SysOrb Network Monitoring System Administrator's Guide

For version 4.6.0

SysOrb Network Monitoring System Administrator's GuideFor version 4.6.0

Table of Contents

List of Tables

Abstract

This document describes the installation and initial configuration of the SysOrb Network Monitoring System. Please refer to the *User's guide for the SysOrb Network Monitoring System* for information about using the system after it is installed correctly. In particular the use of the Web interface is described in detail there.

This document is intended for use by the personnel responsible for *installing, configuring and maintaining* the SysOrb Network Monitoring System Server and Agents on a daily basis. It does not cover actual configuration of monitoring parameters etc.

Chapter 1. System Overview

The purpose of the SysOrb Monitoring System is to monitor servers in a network by talking to the servers and retrieving information about running services, current load and other metrics. This information can be used by the administrators of the network to track down problems with monitored servers, to solve problems before they arise, or as documentation or a forensics tool once problems has occurred.

1.1. SysOrb Components

The SysOrb Network Monitoring System consists of several major software components, the responsibilities of which are outlined below:

- **The SysOrb Server:** This is the center of the SysOrb System. The Server can remotely check networked services (such as web servers and mail servers) via NetChecks, but it is also the central component to which the Agents will report device information and statistics. The SysOrb Server is the central repository in which all statistical and operational data are stored, and it is the entity in the SysOrb System that will actively alert administrators when problems arise in any of the monitored systems.

  In the following text a **node** is any machine or device that can be monitored by SysOrb. A **host** is a node where the SysOrb Agent can be installed.
- **The SysOrb Agent:** In order to actively monitor local devices (such as harddrives, memory, processor statistics etc.) on networked systems, those systems must run the SysOrb Agent. This program will gather operational information from the system on which it is running, and report these data to the SysOrb Server for further processing. In case of failures (such as a harddisk running full, or a crashed service process), the SysOrb Server can actively alert the administrators of the particular machine that reported the failure.
- **The Web Interface:** In order to allow easy access to the monitoring data and then configuration of the SysOrb System from anywhere and from any platform, the Web-based Interface is provided. This User Interface ships as a part of the SysOrb Server, and is usually (but not necessarily) run on the same physical machine as the one running the SysOrb Server. In order to fully use and configure monitored services or devices in the SysOrb System, all you need is a computer with a web-browser capable of accessing the SysOrb Server computer.

1.2. How It Works

To illustrate how the SysOrb components work together, Figure 1 shows a setup of a SysOrb Server, some SysOrb Agents and a SysOrb Web Interface.

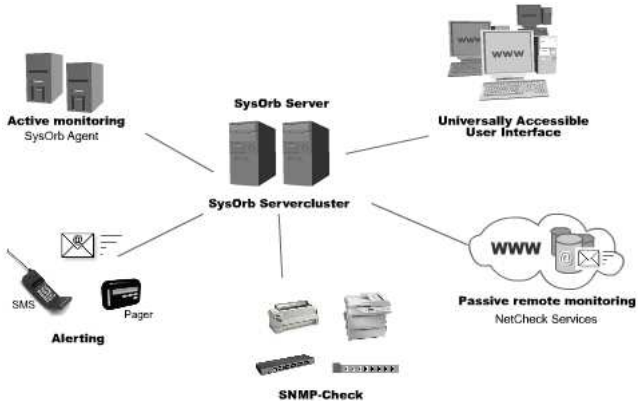


Figure 1: SysOrb Overview

The setup consists of a SysOrb Server, a SysOrb Agent and a SysOrb Web Interface. When a SysOrb Agent checks in on the SysOrb Server information about the SysOrb Agent's current status is stored in the SysOrb Server's database.

The SysOrb Web Interface connects to the SysOrb Server and through this interface the SysOrb user can monitor the network, add new SysOrb Agents and similar operations.

Chapter 2. System Requirements

The two sections below describe the minimum requirements for the SysOrb Agent and the SysOrb Server.

2.1. Agent Requirements

The SysOrb Agent software will run on most hardware used in servers today, and it currently supports most major server operating systems. Because of this the SysOrb Agent software can be run on your servers without interfering with the servers' normal operations.

In general, the SysOrb Agent has very low system requirements. If your server hardware can run the installed operating system, chances are it can run the SysOrb Agent without problems. The actual requirements for the agents will vary depending on the amount of checks configured on the given agent, and the check-in interval configured.

2.2. Server Requirements

The requirements for the SysOrb Server depend on the number of NetChecks, AgentChecks and snmpChecks monitored. If only a small number of these need to be monitored the SysOrb Server can be run on very modest hardware. The more checks monitored, the higher the requirements for the SysOrb Server.

Please note: You would usually want to dedicate a machine to running the SysOrb Server, in order to minimize the risk of failures caused by other software, on your main monitoring Server. The SysOrb Server can, however, easily run on a system which is handling other tasks as well (such as a mail-server or name-server). It is not recommended to install the SysOrb Server on a system that is already **very** busy, as the SysOrb Server will consume resources, and some measurements (such as network response time measurements) will be disturbed and inaccurate if the system is too busy handling other tasks.

Tip: In general, the database space requirements will grow with approximately one megabyte, per monitored check, and one megabyte per monitored node. So monitoring 20 devices on each of 10 different machines, will require approximately 210 megabytes of disk space.

In addition to this, reports in particular will also add to the disk space consumption. Reports are stored efficiently and a fully detailed report over the aforementioned 200 monitored checks will require much less than the 200 megabytes required for the full scale monitoring. Yet, when setting up a system in which many users have access to generate reports, report disk space consumption should be taken into account.

One should, of course, configure a disk space check with suitable warning and alert thresholds, for the filesystem on which the database reside, when configuring the agent on the SysOrb Server.

- Windows 7 or Windows Server 2012 (x86_64):
  - **CPU:** Intel Core2 compatible or better
  - **RAM:** 1 GB free memory for smaller installations (less than a hundred nodes configured). 2-8 GB recommended for larger installations.
  - **Disk:** 100MB for installation and log files and approximately 1 MB per configured check. E.g. an installation with 100 nodes and 100 active checks for each node would require 10 GB of disk space for the database

- Red Hat Enterprise Linux 6 (x86_64):
  - **CPU:** Intel Core2 compatible or better
  - **RAM:** 1 GB free memory for smaller installations (less than a hundred nodes configured). 2-8 GB recommended for larger installations.
  - **Disk:** 30MB for installation and log files and approximately 1 MB per configured check. E.g. an installation with 100 nodes and 100 active checks for each node would require 10 GB of disk space for the database

- Solaris 11 (x86_64):
  - **CPU:** Intel Core2 compatible or better
  - **RAM:** 1 GB free memory for smaller installations (less than a hundred nodes configured). 2-8 GB recommended for larger installations.
  - **Disk:** 50MB for installation and log files and approximately 1 MB per configured check. E.g. an installation with 100 nodes and 100 active checks for each node would require 10 GB of disk space for the database

- Debian GNU/Linux 7.0 (x86_64):
  - **CPU:** Intel Core2 compatible or better
  - **RAM:** 1 GB free memory for smaller installations (less than a hundred nodes configured). 2-8 GB recommended for larger installations.
  - **Disk:** 30MB for installation and log files and approximately 1 MB per configured check. E.g. an installation with 100 nodes and 100 active checks for each node would require 10 GB of disk space for the database

There are many other factors that affect overall system performance. Large installations may need not just *bigger* but also *faster* storage. In general, more memory will help the disk I/O system. Using servers with more than one CPU core is also beneficial to SysOrb, especially in large installations.

Chapter 3. Installing SysOrb

3.1. Installing the SysOrb Server

The SysOrb Server is a fairly complex piece of software, yet the installation itself is simple. Please refer to Chapter 5 for detailed installation instructions for your particular server platform.

The Server may be able to run with its default configuration right after the install completes, but it is *recommended* that the configuration section is consulted before actually running the Server.

The SysOrb Server installation also contains the SysOrb Web interface. This section will not describe setup of any specific web-servers. The SysOrb Web Interface consists of CGI programs, which will work with *any* web-server supporting the CGI standard.

> **Notice for IIS users:** Please make sure that you do not have third-party scripting extensions installed on the web server on which you will run the SysOrb Web interface. It is a common problem that these tools will attempt to interpret the SysOrb CGI programs as *scripts* written in some interpreted language (such as Perl). This will result in a non-functional SysOrb Web interface installation!
>
> The SysOrb Web interface CGI programs are standard Win32 executables, and unless the configuration is overwritten by third party products, a standard IIS will work out of the box with SysOrb.
>
> It is possible to manually override the third party scripting product configuration in the web server, to work around this problem and run both the scripting product and the SysOrb Web interface on the same web server. Please refer to the IIS documentation, or send an e-mail to <`support@sysorb.com`> for assistance in this matter.

You can install the Web Interface wherever you please, you can even run a SysOrb Server on one platform (say, Solaris) and the Web Interface on another (for example Windows). If you do not want to install a Web-server on the SysOrb Server machine, you can install the SysOrb Server package on any other machine, and then disable the SysOrb Server there so that only the web interface remains active.

> **Note:** When the SysOrb Web Interface is installed, you can use it to access and configure the SysOrb Server. The Web Interface will ask you for a username/password. For the default configuration this is:
>
> - **Username:** admin
> - **Password:** admtest
> - **Domain:** . *(or just leave blank)*
>
> After you have logged in for the first time, make sure to change at least the password for the admin user. For more information about how to use the Web Interface, please refer to *User's guide for the SysOrb Monitoring System*.

3.1.1. Microsoft Windows 7 / Server 2012

To install the SysOrb Server on machine running a Microsoft Windows operating system, you must have administrator access to that machine. Please make sure that you are logged on as the *Local* Administrator, or as another user with administrative privileges on the local machine.

The full Server and Web interface installation is contained in the file:

`sysorb-server-4.6.0-6039.win7-amd64.amd64.msi`

Download and run this file, and follow the instructions on screen during the install process.

Note: On the download page for SysOrb it is also possible to download an installation guide for Windows, which contains a screen for screen walk-through of the installation process.

If, after installation, you get an error page in your browser instead of the SysOrb Login page, please check that the *"SysOrb Server"* service is running in the Service Control Manager.

When the installation is finished you need to install your license file, if you have purchased SysOrb. See Section 3.2 for more information about how you do this. If you just want to test SysOrb, a test license is included in the installation.

Although the server is normally run with its default configuration from the installation procedure, it is recommended that you consult Chapter 5 for a more thorough description of all available configuration options.

The web-server should *allow execution* of CGI programs, and should be able to use the file `index.cgi` as the default for a directory. Apart from that, there are no special requirements for the server to work with the SysOrb web interface.

You should be able to see either an error page (showing a white background and an information-icon) from the SysOrb Web Interface if your SysOrb Server is not running, or you should see the login page.

If you cannot get to the SysOrb Login page, please refer to the "IIS Notice" at the very beginning of Section 3.1 for further information.

3.1.2. Red Hat Enterprise Linux 6 or CentOS6

The full Server and Web interface installation for Red Hat Enterprise Linux is contained in the file:

`sysorb-server-4.6.0-6039.centos6-amd64.amd64.rpm`

Note: The Red Hat Enterprise Linux 6 package should also work on all newer releases of Red Hat Enterprise Linux as well as CentOs and Fedora. If that is not the case, please contact Evalesco support.

These files contain the entire SysOrb Server software package, and are available for download from http://www.evalesco.com.

In order to install the SysOrb Server, you must have root access to the system.

Install the SysOrb Server using your favorite RPM utility (for support on the RPM utility, see http://www.redhat.com), or using the plain **rpm** command, e.g.:

rpm -Uvh `sysorb-server-4.6.0-6039.centos6-amd64.amd64.rpm`

You now have a SysOrb Server configured on your system. If you have purchased SysOrb, you will need to install your license file (See Section 3.2 for information about how you do this). See Chapter 5 for information on how to configure your newly installed SysOrb Server.

Type **/etc/rc.d/init.d/sysorbd start** to start the server for the first time. Per default SysOrb is set up to automatically start in runlevel 3 and 5.

The installation script will automatically configure an alias in your apache configuration for the SysOrb Web interface.

Note: The installation script tries to add the Alias to the Apache configuration files in `/etc/httpd/conf/`. If your system does not have the configuration files in this directory edit them manually to export the directory `www` in the SysOrb Web Interface installation directory as `/sysorb/`. Make sure you allow execution of CGI scripts. If you installed the SysOrb Web Interface from the above mentioned RPMs, the SysOrb Web interface installation directory is `/var/lib/sysorb/www`.

Remember to restart the Apache web server after having installed the SysOrb Web Interface. The web server must re-read its configuration in order to recognize the new `/sysorb/` alias.

When this is done, you should be able to see the login page (or an error page if your server is not yet running), by going to the URL http://localhost/sysorb/.

> **Note:** If you are using a SELinux enabled Red Hat distribution, and are using SELinux policies on your web server, you have to perform the following steps for the webinterface to work:
>
> ```
> cd /var/lib/sysorb/www
> chcon -t httpd_sys_content_t *.{png,gif,css,js}
> chcon -t httpd_sys_script_exec_t *.cgi
> ```
>
> Once these commands have been run, the webinterface will work. However, an audit will be generated each time one of the cgi-files are executed. In order to disable this audit, the `selinux-policy-targeted-sources` package must be installed. Once this is installed, perform the following steps:
>
> ```
> cd /etc/selinux/targeted/src/policy/
> echo "dontaudit httpd_sys_script_t ld_so_cache_t:file execute;" >> domains/misc/local.te
> make reload
> ```
>
> After this the audit message will be disabled.

If you have trouble getting this to work, please take a look in the error and access logs from your web-server. These logs often help by pointing out errors in the configuration files.

3.1.3. Debian Linux 7.0 and other Debian based systems

Debian users should download the entire SysOrb Server software package named:

`sysorb-server-4.6.0-6039.debian7-amd64.x64.deb`

These files can be downloaded from http://www.evalesco.com. This should also install on most Debian based distributions, e.g. Ubunutu. Also, you should make sure that you have root access to the system on which you are installing the server software.

Install the packages with command:

dpkg -i `sysorb-server-4.6.0-6039.debian7-amd64.x64.deb`

You now have a SysOrb Server configured on your system. If you have purchased SysOrb, you will need to install you license file (See Section 3.2 for information about how you do this), before starting SysOrb. Please consult Chapter 5 for detailed information about all possible configuration parameters.

Type **/etc/init.d/sysorbd start** to start the server for the first time. Per default SysOrb is set up to automatically start in runlevel 2.

The installation script will automatically configure an alias in your apache configuration for the SysOrb Web interface.

> **Note:** The installation script tries to add the Alias to the Apache configuration files in `/etc/apache/`. If your system does not have the configuration files in this directory edit them manually to export the directory `www` in the SysOrb Web Interface installation directory as `/sysorb/`. Make sure you allow execution of CGI scripts. If you installed the SysOrb Web Interface from the above mentioned debian package, the Web-interface is located in `/var/lib/sysorb/www`.

Please remember to restart the Apache web server after having installed the SysOrb Web Interface. The web server must re-read its configuration in order to recognize the new `/sysorb/` alias.

Once this is done, you should be able to see the login page (or an error page if your server is not yet running) by going to the URL http://localhost/sysorb/.

If you have trouble making this work, please take a look in the error and access logs from your web-server. These logs often help by pointing out errors in the configuration files.

3.1.4. Solaris 11

SysOrb server is avaiable for Solaris 11 on x86_64 platforms.

Package to download:

```
sysorb-server-4.6.0-6039.solaris11-amd64.x86.p5p
```

In order to install the SysOrb Server, you must have root access to the system. The SysOrb Server is distributed as an Oracle Solaris package repository, '.p5p' file.

A normal installation in the global zone is straight forward:

pkg install -g `sysorb-server-4.6.0-6039.solaris11-amd64.x86.p5p` sysorb-server

You now have a SysOrb Server configured on your system. If you have purchased SysOrb, you will need to install you license file (See Section 3.2 for information about how you do this), before starting SysOrb. Please consult Chapter 5 for detailed information about all possible configuration parameters.

Type **svcs sysorbd** to see the status of the SysOrb service. It should have started up automatically after installation.

The SysOrb installation will include a configuration of the Apache 2.2 (and 2.4) server that is supplied with Solaris 11. In order to access the SysOrb web interface, the Apache server must be running. The web server status can be inquired with the command **svcs http:apache22**. If this server is not running, it can be enabled with the command **svcadm enable http:apache22**.

Once this is done, you should be able to see the login page by accessing http://localhost/sysorb/.

3.2. Installing a license file

If you have purchased a license for SysOrb, you need to install it after you finished the installation of SysOrb.

The installation is simple. Just overwrite the old `sysorb.lic` file with the one that you downloaded from the license manager. The default license location on Microsoft Windows is:

`install-dir``\Config\sysorb.lic`

On Debian, and Red Hat Linux systems it is:

```
/etc/sysorb/sysorb.lic
```

And on Solaris it is:

```
/etc/sysorb/sysorb.lic
```

Once the license file is in place, you will need to restart the SysOrb Server in order for it to notice the change of the license. If you are not certain that the installation of the new license worked, you can look in the `server.log` for a line that says:

```
[dbms]: License is valid for xx servers and xx agents.
```


3.3. Upgrading the SysOrb Server

This chapter describes the steps necessary to upgrade you SysOrb server to version 4.6.0.

You simply upgrade SysOrb server by stopping the service, installing the new package, and starting the new SysOrb server afterwards. No database conversion is needed.

3.3.1. Importing the default NodeClasses

SysOrb ships with some default NodeClasses. These classes provides an example of how one can use the NodeClasses to create a hierarchy that can identify many different kinds of machines. Many of the default NodeClasses also contain information about how SysOrb can automatically discover whether a given node belongs to the NodeClass.

> **Note:** You can only import the default NodeClasses when you have upgraded your SysOrb Server to at least version 3.0. Also note, that if you started with SysOrb 3.0, the NodeClasses are included in the default database.

To import the default NodeClasses you must use the command:

sysorb-importer -l username -f nodeclasses.xml

In most Unix systems the sysorb-importer program is located in the path. On Windows systems they are located in `C:\Program Files\SysOrb Server\`. The `nodeclasses.xml` is located in the same directory as the SysOrb database files. On most unix systems this is in `/var/lib/sysorb`. On Windows it is `C:\Program Files\SysOrb Server\Config`.

Upon execution, the sysorb-importer will ask you about the password for the username specified on the command line. Once entered the program will import all the NodeClasses and when it is finished, you can access the NodeClasses from the SysOrb Web Interface.

3.4. Installing the SysOrb Agent

The SysOrb Agent is a very light piece of software, with very few configuration options. It retrieves its configuration from the SysOrb Server during normal operation, so the only configuration actually needed on the agent side (for the basic agent functionality) is the list of SysOrb servers to which the Agent should check in, and a name and a SysOrb domain used by the agent to identify itself to the SysOrb server.

You can use the DNS name of the host as the Agent name, or you can use the host's IP address. Both options will allow the SysOrb server to perform NetChecks and snmpChecks on the host too, provided that the given DNS name or IP address is actually reachable from the SysOrb Server. If you do not need to perform NetChecks or snmpChecks on the host, the Agent name can be chosen freely.

On the SysOrb server you are supposed to create a node with the exact same name as specified in the configuration, in the specified domain.

> **Tip: Node Labels:** It is usually undesirable to have a list of IP addresses displayed in the web interface, as the sole means of identifying one's monitored systems. Yet, it is sometimes a requirement that IP addresses be used for the agent names (for example, if the agent hosts do not *have* DNS names).
>
> Node Labels are a solution to this problem. When configuring the agent, in the web interface, the user is provided with both an agent host name entry field (which is mandatory), and an *optional* "node label" entry field. Any descriptive name can be inserted into the node label field - it will then be displayed in the usual web interface host listings afterwards. The node label does not affect interaction with either NetChecks, SNMPChecks, or the agent - it is purely a displayed name.

To actually tell the SysOrb Server to monitor a specific SysOrb Agent you need to use the SysOrb Web Interface. See *User's Guide to the SysOrb Monitoring System* for more information on how to use the SysOrb Web Interface.

3.4.1. Microsoft Windows 7 / Server 2012

The SysOrb Agent installation package is named:

`sysorb-agent-4.6.0-6039.win7-amd64.amd64.msi`

This file can be downloaded from http://www.evalesco.com. To start the installation procedure, either double click on the downloaded file, or execute the following command in the Command Prompt:

msiexec -i `sysorb-agent-4.6.0-6039.win7-amd64.amd64.msi`

When following the instructions in the installation program, **please notice the dialog asking for SysOrb Server Name, SysOrb Domain and SysOrb Agent Name**. If the SysOrb Server Name is incorrectly entered the SysOrb Agent will not be able to send monitoring information to the SysOrb Server. If the names cannot be entered during installation they can later be changed as described in Chapter 8 of this document.

> **Upgrade note:** If you need to manually upgrade the SysOrb Agent package, you can proceed in two ways. The simplest way is to start by uninstalling the SysOrb Agent, through Add/Remove Programs, and then install the new package. However that will remove the configuration and requires that the key is released on the SysOrb Server. The other solution is to run the following command in the Command Prompt:
>
> **msiexec -i `sysorb-agent-4.6.0-6039.win7-amd64.amd64.msi` REINSTALL=ALL REINSTALLMODE=vomus**
>
> Performing the upgrade this way, will retain all your configuration, and will not require a release of the Agent's key. Please note that the upgrade command above is case sensitive and must be written exactely as shown.

3.4.2. Red Hat Enterprise Linux 6 or CentOS6

Users of Red Hat Enterprise linux should download:

`sysorb-agent-4.6.0-6039.centos6-amd64.amd64.rpm`

You can download these files from http://www.evalesco.com. Also, you should make sure that you have root access to the system on which you are installing the Agent Software.

> **Note:** These packages are for the newer enterprise Red Hat editions. If you are using an older version of Red Hat Linux, please download and install the package for Red Hat Linux 7.2 instead.

Use the RPM utility to install the package. Example:

rpm -Uvh `sysorb-agent-4.6.0-6039.centos6-amd64.amd64.rpm`

This should successfully install the SysOrb Agent on your system.

Please read Chapter 8 before you start the agent. The agent must know where to find its server before it will be able to do anything useful.

In order to start the SysOrb Agent after the installation, type in the command: **/etc/rc.d/init.d/soagent start**. Per default the Agent is configured to start in the runlevels 3 and 5.

3.4.3. Debian GNU/Linux 7.0 and other Debian based systems

Debian 7.0 users can download the installation package named:

`sysorb-agent-4.6.0-6039.debian7-amd64.x64.deb`

The file can be downloaded from http://www.evalesco.com. This should also install on most Debian based distributions, e.g. Ubunutu. Also, you should make sure that you have root access to the system on which you are installing the agent software.

Use the dpkg utility to install the package, e.g.:

dpkg -i `sysorb-agent-4.6.0-6039.debian7-amd64.x64.deb`

This should successfully install the SysOrb Agent on your system.

Please read Chapter 8 before you start the agent. The agent must know the name of it's server, to be able to checkin.

Then type **/etc/init.d/soagent start** to start the agent for the first time. Per default the Agent is set up to automatically start in runlevel 2.

3.4.4. Solaris 11

To install the SysOrb Agent on a machine running Solaris 11 on an x86_64, you should download the file named:

`sysorb-agent-4.6.0-6039.solaris11-amd64.x86.p5p`

Please note, you must have root access to the system on which you are installing this software.

