target, [CGI Configuration File Options](#)

Notifications, [Notifications](#)

Audio Alerts, [Notifications](#)

Filter, [Notifications](#)

notification_timeout=

Notification Timeout, [Main Configuration File Options](#)

nrpe

nrpe, [NRPE](#)

send_nrpe, [NRPE](#)

NSCA

nsca, [NSCA](#)

send_nsca, [NSCA](#)

NSClient++, [Icinga Addons](#)

Monitoring Windows Machines, [Monitoring Windows Machines](#)

O

Object cache file, [Temporary Data](#)

Object Configuration Overview, [Object Configuration Overview](#)

Object Definitions, [Object Definitions](#)

Command, [Command Definition](#)

Contact, [Contact Definition](#)

Contactgroup, [Contactgroup Definition](#)

Host, [Host definition](#)

Host Dependency, [Hostdependency Definition](#)

Host Escalation, [Hostescalation Definition](#)

Hostgroup, [Hostgroup Definition](#)

Module, [Module Definition](#)

Retention, [Object Definitions](#)

Service, [Service Definition](#)

Service Dependency, [Servicedependency Definition](#)

Service Escalation, [Serviceescalation Definition](#)

Service Extended Information, [Serviceextinfo Definition](#)

Servicegroup, [Servicegroup Definition](#)

Time-Saving Tricks For Object Definitions, [Time-Saving Tricks For Object Definitions](#)

Timeperiod, [Timeperiod Definition](#)

Object definitions

Service Extended Information, [Hostextinfo Definition](#)

Object Inheritance, [Object Inheritance](#)

Additive Inheritance of String Values, [Object Inheritance](#)

Cancelling Inheritance of String Values, [Object Inheritance](#)

Implied Inheritance, [Object Inheritance](#)

Implied/Additive Inheritance in Escalations, [Object Inheritance](#)
 Important values, [Object Inheritance](#)
 Multiple Inheritance Sources, [Object Inheritance](#)
 object_cache_file= Object Cache File, [Main Configuration File Options](#)
 obsess_over_hosts= Obsess Over Hosts Option, [Main Configuration File Options](#)
 obsess_over_services= Obsess Over Services Option, [Main Configuration File Options](#)
 ochp_command= Obsessive Compulsive Host Processor, [Main Configuration File Options](#)
 ochp_timeout= Obsessive Compulsive Host Processor, [Main Configuration File Options](#)
 ocsp_command= Obsessive Compulsive Service Processor, [Main Configuration File Options](#)
 ocsp_timeout= Obsessive Compulsive Service Processor, [Main Configuration File Options](#)
 On-Call Rotations, [On-Call Rotations](#)

P

Passive Checks, [Passive Checks](#)
 Passive Host State Translation, [Passive Host State Translation](#)
 passive_host_checks_are_soft= Passive Host Checks Are SOFT, [Main Configuration File Options](#)
 perfdata_timeout= Performance Data Processor Command, [Main Configuration File Options](#)
 Performance Data, [Performance Data](#)
 persistent_ack_comments= Persistent Acknowledge Comments, [CGI Configuration File Options](#)
 physical_html_path= Physical HTML Path, [CGI Configuration File Options](#)
 ping_syntax= Ping Syntax, [CGI Configuration File Options](#)
 Plugins
 Icinga Plugin API, [Icinga Plugin API](#)
 Icinga Plugins, [Icinga Plugins](#)
 Integrating a new plugin, [Icinga Plugins](#)
 Plugins and macros, [Icinga Plugins](#)
 PNP4Nagios, [Icinga Addons](#)
 pnp with Icinga-Web, [Integration of PNP4Nagios into Icinga-Web](#)
 precached_object_file= Precached Object File, [Main Configuration File Options](#)
 process_performance_data= Performance Data Processing, [Main Configuration File Options](#)