In order to install the agent on the global zone of a Solaris 11 system, issue the command:

pkg install -g `sysorb-agent-4.6.0-6039.solaris11-amd64.x86.p5p` sysorb-agent

Your SysOrb Agent should now be installed. Please read Chapter 8 to see how the `/etc/sysorb/agent.conf` agent configuration file should be set up.

The status of the agent service can be inquired using the command **svcs soagent**. The agent service can be enabled using the command **svcadm enable soagent**.

3.5. Unattended installation on Windows

In order to do an unattended on Windows, one must use the `msiexec` program from the command line. The basic syntax of an unattended installation is:

msiexec /passive /i `sysorb-agent-4.6.0-6039.win7-amd64.amd64.msi`

This will install the SysOrb agent using the default configuration options, meaning the SysOrb agent will most likely have to be configured after installation.

To avoid having to configure the SysOrg agent after installation, arguments can be given to the installer changing the default settings. The **msiexec** can be given properties as Property=PropertyValue on the command line.

The SysOrb agent installer uses a number of properties:

- AGENTNAME - The name of this SysOrb Agent.
- SERVERLIST - Comma seperated list of SysOrb servers the agent should check in to.
- AGENTDOMAIN - The domain for this SysOrb agent.
- AUTOUPGRADE - Set to 'true' to
- LOGFILE - The full path to the agent log file.

- LOGCHKCONF - The full path to the log check configuration file.
- SERVERPORT - The port the SysOrb server is listening on.
- LOGDAYS - The number of days between log file rotation.
- LOGLEVEL - The amount of logging.
- AUTOUPGRADE - Set to 'true' to allow autoupgrade of the agent, 'false' otherwise.

So, in order to do an unattended installation of the SysOrb agent with the name *MySysOrbAgent* checking in to the server *MySysOrbServer* disallowing automatic upgrades, one could execute:

msiexec -passive -i `sysorb-agent-4.6.0-6039.win7-amd64.amd64.msi` AGENTNAME=MySysOrbAgent SERVERLIST=MySysOrbServer AUTOUPGRADE=false

Chapter 4. Auto-upgrading the SysOrb Agent

Once the SysOrb Agent is installed on every monitored machine, you will not have to upgrade the Agent manually for each new release of SysOrb. Instead a special package containing the Agent for every platform will be distributed with the SysOrb Server.

Every time the Agent checks in to the Server, it will check whether a new version of itself is available. If that is the case, it will download the new version and upgrade itself without human intervention.

In order to enable this, all you have to do is copy the .spm file containing the new Agent into `/var/lib/sysorb/upgrade` on non-Windows systems, or `C:\Program Files\SysOrb Server\Upgrade` on Windows. The the Agents will start downloading the next time they check in.

> **Troubleshooting tip:** If the agents do not upgrade, you can try enabling the log_upgrade_debug switch on the SysOrb Server and restart it. (See Chapter 5 for details on configuring the Agent) Within the first about 50 lines written in server.log after the restart should be a line like
>
> ```
> Scanning for autoupgrade packages in...
> ```
>
> and immediately after the server will list the package files it finds.

The agent/server protocol is designed with the utmost care for security. The server has to authenticate itself using a 128 bit key, which is also used for encryption of subsequent communication. The chances of a hacker intercepting the transfer of an upgraded Agent executable are minimal. However, if the machine running the SysOrb Server is otherwise compromised, a clever hacker would be able to trick the SysOrb Server into sending any file as an Agent upgrade, thereby gaining access to all the machines running Agents.

If you will not risk this on your mission critical machines, you can instruct individual Agents not to download executables from the SysOrb Server, no matter what the server says. (See Chapter 8 for details on configuring the Agent)

4.1. Auto-upgrade with SysOrb Satellites

The installed auto-upgrade packages are local to the SysOrb Server or Satellite they are installed on. This means that the auto-upgrade packages needs to be installed on all SysOrb Servers or Satellites where agents check-in, in order to upgrade all the agents.

Chapter 5. SysOrb Server Configuration

The SysOrb Server offers a wide range of configuration options for both debugging and troubleshooting, as well as for performance tuning and administrative convenience.

The different ports of the server have similar configuration options, but the actual way in which the configuration options are accessed differ, depending on which platform you are running. On Unix-like systems the configuration is stored in files and on Microsoft Windows the configuration is stored in the system registry.

After a configuration change, you will have to restart the SysOrb Server service (a complete server reboot is not necessary).

5.1. Microsoft Windows

Server configuration is done in a program called **SysOrb Config** which can be found in the start menu. This program includes detailed descriptions of each option.

The program is located in the `Programs` folder specified during the installation.

5.2. Unix-like systems (FreeBSD, Linux and Solaris)

The configuration file is located in:

`/etc/sysorb/server.conf`

on all Unix-like platforms, except FreeBSD where it is located in:

`/usr/local/etc/sysorb/server.conf`

This file contains the configuration for the server. If you change this file you must restart the server for the changes to take effect.

You can however make the server reopen its log file by sending a hangup signal (**SIGHUP**) to the `sysorbd` process. This is used in the logrotate script that accompanies the server on Linux.

The options of primary interest are likely to be the location of the database files, and the logging options. See the reference for details.

5.3. Configuration Options Reference

The configuration directives for the SysOrb Server fall into nine categories:

- **Logging Options** defines where the SysOrb Server should keep its logfiles and what information should be logged.
- **Database Options** defines where the data files for the SysOrb Server's database should be stored.
- **Network Connection Manager Options** defines what TCP port the SysOrb Server should use and how many concurrent connections it will accept.
- **NetCheck Manager Options** defines maximum number of NetChecks (e.g. mail server check).
- **General SysOrb Server Options** defines other options for the SysOrb Server.
- **Alert Dispatcher Options** defines how and to whom alerts will be given.
- **SNMP options Options** defines various apects on how SysOrb should perform SnmpChecks.

- **Grid Options** defines options for a grid of SysOrb servers and satellites.
- **WebAPI Options** defines options for a WebAPI service.

5.3.1. Logging Options

`logdir` (path)

This is the location of the logfile. It is not the filename, but the directory in which the file should be. The file will be named `server.log` within the specified directory.

Default value (Unix): `/var/log/sysorb`

Default value (Windows): *`install-dir`*`\Config`

`log_assert` (boolean)

Whether the server should log critical internal errors.

Default value: `true`

`log_audit` (boolean)

Enable/disable audit logging - logging of all user interactions with the system.

Default value: `false`

`log_dbms_cpu` (boolean)

Whether the database subsystem should log information about how it spends its time.

Default value: `false`

`log_dbms_debug` (boolean)

Whether the database subsystem should log informational messages.

Default value: `false`

`log_dbms_anomaly` (boolean)

Whether the database subsystem should log information about certain situations, e.g. another part of sysorb requesting infomation about an object which does not exist in the database. Please note that there can be perfectly valid reasons for such queries, for instance the NetCheck engine requesting information about a NetCheck wich has just been deleted by a user.

Default value: `false`

`log_uplink_debug` (boolean)

Whether the uplink (rental license and problem reporting) subsystem should log informational messages.

Default value: `false`

`log_tsdb_debug` (boolean)

Whether the time series database subsystem should log informational messages.

Default value: `false`

`log_socketio_debug` (boolean)

Whether the network connection manager should log informational messages.

Default value: `false`

`log_keygen_debug` (boolean)

This option enables informational messages regarding Diffie-Hellman prime/generator pair generation.

Default value: `false`

`log_snmp_debug` (boolean)

Whether the SNMP subsystem should log informational messages.

Default value: `false`

`log_esxi_debug` (boolean)

Whether the ESXi subsystem should log informational messages.

default value: `false`

`log_snmpval_debug` (boolean)

Whether the SNMP subsystem should log informational messages about the values it receive.

Default value: `false`

`log_general_debug` (boolean)

Whether general debugging information should be logged.

Default value: `false`

`log_sched_debug` (boolean)

Whether the event-scheduler should log informational messages. These messages are usually not very interesting.

Default value: `false`

`log_alarm_debug` (boolean)

Whether the central decision-making process (the alarm logic) should log informational messages.

Default value: `false`

`log_alert_debug` (boolean)

Whether the alert dispatcher should log informational messages, which can be used for troubleshooting mail-server or modem dialing problems. Set this to "true" if you have problems getting alerts via e-mail or numerical pager.

Default value: `false`

`log_netcheck_debug` (boolean)

This option enables informational messages from the NetCheck module.

Default value: `false`

`log_icmpcheck_debug` (boolean)

This option enables informational messages from the ICMP (ping) module.

Default value: `false`

`log_ssl_debug` (boolean)

This option enables informational regarding the loading of the SSL library used for SSL based NetChecks..

Default value: `false`

`log_report_debug` (boolean)

This option enables informational messages from the Report module.

Default value: `false`

`log_forecast_debug` (boolean)

This option enables informational messages from the Forecast module.

Default value: `false`

`log_upgrade_debug` (boolean)

This option enables informational messages regarding auto upgrade of the agents.

Default value: `false`

`log_scan_debug` (boolean)

This option enables logging when the SysOrb Server receives scan requests from agent machines.

Default value: `false`

`log_nodeclass_debug` (boolean)

This option enables logging about the NodeClass check confiugration engine.

Default value: `false`

`log_router_debug` (boolean)

This option enables informational messages regarding grid routing.

Default value: `false`

`log_link_debug` (boolean)

This option enables informational messages regarding grid link establishment.

Default value: `false`

`log_station_debug` (boolean)

This option enables informational messages regarding grid station to station communication.

Default value: `false`

`log_gridrpc_debug` (boolean)

This option enables informational messages regarding the grid RPC communication.

Default value: `false`

`log_gridsync_debug` (boolean)

This option enables informational messages regarding grid synchronisation.

Default value: `false`

5.3.2. Database Options

`layout_conf` (filename)

This is the file from which the database back-end reads the layout information which the Web-interface needs.

Default value (Unix): `/var/lib/sysorb/layout.conf`

Default value (Windows): *install-dir*\Config\layout.conf

dbms_tsdb (filename)

This is the file in which the database back-end will store the time-series database data. This file can become rather large if many devices are monitored. The file grows with approximately 1 megabyte per monitored check.

> **Linux-based systems note:** You can specify either a file on your filesystem, or a device-file for a block-device (such as a disk partition or a RAID device) here.

Default value (Unix): /var/lib/sysorb/main.tsdb

Default value (Windows): *install-dir*\Config\main.tsdb

dbms_odb (filename)

This is the file that keeps all configuration information about hosts, services, devices, users, groups, etc. It is the entire hierarchical configuration of your monitored systems and their users. This file also keeps passwords for user accounts, so you should make sure that its permissions are set correctly if you run the SysOrb Server on a multiuser system with other interactive users.

Default value (Unix): /var/lib/sysorb/main.odb

Default value (Windows): *install-dir*\Config\main.odb

dbms_odbj (filename)

This is the journal file for the aforementioned main.odb meta-data database file. The journal provides an efficient method for the SysOrb Server to guarantee data consistency in the meta database. The default size of this file is 8 megabytes, and the size will never change during the lifetime of the database.

Default value (Unix): /var/lib/sysorb/main.odbj

Default value (Windows): *install-dir*\Config\main.odbj

> **Performance tip:** The journal can be placed on a small but fast storage device, in order to improve overall meta database performance. With the modest size of the journal, it is even possible to place it on a battery backed SDRAM device.

dbms_buffer_blocks (number of blocks)

This option specifies how many ODB blocks (default block size is 8 KiB) the **sysorb-dbms** process will keep in memory, to speed up normal database queries and other common operations. The larger the value, the more memory the database process will consume. The memory overhead can be estimated by multiplying the number of blocks with the block size - for example, 4096 blocks will consume approximately 4096*8 KiB = 32 megabytes of memory.

Default value: 4096

tsdb_buffer_entries (number of entries)

This option specifies how many data records (per check) there can be in the write buffer allocated allocated for each check. Enlarging this buffer will give the time series more freedom in choosing when to flush data to disk, possibly increasing performance. The cost of an increase is a larger memory footprint of the **sysorb-tsdb** process.

Default value: 64

`tsdb_buffer_write_min` (number of entries)

This option controls how many entries we want in the entry write buffer before we start flushing entries to disk. The larger this number, the fewer writes to disk will be needed. It is, however, advised that this number stays well below the `tsdb_buffer_entries` parameter, so that the write buffers do not fill up. If a write buffer is filled, flushing of the buffer will be forced, thereby limiting the freedom the database has to choose a convenient time to flush the data.

Default value: one quarter of `tsdb_buffer_entries`

`tsdb_buffer_age_max` (seconds)

This is the maximum allowed age (measured in seconds) of an entry in the write buffer. If an entry older than this is found in a write buffer, entries from that buffer will be flushed (no matter if `tsdb_buffer_write_min` is exceeded or not. The larger this option is, the more freedom the database will have in choosing when to flush data. However, if the SysOrb Server machine loses power or is in some other way shut down without giving **sysorb-tsdb** a chance to flush the write buffers from memory to disk, all records in the write buffers will be lost. This option is used to limit the maximum amount of data loss.

Default value: `600`

`tsdb_buffer_flush_batch` (number of buffers)

The time series database will go through its live write-buffers in a round-robin fashion. Each second, it will consider `tsdb_buffer_flush_batch` buffers, and inspect whether each inspected buffer should be flushed (either because of `tsdb_buffer_age_max` or `tsdb_buffer_write_min`). This option controls how many buffers are inspected each second. The higher the number, the greater the potential peak workload on the database. If this number is low, compared to the number of checks in the system, it can however take a long time for the database to actually discover that a buffer holds entries that are too old and should be flushed.

Default value: `8`

5.3.3. Network Connection Manager Options

`socketio_bind_port` (integer)

This is the TCP port on which the Connection Manager will listen for incoming connections. The standard SysOrb port is 3241, and there should be no reason to change this. If you must use another port, make sure that you change the port number both in the server, the agents and the web interface configuration files.

Default value: `3241`

`socketio_max_connections` (integer)

This is the maximum number of concurrent connections the connection manager will accept. The number should be sufficiently low not to overload the server, and sufficiently high to allow service of both agents and user interface. Both agent and user interface connections are only active when needed, so this number can be less than the number of agents and active users on the SysOrb Monitoring System. If you are seeing many messages in the SysOrb Server log about connections being expired because high or low-water pressure, and your SysOrb Server can handle the higher load, you should raise this number.

Default value: `300`

`socketio_max_inflight` (integer)

This is the maximum number (in thousands) of check-in results the connection manager will accept holding in memory, before it starts refusing agent connections. The purpose of this limit, is to prevent "check-in storms" - a situation in which a large number of agents simultaneously check in a very large number of results, causing the server to slow down, in turn causing even more data to queue up. By limiting the number of results held in memory and asking the agents to "check back later", the server can limit the load and prevent the overload.

If this number is set too low, you may see the server refusing agent checkins even though it is not too heavily loaded.

Default value: `10` (thousands)

`socketio_agentconf_rate` (integer)

This option defines how often a single SysOrb agent may request configuration from the SysOrb server. It is given as a number of seconds. Under normal operation, an Agent will only request this information when re-started. If this limit is reached it usually points towards a problem with an agent restarting over and over. There should be no reason to increase this limit.

Default value: `600`

`socketio_agentconf_burst` (integer)

This option defines how often a single SysOrb agent may request configuration from the SysOrb server. When an agent has requested information this many times in a row, the above rate limit will be enforced.

Default value: `5`

`agent_package_path` (directory)

This option specifies the directory where the Network Connection Manager should look for packages for the agent autoupgrade. The directory is automatically scanned once in a while.

Default value (Unix): `/var/lib/sysorb/upgrade`

Default value (Windows): *`install-dir`*`\Upgrade`

5.3.4. NetCheck Manager Options

`netcheck_limit_children` (integer)

This is the number of simultaneous TCP-based NetChecks that can run at any given time. Checking mail-server or web-server response times, or the response time of any other TCP based service, will cause a check program to run. This number limits how many check programs can run at any given time. The default value should be sufficient for a very large number of NetCheck services, as most checks only run for a few milliseconds each. Setting this value too low will cause checks to be queued but not lost. Setting it too high will consume more system memory and possibly slow down the entire system thereby reducing the accuracy of the NetChecks. It is better to leave this value low than set it too high.

Note: Windows has a hard limit of 30 children, and it is not recommended setting the value above 25.

Default value (Unix): `32`

Default value (Windows): `20`

`netcheck_process_pool_size` (integer)

This option controls the size of the netcheck process pool that SysOrb will spawn. Network checks to be performed are scheduled amongst the processes in the pool in a round-robin fashion. The default value of 0 means that SysOrb will attempt to autodetect a sane value and most users should not need to overrule this.

Default value : `0 (autodetect)`

`custom_netcheck_conf` (filename)

This option points to the configuration file for the custom NetChecks. An example of how such a file should be written can be seen in the custom_netcheck.conf.sample in the same directory as the main configuration file.

`netcheck_ping_spacing` (integer)

This option defines the minimal spacing between ICMP ECHO REQUEST packets in microseconds, increase this if your network equipment drops ICMP packets because of buffer overruns.

Default value: `10000` Î¼s

`dns_cache_secs` (integer)

The maximal time SysOrb will cache the result of DNS lookups in seconds.

Default value: `86400` s

`libssl` (filename)

In the event that the NetCheck engine cannot locate the SSL-library, the excact path to the library can be specified using the libssl option. If the libssl option is set, the NetCheck engine will only try to load the exact specified library. Otherwise it will try a list of possible names for the library, and use the first one that loads.

Default value: `not set`

> **Note:** This option is not available on Windows, as the SSL library is distributed with the SysOrb Server package.

5.3.5. General SysOrb Server Options

`working_directory` (pathname)

This is the directory that will be used as working directory by the server once it is running. It does not matter much what this is set to, if all other filenames use absolute paths (which they should). On UN*X systems and Linux this option lets you set the working directory to somewhere outside of e.g. `/home` thereby allowing you to unmount `/home` without shutting down the SysOrb Server. On Microsoft Windows this option has no other effect than providing a base path for relative filenames.

Default value (Unix): `/var/lib/sysorb`

Default value (Windows): *`install-dir`*

`license_path` (pathname)

The SysOrb server will search this directory for the `sysorb.lic` license file.

Default value (most Unix'es): `/etc/sysorb`

Default value (Windows): *`install-dir`*`\Config`

`variable_file` (filename, Unix only)

This file is used for storing various non-critical values. The `sysorb` user must have write access to this file.

> **Windows note:** This option does not exist on Windows, because the information is stored in the sub-key `Variables` in the registry under `SOServer`.

Default value (most Unix'es): `/var/lib/sysorb/variables.server`

`smem_prefix` (path and filename prefix)

This location prefix is used when building up the absolute filenames of files used internally by SysOrb to set up communications between the various processes that make up the SysOrb server.

Default value (most Unix'es): `/var/lib/sysorb/smem`

Default value (Windows): `SMem`

`smem_size` (size in megabytes)

Total size of the shared memory region used by the SysOrb processes for intercommunication.

A too small value will cause excessive communication overhead on a busy server, while a too large value will waste too much memory and may exceed operating system or architecture limitations on memory mapping sizes.

Default value: `128`

`resource_directory` (pathname)

The resource directory is the directory where pictures and css-files used for email-alerts and -reports are kept.

Default value (most Unix'es): `/var/lib/sysorb/resources`

Default value (Windows): *`install-dir`*`\Resources`

`upload_serverinfo` (boolean)

This option controls whether this SysOrb server should send important information about its health to Evalesco A/S. If enabled the SysOrb server will try uploading the information using the HTTP protocol.

Default value: `false`

`allow_import_export` (boolean)

In order to export or import sensitive data like passwords and agent keys you need to enable this option. This option should _only_ be set to true during import or export. When enabled the system in vulnerable to password and agent key sniffing among other things..

Default value: `false`

`max_scan_processes` (integer)

The maximum number of parallel jobs to run during AutoDiscovery.

Default value: `8`

`allowed_candidate_nodes` (integer)

When a sysorb agent checks in to the server, but is not already configured on the server, a candidate node is created. This candidate node can be viewed in the web interface and used to create a real node. This option allows you to limit the number of candidate nodes that can be on the server at one time. Futher agents will be rejected without notice.

Default value: `10`

`spare_keypool_size` (integer)

The SysOrb server will try to have this number of Diffie-Hellman prime/generator pairs precalculated for future agents checking in.

Only if you have many agents registering with the SysOrb server in quick succession will it make sense to increate this value.

Default value: `5`

`keypool_reuse` (integer)

The SysOrb server will use a Diffie-Hellman prime/generator pair for negotiating secret keys with this many agents before discarding it.

Only if you have many agents registering with the SysOrb server in quick succession will it make sense to increase this value.

Default value: `1`

`passive` (boolean)

If this option is enabled, the SysOrb server will perform no NetChecks, generate no reports, not do AutoDiscovery, or take any other action. It will still accept incoming connection from the web interface and from SysOrb agents. This option is mostly useful for recovery, if one has accidentally started an AutoDiscovery, SNMP scan or the like, that turns out to slow down or crash the SysOrb server.

Default value: `false`

5.3.6. Alert Dispatcher Options

`mail_server` (hostname)

This is the name of the mail-server (a server accepting SMTP connections, and willing to relay for the SysOrb Server). It should be set to the name of a local mail relay on your site.

Default value: `localhost`

`mail_from` (string)

This is the domain name that will be used in the **HELO mail_from** SMTP negotiation with the mail server, when an alert is sent via e-mail.

Default value: `localdomain`

`reply_mail_to` (string)

This is the e-mail address that will be put in the **From:** field of all e-mails sent by the SysOrb Server. It should be a valid e-mail address of the SysOrb Server administrator in the form of: `user` or `user@host`.

Default value: `root@localhost.localdomain`

`alert_sms_subject` (string)

Setting this option to a value, will make all SMS's sent via email to SMS gateways have the same subject. This can be useful, if the email to SMS gateway requires that the email has identification in the subject.

> **Note:** Even with this option set, it is still possible to override the subject individually by "Forcing" the subject on a specific path, through the web-interface.

`alert_defer_at_boot` (minutes)

If the SysOrb Server has a lot of agents configured, it sometimes sends out premature alerts, for agent checkin etc. In order to prevent this, the alert_defer_at_boot can specify for how long in minutes the SysOrb Server will defer sending out alerts, when it is started.

Default value: `2`

`alert_retry_delay` (seconds)

If an alert transmission fails, this is the delay in seconds that the SysOrb Alert module will wait until it re-tries the transmission.

Default value: `30`

`alert_attempts` (integer)

This option specifies the maximum number of times the SysOrb Alert module will attempt the transmission of a single alert message.

Default value: `5`

`alert_to_incident_log` (boolean)

This option controls wether SysOrb will log alert path usage to the nodes incident log.

Default value: `false`

`agent_checkin_delay` (integer)

This option controls when agent checkin reached ALERT level.

Default value: `30`

`webinterface_url` (string)

This is the URL for the Web-interface that will be put in the alert messages, for the user to click on. It should point on the preferred Web-interface for the server.

Default value: `http://localhost/sysorb`

`modem_port` (filename)

This is the name of the modem device on your system. On Unix-like platforms, please make sure that the `sysorb` user has both read and write access to the device pointed to by this option, as the server will otherwise be unable to use the modem.