Q

Quickstart
 Icinga and IDOUtils on FreeBSD, [Icinga and IDOUtils Quickstart on FreeBSD](#)
 Icinga on FreeBSD, [Icinga Quickstart FreeBSD](#)
 Icinga on Linux, [Icinga Quickstart](#)
 Icinga with IDOUtils, [Icinga with IDOUtils Quickstart](#)

Overview, Quickstart Installation Guides

R

RAM disk, Temporary Data

Reachability

Determining Status and Reachability of Network Hosts, [Determining Status and Reachability of Network Hosts](#)

Redundancy

Redundant and Failover Network Monitoring, [Redundant and Failover Network Monitoring](#)

refresh_rate=

CGI Refresh Rate, [CGI Configuration File Options](#)

resource_file=

Resource File, [Main Configuration File Options](#)

retained_contact_host_attribute_mask=

Retained Contact Host Attribute Mask, [Main Configuration File Options](#)

retained_contact_service_attribute_mask=

Retained Contact Service Attribute Mask, [Main Configuration File Options](#)

retained_host_attribute_mask=

Retained Host Attribute Mask, [Main Configuration File Options](#)

retained_process_attribute_mask=

Retained Process Attribute Mask, [Main Configuration File Options](#)

retained_process_host_attribute_mask=

Retained Process Host Attribute Mask, [Main Configuration File Options](#)

retained_scheduling_info=

Use Retained Scheduling Info, [Main Configuration File Options](#)

retained_service_attribute_mask=

Retained Host and Service Attribute Mask, [Main Configuration File Options](#)

retain_state_information=

State Retention Option, [Main Configuration File Options](#)

retention_update_interval=

Automatic State Retention Update, [Main Configuration File Options](#)

S

Same Host Dependencies, [Time-Saving Tricks For Object Definitions](#)

Same Host Dependencies With Servicegroups, [Time-Saving Tricks For Object Definitions](#)

Scheduling

Configuration Options, [Service and Host Check Scheduling](#)

Host Check Directives, [Service and Host Check Scheduling](#)

Host Checks, [Service and Host Check Scheduling](#)

Initial Scheduling, [Service and Host Check Scheduling](#)

Inter-Check Delay, [Service and Host Check Scheduling](#)

Maximum Concurrent Service Checks, [Service and Host Check Scheduling](#)

Normal Scheduling, [Service and Host Check Scheduling](#)

Scheduling Delays, [Service and Host Check Scheduling](#)

Scheduling During Problems, [Service and Host Check Scheduling](#)

Scheduling Example, [Service and Host Check Scheduling](#)

Service and Host Check Scheduling, [Service and Host Check Scheduling](#)

Service Definition Options That Affect Scheduling, [Service and Host Check Scheduling](#)

Service Interleaving, [Service and Host Check Scheduling](#)

Time Restraints, [Service and Host Check Scheduling](#)