Default value (Unix): `/dev/modem`

Default value (Windows): `COM1`

`modem_options` (string)

This comma separated list specifies the capabilities of the attached modem. The possible elements are

page

The modem is able to dial a numerical pager. (Any modem should have this capability).

sms

Indicates that the modem supports the GSM 07.07 and GSM 07.05 standards.

Default value: `page`

`modem_init` (string)

This string is sent to the modem before a number is dialed.

Default value: `ATE0 S7=20 S8=2`

`modem_sim_pin` (string)

If you use the Siemens M20 GSM box for sending SMS messages directly to your cellular phone, you must supply the PIN code for the SIM card here, if one is needed for the SIM card to work.

`modem_baud_rate` (integer)

Use this option to set the baud-rate of the serial port SysOrb uses to communicator with the modem. The allowed values are 200, 300, 600, 1200, 2400, 3800, 9600, 19200, 38400.

Default value: `19200`

`modem_pager_timeout` (seconds)

Use this option to set the maximal time to elapse between the ATD and ATH commands, when SysOrb is dialing a numerical pager (or an analog phone.)

Default value: `30`

`agent_checkin_delay` (seconds)

Allows the user to define how much time SysOrb server gives an agent before SysOrb server generate "late for checkin" alert

Default value: `30`

> **Note:** You might experience "false" agent late for checkin alarms e.g. during the night. These alerts go away after a few minutes however it is annoying for a person on night call to get woken up, just to see a clean SysOrb alert list with no alerts.

> **Cause:** These alarms are not a result of buggy SysOrb, but a natural result of e.g. a network under heavy pressure. It might be the case that a lot of backup jobs are running and this is taking up all the bandwidth in the network. This can prevent the agent from checking in to the SysOrb server in time. Once there is enough available bandwidth the SysOrb agent will check in again.

5.3.7. SNMP Options

`max_snmp_table_entries` (integer)

If SysOrb sees more that this number of entries in one SNMP table during a scan, the rest will be ignored.

You may increase this value if any of your SNMP devices has huge tables. Please note that the SysOrb web interface may navigate slowly in such cases.

Default value: `500`

`snmp_max_retrans` (integer)

The number of retransmits that will be done if no response is received for SNMP probes during normal operation.

Normally you should never need to change this value, but if your network or the probed device is frequently dropping UDP packets it may be necessary to increase this value slightly.

Default value: `3`

`snmp_max_retrans_walk` (integer)

The number of retransmits that will be done if no response is received for SNMP probes during a SNMP walk of a device.

Normally you should never need to change this value, but if your network or the probed device is frequently dropping UDP packets it may be necessary to increase this value slightly.

Default value: `5`

`snmp_timeout_initial` (integer)

SNMP probe packets are timed out and retransmitted if they do not receive a timely reply (number of retransmits are controlled by 'snmp_max_retrans' and 'snmp_max_retrans_walk'). For each retransmit the timeout is doubled, up to a maximum ceiling specified by 'snmp_timeout_max'. This value specifies the initial timeout for the first packet, before retransmit, in milliseconds.

You should only ever raise this value if you experience that SNMP probes often fail and you suspect this to be due to dropped UDP packets on the network rather than a problem with the probed device.

Default value: `500`

`snmp_timeout_max` (integer)

SNMP probe packets are timed out and retransmitted if they do not receive a timely reply. For each retransmit the timeout is doubled, up to a maximum ceiling specified by this parameter. This value is in milliseconds.

You should only ever raise this value if you experience that SNMP probes often fail and you suspect this to be due to dropped UDP packets on the network rather than a problem with the probed device.

Default value:

`snmp_packet_spacing` (integer)

This option defines the minimal spacing between SNMP packets in microseconds, increase this if your network equipment drops SNMP packets because of buffer overruns.

Default value: `10000` Î¼s

5.3.8. Grid Options

`ssec_keylength` (bits)

This SysOrb station (server) will generate RSA station keys with this many bits.

Default value: `1024`

`ssec_min_keylength` (bits)

This SysOrb station will require RSA keys from other stations to be at least this many bits.

Default value: `1024`

`ssec_secret_keylength` (bits)

This SysOrb station will generate AES session keys with this many bits

Default value: `256`

`ssec_secret_min_keylength` (bits)

This SysOrb station will require AES session keys from other stations to be at least this many bits.

Default value: `128`

`ssec_secret_lifetime` (seconds)

Determines how often session keys should be replaced

Default value: `600`

`grid_default_ttl` (integer)

Default TTL for grid messages

Default value: `8`

`master_notify_queue` (integer)

The maximum number of pending updates that a SysOrb master will keep track of. If the satellite fails to acknowledge them, the master will kick the satellite.

Default value: `8192`

`satellite_tsdb_async` (integer)

The maximum number of timeseries being propagated from the satellite at one time

Default value: `32`

5.3.9. WebAPI Options

`wapi_enabled` (boolean)

This option enables WebAPI service

Default value: `false`

`wapi_port` (integer)

This option defines a port WebAPI uses to serve requests.

Default value: `8088`

`wapi_use_ssl` (boolean)

This option tells WebAPI to use ssl.

Default value: `false`

`wapi_ssl_cert_path`(string)

This option defines path to ssl certificate files.

`wapi_threads` (integer)

This option defines the number of threads to serve user requests in parallel.

Default value: `1`

5.4. Troubleshooting HTTPS/IMAPS/POP3S problems

The SysOrb Servers SSL support uses the OpenSSL libraries. On all platforms except Windows, it uses the version of OpenSSL that ships with the operating system. This chapter describes how to troubleshoot SSL check problems on the Unix-like platforms.

The most common problem is that the check shows "No Connection" even though the service is accessible. It usually happens because the OpenSSL library is not installed on the host OS. If this is the case, you will be seeing "No connection" errors in the SysOrb Web Interface, and the server log will contain the following message:

```
Could not load SSL-library. Please check the administrators guide for information
about how to fix this problem
```

The first thing to check is that the OpenSSL library is installed on the host OS. How to ensure this can be seen in the list below:

Most Linux and FreeBSD

On the platforms that aren't mentioned below, the SysOrb Server package depends on the OpenSSL package being installed. So if the SysOrb Server package can be installed, the OpenSSL library is also installed.

Debian

`libssl1.0.0`, or newer package should be installed. You can check this by executing the following command:

dpkg -l 'libssl1*'

One of the output lines should start with

`ii`
and mention one of the package names listed above.

If this is not the case, you should execute the following command to find the newest version of libssl that is available for your system:

apt-cache search libssl1

Then you can install the package by executing:

apt-get install <package name>

Solaris

On Solaris 11 OpenSSL is provided as a component of the operating system.

It is not necessary to restart the SysOrb Server, if the OpenSSL library was installed while the SysOrb Server was running.

If the OpenSSL library is installed, and SysOrb still cannot find the OpenSSL library, you can specify the exact location of libssl.so (or similar named), by using the `libssl` server configuration option. See Section 5.3.4 for information about this option.

If you instead get a message stating

```
Error loading library
```

or

```
Error dynamically loading function
```

it means that your version of OpenSSL is incompatible with SysOrb. In this case you should either upgrade to a newer version provided by your OS provider, or download the source code from http://openssl.org and compile it

by hand. Instructions on how to do this can be found on the OpenSSL homepage. Note that after the installation, it might be necessary to use the `libssl` to get the SysOrb Server to use the correct version of openssl.

5.5. On the fly debug logging

SysOrb has the ability to add debug logging on the fly without having to restart the service. The extra debug information will be added in the "server.log" file.

To add debug logging, simply create a new empty file in the log director, naming it according to the information you want to add to the serverlog. It is important that the naming of the files, does not include any extensions. Once they have been placed in the correct directory, the logging should start to add data to the server.log

Default settings for where log files are placed, and where the log check-files should be added, are:

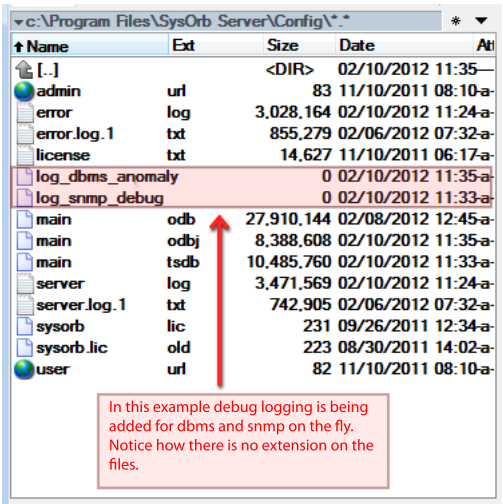
- **Default log directory (Unix)**: */var/log/sysorb*
- **Default log directory (Windows)**: *install-dir\Config*

To avoid that you get huge log files, you should only add debug logging when needed. Once you do not need the debug logging, you simply delete the files again, and the logging will stop.

Below is a list of filenames that can be used to add the debug logging.

Look in Section 5.3.1 to get more detailed information about the debug logging options.

log_dbms_debug
log_tsdb_debug
log_dbms_anomaly
log_socketio_debug
log_keygen_debug
log_snmp_debug
log_snmpval_debug
log_general_debug
log_sched_debug
log_alarm_debug
log_alert_debug
log_broadcast_debug
log_brdc_frag_debug
log_time_debug
log_netcheck_debug
log_icmpcheck_debug
log_ssl_debug
log_report_debug
log_forecast_debug
log_upgrade_debug
log_scan_debug
log_nodeclass_debug
log_router_debug
log_link_debug
log_station_debug
log_gridrpc_debug
log_gridsync_debug
log_uplink_debug



Example: An example on how on the fly debugging is set up.

Chapter 6. SysOrb Web Interface Configuration

The Web Interface needs to know how it can contact the SysOrb Server. Note that the Web Interface can run on any system with network access to the SysOrb Server system and that it does not have to actually run on the same physical machine, although this is by far the most common configuration.

The configuration of the SysOrb Web Interface is stored in `/etc/sysorb/cgi.conf` on all Unix-like systems, except FreeBSD where the file is stored in `/usr/local/etc/sysorb/cgi.conf`, and Mandrake where the file is located in `/etc/opt/sysorb-server/cgi.conf`. On Microsoft Windows configuration is performed by the SysOrb Config program found in the start menu.

6.1. General Web Interface Options

`server_list` (list of hosts)

The list is a comma-separated list of hostnames, **without whitespace**. The web interface first tries all the hosts in the list, and then only displays those that answered. If the servers are running on different ports, this can also be specified by adding `:portno` after the hostname. If no portnumber is given the default is `server_port`.

Default value: `localhost`

`server_port` (integer)

This is the port number on which the SysOrb Server listens for incoming connections. The value should only be changed if the server has been told to use another port number as well. It should not be necessary to change these port numbers, but it is nonetheless possible if needed.

Default value: `3241`

`default_domain` (domain-identifier)

This option specifies which domain should be presented in the domain box on the login page. If this option is unspecified it defaults to the empty string, which is also the root domain.

Default value: `.` (dot)

Chapter 7. SysOrb Web Server Configuration

On Windows it is possible to install a Web Server with a small feature set, that is customized for use with the SysOrb Web Interface. It is provided as an alternative to installing IIS, but for the larger installations the use of IIS is still recommended, because if its larger featureset.

The SysOrb Web Server has few configuration options, and most should be set to reasonable values by the installation program.

7.1. Web Server Configuration Reference

The following sections list the available configuration options for the Web Server. Note that usually the Web Server is ready for use after the installation.

7.1.1. File Options

`html_dir` (directory)

This option specifies the directory from which the static files should be served.

Default value: *`install-dir`*`\www`

`cgi_dir` (directory)

This options specifies the directory from which the CGI-programs should be served.

Default value: *`install-dir`*`\www`

`cgi_ext` (file extension)

The file extension used to determine if a request is for a CGI-program or for a file.

Default value: `cgi`

`index_file` (file name)

The name of the file that should be served if no file is specified.

Default value: `index.cgi`

7.1.2. Logging

`log_file` (file name)

The name of the file that the Web Server should log to.

Default value: *`install-dir`*`\error.log`

`tick_time` (seconds)

How often the Web Server should write "MARK" messages to the log file. Use 0 to disable writing of MARK messages

Default value: 0

`log_level` (integer)

The amount of logging that is written to the log file. Write 0, to disable logging, 3 to only log warning messages, 7 to log warning and informational messages, and 15 to log everything including debug messages.

Default value: 7

`log_rotate_time` (seconds)

How often the log-file is rotated, in seconds.

*Default value:*604800 (7 days)

7.1.3. Server/network options

`port` (integer)

Use this option to set the port number through which the Web Server should be available.

*Default value:*Determined during the installation

`server_string` (string)

The server identification string. This string is sent whenever a page is requested from the Web Server. Change this option to make the SysOrb Web Server present itself differently.

Default value: SysOrb Httpd v.4.6.0 build 6039 on Windows NT (IA-32)

`max_connections` (integer)

The maximum number of open connections to the Web Server. When this number of connections is reached, new connections will not be accepted.

Default value: 32

`time_out` (seconds)

The number of seconds a connection is allowed to be idle, before the Web Server automatically closes it.

Default value: 60

Chapter 8. SysOrb Agent Configuration

The agent has few configuration options. It only needs to know about a few simple things, such as how to contact the SysOrb Server. The real configuration of what checks to perform, when to check in and so on is configured on the SysOrb Server, and migrated to the agent on demand.

The agent configuration is stored in the file `/etc/sysorb/agent.conf` on all Unix-like systems. On Microsoft Windows systems this information is stored in the system registry, and is made accessible through the **SysOrb Config** program installed with the SysOrb Agent.

The agent must be restarted in order to reread the configuration. However, reconfiguration of the agent options is almost never needed after the agent is set up the first time.

On Unix systems a hangup signal (`SIGHUP`) can be sent to the agent to re-open the log file. This is used by the logrotate script on Linux, and can be employed similarly on FreeBSD and Solaris. On Windows the log-rotation is handled by the agent.

8.1. Agent Configuration Reference

Most of the options in the SysOrb Agent configuration are specific to the site where the SysOrb runs and, as such, no appropriate default values can be given. When installing the SysOrb Agent remember to check that the configuration options matches the environment in which the SysOrb Agent works.

The first three options need to be changed on almost all new installations.

`name` (hostname)

This is the name the agent will use to identify itself to the server. This must be the exact same name as a host created on the SysOrb Server.

`domain` (domain-identifier)

This is the domain the agent uses to identify itself to the server with. This must be the exact same domain as the agent was configured with in the server.

The following options rarely needs to be changed:

`server_port` (integer)

The port number on which the SysOrb Server will be listening for incoming connections. This number should not be changed, unless there is some special reason for using another port.

Default value: `3241`

`logfile` (filename)

This is the name of the logfile the agent will use to write informational messages and errors in. You should keep the default setting unless you want to place the logfile somewhere else.

Default value (Unix): `/var/log/sysorb/agent.log`

Default value (Windows): *`install-dir`*`\Config\agent.log`

`variable_file` (filename, Unix only)

This is the name of the variable file, which the agent uses to store non-critical values. The `sysorb` user must have write access to this file.

> **Windows note:** This option does not exist for Windows, because all the information is stored in the registry.

Default value (Unix): `/var/lib/sysorb/variables.agent`

`log_level` (0-3)

This is the logging level to be used by the agent. The higher the number, the less the agent will log. The default is `1`, where most uncommon messages are logged, as well as all errors. For troubleshooting purposes, you may wish to set the log level to `0`, where the agent will log a lot of routine events.

Default value: `1`

`log_days` (integer)

This option tells the agent how many days should pass before it changes logfile. The old logfile will be renamed to *`logfile`*`.X`, where `X` is increasing as the files get older, and *`logfile`* is the value specified in the logfile parameter. If set to 0, the logfile is never changed by the agent.

> **Unix note:** Many Unixes have a build in tool, which handles the change of logfiles. If your Unix does not have such a tool, you should enable log_days

Default value (Windows): `7`

Default value (Unix): `0`

`max_log_files` (integer)

This option determines how many changed logfiles will be kept.

Default value (Windows): `5`

Default value (Unix): `0`

`allow_autoupgrade` (boolean)

The agent will only attempt to download and install a new version of itself if this flag is set to true. See Chapter 4 for further explanation.

Default value: `true`

`use_psapi` (boolean, Windows only)

Choose which way the SysOrb Agent retrieves the list of running processes. The default `false` uses the same way as the Windows Task Manager, and should work on all machines. However if you experience trouble with the process reporting, try switching this on.

> **Warning:** Changing this option will change the name of some of the running processes, since the two different way, also uses different naming. So when changing this option, the agent must be rescanned and it should be made certain that all the process presence checks still work.

Default value (Windows): `false`

`perf_cnt_level` (string, Windows only)

This option changes the detail level of the performance counters reported to the SysOrb Server. There are four different levels: `Novice`, `Advanced`, `Expert` and `Wizard`. Changing this from the default of `Advanced` will change the number of performance counters that can be checked. The `Novice` level will return the lowest number of performance counters, and the `Wizard`-level will return the highest.

Note: After this option has been changed, the agent must be rescanned from the Web-interface in order for the new performance counters to show up.

Default value (Windows): `Advanced`

`max_data_retention_days` (integer)

If the agent is unable to contact the server, it will still generate monitoring data and save it. If the server is unavailable for a longer time, the agent can end up using large amounts of memory. Use this option to set how many days should pass without server connection before the agent starts discarding the monitoring data. Set to 0 or negative value to disable discarding of monitoring data.

Default value: 7

`custom_chk_conf` (filename)

If this option is set, the agent will try to read the custom check configuration file specified by the option. For information about the custom check file format, see Section 8.3.

Default value: (none)

`actions_conf` (filename)

If this option is set, the agent will try to read the custom AgentAction configuration file specified by the option. For information about the custom AgentAction file format, see Section 8.4.

Default value: (none)

`log_chk_conf` (filename)

If this option is set, the agent will try to read the log check configuration file specified by the option. For information about the log check file format, see Section 8.6.

Default value: (none)

8.2. Releasing keys to force registering at SysOrb server

If you for some reason would like an agent to check-in to another server than the one it already does, you have to release the authentication key for the agent. Both the server and the agent has the key, which is used for security purposes.

8.2.1. Releasing on agent

To release the agent key on a Unix-like system, you must delete you the variables file while the SysOrb agent daemon is stopped. The location of the variables file is configured in the agent.conf file on the host where the agent runs (ususally `/var/lib/sysorb/variables.agent`).

To release the key on a windows open the SysOrb configuration utility located in *Start menu->Programs->SysOrb->SysOrb configuration*. In the configuration tree choose *SysOrb agent->General->Release Key*. In the dialog on the right side of the configuration dialog click the *Release Key* button.

To release the agent key on a NetWare system, you must delete you the variables file while the SysOrb agent NLM is unloaded. The location of the variables file is configured in the AGENT.CFG file on the host where the agent runs (ususally `\SYSORB\AGENT.VAR`).

8.2.2. Releasing on server

You can also release the key on the server to force the server and agent to make a new key.

To release the key on server, do the following in the SysOrb web interface:

- Click the Configure button in the navigation menu.
- Find the host you want to have release the key for in the configuration tree.
- Click edit on the edit link associated with the host.
- Select Edit Node.
- If the server and agent has negotiated a key you will be able to find a Agent key label. Click on the Release key on the right to release the key on the server. If the label is not present, it is because the server does not have a key for the host in question.

8.3. Configuration of custom checks

If the built-in checks of the SysOrb Agent does not suffice for you particular setup, you may write extension scripts that can perform the check.

You can write these scripts in any language you desire. The only requirement is that the script must deliver the result of the check to the SysOrb Agent in one of two ways.

- If the check is either good or bad, the script must tell the SysOrb agent by the exit code. A non-zero exit code indicates a bad result, while an exit code of zero indicates a good result. How exit with specific exit codes depends on the script language used.
- If the check can return a numeric value within some range, it must output that value decimally to standard output on a line like: "Result: 1234". Note that the colon must be present, and that the Agent is case sensitive. The SysOrb Agent will parse the output from the script, putting any line not conforming to the above format into the agent log file.

The custom checks that an Agent is able to execute is configured using a file on the machine running the SysOrb Agent. It may have been more convenient to be able to do it from the web interface, but that would open serious security risks, as any SysOrb user, with capabilities to configure checks, would be able to have the agent execute arbitrary shell commands.

In order to use custom checks you must first tell the SysOrb Agent where to find the custom check configuration file. On non-windows systems this is done by adding a line like

```
custom_chk_conf=/etc/sysorb/custom.conf
```

to the agent configuration file (usually located in `/etc/sysorb/agent.conf`). On Windows you must open the SysOrb Configuration utility located in *Start menu->Programs->SysOrb->SysOrb Configuration*. In the configuration tree choose *SysOrb Agent->Configuration files->Custom checks* and enter the path where you are going to put the new configuration file.

After changing this path, the Agent must be restarted, but wait until you have put something into the new file.

You must edit the new file `custom.conf` in order to configure which custom checks the agent can perform. The format of the file is most easily illustrated with a few examples of custom checks:

New to 4.4: it is now possible to get extended result description from boolean check: just add result_pattern string to the check config with appropriate pattern. Then this description will be displayed in the check result.

Example 8-1. Good/bad check

```
# Comment lines begin with #
[check name]
command=/path/to/the/execfile parameters
timeout=10
```

The value between [and] gives the name of the check. command is the command sent to the shell, so normal shell expansion works. The timeout parameter is optional, and the default value is 10 seconds. Please note that the SysOrb Agent stops processing other tasks (ordinary checks and communication with the server) while a custom check is carried out, so timeout shouldn't be more than half of the checkin frequency for that agent.

Example 8-2. Numeric integer check

```
[check name]
type=integer
unit=kB
command=/path/to/the/execfile parameters
timeout=10
```

The line type=integer tells the SysOrb Agent, that this is a numeric integer check, that will output a line on the form `Result: 1234`, the unit property tells the agent, that the numeric output should be interpreted as a number of kilobytes. This information is passed along to the SysOrb Server, to allow putting units on the Y-axis in graphs and elsewhere.

Example 8-3. Numeric real check

```
[check name]
type=real
unit=s
command=/path/to/the/execfile parameters
timeout=10
```

The line type=real tells the SysOrb Agent, that this is a numeric check, that will output a line on the form `Result: 12.03`, the unit property tells the agent, that the numeric output should be interpreted as a number of kilobytes. This information is passed along to the SysOrb Server, to allow putting units on the Y-axis in graphs and elsewhere.

Example 8-4. Differential numeric integer check

```
[check name]
type=diff_integer
unit=kB/s
command=/path/to/the/execfile parameters
timeout=10
```

The line type=diff_integer tells the SysOrb Agent, that this script outputs an integer like a numeric integer check. For each pair of successive invocations, the SysOrb agent will subtract the two results, and divide the difference with the elapsed time. This is useful if your script for instance outputs the number of bytes which have passed through an interface since some fixed point in time, but you want the check to show the number of bytes per second passing through right now.

Example 8-5. Differential numeric real check

```
[check name]
type=diff_integer
unit=kB/s
command=/path/to/the/execfile parameters
timeout=10
```

The line type=diff_real tells the SysOrb Agent, that this script outputs an real like a numeric real check. For each pair of successive invocations, the SysOrb agent will subtract the two results, and divide the difference with the elapsed time. This is useful if your script for instance outputs the number of bytes which have passed through an interface since some fixed point in time, but you want the check to show the number of bytes per second passing through right now.

Whenever you edit this file, you must push open a web interface and push Rescan on the AgentChecks tab on the Configure page for the host in question. You do not need to restart the SysOrb Agent, unless you have changed the location of this file.