Security Considerations, [Security Considerations](#)
 Service Checks, [Service Checks](#)
 Service Definition, [Service Definition](#)
 Service Definitions, [Time-Saving Tricks For Object Definitions](#)
 Service Dependency Definitions, [Time-Saving Tricks For Object Definitions](#)
 Service Escalation Definitions, [Time-Saving Tricks For Object Definitions](#)
 Servicedependency Definition, [Servicedependency Definition](#)
 Serviceescalation Definition, [Serviceescalation Definition](#)
 Servicegroup Definition, [Servicegroup Definition](#)
 service_check_timeout=
 Service Check Timeout, [Main Configuration File Options](#)
 service_check_timeout_state=
 Service Check Timeout State, [Main Configuration File Options](#)
 service_freshness_check_interval=
 Service Freshness Check, [Main Configuration File Options](#)
 service_interleave_factor=
 Service Interleave Factor, [Main Configuration File Options](#)
 service_inter_check_delay_method=
 Service Inter-Check Delay, [Main Configuration File Options](#)
 service_perfdata_command=
 Service Performance Data Processing, [Main Configuration File Options](#)
 service_perfdata_file=
 Service Performance Data File, [Main Configuration File Options](#)
 service_perfdata_file_mode=
 Service Performance Data File, [Main Configuration File Options](#)
 service_perfdata_file_processing_command=
 Service Performance Data, [Main Configuration File Options](#)
 service_perfdata_file_processing_interval=
 Service Performance Data, [Main Configuration File Options](#)
 service_perfdata_file_template=
 Service Performance Data File, [Main Configuration File Options](#)
 service_perfdata_process_empty_results
 Process Empty Service Performance Data Results, [Main Configuration File Options](#)
 showlog_current_states=
 Show current states, [CGI Configuration File Options](#)
 showlog_initial_states=
 Show initial states, [CGI Configuration File Options](#)
 show_tac_header=
 Show header with tactical information, [CGI Configuration File Options](#)
 show_tac_header_pending=
 Show header including pending counts, [CGI Configuration File Options](#)
 sleep_time=
 Inter-Check Sleep Time, [Main Configuration File Options](#)
 SNMP Trap Integration, [SNMP Trap Integration](#)
 soft_state_dependencies=
 Soft State Dependencies, [Main Configuration File Options](#)
 splunk_url=
 Splunk URL, [CGI Configuration File Options](#)
 stalking_event_handlers_for_hosts=
 Stalking Event Handlers for Hosts, [Main Configuration File Options](#)
 stalking_event_handlers_for_services=
 Stalking Event Handlers for Services, [Main Configuration File Options](#)
 Starting and Stopping Icinga, [Starting and Stopping Icinga](#)

State Stalking, [State Stalking](#)
 State Types, [State Types](#)
 state_retention_file= State Retention File, [Main Configuration File Options](#)
 Status file, [Temporary Data](#)
 statusmap_background_image= Statusmap CGI Background, [CGI Configuration File Options](#)
 status_file= Status File, [Main Configuration File Options](#)
 status_log= Status Log,
 status_update_interval= Status File Update Interval, [Main Configuration File Options](#)
 syslog_local_facility= Syslog Local Facility Logging Value, [Main Configuration File Options](#)
 System requirements, [About Icinga](#)

T

Tables

acknowledgements, [Historical Tables](#)
 commands, [Configuration Tables](#)
 commenthistory, [Historical Tables](#)
 comments, [Current Status Tables](#)
 configfiles, [Configuration Tables](#)
 configfilevariables, [Configuration Tables](#)
 conninfo, [Debugging Tables](#)
 contactgroups, [Configuration Tables](#)
 contactgroup_members, [Configuration Tables](#)
 contactnotfications, [Historical Tables](#)
 contactnotificationmethods, [Configuration Tables](#)
 contactnotificationsmethods, [Historical Tables](#)
 contacts, [Configuration Tables](#)
 contact_addresses, [Configuration Tables](#)
 contact_notificationcommands, [Configuration Tables](#)
 customvariables, [Configuration Tables](#)
 customvariablestatus, [Current Status Tables](#)
 downtimehistory, [Historical Tables](#)
 eventhandlers, [Historical Tables](#)
 externalcommands, [Historical Tables](#)
 flappinghistory, [Historical Tables](#)
 hostchecks, [Historical Tables](#)
 hostdependencies, [Configuration Tables](#)
 hostescalations, [Configuration Tables](#)
 hostescalation_contactgroups, [Configuration Tables](#)
 hostgroups, [Configuration Tables](#)
 hostgroup_members, [Configuration Tables](#)
 hosts, [Configuration Tables](#)
 hoststatus, [Current Status Tables](#)
 host_contactgroups, [Configuration Tables](#)
 host_parenthosts, [Configuration Tables](#)
 instances, [Central Tables](#)
 logentries, [Historical Tables](#)