Sometimes you do not write the script yourself, and there is no easy way of having it output its result on a line like `Result: 1234`. In that case you can instruct the SysOrb Agent to look for the result in lines containing other text. You do that by means of a POSIX regular expression. The syntax of regular expression can be found at: http://www.opengroup.org/onlinepubs/007908799/xbd/re.html

Example 8-6. Script outputting `Response time: 1234 ms`

```
[resp_time]
type=integer
command=measure-command
result_pattern=^Response time: ([[:digit:]]+) ms$
timeout=10
```

The result_pattern must contain exactly one set of parentheses enclosing the pattern matching the numerical result of the check.

Example 8-7. Count the number of files in /tmp

```
[temp_count]
type=integer
command=find /tmp -maxdepth 1 | wc -l
result_pattern=([[:digit:]]+)
timeout=10
```

This result_pattern will match any line containing numerical characters. The SysOrb Agent will then pick the first line containing numbers, and if there are more than one number on that line, the leftmost of them.

8.4. Configuration of AgentActions

Starting with SysOrb version 2.4, you can now define AgentActions, which is actions carried out on the Agent machine, when a user orders this through the Web Interface. These actions can help ease remote administration, as they can be defined to carry out common tasks on machines. An AgentAction could as an example restart a webserver, or reboot the target machine.

In order to use AgentActions you will first have to configure the Agent to read the actions from a configuration file. The name of this file is specified through the `actions_conf` option in the agent configuration (see Section 8.1 for information about how this is done).

The fileformat for the AgentAction configuration file resembles the fileformat of the Custom Check configuration file very much. The format consists of a block as following for each action:

```
# Comment lines begin with #
[action name]
command=/path/to/the/execfile parameters
```

The action name between the [and] is the name displayed to the user of the Web Interface. The `command` parameter is the command which is executed. On Unix-like systems this command is passed to the shell, so normal shell syntax can be used.

As an example, the following configuration file will create an action that will restart IIS on a Windows 2000 Server installation.

Example 8-8. IIS restart AgentAction

```
# Restart IIS.
[Restart IIS]
command=iisreset
```

Once the AgentAction configuration file is in place, you must rescan the agent before the actions appear in the Web Interface. When they appear, they will be accessible from the Node overview page, and from the AgentCheck configuration pages.

8.5. Configuration of Automated Actions - Remedy

Introduced in SysOrb 4.0, remedy is a functionality that makes it possible for SysOrb to automatically carry out an agent action. A typical situation where remedy is used, could be to automatically restart a service if it stops.

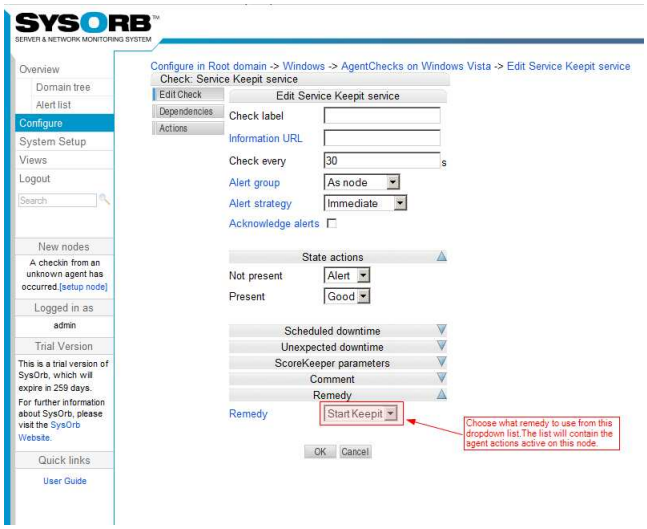
To set up a remedy, the first thing to do is to set up a Agent actions, look in Section 8.4 if you want to know more about this topic.

After this, configure the check, by choosing the action that you want to happen when the check goes into alert in the remedy dropdown. Once this is done, remedy should run as soon as the check goes into alert, triggering the action that have been selected.

Once an automatic action has been carried out by SysOrb, the remedy will be disabled for 30 minutes, which means, that if a second alert comes up, the check will go into alert state, and follow through as normal.

To come with an example: A server is monitoring a service, that HAS to run all the time. However, due to some issues on the server, the service will shut down every now and again, therefore a agent action that restarts the service have been set up. Remedy will make sure, that if the service only shuts down one time, it will automatically restart it with the agent action, however, if the service goes down a second time within 30 minutes (while remedy is disabled) it will go into alert. This is done to make sure that if the service goes down for good, and do not recover, SysOrb will tell you.

At the same time, when remedy is executed, there is automatically introduced a 5 minutes unexpected downtime for the whole node, this is done in order to avoid alarms from other checks which might arise because of the action being executed. A common situation could be: A web site starts running poorly. SysOrb remedy is set to re-starts IIS, if a site is responding above a certain limit. Restarting IIS causes all sites on the host to temporarily malfunction, but we do not want to have alarms from all sites. If restarting the web server has solved the problem, everything will be back to normal when the node comes out of downtime. The incident log on the given node will tell you when SysOrb executed the remedy.



8.6. Configuration of LogChecks

The SysOrb Agent can scan logfiles on the host machine, and report any error messages or unexpected event to the SysOrb Server, providing a quick overview of all machines on your network.

The files that the Agent can monitor this way must be specified in a configuration file on the machine running the SysOrb Agent. This is a safety measure, guarding agaist a compromised SysOrb Server being able to retrieve a copy of any file on the hosts running SysOrb Agents.

The name of the file containing the definitions of the allowed LogChecks is given in the Agent configuration option `log_chk_conf` (see Chapter 8 for more information). It is usually called `log.conf`, a sample file allowing a few system log files to be scanned is included in the SysOrb Agent package.

The file contains a number of lines that each specifies a path that is to be searched for log files to monitor. This can be a specific file path, such as `C:\logs\error.log` or `/var/log/messages`. But it can also contain the following special wildcard characters. '?' can match any character. '*' matches any character string of any length. '**' matches any directory path. Thus, the path `C:\logs\log??\**\error.log` matches `C:\logs\log01\sub\dir\error.log` and `C:\logs\log02\sub\sub\dir\error.log`, but not `C:\logs\log002\sub\dir\error.log`.

You can also specify to scan for eventlogs by setting the path o 'eventlog:*'. This will scan for all eventlogs available, but you can replace '*' with a specific event log name.

Note that for these logs to actually be monitored, you will have to configure them on the SysOrb server, setting up how often the check should be performed, along with various other parameters. (See the *User's Guide* for information on how to activate LogChecks.)

Example 8-9. LogCheck declaration file

```
/var/log/messages
/var/log/apache/virtual-domain-*/error.log
eventlog:*
```

This will allow the user to configure checks on the file `/var/log/message`, all virtual domain error logs in `/var/log/apache/virtual-domain-*/error.log` and all eventlogs.

8.7. Enabling IPMI hardware monitoring

The agent supports monitoring of hardware parameters such as *fan speeds*, *temperatures*, *voltages* etc. by means of the IPMI (Intelligent Platform Management Interface) standard on supported server and operating system combinations.

Hardware monitoring with IPMI is supported on these operating systems:

Windows Server systems prior to Windows 2003 R2

Intel provides an IPMI driver free of charge which can be used on these systems. The driver can be downloaded from the Intel driver download page. This driver is fully supported by the SysOrb Agent.

Windows Server 2003 R2 (and later)

Microsoft provides an IPMI driver as part of the operating system. It is no installed by default, but can be installed by following the instructions from the TechNet article "Enabling Hardware Management". This driver is fully supported by the SysOrb Agent.

Linux kernel 2.6 based systems

Most modern Linux distributions ship with Linux kernels from the 2.6 series, and include an OpenIPMI driver. This driver is fully supported by the SysOrb Agent.

Once the appropriate driver is installed, the SysOrb Agent can be re-scanned, and all IPMI sensors available on the given hardware provided by the operating system driver will appear in the web interface under "Agent Checks".

Chapter 9. Grid configuration

9.1. Grid overview

The SysOrb Grid functionality allows for centralized secure monitoring of remote networks. A "satellite" can be installed on a remote network, and perform NetChecks, SNMPChecks and receive Agent Check-ins on that remote LAN. The satellite can securely deliver all locally gathered check results back to a "master", the single centralized SysOrb Server.

A grid will contain *one master*, and *one or more satellites*. These are collectively called "Stations" in the grid. It must be possible, network wise, for either the master to initiate a TCP connection to the satellite, or, for the satellite to initiate a TCP connection to the master. Therefore, it is actually possible to monitor multiple LANs sharing the *same* IP ranges (for example the 10.0.1.X series), as long as the satellites can connect to the master.

If the network connectivity between a satellite and the master fails, the satellite will continue to run all configured checks on its local network. It will hold all measurements gathered in its local database, until the network connectivity comes back up. The satellite does not hold measurement data locally for any longer than what is necessary - but since it does include the full featured database already used in the common SysOrb Server product, it can spool measurement data to disk efficiently for very long periods of time, if the network outage is prolonged.

The high levels of security in the grid networking layer, combined with the autonomy of the grid satellites, allows the use of satellites for running SNMPChecks and NetChecks on remote locations, securely on the LAN, and reporting back these results in a safe manner to a central SysOrb Server (a Grid Master). Even though a VPN could also be used to secure the otherwise inherently insecure SNMPChecks and NetChecks, a VPN would not allow for continued monitoring in case of network outages, and the VPN itself would incur a significant and unpredictable latency to the checks. In other words, the grid solution with satellites allow for consolidated monitoring scenarios that are not possible to achieve using other conventional technologies, as, for example, VPNs.

9.1.1. Grid IDs - the GID

As mentioned, the IP addresses of satellites need not be unique across your SysOrb installation, and therefore cannot be used to uniquely identify a satellite in the grid. When configuring NetChecks and SNMPChecks it is possible to specify which station should execute the check - in order to accommodate this, there must be some form of unique identification of every single station in the grid.

A "Grid ID" scheme has been developed for this purpose. Grid IDs, or GIDs, are unique numbers used to identify a station in the grid. A GID consists of four two-digit hexadecimal numbers; for example "0A:3D:42:F1".

In order to accommodate future grid interoperability between existing installations, it is highly desirable that all SysOrb installations use globally unique GIDs. This is not a requirement as such, and any installation is free to use whatever GIDs it wants to - but Evalesco A/S *encourages* our users to use globally unique GIDs. In order to facilitate such a global numbering scheme, it is now possible for every customer to allocate a "GID Prefix" on the Evalesco Web site. The prefix consists of the first three numbers of the four-number GID. This prefix is guaranteed to be globally unique. Once the GID prefix is acquired, the user can freely assign the last of the four numbers - this allows for a grid with 256 stations (using numbers from 00 to FF), which should be enough for most installations. If more stations are required in a grid, more prefixes can be allocated for that customer.

If a prefix of "02:FE:C7" is assigned to a customer, that customer can use, for example, "02:FE:C7:01" as the GID for the master station, "02:FE:C7:02" as the GID for the first satellite, and so forth...

9.1.2. Links

In order to receive configuration from the master, and in order to report results back to the master, the satellite must have some form of network connectivity to the master. In SysOrb Grid terminology, we call this connectivity a

"link". A link is a network connection between a single master and a single satellite.

Links can be initiated from the master, or from the satellite, or can be configured to be initiated from any of the two. A link will actually be constituted by a TCP connection from a random port on the initiator machine to the common port 3241 on the receiver machine. If you have configured your SysOrb Server to use a port other than the default 3241, you will of course need to adjust your link configurations accordingly.

The most common setup, is to configure the links so that it is the satellites (which are often on closed networks, often using the same 10.X.X.X IP address ranges) initiate the connection to the master (which is usually located on a public IP address). The connections initiated from the satellites can be NAT'ed out thru a firewall, for example.

9.1.3. Endpoint security

A grid will commonly consist of machines located far away from each other, with links going over common insecure Internet lines. The grid networking layer will ensure *authenticity*, *integrity* and *secrecy* of all communication within the grid. It is therefore not necessary to employ VPN solutions or private protected lines connecting the stations within the grid.

Every station in the grid will have its own RSA public and private key pair. This key pair is automatically generated when a station is first set up, and it can be viewed in the web interface (under "System Setup" and "This Station"). The RSA key pair will by default be 1024 bits, but this can be configured freely between 512 bits and 8192 bits.

The first time two stations establish a link between them, they will exchange their public keys. This communication *cannot* be authenticated, as the stations share no secrets prior to this exchange. A "man-in-the-middle" attack is thus possible during this initial exchange. It is therefore advised that one verifies that the public key received from a remote station is in fact the correct public key, by comparing the remote station key (found in the web interface under "System Setup" and "Stations", selecting the given remote station) as received on the local station, with the actual station key of the remote station (found as described above, on the remote stations web interface, under "This Station").

Once it is established that the public keys have been correctly exchanged during this very first connection of the stations, security will be maintained in all future communication.

Based on the RSA keys exchanged, the stations will negotiate a symmetric key for all following communication. This symmetric key establishment is referred to as "Endpoint Security" - it is the guarantee for authenticity, integrity and secrecy of all communication between the pair of stations. Per default, the endpoint security will consist of a shared 256 bit AES key. It is possible to configure stations to use 128 or 192 bit keys instead.

9.2. Configuring the Master

Your grid master will be the station on which all day-to-day operations happen. It is the station that will send out alerts, and it is the station that will store all historical data for every check running on the grid. Except for the initial satellite setup, all configuration of every monitored node in your grid will also be done via. the web interface on the grid master.

9.2.1. Installation and licensing

The grid master functionality is included in the standard SysOrb Server. Therefore, installing a grid master is simply the installation of a normal SysOrb Server, as described earlier in this manual.

The grid master functionality is enabled in the server with a special license file. Trial licenses are offered specially, and full licenses can be purchased just like the non-grid licenses. Please contact your sales representative or Evalesco A/S for details.

9.2.2. Configuring the master station

In the web interface, under "System Setup", enter the "This Station" page. Here you can specify a descriptive name for the station (the master), and you can enter the unique GID. Please use your allocated GID prefix, and pick the last number in the GID as you wish (as long as it is unique within your installation).

Once a descriptive name and the proper GID is entered, the station as such is configured to participate in your grid. The cryptographic keys on the configuration page will be blank initially, but once you have configured the station, the system will begin to generate a proper RSA key pair. After a few minutes (depending on the speed of the system), you should be able to see the signature and encryption keys listed in the web interface.

9.2.3. Adding a satellite to the master's configuration

The master must now be made aware of a satellite. We have not yet configured the satellite itself, but we can still let the master know that one is going to exist in the future.

Under "System Setup", please select the "Stations" page. This is for configuring all remote stations, as seen from this local station. Select "Add station" to add a satellite.

The "name" field cannot be filled out, as the name of the remote station will be fetched automatically once the master and the satellite establish their connection. The GID field, however, must be filled out. This field must contain the GID of the station you will be adding. Again, please make sure that you select a unique GID for your station, preferably using your allocated GID prefix.

9.2.4. Configuring the link between the master and the satellite

In order to facilitate communication between the master and the server, we must now let the master know how it can contact, or how it can be contacted by, the satellite. This is done under "System Configuration" and the "Links" page.

Select "Add Link" to define a link between the master and the satellite. You can enter a descriptive name for this link. You must now select whether the master may accept incoming connections from the remote station over this link, and whether the master may attempt to initiate connections to the remote station over the link.

For masters, it is most common to specify that they are allowed to accept incoming connections. Masters will often not be able to connect directly to the remote satellites, as these will often be NAT'ed away behind firewalls on remote networks. If, however, your master must attempt to establish the connection to the satellite, you must specify a destination IP address (and optionally a port number if your satellite deviates from the default configuration of using port 3241).

Now, that we have configured the master, made the master aware of the satellite, and configured the master's connectivity to the satellite, all of the grid specific communication is actually complete. We can now go ahead and actually configure the satellite station that we have just made the master aware of.

9.2.5. Exporting domains to the satellite

Information can be exported to the master, on a domain-tree basis. For each domain, one can specify a list of GIDs that are allowed to "mount" said domain. A common scenario is to create a domain for each remote satellite location, and specify the corresponding satellite GID in the allowed export list for the given domain.

Please create a domain for your remote satellite location (under "Configure", select "Add domain", and add a new domain as usual). For the rest of this example, let us assume the domain is called "remote" and that it is located right under the root domain. Please be sure to specify the GID of the satellite in the export list for the domain.

The satellite will now be allowed to retrieve all node and check configurations from the domain and from all domains under this domain. The satellite will still have to be configured to actually "pull" this information from the

master, but specifying the GID of the satellite in the export list of the domain is a prerequisite to actually allowing the export of information to the given satellite.

9.3. Configuring the Satellite

Configuring the satellite is very similar to configuring the master. You must perform a number of steps here using the web interface on the satellite system. However, there will be no use for the web interface once the satellite is configured. It would therefore be wise to un-install or disable the web interface on the satellite once the configuration is done. Please also note that the satellite will have its own administrator user, with the default "admin" login name and default password. *The password on this account should be changed to prevent others from reconfiguring the satellite station!*

9.3.1. Installation and licensing

Grid Satellite functionality is also included in the standard SysOrb Server packages. The satellite functionality is enabled, like in the case of the master, with a special license file.

9.3.2. Configuring the satellite station

Like we configured the master for its participation in the grid, with a descriptive name and a unique GID, we must now configure the satellite with its parameters for grid operation.

In the web interface, under "System Setup", enter the "This Station" page. Here you can specify a descriptive name for the station (the satellite), and you can enter the unique GID. Please make sure that you use the same GID as you did when you configured this satellite previously on the master station.

9.3.3. Adding the master to the satellite's configuration

As we told the master about this satellite, we must also tell this satellite about the existence of the master. Under "System Setup", please select the "Stations" page. Select "Add station" to add the master.

The GID field must contain the GID of the station you are adding - this should be the GID of the master station.

9.3.4. Configuring the link between the satellite and the master

We have already specified on the master, how communication should commence between the satellite and the master. We must now tell the satellite how this happens as well - keeping in mind that from the satellites point of view, this will be a "reversed" configuration compared to the master. For example, if the master must only accept connections and never initiate, the satellite must be told to initiate and never accept. Like on the master, the link configuration is performed under "System Configuration" and the "Links" page.

Select "Add Link" to define a link between the satellite and the master. You can enter a descriptive name for this link. Now, enter the "reversed" configuration of what was done on the master side. Assuming the master was configured to accept and never initiate connections, you must now specify the opposite for the satellites link configuration, and you must supply an IP address for the satellite to connect to.

The satellite will expect, that when it connects to the IP address that you specify, it will find a station with the GID that you specify for this link. Please be sure that the satellite is actually allowed to initiate a TCP connection to the given port on the given IP address, and that this actually is the proper IP address of your master station.

9.3.5. Mounting domains from the master

Earlier, a domain called "remote" was created on the master station, and the GID of this satellite was added to the domain export list. We can now tell the satellite to actually mount this domain - to retrieve all configuration information under the domain, store it in the local database, and to perform any checks that might be configured on nodes under that domain.

Under "Configure", select "Add Mountpoint". Now select the GID of the master to import information from. Also specify a descriptive name for the mount - this name does not have to be the same as the name of the domain on the master. Finally, you must specify the actual domain, as named on the master, to mount. Using the example "remote" domain, you can specify ".remote" in this field, to state that you wish to import the domain named "remote" located directly under the root domain on the master.

Domains are separated with dots. If, for example, the satellite should mount a domain called "foo", under the domain "bar", under the root domain, the path would be ".bar.foo".

As soon as the satellite and the master establish their link and negotiate their end-point security, the satellite will attempt to retrieve all configuration information for the configured mountpoint. All of this is happening as background operations, and you will not actually have to wait for any of these operations to complete. You can follow the progress of the link establishment, the security negotiations, and the actual database synchronization happening in the mount process, by using the logs under the "Stations" and "Links" pages under "System Setup" on both the satellite and the master stations.

This concludes all "system level" configuration of your master and satellite setup. All remaining configuration will be done via. the web interface on the master system exclusively. The satellite will be notified by the master of any configuration changes, and will automatically start performing any checks that are configured to be run on the satellite.

9.4. Getting started with remote monitoring

You can specify, either on a check-by-check basis, per node, or per domain, from which grid GID SNMPChecks and NetChecks should be performed. In order to get started with your satellite quickly, we suggest that you enter the configuration page for the exported domain, using the web interface on your master station, and specify that all checks should be performed on the GID of the satellite. Enter "Configure" and "Edit" for the exported domain, in order to do this.

Now, you can run an "Auto discovery" under the exported domain. This will start an auto-discovery process on the remote satellite, and the satellite will then configure any nodes that it discover on the remote network. These nodes will appear as they are found, in the web interface on the master.

From here on, you can add new checks, re-configure the ones already set up, add or delete nodes, etc. etc. All reconfiguration that you perform via. the web interface on the master, will be propagated to the relevant remote satellites. The satellites will report back check results to the master, where you can access the graphs and where alerts can be generated. In fact, as soon as the grid is configured, the day to day administration is very similar to the common setup with only one SysOrb Server, except that you now have the ability to run NetChecks and SNMPChecks efficiently and securely on remote locations.

Chapter 10. SysOrb Server maintenance

10.1. Adding MIB files

In order to monitor nodes using snmpChecks the SysOrb server must know about the name, measuring scale etc. of the parameters to be monitored. These informations is stored in MIB (Management Information Base) files.

SysOrb comes with a number of MIB's pre-installed. These MIB's cover standard parameters such as traffic statistics for routers and switches, and some common printer parameters. You should be able to set up basic monitoring of almost any SNMP enabled router, switch or printer without adding other MIB files.

Should you, however, wish to monitor vendor-specific parameters on your network equipment, you will need to add the relevant MIB files to the ones already in SysOrb. This is done while the SysOrb server is running and operational.

To add a MIB file named `xyz.mib` to a running SysOrb server, execute the command **sysorb-mibparser -l admin `xyz.mib`**, assuming a SysOrb user named admin exists in the root domain, and has appropriate privileges. The sysorb-mibparser utility will then extract the information from the MIB file and insert them into the running SysOrb database. Afterwards SysOrb does not need the file `xyz.mib` any more.

After adding the needed MIB files, you must perform a SNMP scan (again) of the nodes supporting any of the newly added parameters. You do that through the SysOrb web interface by clicking configure, browsing to find the node, clicking snmpChecks and finally clicking on the rescan button at the bottom of the page.

The full syntax of sysorb-mibparser is:

sysorb-mibparser [-h] [-s *server*] [-p *port*] -l *login* [-P *password*] *-e | -r | mib-file*...

-s server

Instructs the utility which SysOrb server to connect to. Defaults to the local machine.

-p port

If you SysOrb server uses a non-default port, use this switch.

-l login

You must give the name of a SysOrb user from the root domain having appropriate privileges.

-P password

You may specify a password on the command line to avoid having to type it afterwards, this is especially useful in scripts. However, you should mind the security implications of leaving a plaintext password in a script file.

-h

Show a help message explaining the command line and switches.

-e

Enumerate imported MIB modules.

-r

Remove unused MIB modules.

10.2. Custom NetChecks

In some cases the NetCheck capabilities of the SysOrb Server is not good enough to satisfy a specific need. In these cases a Custom NetCheck can be created.

A Custom NetCheck functions the same way as a normal NetCheck. That is, the result from a Custom NetCheck is the amount of time it took to perform the check. They can of course be used for other purposes, but a Custom NetCheck always returns a number as its result.

This number must be an integer, and represents the number of milliseconds it took to perform the check, and must be written to output by the Custom NetCheck, in order for SysOrb to find it. The format of the line containing this number, can be specified for each check.

If SysOrb did not locate this line in the output from the Custom NetCheck before it exits, the check is marked as giving no reply. If the check does not write the line, in ten seconds, plus the upper limit for the check, the check will be marked as timed out.

The Custom NetCheck is executed as stated in the configuration, with the name of the node to perform the check on, given as the first parameter.

10.2.1. Configuring Custom NetChecks

In order to use Custom NetChecks, you must first tell the SysOrb Server where to find the Custom NetCheck configuration file. On non-windows systems this is done by adding a line like

```
custom_netcheck_conf = "/etc/sysorb/custom_netcheck.conf"
```

to the SysOrb Server configuration file (usually located in `/etc/sysorb/server.conf`). On Windows you must open the SysOrb Configuration utility located in *Start menu->Programs->SysOrb->SysOrb configuration*. In the configuration tree choose *SysOrb Server->NetCheck Manager->Custom NetChecks*, and enter the path where you are going to put the new configuration file.

The format of the Custom NetCheck configuration file is easily illustrated by a simple example:

Example 10-1. Custom NetCheck

```
# Comment lines start with a #
[NetCheck Name]
command=command to execute
result_pattern=([[:digit:]]+)
```

The value enclosed in [and] are the name of the Custom NetCheck, as it will be displayed in the Web-interface. The `command`, is the command to execute, in order to perform the check. The `result_pattern` is a regular expression, that tells SysOrb how to retrieve the result from the check. See Section 8.3 for more information about regular expressions.

Once the Custom NetCheck has been configured, the SysOrb Server must be restarted, in order for it to discover the new checks.

10.3. Custom AlertPaths

In addition to the built-in ways, that the SysOrb server can alert its users (Email, SMS, paging), you can also have SysOrb execute a script of your choice. This could be used to send SMS through a gateway of your provider, to put events into your help-desk system, to send an instant message, or whatever you can do through any command line utility.

The SysOrb server passes information about the alert to be sent exclusively through environment variables. Before instructing SysOrb to invoke the script, you probably want to debug it by running it on the command line with the environment variables manually set. You will find a list with descriptions of each of the variables at the end of this section.

In order to be able to set up the script to be invoked by SysOrb, you must have a running SysOrb server. Log into the web interface using an administrator account, and click the System Setup button. Then select Custom Alerts and click New type of Custom AlertPath. All you have to supply is a name of your choice for this type of custom alerts, and the command for running the script.

After registering the script with SysOrb. You can select any user, click Edit, and the you should see a new option called Add custom path. Try creating a path, and testing it by clickng Test next to the new path on the list.

10.3.1. Script environment

Here is the list of the environment variables passed to the custom alert script.

SO_TYPE

This variable contains one of the the following values:
`warning` The script should send a warning
`alert` The script should send an alert
`test` The script should test the alert path
`scheduled_downtime` The script should send a message, stating that the check is entering unexpected downtime.

SO_USER_NAME

The name of the user to receive this alert.

SO_USER_DOMAIN

The dot separated name of the domain to which the user belongs.

SO_DESTINATION

The alert path destination, as entered in the web interface, when the user created his AlertPath, that is using this script. If your script sends SMS'es through some in-house gateway, then you probably want the destination field to contain the phone number of the phone. Other scripts may require other information in the destinarion field, some may not need it at all.

SO_NODE_LABEL

The full name of the Node in question, as shown on the overview pages.

SO_NODE_DNS

The text from the dns-name/ip-address field of the node in question.

SO_CHECK_TYPE

This variable contains one of the the following values:
`checkin` This alert is due to late checkin of the Node, and the rest of the SO_CHECK_... variables are not set, as this m
`netchecks` This alert is due to a NetCheck.
`agentchecks` This alert is due to an AgentCheck.
`snmpchecks` This alert is due to a snmpCheck.

SO_CHECK_ACCUMULATION

Depending on whether this check is an accumulation or not, this variable will contain `yes` or `no`

SO_CHECK_LABEL

A descriptive name of the check in question.

SO_CHECK_NAME

A dot separated string, which identifies the Check relative to its Node. This is not suitable for inclusion in text presented to the user, use SO_CHECK_LABEL for that. Instead this is mostly useful for conditions in the script, if you for instance want to treat all ICMP checks different than other NetChecks.

SO_CHECK_URL

A http url linking to the SysOrb web interface page for this check.

SO_CHECK_INFO_URL

An optional http url given by the SysOrb user with additional information about this check.

SO_RESULT_STR

The last measured value from the check, printed in human readable format, e.g. `2.34 MB`.

SO_RESULT_UNIT

The unit of the measurement of this check, e.g. `B` (bytes).

SO_RESULT_VALUE

The last measured value from the check, measured in units given by SO_RESULT_UNIT, e.g. `2453666.00`

SO_ALERT_IF_ABOVE, WARN_IF_ABOVE, WARN_IF_BELOW, ALERT_IF_BELOW

The warn/alert limits, measured in units given by SO_RESULT_UNIT. If any of the limits are disabled, the corresponding variable will be unset.

10.4. Creating a new database

One can re-create a clean SysOrb database using the **sysorb-createdb** tool. This is required if one wishes to change the database block sizes, or if one wishes to run the database on raw disk devices (or partitions) rather than on a file system (on operating systems where this functionality is available).

The database creation tool will create two or three files, depending on chosen options. The files are:

`main.odb:`

The meta database.

`main.odbj:` (optional)

External journal for main.odb.

`main.tsdb:`

The time series database

The tool accepts a number of command line options, described blow:

-h:

Print a short help message summarizing usage.

-i:

Start the tool in "interactive" mode - this will allow for a more fine grained control over the created database. It is not usually necessary to tweak these settings, and setting them wrong can result in a non-functional database. Use this option with care.

-d `path`:

Specify a path in where the database files will be created. The default is to create the files in the current working directory.

-c:

Create a completely empty database. This option is not useful in end-user scenarios, as the database will not include the default administrative user, and thus it will not be possible to log into a server started with the empty database.

-f

Force - no questions will be asked by the tool.

A number of advanced settings can be specified in the "interactive" mode of the **createdb** tool. When using the tool in the standard non-interactive mode, good defaults are chosen. It should not usually be necessary for an administrator to specify these options manually, but there are certain situations where such flexibility is useful.

Most notably, when creating a database on "raw" devices (partitions or volumes, rather than in files on a file system), it is necessary to use the interactive mode in order to specify the individual device locations (as the database needs two or three locations, and if volumes or partitions are used, two or three distinct locations must be specified).

In interactive mode, the user will be prompted for each of the advanced settings in turn:

- *Journal type:* The meta database can use either an internal or an external journal. An external journal allows the administrator to place the journal on a smaller but faster storage device.
- *Journal size:* A too small journal will, sooner or later, cause the SysOrb Server to perform an emergency shutdown. It is absolutely vital, that the journal is not chosen too small. The default journal size is 8 Megabytes, which is sufficient. If the journal can be placed on a very fast storage device, it may be beneficial to raise the journal size somewhat. It is not possible to give hard numbers for this though - a busy metadatabase with the journal on a fast storage device, may see speed improvements from a 16 Megabyte journal. It is generally recommended to just go with the default size of 8 Megabytes though.
- *Number of chains:* The time series database relies on "phase chains" rather than a journal to guarantee consistency over unexpected system shutdowns. A number of chains is allocated - the database needs at least one chain as the "consistent" chain, and one as the "active" chain. Thus, the number of chains can not be set lower than two. Setting the number of chains higher does in theory not improve reliability - it does however give the operating system and hardware some amount of "slack" - the default is three chains.
- *Block size:* A uniform block size is used in both the time series database and the meta database. Meta data objects cannot span blocks in the database, therefore the block size must be large enough for the largest possible object. It is not recommended to set the block size lower than 4 kilobytes, although a minimum of 1 kilobyte is possible using the createdb tool. Both databases use 32-bit block indexing, and are therefore limited in size to approximately four billion blocks. A block size of 8 kilobytes thus results in a maximum database size (for each of the metadata and time series databases) of 32 terabytes. Increasing the block size further will raise this limit. The default block size is 8 kilobytes.
- *Estimated number of checks:* As a convenience, the createdb tool can - upon database creation - allocate space in the time series database for some given number of checks. This is only an initial allocation, and the database will be grown later as needed. The default of 1000 checks will result in a database of approximately one gigabyte in size. For small installations, use a smaller number. Again, the database will be grown as needed, anyone can put "10" here - which will make the creatdb run significantly faster.

The following will create a standard database with the default "admin" (password: "admtest") login in the directory /mnt/database/:

sysorb-createdb -d /mnt/database -f

The following will create a completely empty database:

sysorb-createdb -d /mnt/database -f -c

Chapter 11. SysOrb Import and Export tools

The SysOrb importing and exporting tools, called `sysorb-importer` and `sysorb-exporter` is provided to allow the user to import and export parts of the database. This could be either for importing in another system, or in order to create nodes/checks in SysOrb, from the commandline.

11.1. `sysorb-exporter`

The `sysorb-exporter` can be used to export a part of the SysOrb database to XML. Once the data is exported to XML, it can easily be manipulated and imported into SysOrb again, or it can be imported into other programs.

The syntax of the `sysorb-exporter` tool is:

sysorb-exporter [-h] [-s *server*] [-p *port*] [-d *domain*] -l *login* [-P *password*] [-f *filename*] [-e *domain*] [-i *classes*] [-I *classes*]

-h

Displays a text on how to use the tool

-s server

Instructs the utility which SysOrb server to connect to. Defaults to the local machine.

-p port

If you SysOrb server uses a non-default port, use this switch.

-d domain

Specifies the domain in which the user that is used for the login is placed.

-l login

The login name of the user.

-P password

The password for the login.

-f filename

This option specifies the output file. If not specified standard out will be used.

-e domain

The domain to export. I.e. if the users login is in the root domain, and the domain that should be exported is further in the tree, this option is useful.

-i classes

Ignores the listed classes in the export. This is a comma-separated list, that allows to stop the export at a specific level, e.g. at the node level.

-I classes

Include only the listed classes. This is a comma-separated list, that allows you to restrict the scope of the export operation. E.g. **-I Domain,Node** will cause only domains and the nodes to be exported, not any of the checks or other sub-objects of either nodes or domains. Appending a + to a class name will cause all sub-objects to

be included, e.g. **-I Domain,Node+** will cause all nodes with checks and other settings to be exported, while still excluding users and other objects present in domains.

11.2. `sysorb-importer`

The `sysorb-importer` can import XML-data exported from the `sysorb-exporter`, or XML-data generated by scripts or programs, as long as they use the same format as the `sysorb-exporter`.

The syntax of the tool is:

sysorb-importer [-h] [-s *server*] [-p *port*] [-d *domain*] -l *login* [-P *password*] [-f *filename*] [-m]

-h

Displays a text on how to use the tool

-s server

Instructs the utility which SysOrb server to connect to. Defaults to the local machine.

-p port

If you SysOrb server uses a non-default port, use this switch.

-d domain

Specifies the domain in which the user that is used for the login is placed.

-l login

The login name of the user.

-P password

The password for the login.

-f filename

This option specifies the output file. If not specified standard out will be used.

-i domain

The domain to import into. I.e. if the users login is in the root domain, and the domain that should be imported into is further down the tree, this option is useful.

-m

When this option is specified, the `sysorb-importer` does not remove the outmost domain in the XML file, which is done otherwise.

11.3. Examples of how to use

In the following we will demonstrate how the importer and exporter can be used to automate the creation of nodes in SysOrb. These concepts can easily be expanded to allow automatic creation of other objects is SysOrb.

In this example we will assume that there exists a domain in the root-domain called `Test`, and in that domain a node called `template` is placed.

In order to export the contents of the Test domain, the command in the first line of the following example is used. The program will respond with something like the following lines.

Example 11-1. Exporting contents of the Test domain

```
$ sysorb-exporter -l admin -P admtest -e Test -i CheckGroup -f template.xml
Exporting database
Exporting domain Test.
Exporting statistical data (1.3266410235). Please wait.


Done
```

The -i CheckGroup is included in order to keep the exporter-tool from exporting the Checks configured on the node.

The above command will export a file called `template.xml` which will contain the following (note that there will be some differences between different exports):

Example 11-2. Contents of the exported file

```
<?xml version="1.0"?>
<Meta version="1.0" description="SysOrb exported database" timestamp="2004-04-13-T15:52:38">
</Meta><Object id="1.3266410236" type="Domain">
  <Property type="Enumeration(14)" name="downNotify" value="1"/>
  <Property type="String" name="label" value=""/>
  <Property type="String" name="name" value="Test"/>
  <Property type="Computed String" name="siblingName" value="Test"/>
  <Object id="1.3266410235" type="Node">
    <Property type="Boolean" name="acknowledgeAlerts" value="false"/>
    <Property type="Integer" name="alert" value="1000"/>
    <Property type="Integer" name="alertCeiling" value="1100"/>
    <Property type="Enumeration(12)" name="alertStrategy" value="1"/>
    <Property type="Unsigned Integer" name="checkinFrequency" value="30"/>
    <Property type="Integer" name="checkinScore" value="-5"/>
    <Property type="Enumeration(14)" name="downNotify" value="1"/>
    <Property type="String" name="label" value=""/>
    <Property type="Integer" name="missedCheckinScore" value="10"/>
    <Property type="String" name="name" value="template"/>
    <Property type="ObjectSet" name="nodeClasses" value=""/>
    <Property type="Computed String" name="siblingName" value="template"/>
    <Property type="String" name="snmpCommunity" value="public"/>
    <Property type="Enumeration(20)" name="snmpVersion" value="0"/>
    <Property type="Integer" name="warn" value="200"/>
    <Property type="Integer" name="warnCeiling" value="300"/>
  </Object>
</Object>
```

Before this file can be used to create new nodes in SysOrb, the name of the node must be changed. This is the property called `name`. Just change the value to something other than `template`. The following example shows the modified file, which can be used to create a node called `New Node`.

Example 11-3. Modified XML file, ready for import

```
<?xml version="1.0"?>
<Meta version="1.0" description="SysOrb exported database" timestamp="2004-04-13-T15:52:38">
</Meta><Object id="1.3266410236" type="Domain">
  <Property type="Enumeration(14)" name="downNotify" value="1"/>
  <Property type="String" name="label" value=""/>
  <Property type="String" name="name" value="Test"/>
  <Property type="Computed String" name="siblingName" value="Test"/>
  <Object id="1.3266410235" type="Node">
    <Property type="Boolean" name="acknowledgeAlerts" value="false"/>
    <Property type="Integer" name="alert" value="1000"/>
    <Property type="Integer" name="alertCeiling" value="1100"/>
    <Property type="Enumeration(12)" name="alertStrategy" value="1"/>
    <Property type="Unsigned Integer" name="checkinFrequency" value="30"/>
    <Property type="Integer" name="checkinScore" value="-5"/>
    <Property type="Enumeration(14)" name="downNotify" value="1"/>
    <Property type="String" name="label" value=""/>
    <Property type="Integer" name="missedCheckinScore" value="10"/>
    <Property type="String" name="name" value="New Node"/>
    <Property type="ObjectSet" name="nodeClasses" value=""/>
    <Property type="Computed String" name="siblingName" value="template"/>
    <Property type="String" name="snmpCommunity" value="public"/>
    <Property type="Enumeration(20)" name="snmpVersion" value="0"/>
    <Property type="Integer" name="warn" value="200"/>
    <Property type="Integer" name="warnCeiling" value="300"/>
  </Object>
</Object>
```

Using the modified file it is possible to create a node called `New Node` in any domain on the SysOrb Server. To create the node in the domain called `New Domain` (must be created before the command is executed), use the following command:

Example 11-4. Creating a node using `sysorb-importer`

```
$ sysorb-importer -l admin -P admtest -i "New Domain" -f template.xml
Importing database into domain 1.1004032985 and replacing it.
Importing configuration..                    Done
```

It is possible to expand on this to create domains, checks etc. from the command line. It is also possible to export the statistical data gathered by SysOrb if it must be processed by some other tool. However in this case `sysorb-tool` is a better solution, which is described in the next chapter.

Chapter 12. SysOrb Tool

The SysOrb Tool is provided in order to access the SysOrb database using a command line tool.

The syntax of the tool is:

sysorb-tool [-h] [-s *server*] [-p *port*] [-d *domain*] -l *login* [-P *password*] [-f *filename*] command [command-options] path | -i id

-h

Displays a text on how to use the tool

-s server

Instructs the utility which SysOrb server to connect to. Defaults to the local machine.

-p port

If you SysOrb server uses a non-default port, use this switch.

-d domain

Specifies the domain in which the user that is used for the login is placed.

-l login

The login name of the user.

-P password

The password for the login.

-f file

This option specifies the input or output file. If not specified standard in or out will be used, according to the command used.

The **command** argument determines the output of the program. Currently there are seven different commands available:

select

The **select** command is used for extracting statistical information about a check or the incident log from a node, and output it to a CSV file. It is possible to specify the time interval that should be exported. In this case the path argument is a path to a check.

insert

The **insert** command allows the user to insert data into a check. The data is read from an CSV file, in the same format used by the **select** command.

listdomains

listdomains is used to extract a list of domains. Which domains are listed are determined by the path, which must point to a domain. The command then lists all subdomains in this domains. This can also be done recursively, so the subdomains of the subdomains also are listed.

listnodes

listnodes will list the nodes located under the domain, specified by the path.

listchecks

listchecks will, if given a node as parameter, list all the checks on this node. If given a domain, all the checks on the nodes in the domain will be listed.

listactions

listactions command lists all the actions on a node or domain identified either by an ID, or by a path. Output is the same format as **listdomains** uses. If the command is run on a domain, all actions on nodes directly under the selected domain will be displayed. If a node is used, all the actions are displayed.

listproperties

listproperties will list all the properties of the object identified by the path. This can be both Domains, Nodes and Checks.

update

update will update the value of one or more properties in the object identified by the path.

resetscores

resetscores command can be used to reset ScoreKeeper scores on a node or check.

setdowntime

setdowntime command can be used to set unexpected downtime on a domain, a node or a check.

The path is a string identifying either a domain, a node or a check. If the path identifies a domain, the path only consists of one item. This is a '.' seperated list of the domain names, leading to the domain. So a path to the 'Routers' domain located in the 'Backbone' domain would be 'Backbone.Routers'.

If the path identifies a node, it will consist of two parts seperated by a ','. The first path will identify the domain in which the node is located, in the same way as specifying a path to a domain. The second part is the node's name (not label) in SysOrb. I.e. a path to the node 'mainrouter' in teh 'Routers' subdomain would be 'Backbone.Routers,mainrouter'.

A third part is added when the path identifies a check. This part identifies the check for the SysOrb Server. However constructing thi part by hand is not easy, so the listchecks command should be used in order to obtain thie part of the path.

Instead of a path, the **-i** switch can be used. This switch allows you to give an ID to the domain/node/check instead of the path. The id's can be retrieved by the list commands.

12.1. The select command

select [-t *timeformat* | -u] [-b *starttime*] [-e *endtime*]

The **select** command allows you to extract statistical information about a specific check, or the incident log from a node, into a CSV (Comma Seperated Value) file.

-t timeformat

Using this switch allows you to change the format of the time, when written to the CSV file. How to do this is documented in Section 12.1.1.

-u

Using this switch will format the time as seconds passed since '1970-01-01 00:00:00 GMT'. Note that only one of the **-u** and **-t** switches can be used at a time.

-b starttime

The earliest point in time, for which data should be extracted. This should be specified as either as the number of seconds passed since '1970-01-01 00:00:00 GMT', or as ISO time (YYYY-MM-DD-Thh:mm:ss). If **-b** is not given, the extraction will include the first data record, present in SysOrb.

-e endtime

The latest point in time, for which data should be extracted. The format of *endtime* is the same as for *starttime* for the **-b** switch.

12.1.1. How to specify the *timeformat*

If you have specific requirements as to how the time stamps should be formatted, the **-t** option allows you to customize the output.

The *timeformat* is a string, in which some special character sequences will be replaced with specific parts of the date.

The following sequences are supported (note that the specific output of some of these sequences will depend on the language and region setting of the node where sysorb-tool is executed):

%a

The abbreviated weekday name for the date.

%A

The un-abbreviated weekday name.

%b

The abbreviated month name for the date.

%B

The full month name.

%c

The preferred date and time format, according to the languange and region settings of the machine.

%d

The day of the month. (01-31).

%H

The hour on a 24-hour clock, as a number (00-23).

%I

The hour on a 12-hour clock, as a number (01-12).

%j

The day of the year as a number (001-366).

%m

The month as a number (01-12).

%M

The minute as a number (00-59).

%p

AM/PM for 12-hour clocks.

%S

The seconds as a number (00-61). The number of seconds can reach 61, in order to account for leap-seconds.

%U

The week number of the year (00-53). This will start the numbering with the first sunday as the first day of week 01.

%w

The weekday as a number (0-6). Sunday is 0.

%W

The week number of the year (00-53). This will start the numbering with the first monday as the first day of week 01.

%x

The preferred date format, according to the languange and region settings of the machine.

%X

The preferred time format, according to the languange and region settings of the machine.

%y

The year without the century as a number (00-99).

%Y

The year with the century as a number

%%

A '%' will be output.

12.2. The insert command

insert

The **insert** command reads a CSV file, and imports the data from that file, into a specific check in the SysOrb database.

The command takes no options, and it expects the timeformat of the CSV file to be either the default format outputted from **select**, or the format obtained when **-u** is used. The command will automatically determine which of the formats are used for the CSV file.

12.3. The listdomains command

listdomains [-o N | i | p | d] [-r]

The **listdomains** command will list the sub-domains of the specified domain.

-o *N|i|p|d*

This switch controls the produced output. Each of the letters will make the command output a different property of the domains being listed. The order of the letters etermins the order of the output. The letters have the following meanings:

N

Will display the human readable name for the domain.

i

Will display the ID of the domains.

p or d

Will display the path to the domains, usable as a parameter for the **listnodes** or **listdomains** commands.

-r

If this switch is given, not only direct subdomains will be listed, but also subdomains of the subdomains.

12.4. The listnodes command

listnodes [-o N | i | p | d | n] [-r]

The **listnodes** command will list the nodes of the specified domain.

-o *N|i|p|d|n*

This switch controls the produced output. Each of the letters will make the command output a different property of the nodes being listed. The order of the letters determine the order of the output. The letters have the following meanings:

N

Will display the human readable name for the node.

i

Will display the ID of the node.

p

Will display the full path to the node, usable as a parameter for the **listchecks** command.

d

Will display the path to the domain in which the node is located.

n

Will display the path to the node, without the domain path.

-r

If this switch is given, not only nodes placed directly in the selected domain will be listed, but also nodes in the subdomains.

12.5. The listchecks command

listchecks [-o N | i | p | d | n | c] [-r]

The **listchecks** command will list the checks of the specified node or domain.

-o *N|i|p|d|n|c*

This switch controls the produced output. Each of the letters will make the command output a different property of the checks being listed. The order of the letters etermins the order of the output. The letters have the following meanings:

N

Will display the human readable name for the check.

i

Will display the ID of the check.

p

Will display the full path to the check, usable as a parameter for the **select** or **insert** commands.

d

Will display the path to the domain in which the check is located.

n

Will display the path to the node, on which the check is located.

c

Will display the path to the check, without the domain and node path.