notifications, [Historical Tables](#)
 objects, [Central Tables](#)
 processevents, [Historical Tables](#)
 programstatus, [Current Status Tables](#)
 runtimevariables, [Current Status Tables](#)
 scheduleddowntime, [Current Status Tables](#)
 servicechecks, [Historical Tables](#)
 servicedependencies, [Configuration Tables](#)
 serviceescalations, [Configuration Tables](#)
 serviceescalation_contactgroups, [Configuration Tables](#)
 servicegroups, [Configuration Tables](#)
 servicegroup_members, [Configuration Tables](#)
 services, [Configuration Tables](#)
 servicestatus, [Current Status Tables](#)
 service_contactgroups, [Configuration Tables](#)
 statehistory, [Historical Tables](#)
 systemcommands, [Historical Tables](#)
 timedeventqueue, [Current Status Tables](#)
 timedevevents, [Historical Tables](#)
 timeperiods, [Configuration Tables](#)
 timeperiod_timeranges, [Configuration Tables](#)
tab_friendly_titles=
 Show object type in tab title, [CGI Configuration File Options](#)
tac_show_only_hard_state=
 Tac Show Only Hard State, [CGI Configuration File Options](#)
 TCP Wrapper Integration, [TCP Wrapper Integration](#)
 Temporary Data, [Temporary Data](#)
temp_file=
 Temp File, [Main Configuration File Options](#)
temp_path=
 Temp Path, [Main Configuration File Options](#)
 Thruk, [Icinga Addons](#)
 Time Periods, [Time Periods](#)
 Timeperiod Definition, [Timeperiod Definition](#)
translage_passive_host_checks=
 Translate Passive Host Checks, [Main Configuration File Options](#)
 Tuning Icinga For Maximum Performance, [Tuning Icinga For Maximum Performance](#)

U

Update/upgrade Icinga-Web database, [Upgrading Icinga-Web and Icinga-Web Database](#)
 Update/upgrade of Icinga-Web, [Upgrading Icinga-Web and Icinga-Web Database](#)
 Upgrading, [Upgrading \(to\) Icinga](#)
 from a previous Icinga version, [Upgrading \(to\) Icinga](#)
 from Nagios version 2.x, [Upgrading \(to\) Icinga](#)
 from Nagios version 3.x, [Upgrading \(to\) Icinga](#)
 Upgrading IDOUtils, [Upgrading IDOUtils Database](#)
url_html_path=
 URL HTML Path, [CGI Configuration File Options](#)
use_aggressive_host_checking=
 Aggressive Host Checking, [Main Configuration File Options](#)
use_agressive_host_checking=
 Aggressive Host Checking, [Main Configuration File Options](#)

v

use_authentication=
 Authentication Usage, [CGI Configuration File Options](#)
use_embedded_perl_implicitly=
 Embedded Perl Implicit Use, [Main Configuration File Options](#)
use_large_installation_tweaks=
 Large Installation Tweaks, [Main Configuration File Options](#)
use_logging=
 Enable/disable CGI command logging, [CGI Configuration File Options](#)
use_regexp_matching=
 Regular Expression Matching, [Main Configuration File Options](#)
use_retained_program_state=
 Use Retained Program State, [Main Configuration File Options](#)
use_syslog=
 Syslog Logging Option, [Main Configuration File Options](#)
use_syslog_local_facility=
 Syslog Local Facility Logging Option, [Main Configuration File Options](#)
use_timezone=
 Timezone Option, [Main Configuration File Options](#)
use_true-regexp_matching=
 True Regular Expression Matching, [Main Configuration File Options](#)

V

Verifying Your Configuration, [Verifying Your Configuration](#)
Volatile Services, [Volatile Services](#)

W

Webinterface, [Installation of the Icinga-Web Frontend](#)
Errors in the, [Installation of the Icinga-Web Frontend](#)
What is Icinga?, [About Icinga](#)
What's new in Icinga, [What's New in Icinga 1.4](#)

[Prev](#)

[Database changes/alterations](#)

[Home](#)

© 2009-2011 Icinga Development Team, <http://www.icinga.org>