-r

If this switch is given, not only checks placed on nodes which are directly in the selected domain will be listed, but also checks on nodes in the subdomains. This option has no effect if the path specifies a node instead of a domain.

12.6. The listactions command

listactions [-o N | i | p | d | n | c] [-r]

The **listactions** command lists all the actions on a node or domain identified either by an ID, or by a path. Output is the same format as **listdomains** uses. If the command is run on a domain, all actions on nodes directly under the selected domain will be displayed. If a node is used, all the actions are displayed.

-o *N|i|p|d|n|c*

This switch controls the produced output. Each of the letters will make the command output a different property of the checks being listed. The order of the letters etermins the order of the output. The letters have the following meanings:

N

Will display the human readable name for the action.

i

Will display the ID of the action.

p

Will display the full path to the action, usable as a parameter for the **select** or **insert** commands.

d

Will display the path to the domain in which the action is located.

n

Will display the path to the node, on which the action is located.

c

Will display only the check part of the path.

-r

If the command is run on a domain, it will list all active actions on nodes in all subdomains. If used on a node, the option will have no effect.

12.7. The listproperties command

listproperties {*path-specification*} [-a] [-t] {*property-name*}...

The **listproperties** command will list some or all of the properties of a Domain, Node or Check together with the value of said properties..

-a

When this switch is given, all properties of the given Domain, Node or Check will be listed, together with their value.

-t

When this switch is given, the type of each property will be listed together with the propertys name and value

property-name...

If the **-a** argument is not given, the **listproperties** will only list the properties where the name is explicitly given on the command line.

12.8. The update command

update {*path-specification*} {*property-name*=*new-property-value*}...

The **update** command will update one or more properties of a Domain, Node or Check, if necessary.

The commands arguments is a list of properties together with the new value of the property.

If you wish to alter the name of a domain, the parameter **name="New Name"** will change the name property of the specified domain to "New Name". When you use parameters that contains spaces, make sure that you quote according to the rules of your shell.

12.9. The resetscores command

resetscores {*path-specification*}

The **resetscores** command can be used to reset ScoreKeeper scores on a node or check.

12.10. The setdowntime command

setdowntime {*path-specification*} {*interval*} [-c]

The**setdowntime** command can be used to set unexpected downtime on a domain, a node or a check.

interval

The interval is given as an integer number of seconds from the time of invocation.

-c

Comment.

12.11. Understanding the CSV file-format

`sysorb-tool` has 3 different CSV formats, depending on which kind of check is selected.

The first line of each CSV file identifies the columns of the file. After this line, comes the real data from the check. Each line corresponds to one record in the SysOrb database. A record can contain several measurements. This is because SysOrb will "compress" older data, by combining several measurements into one record.

A measurement can be either good or bad. If it is good, it means that SysOrb actually have obtained a value from the check. If it is bad, it means that SysOrb was unable to obtain a value from the check. For example this is the case when SysOrb tries to perform a HTTP check, but cannot connect to the server.

12.11.1. Continuous checks

Continuous checks are the checks that are plotted as a line graph in the SysOrb Web-interface, plus the uptime graphs. E.g. all NetChecks are Continuous graphs.

A CSV file of a continuous check contains 7 different columns.

Begin

The time at which the first measurement contained in the record is started.

End

The end time of the last measurement contained in the record.

Min

The minimum value of the good measurements.

Avg

The average value of the good measurements included in the record.

Max

The maximum value of the good measurements.

ngood

The number of good measurements contained in the record.

nbad

The number of bad measurements contained in the record.

12.11.2. State checks

State check are the checks that can be in different states. E.g. Process Presence check, RAID state check etc. are all State checks.

Internally in SysOrb the state of these checks are represented by a bit-mask. So when SysOrb "compresses" several state measurements into one, more bits get set in the bit-mask of the resulting record.

A state check CSV file contains 5 different fields:

Begin

The time at which the first measurement contained in the record is started.

End

The end time of the last measurement contained in the record.

Mask

A bitmask where all the bits of the different states seen in the period of the record are seen.

ngood

The number of good measurements contained in the record.

nbad

The number of bad measurements contained in the record.

12.11.3. String checks

String checks are all the kind of checks that produce strings. Currently Log Checks and Incident logs for Nodes are the only kinds.

A CSV file of a string check contains the 4 different fields:

Time

The time at which the string was encountered.

Severity

An indicator of how severe the text message is. Currently three values are supported:

- **1** - Good
- **2** - Warning
- **4** - Alert

Text

The text of the measurement.

12.12. Example scripts

In this section we will present two scripts which automate the extraction of all data in SysOrb (except Incident Logs) through the `sysorb-tool`.

One script is for bash, and should work with all Unix'es where bash is installed. The other script is for Windows.

Example 12-1. Example extracting all data using bash

```
#!/bin/bash

#
# Change these variables to match your setup.
#
SOTOOL="/usr/bin/sysorb-tool"
USERNAME=admin
PASSWORD=admtest
DOMAIN=.
SERVER=localhost

# This directory will be created by the script. If such a file or
# directory already exists, it will be deleted.
OUTPUTDIR=output

STDCMDLINE="${SOTOOL} -l ${USERNAME} -P ${PASSWORD} -s ${SERVER} -d ${DOMAIN}"


# Prepare the output directory.
if [ -e "$OUTPUTDIR" ]; then
    rm -rf "$OUTPUTDIR"
fi;

mkdir -p "$OUTPUTDIR"

# The outer loop extracts a list of all the nodes on the server.
$STDCMDLINE listnodes -r -o iN . | while read i; do
```



```
    # The id of the node is the characters before the first space
    NODEID=`echo $i | cut -d ' ' -f 1`

    # The name of the node is the rest of the characters. Remember to
    # translate /'es to _, as /'es are illegal in filenames.
    NODENAME=`echo $i | cut -d ' ' -f 2- | tr / _`

    echo "Exporting $NODENAME"

    # The inner loop extracts all the checks on one of the nodes
    $STDCMDLINE listchecks -o iN -i $NODEID | while read j; do

 # The check's id and the checks name, is extracted in the same
 # way as the nodes.
 CHECKID=`echo $j | cut -d ' ' -f 1`
 CHECKNAME=`echo $j | cut -d ' ' -f 2- | tr / _`

 # Extract all data for the check. If there is only need for a
 # period of data, the -b and -e options to the select command
 # can be used, together with the unix date command.
 $STDCMDLINE -f "$OUTPUTDIR/$NODENAME - $CHECKNAME".csv select -i $CHECKID
    done;
done;
```

Example 12-2. Example extracting all data on Windows

```
@ECHO off

REM The parameters to the sysorb-tool. The current one logs in as the user
REM admin with the password admtest in the domain . on the server localhost
SET params="-l admin -P admtest -d . -s localhost"

REM Change sotool to point to the path of the SysOrb tool.
SET sotool="%ProgramFiles%\SysOrb Server\sysorb-tool.exe"

REM Execute the sysorb-tool in order to extract the list of all the domains.
%sotool% %params% -f %tmp%\nodes.txt listnodes -r -o iN .

REM Now use the nodes.txt file as a parameter to the for loop

FOR /F "tokens=1*" %%i IN (%tmp%\nodes.txt) DO (
 REM echo Processing node %%j

 REM Extract a list of all checks present on the node
 %sotool% %params% -f %tmp%\%%i-checks.txt listchecks -o iN -i %%i

 REM If there are no checks on the node, the file will not be created.
 IF EXIST %tmp%\%%i-checks.txt (
  FOR /F "tokens=1*" %%k IN (%tmp%\%%i-checks.txt) DO (
   ECHO Extracting data for %%l on %%j to file %%k.csv

   REM Store the name of the check in the first line
   REM of the CSV file.
   ECHO Node,CheckName > %%k.csv
   ECHO %%j,%%l >> %%k.csv

   REM Extract the data for the check.
   %sotool% %params% select -i %%k >> %%k.csv
  )
```

```
    DEL /F %tmp%\%%i-checks.txt
   )
  )

  rem DEL /F %tmp%\nodes.txt
```

Chapter 13. Acknowledgments

SysOrb uses parts from other software projects.

13.1. GD Graphics Library

Portions copyright 1994, 1995, 1996, 1997, 1998, 1999, 2000 by Cold Spring Harbor Laboratory. Funded under Grant P41-RR02188 by the National Institutes of Health.

Portions copyright 1996, 1997, 1998, 1999, 2000 by Boutell.Com, Inc.

Portions relating to GD2 format copyright 1999, 2000 Philip Warner.

Portions relating to PNG copyright 1999, 2000 Greg Roelofs.

Portions relating to libttf copyright 1999, 2000 John Ellson (ellson@lucent.com).

Portions relating to JPEG copyright 2000, Doug Becker and copyright (C) 1994-1998, Thomas G. Lane. This software is based in part on the work of the Independent JPEG Group.

Portions relating to WBMP copyright 2000 Maurice Szmurlo and Johan Van den Brande.

Permission has been granted to copy, distribute and modify gd in any context without fee, including a commercial application, provided that this notice is present in user-accessible supporting documentation.

This does not affect your ownership of the derived work itself, and the intent is to assure proper credit for the authors of gd, not to interfere with your productive use of gd. If you have questions, ask. "Derived works" includes all programs that utilize the library. Credit must be given in user-accessible documentation.

This software is provided "AS IS." The copyright holders disclaim all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to this code and accompanying documentation.

Although their code does not appear in gd 1.8.4, the authors wish to thank David Koblas, David Rowley, and Hutchison Avenue Software Corporation for their prior contributions.

13.2. OpenSSL Library

13.2.1. OpenSSL License

Copyright (c) 1998-2007 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment: "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:

   "This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT “AS IS” AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

13.2.2. Original SSLeay license

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are aheared to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young’s, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

   "This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

   The word ’cryptographic’ can be left out if the rouines from the library being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement: "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

Chapter 14. WebAPI

14.1. Overview

From SysOrb version 4.2 a RESTfull API is shipped with SysOrb. The API allows integrating SysOrb with 3rd party services or extends its functionality. This chapter provides documentation of how to use SysOrb RESTfull API..

14.1.1. Definitions

- REST - Representational State Transfer. It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.
- Web server - Web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver Web content that can be accessed through the Internet.
- Authentication - is the process of determining whether someone or something is, in fact, who or what it is declared to be.
- Authorization - function of specifying access rights to resources
- CRUD - create, read, update and delete (CRUD) are the four basic functions of persistent storage.
- URI - A URI identifies a resource either by location or name. More often than not, most of us use URIs that defines a location to a resource. However, a URI does not have to specify the location of a specific representation.
- URL - A URL is a URI but a URI is not a URL. A URL is a specialization of URI that defines the network location of a specific representation for a given resource.

14.1.2. Overall description

In addition to standard Web browser interface SysOrb will expose its subset of its functionality via RESTful API accessible via HTTP(S) transport. API operations will be invoked at specific server endpoints (HTTP resources).

All API requests are completely independent from each other, i.e. we don't maintain any session specific state machine on server - that is an essence of REST approach. The duty of maintaining the state of session (if needed) lies on client. For example, in order to match the state before request send to server and after getting a response, the client can send some context specific information (cookie) related to request and server, after processing the request returns response as well as unmodified context data (cookie, etc) attached to request.

Another distinct feature of REST approach that should be supported by SysOrb Web API is "CRUD" approach to accessing the data that matches to standard HTTP methods: Create=POST, Read=GET, Update=PUT, Delete=Delete.

Thus, using this approach, the majority of requests will work with server as with data storage. In some the cases when some procedure needs to be invoked on SysOrb server (like "rescan") - POST request (with arguments if needed) will be sent to corresponding endpoint.

Initial vision on SysOrb Web API implementation is to make as simple bridge to access SysOrb database via HTTP+XML, i.e. don't make significant data post processing/aggregation (like standard CGI of Web interface does). However, provisions should be made in API implementation to have the possibility for creating complex responses (combined from several data sources).

14.1.3. External interfaces

All the communication with users of the API (either mobile clients or other servers) will done via standard HTTP(S) protocol.

14.1.4. Internal (system) interfaces

"sysorb-webapi" module is the main point for accepting requests, parsing them, querying data from one of SysOrb databases, preparing response and sending it back to client

Designed and developed "sysorb-webapi" module (executable) is expected to be the main outcome of this specification.

Sysorb-webapi should be started by sysorbd continue running together with other sysorb processes (like sysorb-dbms, sysorb-tsdb).

For communication with SysOrb databases sysorb-webapi will use the same shared memory based IPC protocol (see sibling manager) that is used by other SysOrb processes.

14.1.5. Protocols

XML is used for data transferred via HTTP.

14.2. Getting Started

14.2.1. Understanding objects hierarchy in SysOrb

There are a lot of objects in SysOrb, below gives an overview of how it is structured and explain which object can currently be accessed via the Web API.

Figure 14-1.

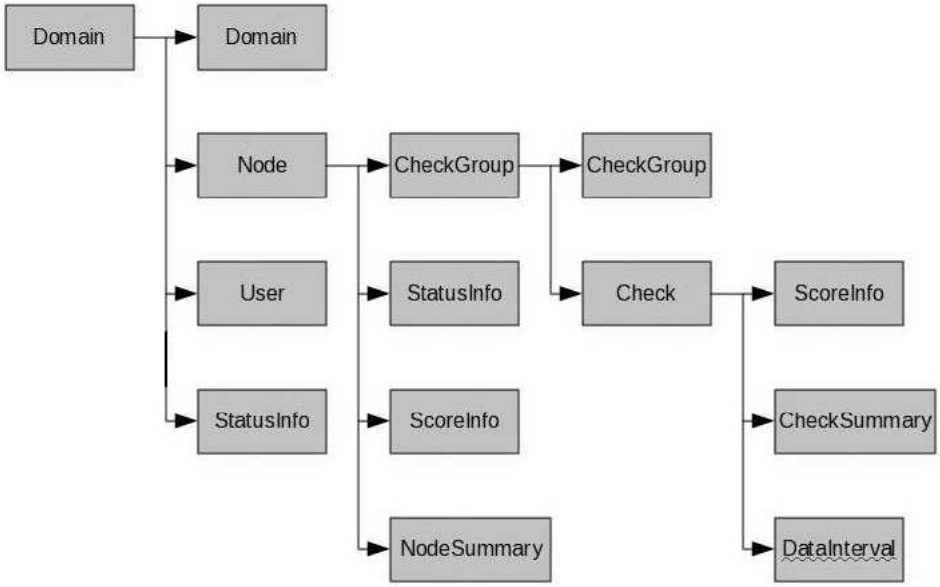


Figure 1: SysOrb Objects Hierarchy

14.2.1.1. Domain

Domains are a logical way of distinguishing between separate parts of the network.

Properties:

Table 14-1.

Name	Type	Editable	Default	Description
id	string			Object id
version	string			Object version
name	string	+		Object name
siblingName	string			Internal name
treeLabel	string		"name"	Composed using "label" property and "name".
longLabel	string		"name"	Composed using "label" property and "name".
stationLocale	uint		Null	Id of satellite assigned to this domain.
dependsOn	object set			The list of objects from which this domain depends
alertGroup	object	+	Null	Alert Group assigned to this Domain
infoUrl	string	+	Null	Link to the external resource for this Domain
comment	string	+	Null	Any additional comment
label	string	+		Description of Domain
maxTotalNodes	uint	+	Null	The maximum number of nodes that are allowed in this domain including subdomains.

Name	Type	Editable	Default	Description
maxNetCheckNodes	uint	+	Null	The maximum number of nodes that can have NetCheck configured on them, excluding nodes that may use NetChecks as add-on to AgentChecks or SnmpChecks.
maxSnmpCheck10Nodes	uint	+	Null	The maximum number of nodes that can have between 1 and 10 SnmpChecks configured, excluding nodes that may use SnmpChecks as add-on to AgentChecks. (Such nodes may also have an unlimited number of NetChecks.)
maxSnmpCheckUnlimitedNodes	uint	+	Null	The maximum number of nodes that can have 11 or more SnmpChecks configured, excluding nodes that may use SnmpChecks as add-on to AgentChecks. (Such nodes may also have an unlimited number of NetChecks.)
maxAgentCheckNodes	uint	+	Null	The maximum number of nodes that can have AgentChecks configured. Whenever a key has been negotiated for a Node it is considered a AgentCheck node. A key is automatically negotiated for a Node the first time a SysOrb Agent checks in to the SysOrb server. In order to free the key again, go to the configuration page of the node and click the 'release key' button. If no such button is visible then the node has no key assigned. It is not possible to perform any AgentChecks without a key assigned to the node. (AgentCheck'ed nodes may also have an unlimited number of SnmpChecks and NetChecks.)
maxLeechingStations	uint	+	Null	The maxinum number of satellites allowed in this domain.
maxAgentLimitNodes	uint	+	Null	The maximum number of nodes that can have an agent with a limited number of AgentChecks configured. Whenever a key has been negotiated for a Node it is considered a agent checked node. The number of checks active on the node, determines whether the node counts as a regular AgentChecked node or as an AgentLimited node. A key is automatically negotiated for a Node the first time a SysOrb Agent checks in to the SysOrb server. In order to free the key again, go to the configuration page of the node and click the 'release key' button. If no such button is visible then the node has no key assigned. It is not possible to perform any AgentChecks without a key assigned to the node. (AgentLimited nodes may also have an unlimited number of SnmpChecks and NetChecks.)
maxEsxiNodes	uint	+	Null	The maximum number of Esxi nodes.
countNetChecks	uint			The current number of configured NetChecks.

Name	Type	Editable	Default	Description
countAgentCheckNodes	uint			The current number of nodes with AgentChecks configured
countAgentChecks	uint			The current number of configured AgentChecks
countSnmpChecks	uint			The current number of configured SnmpChecks
countNodes	uint			The current number of Nodes in this domain including subdomains.
countNetCheckNodes	uint			The current number of Nodes with configured NetChecks.
countSnmpCheck10Nodes	uint			The current number of Nodes with configured number of SnmpChecks less than 10.
countLeechingStations	uint			The current number of configured satellites.
countAgentLimitNodes	uint			The current number of AgentLimitNodes
countEsxiNodes	uint			The current number of Nodes with configured EsxiChecks.
countEsxiChecks	uint			The current number of configured EsxiChecks.

14.2.1.2. Node

Node represents a single machine/host/device in the network.

Properties

Table 14-2.

Name	Type	Editable	Default	Description
id	string			Object id
version	string			Object version
parentId	string			Object parent
name	string	+		Object name
siblingName	string			Internal name
treeLabel	string		"name"	Full name: "label (name)"
longLabel	string		"name"	Full name: "label (name)"
stationLocale	uint		Null	Id of satellite assigned to this node.
dependsOn	object set			The list of objects from which this node depends
checkinFrequency	uint	+	Null	In case of Agent installed on the Node determine the frequency Agent will check in to the server.
alertGroup	object	+	Null	Alert Group assigned to this Domain

Name	Type	Editable	Default	Description
checkinScore	int	+	-5	If an agent has been configure on this node then as long as it is checking in on time the score of the node will be decreased by this amount every five seconds. Per default this is set to -5 which means that as long as the node checks in it will decrease the score by 5 every five seconds.
missedCheckinScore	int	+	10	f an agent has been configure on this node then as long as it is not checking in the score of the node will be increased by this amount every five seconds. Per default this is set to 10 which means that as long as the node does not check in it will increase the score by 10 every five seconds.
warn	int	+	200	If the score associated with the node exceeds the number specified here, a warning will be sent to the AlertGroup specified for the node. The score of the node can be in- and decreased by the checkin/no-checking settings for the node. Each check of this node also have a AlertGroup setting, and its own score and scorekeeper parameters. If a check is in a warn or alert state, this check will be included in the alert message.
warnCeiling	int	+	300	The score associated with the node or scores associated with checks on this node cannot exceed the number specified here by a warning score increment but only by an alert score increment (as specified in the misc. check configurations). The score of the node can be in- and decreased by the checkin/no-checking settings for the node or by the scorekeeper parameters for each check.
alert	int	+	1000	If the score associated with the node exceeds the number specified here, an alert will be sent to the AlertGroup specified for the node. The score of the node can be in- and decreased by the checkin/no-checking settings for the node. Each check of this node also have a AlertGroup setting, and its own score and scorekeeper parameters. If a check is in a alert state, this check will be included in the alert message.
alertCeiling	int	+	1100	The score associated with the node or scores associated with checks on this node cannot exceed the number specified here. The score of the node can be in- and decreased by the checkin/no-checking settings for the node or by the scorekeeper parameters for each check.
snmpCommunity	string	+	public	SNMP Community for device. Default value is "public".
smnpVersion	enum (SnmpVersion)	+	Autodetect	Version of SNMP protocol to communicate with device

Name	Type	Editable	Default	Description
snmpPort	uint	+	161	Port SysOrb should use to communicate with device
snmpScanInProgress	bool		false	The mark that snmp scan on this node is in progress
snmpScanned	bool		false	'True' means that snmp scan on this node was finished
nodeClasses	object set			List of NodeClasses of this node
resetScore			Null	This property used to reset score on the node. Doesn't supported by current version of API
resetAllScores			Null	This property used to reset all score on the node. Doesn't supported by current version of API
esxiEndPoint	string	+	sdk	EndPoint string to connect to the ESXi server.
esxiUser	string	+		Username to access the ESXi server
esxiPassword	string	+		User password to access the ESXI server
esxiPort	uint	+	443	Port SysOrb should use to communicate with ESXi server
alertStrategy	enum (Alert-Strategy)	+	Immediate	Scorekeeper strategy will keep a score for the node, that is always larger that 0 and is in-/decreased by the state of the monitored checks/node. Only if the score reaches some configurable threshold an alert will be sent. Immediate strategy will send a alert immediately when a monitored check/node is in a bad state. ScoreKeeper is best used for checks that should have some kind of fuzzy thresholds, like ping check where a couple of failed pings is usually not of any concern. Immediate is used for something like LogChecks where a log file in a bad state should be alerted immediately.
infoUrl	string	+	Null	Link to the external resource for this Node
comment	string	+	Null	Any additional comment
acknowledgeAlerts	bool		false	If this option is selected, each alert have to be acknowledged by a SysOrb user before the associated check/node is considered in a good state again. This is useful for ensuring that all alerts is handled by a user. It is especially useful for LogChecks where a bad log entry line should always signal a bad log, ie. a good log entry line read sometime after the bad one should not signal that the LogCheck is in a good state again.
lastAcknowledge	timestamp		Null	The time this node was acknowledged.
label	string	+		Extra description for this Node
countNetChecks	uint		0	The number of active NetChecks
countAgentChecks	uint		0	The number of active AgentChecks
countSnmpChecks	uint		0	The number of active SnmpChecks

Name	Type	Editable	Default	Description
countEsxiChecks	uint		0	The number of active EsxiChecks

14.2.1.3. CheckGroup

CheckGroups are objects that cannot be created or modified by the user, and used to organize checks into hierarchy.

Each node has 4 predefined CheckGroups one for each check type:

- NetCheckGroup (id is 1.XXX.0)
- AgentCheckGroup (1.XXX.1)
- SnmpCheckGroup (1.XXX.2)
- EsxiCheckGroup (1.XXX.3)

14.2.1.4. Check

There are three different types of checks in SysOrb with own set of properties:

- Continuous - is the checks that have numeric data results such as response time, disk free space, CPU usage, etc.
- Enumeration - such checks has a results in a number of cases, such as process (present, absent), RAID (OK, degraded, failed) etc.
- Incident - these checks have a text line as a result (LogChecks).

Properties below are common for all checks in SysOrb:

Table 14-3.

Name	Type	Editable	Default	Description
id	string			Object id
version	string			Object version
parentId	string			Object parent
name	string	+		Object name
siblingName	string			Internal name
treeLabel	string		"name"	Full name: "label (name)"
longLabel	string		"name"	Full name: "label (name)"
stationLocale	uint		Null	The Grid Station that should perform the check. Default value uses the setting from the node containing this check.
inherits	object		Null	The id of the check configured in the NodeClass this check was inherited from.
hasInherits	bool		false	Shows if the check has inherited properties
dependsOn	object set			The list of objects from which this check depends
anyOverriden	bool		false	Shows if the check has overriden properties

Name	Type	Editable	Default	Description
isContinuous	bool		true	'True' if check is of type Continuous
resetScore				Uset to reset score on the check
infoUrl	string	+	Null	Link to the external resource for this Check
comment	string	+	Null	Any additional comment
alertGroup	object	+	0.2	Alert Group assigned to this Check
alertStrategy	enum (Alert-Strategy)	+	ScoreKeeper	Scorekeeper strategy will keep a score for the node, that is always larger that 0 and is in-/decreased by the state of the monitored checks/node. Only if the score reaches some configurable threshold an alert will be sent. Immediate strategy will send a alert immediately when a monitored check/node is in a bad state. ScoreKeeper is best used for checks that should have some kind of fuzzy thresholds, like ping check where a couple of failed pings is usually not of any concern. Immediate is used for something like LogChecks where a log file in a bad state should be alerted immediately.
acknowledgeAlerts	bool	+	false	If this option is selected, each alert have to be acknowledged by a SysOrb user before the associated check/node is considered in a good state again. This is useful for ensuring that all alerts is handled by a user. It is especially useful for LogChecks where a bad log entry line should always signal a bad log, ie. a good log entry line read sometime after the bad one should not signal that the LogCheck is in a good state again.
lastAcknowledge	timestamp		Null	The time this check was acknowledged.
goodScore	int	+	-5	The amount that the score associated with the node should be decreased when this check is in a good state. The default of -5 will decrease the score by 5. The Node configuration page is used for specifying when alerts should be sent. Notice that this value is only used for ScoreKeeper alert strategy.
warnScore	int	+	6	The amount that the score associated with the node should be increased when this check is in a warn state. The default of 6 will increase the score by 6. The Node configuration page is used for specifying when alerts should be sent. Notice that this value is only used for ScoreKeeper alert strategy.
alertScore	int	+	20	The amount that the score associated with the node should be increased when this check is in a alert state. The default of 20 will increase the score by 20. The Node configuration page is used for specifying when alerts should be sent. Notice that this value is only used for ScoreKeeper alert strategy.

Name	Type	Editable	Default	Description
remedyId	object set		Null	Note that once the automatically execution of the selected action is carried out the whole node will be set into 5 min. downtime. This is done in order to avoid alarms from other checks which might arise because of the action being executed.
forecastEnabled	bool	+	false	When forecasts are enabled you can get warnings on predicted failures.
warnOnForecast	bool	+	false	Once a forecast is generated, the forecaster can look through the forecast values and warn administrators of any potential future problems the forecast may indicate.
forecastAlertTime	enum (ForecastAlerttime)	+	1 hour	Depending on the nature of the data that are forecast, forecasts will turn out to be reliable for longer or shorter periods of time. You may find, that forecasts for certain checks are only reliable for, say, two hours. Other forecasts may be reliable for days. Using this option you can specify how long into the future you want the forecaster to look for conditions that may result in a warning condition. Specifying "1 Hour" means, that while the forecast may be generated for several days into the future, only the first hour of the forecast will be examined for conditions that can result in a warning being sent to the administrators.
forecastAlertEstError	enum (ForecastEsterror)	+	5%	The forecaster can estimate the error margin on the forecast data, based on simulations on past observations. This option lets you specify the maximal error estimate that a forecast can have, in order to be used as basis for forecast warnings. For example, if you specify "5%", a forecast with an estimated error of "7%" cannot result in warnings being sent to administrators. Any forecast with an error above the threshold set here, will be deemed unfit for use as basis for forecast warnings.
frequency	uint	+	30	Time in seconds between check executions
active	bool			'True' indicates that check is active

Properties common for Continuous checks:

Table 14-4.

Name	Type	Editable	Default	Description
plotMin	real	+	Null	The minimum value on the Y axis that should be plotted on graphs for this check.
plotMax	real	+	Null	The maximum value on the Y axis that should be plotted on graphs for this check.
warnIfAbove	real	+	500	The threshold when check results should be interpreted as in warning

Name	Type	Editable	Default	Description
alertIfAbove	real	+	1000	The threshold when check results should be interpreted as in alerting
supportsMultiCheck	bool		true	Determines if this check can be used in multicheck configurations. Not supported in the current API.
unit	string			Unit that is used to measure the check. Default value depends on check type
preferredUnit	string			The preferred unit is used in alert notifications, graph viewing etc. instead of the unit that is used to measure the check. E.g. if the check is performed and results in a bytes/sec value, the displayed value can be changed to bits/sec by using this property

Properties common for Enumeration checks:

Table 14-5.

Name	Type	Editable	Default	Description
noStates	uint			Determines the number of states of this check.
XXXX	enum (StateAction)			Determines the status (Good, Warn, Alert, Unknown) for each state of the check. It noStates = 3, there are 3 properties: stateAction0, stateAction1, stateAction2
XXXX	string			Determines the text description for each state of the check.

14.2.1.4.1. NetChecks

These checks can be performed without a SysOrb Agent being installed.

NetChecks on services are performed by using one of these 8 protocols:

- DNS - Asks a DNS-server to translate a hostname to an IP-address, and matches the returned IP-address with a user specified list of valid IP's.
- FTP - Tries to log on the node's FTP-server with a specified username and password. If no username is specified, it is only checked that the FTP-banner is returned.
- HTTP - Connects to an HTTP server on the node and tries to retrieve an URL defined as part of the check.
- ICMP - Sends an ICMP PING to the node and listens for a reply. This is basicly the same as the ping-program does.
- IMAP - Connect to the node and examines whether the IMAP server is running or not. This is done by logging in and logging out again.
- POP3 - Connect to the node and examines whether the POP3 server is running or not. This is done by logging in and logging out again.
- SMTP - Connects to the node and examines whether there is an SMTP server running there or not.
- Generic TCP - This simply tries to make a TCP connection to the specified port on the node. If a connection can be opened this check is considered successful.

Common properties for all NetChecks:

Table 14-6.

Name	Type	Editable	Default	Description
remoteName	string			Internal name of netcheck

Properties for HTTP checks:

Table 14-7.

Name	Type	Editable	Default	Description
remoteType	enum (Rem-check-Type)		HTTP	The type of NetCheck
port	uint	+	80	The port SysOrb should send request
httpUrl	string	+	Null	When SysOrb performs a check on an HTTP service it will issue a simple GET request for this URL.
httpStatusRegex	string	+	^2..	After SysOrb has issued a GET request for the URL specified, the web server should return a HTTP status string consisting of three digits. In this field you can specify a pattern that the status must match for the check to be considered OK. This pattern is a POSIX regular expression. For information about regular expressions, please consult:
httpContentRegex	string	+		This field is used for specifying a pattern that must match the content of the retrieved document. The pattern is a POSIX regular expression. For information about regular expressions, please consult:
httpVersion	enum (HttpVer-sion)	+	1.1	This is the HTTP protocol version that SysOrb will use to query the web server.
useSsl	bool	+	false	Determines if this check should establish an SSL-tunnel before performing the check. This is necessary when checking e.g. secure HTTP servers
httpUserAgent	string	+	Null	If specified, this is the value of the HTTP User-Agent header that SysOrb will send with the request. If you need, for some reason, to send an empty User-Agent header you can achieve this by entering a single space in the input field. The string you specify should conform to the rules of RFC 2616 section 14.43 (

Properties for DNS checks:

Table 14-8.

Name	Type	Editable	Default	Description
remoteType	enum (Remcheck-Type)		DNS	The type of NetCheck
port	uint	+	53	The port SysOrb should send request
dnsHost	string	+	Null	Host to lookup
dnsIp	string	+	Null	The IP address that the host name lookup should resolve to.

Properties for FTP checks:

Table 14-9.

Name	Type	Editable	Default	Description
remoteType	enum (Remcheck-Type)		FTP	The type of NetCheck
port	uint	+	21	The port SysOrb should send request
user	string	+	Null	User name to login to FTP server
checkPassword	string	+	Null	User password to login to FTP server

Properties for Generic TCP check:

Table 14-10.

Name	Type	Editable	Default	Description
remoteType	enum (Remcheck-Type)		Generic TCP	The type of NetCheck
port	uint	+	0	The port SysOrb should send request
connClose	bool	+	false	Enable this option if the check should wait for the other end to close the connection.

Properties for ICMP checks:

Table 14-11.

Name	Type	Editable	Default	Description

Name	Type	Editable	Default	Description
remoteType	enum (RemcheckType)		ICMP	The type of NetCheck
checkAttempts	uint	+	1	Ping attempts specifies the number of times the ICMP checker is allowed to send an ECHO REQUEST packet to the node, in order to get a successful ECHO REPLY message back. The default is 1 attempt, but for congested networks or busy nodes, it can be beneficial to increase this number in order to avoid false alarms.

Properties for IMAP checks:

Table 14-12.

Name	Type	Editable	Default	Description
remoteType	enum (RemcheckType)		IMAP	The type of NetCheck
checkPassword	string	+	Null	Password used to access IMAP server
port	uint	+	143	The port SysOrb should send request
useSsl	bool	+	false	Determines if this check should establish an SSL-tunnel before performing the check. This is necessary when checking e.g. secure HTTP servers
user	string	+	Null	User name to access IMAP server

Properties for POP3 checks:

Table 14-13.

Name	Type	Editable	Default	Description
remoteType	enum (RemcheckType)		POP3	The type of NetCheck
checkPassword	string	+	Null	Password used to access POP3 server
port	uint	+	110	The port SysOrb should send request
useSsl	bool	+	false	Determines if this check should establish an SSL-tunnel before performing the check. This is necessary when checking e.g. secure HTTP servers

Name	Type	Editable	Default	Description
user	string	+	Null	User name to access POP3 server

Properties for SMTP checks:

Table 14-14.

Name	Type	Editable	Default	Description
remoteType	enum (Rem-check-Type)		Generic TCP	The type of NetCheck
port	uint	+	25	The port SysOrb should send request
smtpExecuteVrfy	bool	+	false	Should or not execute VERIFY command

14.2.1.4.2. AgentChecks

Properties:

Table 14-15.

Name	Type	Editable	Default	Description
agentId	string			The id of agent
checkType	enum			The type of the check. See AgnetCheckType in Enumeration secion
executeAction	object			Action assigned to this check (in case of AgentAction check)
descriptionGood	string			Check 'good' status description
descriptionWarn	string			Check 'warn' status description
descriptionAlert	string			Check 'alert' status description
warnIfBelow	real	+		The bottom threshold of the check value when Warning should be triggered
alertIfBelow	real	+		The bottom threshold of the check value when Alert should be triggered
warnIfAbovePct	real	+		The top threshold (in percent) of the check value when Warning should be triggered
alertIfAbovePct	real	+		The top threshold (in percent) of the check value when Alert should be triggered
warnIfBelowPct	real	+		The bottom threshold (in percent) of the check value when Warning should be triggered
alertIfBelowPct	real	+		The bottom threshold (in percent) of the check value when Alert should be triggered
rangeMin	real	+		The max value to draw in SysOrb graphs
rangeMax	real	+		The min value to draw in SysOrb graphs

Name	Type	Editable	Default	Description
separatorRegex	string	+	\\r?\\n	Log entries are identified by looking for this log entry separator. In many cases a newline is the separator. The separator is specified using POSIX regular expressions. For information about regular expressions, please consult:
separatorInclusion	enum			This field specifies if the separator should be a part of the log entry. You can choose to include it in the log entry before the separator, in the log entry after the separator or to discard the separator.
silenceWarn	uint			The number of seconds the logcheck may be silent before a warning is sent. The LogCheck is considered silent as long as no rules with a 'log' target is matched.
silenceAlert	uint			The number of seconds the logcheck may be silent before an alert is sent. The LogCheck is considered silent as long as no rules with a 'log' target is matched.
parasiteOn	id			If you set this field, this LogCheck will leech its entries from the given LogCheck, only if no rules in this check matches an entry will it be returned to the original LogCheck, for processing using its rules. This is mostly useful when configuring LogChecks using NodeClasses.
parasitePattern	string		'.*'	If you have chosen this check to be a parasite on another LogCheck, then the entries "leeched" from the other LogCheck will be restricted to the ones matching this pattern.

14.2.1.4.3. SnmpChecks

Properties:

Table 14-16.

Name	Type	Editable	Default	Description
mibObject	string			Id of the object that contain SNMP OID

14.2.1.4.4. EsxiChecks

Properties:

Table 14-17.

Name	Type	Editable	Default	Description
esxiType	enum			The type of esxi check.

14.2.1.5. User

Properties:

Table 14-18.

Name	Type	Editable	Default	Description
id	string			Object id
version	string			Object version
parentId	string			Object parent
name	string	+		Object name
siblingName	string			Internal name
treeLabel	string		"name"	Full name: "label (name)"
longLabel	string		"name"	Full name: "label (name)"
realName	string	+	"name"	Full name
password	string	+	Null	User password. When requested always contain Null value.
capAck	bool	+	false	Enabling this allows the the user to set downtime, acknowledge alerts and reset scores on both nodes and checks, but not otherwise change any node or check settings.
capAction	bool	+	false	This capability allows the user to start an AgentAction on a node.
capChecks	bool	+	false	Allows the user to configure what checks should run on the different nodes in the accessible domains. The user is also allowed to create and edit NodeViews on all accessible nodes. Lastly, it allows the user to acknowledge alerts, reset scores and configure downtime for checks.
capClusterConf	bool	+	false	This capability only affects SysOrb users in the root domain. When this is enabled the user will be able to setup some server/cluster-wide parameters of the SysOrb server, currently only which Custom AlertPaths will be available, and what command to execute. **Security warning**: Enabling this capability effectively allows the user to execute abitrary shell commands on the SysOrb server (through Custom AlertPaths).
capDomains	bool	+	false	Allows the user to edit or add new subdomains to his or her Origin Domain. It also allows the user to create or edit QuickLinks and Report headers/footers in his Origin Domain and all subdomains to this. If the user is located in the root domain, it also allows him/her to import MIB-files into the SysOrb Server.
capGridConf	bool	+	false	Allows the user to create stations, links, mount points and exports.**Note:** This capability only affects users in the root domain.
capGroups	bool	+	false	Lets the user administer groups and assign alert paths to these in the accessible domains.

Name	Type	Editable	Default	Description
capNodes	bool	+	false	Lets the user configure nodes in his or her Origin Domain and all the subdomains. It also allows the user to edit and configure NodeClasses created in the Origin domain or one of its subdomains. This option also allows the user to acknowledge alerts, reset scores, and configure downtime, but only for nodes.
capRead	bool	+	true	Allows the user to view information
capReports	bool	+	false	Lets the user create templates for reports and generate reports from them. Without this option, the user is not allowed to generate or edit reports, even if they have "Public Editable" set.
capSelf	bool	+	true	Allows the user to edit his or her own information, including his or her password, and to delete the user account. This does
capSetCap	bool	+	false	Lets the user change anything in the domain and its subdomains. This is effectively a way of giving the user full administrative rights in a domain.**Note:** No amount of capabilities can allow a user to access higher level domains. It is therefore perfectly safe to give customers logins with administrative privileges in their own domain. **Security warning:** Enabling this capability for a user in the root domain will allow that user to give himself Server setup capability, which will allow him to run arbitrary shell commands on the SysOrb server.

Name	Type	Editable	Default	Description
capUsers	bol	+	false	A user with this capability can also create new users, but without the "capSetCap" capability, the created users will only have the rights to "capRead", and to "capSelf". Furthermore, this capability together with "capSelf", will allow the user to edit and view other users Views. Without the capability to "capSelf", other users private views can be seen, but not edited. In combination with to "capReports", this capability allows the user to view and edit the private reports of other users. However, the user is only allowed to edit a private view or report, if the owner of the view or report is from the same domain or a subdomain of the users domain. E.g. if a user from the Root domain creates a private report in the Customer.A domain, then a user with all capabilities enabled, will not be able to edit this report. This can only happen if the view or report has "Public edit" enabled.

14.2.1.6. StatusInfo

Properties:

Table 14-19.

Name	Type	Editable	Default	Description
checkin	enum (Status)			Current checkin status
countAlert	uint			The number of checks in Alert state
countOk	uint			The number of checks in Ok state
countWarn	uint			The number of checks in Warning state
network	enum (Status)			Overall status of Network Checks
system	enum (Status)			Overall status of Agent Checks
worst	enum (Status)			The worst status of the Domain
worstDomain*	string			The id of Domain in worst state
worstNode*	string			The id of Node in worst state

Note: Properties marked with "*" only for Domain status info

14.2.1.7. ScoreInfo

Properties:

Table 14-20.

Name	Type	Editable	Default	Description
enteredAlert	timestamp			The time Node/Check had entered Alert state
enteredWarn	timestamp			The time Node/Check had entered Warning state
state	enum			The current state of the Node/Check
status	enum (Status)			The current status of the Node/Check

14.2.1.8. NodeSummary

Properties:

Table 14-21.

Name	Type	Editable	Default	Description
badChecks	object set			List of checks id with bad results
lastCheckin	timestamp		Null	The time of agent last checkin
lastEsxi	timestamp		Null	The time of ESXi last check
lastEsxiRef	id		Null	The id of ESXi last check
lastService	timestamp		Null	The time of netcheck last check
lastServiceRef	id		Null	The time of netcheck last check
lastSnmp	timestamp		Null	The time of SNMP last check
lastSnmpRef	id		Null	The time of SNMP last check
nextCheckin	timestamp		Null	The time of next agent checkin
tag	string			Internal
unknownChecks	object set			The list of objects in Unknown state

14.2.1.9. CheckSummary

Properties:

Table 14-22.

Name	Type	Editable	Default	Description
explanation	string		No data present	Description of check result
lastAlert	timestamp		Null	The time of last alert state
lastCheck	timestamp		Null	The time of last check execution
lastGood	timestamp		Null	The time of last good stat
lastWarn	timestamp		Null	The time of last warning state
nextCheck	timestamp		Null	The time of next check execution
result	string		No data present	Check result
tag	string		Null	Internal

14.2.1.10. Enumeration

Table 14-23.

Type	Name	Value	Text
65537	AlertMethod	0	HTML e-mail
		1	SMS via. e-mail gateway
		2	Num. Pager
		3	Direct SMS
		4	Plaintext e-mail
		5	Custom
65538	StateAction	0	Good
		1	Warn
		2	Alert
65540	RemcheckType	1	HTTP
		2	ICMP
		3	DNS
		4	SMTP
		5	POP3
		6	Generic TCP
		7	FTP
		8	IMAP
		9	NTP
		10	NNTP
		11	NFS
		12	NIS
		13	PORTMAP

Type	Name	Value	Text
		14	RPC
		15	Custom
65541	Status	0	N/A
		1	Good
		2	(Good)down
		3	(Good)depend
		4	(Unknown)down
		5	(Unknown)depend
		6	(Warning)down
		7	(Warning)depend
		8	(Alert)down
		9	(Alert)depend
		10	Uninitialized
		11	Unknown
		12	Warning
		13	Alert
65546	ReportPeriod	1	Yesterfday
		2	Last week
		3	Last month
		4	Last year
		5	All time
		6	Last quarter
		7	Last 12 Hours
65548	AlertStrategy	0	Immediate
		1	ScoreKeeper
65550	DownNotify	0	None
		1	People receiving warnings
		2	People receiving alerts
65551	ForecastAlerttime	0	1 hour
		1	2 hour
		2	6 hour
		3	12 hour
		4	1 day
		5	2 day
65552	ForecastEsterror	0	5%
		1	10%
		2	15%
		3	20%
		4	25%
65553	RecurrenceDay	0	Never
		1	Day
		2	Week day
		3	Month day

Type	Name	Value	Text
		4	Last day of month
		5	Last day of quarter
65554	AgnetCheckType	0	Integer Continuous Check
		1	Process Presence Check
		2	Raid Check
		3	Boolean Check
		4	Uptime check
		5	SMART Status Check
		6	Log Check
		7	Floating Point Continuous Check
		8	Agent Action
		9	Advanced Log Check
		10	SAF-TE Slot
		11	SAF-TE Fan Check
		12	SAF-TE Power Supply Check
65556	SnmpVersion	0	Autodetect
		1	SNMPv1
		2	SNMPv2
65560	LogCheckSeparatorInclusion	0	Append to previous entry
		1	Prepend to next entry
		2	Discard the separator
65561	Period	0	1 year
		1	6 months
		2	3 months
		3	1 month
		4	1 week
		5	1 day
		7	1 hour
65562	RuleType	0	FTP Greeting
		1	SMTP Greeting
		2	POP3 Greeting
		3	HTTP reply header
		4	SNMP OID Prefix
		5	Is Pingable (ICMP)
		6	Has a DNS service
		7	IMAP Greeting
		8	Can connect to port
		9	Node info match
		10	HTTPs reply header
		11	POP3S Greeting
		12	IMAPS Greeting

Type	Name	Value	Text
65563	MinDelayScope	0	Per node
		1	Per check
65564	YesNoDefault	0	Yes
		1	No
		2	Default
65566	ReportType	1	System availability
		2	Incidents
		3	Response time
		4	Current licenses
		5	Current configuration
65568	HttpVerion	0	1.0
		1	1.1

14.2.1.11. Timeseries

All check results (timeseries) stored in database in DataInterval objects. There are 3 types of timeseries, depending on check type: Continuous, Enumeration and Incident.

In common case each 2 sequent DataIntervals should be connected (in case of drawing). But in some cases (server was down or check had no results for some period) DataIntervals shouldn't be connected. For that cases SysOrb has special rule:

Consider four data segments, where A, B, C and D is a record in this functions input vector:

A1 - A2 B1 - B2 C1 - C2 D1 - D2

(B2 - C1) is a hole if:

|C1B2| > 2.5 * (|A2B1|)

AND

|C1B2| > 2.5 * (|D1C2|)

If segment A or segment D is missing, the corresponding expression in the above is simply not considered. If we have less than three segments to consider, they will not be connected.

14.2.1.11.1. Contimuous

Properties:

Table 14-24.

Name	Type	Editable	Default	Description
timestart	timestamp			Start of data interval
timeend	timestamp			End of data interval
ngood	int			Number of check results in this interval in a good state

Name	Type	Editable	Default	Description
nbad	int			Number of check results in this interval in a bad state
min	real			Minimum value of check result in the given interval
avg	real			Average value of check result in the given interval
max	real			Maximum value of check result in the given interval

14.2.1.11.2. Enumeration

Properties:

Table 14-25.

Name	Type	Editable	Default	Description
timestart	timestamp			Start of data interval
timeend	timestamp			End of data interval
ngood	int			Number of check results in this interval in a good state
nbad	int			Number of check results in this interval in a bad state
mask	int			State mask according to the statuses of the check

14.2.1.11.3. Incident

Properties:

Table 14-26.

Name	Type	Editable	Default	Description
time	timestamp			Time when incident was stored
severity	int			Incident severity
message	string			Incident message

14.2.2. Configuring SysOrb Web Service

By default SysOrb WebAPI service is disabled. To enable it locate an option "wapi_enabled" in the config file, set it to "true" and restart SysOrb. Other configuration options:

wapi_port - port used by Web service to serve requests.

wapi_use_ssl - set to "true" to enable ssl.

wapi_threads - the number of threads used by web service to serve requests.

wapi_ssl_cert_path - the path to ssl certificate files.

ios_cert_path - the path to iOS certificate files to allow push notifications.

14.2.3. Retrieving access token

This API requires access token in the Authorization header in all functions. To get this token call GET /token function:

Example 14-1. Request access token

```
// uname is the user name
// upass is the user password
// udom is the user domain
var auth = btoa(uname + ":" + upass + ":" + udom); // build authentication string in Base64 en
var url = 'localhost:8080/token';
request = new XMLHttpRequest();

request.open('GET', url, true);
request.setRequestHeader('authorization', 'Basic ' + auth);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Process XML to extract user domain id and access token
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Web Service makes token invalid after 90 minutes of user inactivity.

14.2.3.1. Signing off

Call delete /token function to gracefully close session:

Example 14-2. Sign off

```
// utoken is the access token

var api_url = 'localhost:8080/token';
request = new XMLHttpRequest();

request.open('DELETE', api_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
```

```
  if (request.readyState == 4) {
    if (request.status == 200) {
      // User signed off successfully
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.4. Working with Domains

14.2.4.1. Iterate through Domains hierarchy

Use GET /domains function to get all subdomains one level below of the domain passed to a function. As a start point use domain id from token request - this is the Domain user logged in. Then using object ids from the response it is possible to request other subdomains and so on.

Example 14-3. Get a list of subdomains

```
// udom - is the Domain id which we want to get subdomains
// utoken - access token

//
var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// Call domains function with udom parameter
request.open('GET', api_url + '/domains/' + udom, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of domains with their object id,
      // version and label, parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```


14.2.4.2. Get a list of Domains with StatusInfo

StatusInfo object shows current state of Domain.

Example 14-4. Get Domain StatusInfo

```
// udom - is the Domain id which we want to get subdomains with StatusInfo objects
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to StatusInfo function
var si_url = '/domains/' + udom + '/statusinfo';

// Call get statusinfo function with udom parameter
request.open('GET', api_url + si_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of domains with StatusInfo object,
      // parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.4.3. Create a new Domain

To create a new domain first of all ask for domain template. Domain template is the object that contain a list of properties can be edited by the user.

Example 14-5. Get Domain template

```
// udom - is the Domain where we want to create a new one
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to template function
var template_url = '/domains/' + udom + '/template';

request.open('GET', api_url + template_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);
```

```
request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of editable properties,
      // parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Once received editable properties, fill the request xml to create a domain.

On success status code 201 Created returned. Read the object id in the 'location' header and version in the 'content-version' header.

Example 14-6. Create a new Domain

```
// udom - is the Domain where we want to create a new one
// utoken - access token
// req_xml is xml that contain a list of properties with values

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/domains/' + udom;

request.open('POST', api_url + func_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 201) {
      // Domain was created, it's id is in the 'location' header
      // and version is in the 'content-version'
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```


14.2.4.4. Update a Domain

To update properties of the existing domain get domain template like described in Create a new Domain section.

Then fill the request xml with properties should be changed and call PUT domains function.

On success a new version of the domain is returned in the 'content-version' header.

Example 14-7. Edit domain properties

```
// domain_id - is the domain we want to edit
// version - current version of the domain
// utoken - access token
// req_xml is the xml that contain a list of properties we want to change

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/domains/' + domain_id;

request.open('PUT', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Domain was updated, the new version is in the 'content-version'header
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.4.5. Delete a Domain

Example 14-8. Delete a Domain

```
// domain_id - is the domain we want to delete
// version - current version of the domain
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();
```

```
// URL to the function
var func_url = '/domains/' + domain_id;

request.open('DELETE', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Domain was deleted
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.5. Working with Nodes

14.2.5.1. Get a list of Nodes

Example 14-9. Get a list of Nodes

```
// udom - is the Domain id which we want to get Nodes
// utoken - access token

//
var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// Call nodes function with udom parameter
request.open('GET', api_url + '/nodes/' + udom, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of nodes with their object id,
      // version and label, parse it.
      ....
    }
    else {
```



```
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.5.2. Create a new Node

To create a new node first of all ask for node template. Node template is the object that contain a list of properties can be edited by the user.

Example 14-10. Get Node template

```
// udom - is the Domain where we want to create a new Node
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to template function
var template_url = '/nodes/' + udom + '/template';

request.open('GET', api_url + template_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of editable properties,
      // parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Once received editable properties, fill the request xml to create a node.

On success status code 201 Created returned. Read the object id in the 'location' header and version in the 'content-version' header.

Example 14-11. Create a new Node

```
// udom - is the Domain where we want to create a new node
// utoken - access token
// req_xml is xml that contain a list of properties with values

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/nodes/' + udom;

request.open('POST', api_url + func_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 201) {
      // Node was created, it's id is in the 'location' header
      // and version is in the 'content-version'
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.5.3. Update a Node

To update properties of the existing node get node template like described in Create a new Node section.

Then fill the request xml with properties should be changed and call PUT nodes function.

On success a new version of the node is returned in the 'content-version' header.

Example 14-12. Update Node properties

```
// node_id - is the id of node we want to edit
// version - current version of the node
// utoken - access token
// req_xml is the xml that contain a list of properties we want to change

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/nodes/' + node_id;

request.open('PUT', api_url + func_url, true);
```



```
// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Node was updated, the new version is in the 'content-version' header
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.5.4. Delete a Node

Example 14-13. Delete a Node

```
// node_id - is the id of node we want to delete
// version - current version of the node
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/nodes/' + domain_id;

request.open('DELETE', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Node was deleted
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};
```

```
request.send(null);
```

14.2.6. Working with CheckGroups and Checks

14.2.6.1. Iterate through CheckGoups hierarchy

There are four predefined groups of checks in SysOrb: Network Checks, Agent Checks, SNMP Checks ans ESXi checks. These groups hierarchically are right under the Node object, so to start traverse the check groups use node id as entry point. Then using object ids from the response it is possible to request other check groups/checks and so on.

Example 14-14. Read the check groups

```
// node_id - entry point to start traversing the checks tree
// utoken - access token

//
var api_url = 'localhost:8080';
request = new XMLHttpRequest();

request.open('GET', api_url + '/subtree/' + node_id, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of checkgroup/check objects with their
      // object id, version and label
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.6.2. Get a list of active checks

Previous function returns all checks - active and inactive.

To get only active checks use GET /ckecks/<filter> function. This function accepts a filter string which helps to sort out different types of check: e.q. if you need to get all active network checks use 'type=netchecks' as a filter. Refer to XXX to get detailed info. As a result service returns a list of active network checks with their CheckSummary

object. CheckSummary object shows detailed information about check status, such as check result with explanation string, time of last checks and time when next check scheduled, etc.

Example 14-15. Get all active checks

```
// node_id - this function requires node id to apply filter
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// Notice that we don't specify filter - by default function returns all active checks
request.open('GET', api_url + '/checks/' + node_id, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of active checks with CheckSummary
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Also this function accepts optional parameter 'status', with values 'alert' and 'warn'. For example 'status=alert' will return a list of active checks in ALERT status.

Note: if parameter 'status' is used function return ScoreInfo object instead of CheckSummary in previous example.

ScoreInfo object shows the current status of the check and time when check entered that status.

Example 14-16. Get active ESXi checks in WARNING state

```
// node_id - this function requires node id to apply filter
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// Get active ESXi checks that are in WARN state
var filter = '/type=esxichecks&status=warn';
request.open('GET', api_url + '/checks/' + node_id + filter, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of active checks with ScoreInfo
      ....
    }
```

```
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.6.3. Create a Check

As usually you need to get check template first. But mention that depending on check type (Network Check and others) you need to pass SiblingName (for Network Checks), because Network Checks are not created initially. Let's demonstrate this in examples below.

Example 14-17. Request a check template for Network Check

```
// check_grp - notice that we need to pass a Network Checks group.
// utoken - access token
// sibling - is the sibling name of the check we want to get template.
// Use 'get /subtree' to get a full list of network checks with their SiblingNames

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to template function
var template_url = '/checks/' + check_grp + '/template/' + sibling;

request.open('GET', api_url + template_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of editable properties,
      // parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Now, using given template we can create a check.

Example 14-18. Create a check

```
// id - parent check group id
// utoken - access token
// req_xml is xml that contain a list of properties with values
// sibling - is the sibling name of the check we want to create

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/checks/' + id + '/template/' + sibling;

request.open('POST', api_url + func_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 201) {
      // Check was created, it's id is in the 'location' header
      // and version is in the 'content-version'
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.6.4. Update a Check

Once created (by Scan process, or as described above) any check can be updated. As usually you need to get a list of editable properties:

Example 14-19. Get a lis of editable properties of the given check

```
// check_id - id of check we want to get a list of editable properties.
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to template function
var template_url = '/checks/' + check_id + '/template/';

request.open('GET', api_url + template_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
```

```
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of editable properties,
      // parse it.
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Example 14-20. Update a check

```
// check_id - is the id of check we want to edit
// version - current version of the check
// utoken - access token
// req_xml is the xml that contain a list of properties we want to change

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/checks/' + check_id;

request.open('PUT', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Check was updated, the new version is in the 'content-version' header
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```


14.2.6.5. Delete a Check

Example 14-21. Delete a Check

```
// check_id - check we want to delete
// version - current version of the check
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/checks/' + check_id;

request.open('DELETE', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Check was deleted
      ....
    } else if (request.status == 202) {
      // Check was made inactive - new version is in content-version header
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Note: This function delete permanently only checks that was created from CheckTemplate objects. All other checks are made inactive.

14.2.7. Managing Users in SysOrb

14.2.7.1. Get a list of Users of a Domain

Use get /users function to get all Users in the Domain

Example 14-22. Get a list of Users

```
// domain_id - Domain we want to get a Users list
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

request.open('GET', api_url + '/users/' + domain_id, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of users registered in the domain
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

To get user permissions use get /object function (only if user have capability to view/change user capabilities).

14.2.7.2. Create a new User

To create a new user first of all ask for user template. User template is the object that contain a list of properties can be edited by the user.

Example 14-23. Get User template

```
// udom - is the Domain where we want to create a new User
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to template function
var template_url = '/users/' + udom + '/template';

request.open('GET', api_url + template_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of editable properties,
      // parse it.
      ....
    }
    else {
```



```
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

Once received editable properties, fill the request xml to create a new user.

On success status code 201 Created returned. Read the object id in the 'location' header and version in the 'content-version' header.

Example 14-24. Create a new User

```
// udom - is the Domain where we want to create a new user
// utoken - access token
// req_xml is xml that contain a list of properties with values

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/users/' + udom;

request.open('POST', api_url + func_url, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 201) {
      // USer was created, it's id is in the 'location' header
      // and version is in the 'content-version'
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.7.3. Update a User

To update properties of the existing user get user template like described in Create a new User section.

Then fill the request xml with properties should be changed and call PUT users function.

On success a new version of the user is returned in the 'content-version' header.

Example 14-25. Update a User

```
// user_id - is the id of user we want to edit
// version - current version of the user
// utoken - access token
// req_xml is the xml that contain a list of properties we want to change

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/users/' + node_id;

request.open('PUT', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
request.setRequestHeader('content-version', version);
request.setRequestHeader('content-type', 'text/xml');

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // User was updated, the new version is in the 'content-version' header
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(req_xml);
```

14.2.7.4. Delete a User

Example 14-26. Delete a User

```
// user_id - is the id of user we want to delete
// version - current version of the user
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

// URL to the function
var func_url = '/users/' + domain_id;

request.open('DELETE', api_url + func_url, true);

// Setup required headers
request.setRequestHeader('authorization', 'Basic ' + utoken);
```



```
request.setRequestHeader('content-version', version);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // User was deleted
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.2.8. Enumerations in SysOrb

14.2.9. Common functions applied to most objects in SysOrb

14.2.9.1. Reading the object's properties

To get complete list of properties for any object in SysOrb use get /objects function.

Example 14-27. Get object properties

```
// id - opject id want to get a list of properties
// utoken - access token

var api_url = 'localhost:8080';
request = new XMLHttpRequest();

request.open('GET', api_url + '/objects/' + id, true);
request.setRequestHeader('authorization', 'Basic ' + utoken);

request.onreadystatechange = function() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // We got xml which contain a list of properties of the object
      ....
    }
    else {
      // Handle errors
      ....
    }
  }
};

request.send(null);
```

14.3. Reference

14.3.1. Authentication

In general, requests that require authentication can be authenticated simply by providing the username, password and domain using Basic HTTP authentication. Since the publicly available protocol is provided only over HTTPS the username/password combination is never transmitted in clear text.

All requests against the WebAPI must be authenticated (either by means of an access token or a real username/password/domain).

14.3.1.1. Method GET /token/

Creates a new valid access token.

User name, password and domain name separated by a single colon (":") should be encoded in Base64 and provided in the 'authorization' header

Example 14-28. Authorization header

```
authorization: Basic encodeBase64(uname:upass:.)
```

Reply

If user authenticated successfully return status code 200 Ok and a document with user domain, user id and access token.

Example 14-29. Reply body schema

```
element authenticated {
 element domain { text }
 & element user { text }
 & element token { text }
}
```

14.3.1.2. Method DELETE /token/

End session for the user with given token.

Reply

200 OK

14.3.2. Database

14.3.2.1. Domains

14.3.2.1.1. Method GET /domains/[id]

Returns a list of domains under the given domain.

Reply

The reply is a document with the list of domains including object id, version and tree label.

Example 14-30. Reply body schema

```
element domains {
 element object {
  element class { test }
  & element id { text }
  & element version { text }
  & element treelabel { text }
 }*
}
```

14.3.2.1.2. Method GET /domains/[id]/statusinfo

Returns a list of domains under the given domain with their statusInfo objects.

Reply

Example 14-31. Reply body schema

```
element domains {
 element object {
  element class { test }
  & element id { text }
  & element version { text }
  & element treelabel { text }
  & element infourl { text }
  & element statusinfo {
  element property {
    element name { text }
    & element value { text }
   }*
  }
 }*
}
```

14.3.2.1.3. Method GET /domains/[domain_id]/alertlist

Return a list of objects in alert/warning/downtime state.

Reply

Example 14-32. Reply body schema

```
element alertlist {
  element alerts {
    element domain {
      element id { text }
      & element label { text }
      & element comment { text } ?
      & element infourl { text } ?
      & element node {
        element id { text }
        & element label { text }
        & element comment { text } ?
        & element infourl { text } ?
        & element status { text } ?
        & element check {
          element id { text }
          & element label { text }
          & element comment { text } ?
          & element infourl { text } ?
          & element status { text }
        }*
      }*
    }*
  }
  & element warnings {
    element domain {
      element id { text }
      & element label { text }
      & element comment { text } ?
      & element infourl { text } ?
      & element node {
        element id { text }
        & element label { text }
        & element comment { text } ?
        & element infourl { text } ?
        & element status { text } ?
        & element check {
          element id { text }
          & element label { text }
          & element comment { text } ?
          & element infourl { text } ?
          & element status { text }
        }*
      }*
    }*
  }
  & element downtime {
    element domain {
      element id { text }
      & element label { text }
      & element comment { text } ?
      & element infourl { text } ?
```

```
      & element node {
        element id { text }
        & element label { text }
        & element comment { text } ?
        & element infourl { text } ?
        & element status { text } ?
        & element check {
          element id { text }
          & element label { text }
          & element comment { text } ?
          & element infourl { text } ?
          & element status { text }
        }*
      }*
    }*
  }
}
```

14.3.2.1.4. Method GET /domains/[domain_id]/template

Reply

Example 14-33. Reply body schema

```
element template {
  element class { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
 }*
}
```

14.3.2.1.5. Method POST /domains/[id]

Create a domain under the given domain.

Request

Example 14-34. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created domain is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.1.6. Method PUT /domains/[id]

Update properties of a given domain.

Request

Version of the domain to update should be provided in the content-version header of a request.

Example 14-35. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated domain is returned in the content-version header of a successful 200 response.

14.3.2.1.7. Method DELETE /domains/[id]

Delete domain with given id.

Request

Version of the domain should be provided in the content-version header.

Reply

Returns 200 OK on success.

14.3.2.2. Nodes

14.3.2.2.1. Method GET /nodes/[id]

Returns a list of nodes under the given domain.

Reply

Example 14-36. Reply body schema

```
element nodes {
 element object {
  element class { test }
  & element id { text }
  & element version { text }
  & element treelabel { text }
 }*
}
```


14.3.2.2.2. Method GET /nodes/[domain_id]/[class_name]

Returns a list of nodes under the given domain with their <class_name> object.

Possible values are: statusinfo, scoreinfo, summary.

Reply

Example 14-37. Reply body schema

```
element nodes {
  element object {
    element class { test }
    & element id { text }
    & element version { text }
    & element treelabel { text }
    & element infourl { text }
    & element object {
      element class { text }
      & element property {
        element name { text }
        & element value { text }
      }*
    }
  }*
}
```

14.3.2.2.3. Method GET /nodes/[domain_id]/template

Reply

Example 14-38. Reply body schema

```
element template {
  element class { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
  }*
}
```

14.3.2.2.4. Method POST /nodes/[id]

Create a node under the given domain.

Request

Example 14-39. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created node is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.2.5. Method PUT /nodes/[id]

Update properties of a given node.

Request

Version of the node to update should be provided in the content-version header of a request.

Example 14-40. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated node is returned in the content-version header of a successful 200 response.

14.3.2.2.6. Method DELETE /nodes/[id]

Delete node with given id.

Request

Version of the node should be provided in the content-version header.

Reply

Returns 200 OK on success.

14.3.2.3. Objects

14.3.2.3.1. Method GET /objects/[id]

Returns a list of properties of the given object.

Reply

Example 14-41. Reply body schema

```
element object {
  element class { text }
  & element id { text }
  & element version { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
  }*
}
```

14.3.2.3.2. Method GET /objects/[id]/parents

Returns a list of parent objects of the given object beginning from the root.

Reply

Example 14-42. Reply body schema

```
element parents {
  element object {
    element class { text }
    & element id { text }
    & element version { text}
    & element treelabel { text }
  }*
}
```

14.3.2.3.3. Method GET /objects/[id]/scoreinfo

Returns a list of properties of the scoreInfo object.

Note: request is applicable for node and check objects.

Reply

Example 14-43. Reply body schema

```
element scoreinfo {
  element class { text }
  element property {
```

```
    element name { text }
    & element value { text }
  }*
}
```

14.3.2.3.4. Method GET /objects/[id]/statusinfo

Returns a list of properties of the statusInfo object.

Note: request is applicable for domain and node objects.

Reply

Example 14-44. Reply body schema

```
element statusinfo {
  element class { text }
  element property {
    element name { text }
    & element value { text }
  }*
}
```

14.3.2.3.5. Method GET /objects/[id]/nodesummary

Returns a list of properties of the nodeSummary object.

Note: request is applicable for node objects.

Reply

Example 14-45. Reply body schema

```
element nodesummary {
  element class { text }
  element property {
    element name { text }
    & element value { text }
  }*
}
```

14.3.2.4. Checks and CheckGroups

Each node has a tree of CheckGroup and Check objects, and Checks are the leafs of the tree.

To request properties of any object use GET /object/ API call, to navigate through the tree use the following method.

14.3.2.4.1. Method GET /subtree/[id]

Returns a list of children checkgroup/check/check template objects under the given object (node or checkgroup).

Reply

Example 14-46. Reply body schema

```
element objects {
  element object {
    element class { text }
    & element id { text }
    & element version { text}
    & element treelabel { text }
    & element siblingname { text } ?
    & element active { bool } ?
  }*
}
```

Note: Optional element <active> valid only for checks, and contain 'true' if check is active.

Note 2: Optional element <siblingName> valid only for check templates.

In case of check template element class will be CheckTemplate.

14.3.2.4.2. Method GET /checks/[check_id]/summary

Returns a properties of checkSummary object of the given check.

Reply

Example 14-47. Reply body schema

```
element checksummary {
  element class { text }
  & element property {
     element name { text }
     & element value { text }
  }*
}
```

14.3.2.4.3. Method POST /checks/[check_id]/timeseries

Returns a number of timeseries of a given check for a period of time.

Request

Example 14-48. Request body schema

```
element timeseries {
  element timestart { timestamp }
  & element timeend { timestamp }
  & element limit { integer }
  & element offset { integer }
}
```

Reply

Depends on check type.

Example 14-49. Reply body schema for Uptime, Accumulation, Continuous

```
element timeseries {
  element type { text }
  & element datainterval {
    element timestart { timestamp }
    & element timeend { timestamp }
    & element ngood { integer }
    & element nbad { integer }
    & element min { float }
    & element avg { float }
    & element max { float }
  }*
}
```

Example 14-50. Reply body schema for Enum

```
element timeseries {
  element type { text }
  & element datainterval {
    element timestart { timestamp }
    & element timeend { timestamp }
    & element ngood { integer }
    & element nbad { integer }
    & element mask { integer }
  }*
}
```

Example 14-51. Reply body schema for Incident

```
element timeseries {
  element type { text }
  & element datainterval {
    element time { timestamp }
    & element severity { integer }
    & element message { text }
  }*
}
```


14.3.2.4.4. Method GET /checks/[node_id]/[filter]

Returns a list of active checks for a given node, filtered by <filter>

Filter:

filter = "type=" type ["&status=" status]

type = "all" | "netchecks" | "agentchecks" | "snmpchecks" | "esxichecks"

status = "alert" | "warn"

If <status> is not present, return all active checks with checksummary object, otherwise return all active checks with <status> and their scoreinfo objects.

Reply

Example 14-52. Reply body schema

```
element checks {
  element object {
    element class { text }
    & element id { text }
    & element version { text }
    & element treelabel { text }
    & element group { text }
    & element checksummary {
      element class { text }
      & element property {
        element name { text }
        & element value { text }
      }*
    }
  }*
}
```

Example 1

Return all active netchecks for node 1.123456:

GET /checks/1.123456/type=netchecks

Example 14-53. Reply body schema

```
element checks {
  element object {
    element class { text }
    & element id { text }
    & element version { text }
    & element treelabel { text }
    & element group { text }
    & element checksummary {
      element class { text }
      & element property {
        element name { text }
```

```
        & element value { text }
      }*
    }
  }*
}
```

Example 2

Return all active netchecks for node 1.123456 that is in the alert status:

GET /checks/1.123456/type=netchecks&status=alert

Example 14-54. Reply body schema

```
element checks {
  element object {
    element class { text }
    & element id { text }
    & element version { text }
    & element treelabel { text }
    & element group { text }
    & element scoreinfo {
      element class { text }
      & element property {
        element name { text }
        & element value { text }
      }*
    }
  }*
}
```

14.3.2.4.5. Method GET /checks/[id]/template/<siblingname>

Returns a list of properties that can be filled in order to create/update check.

If parameter <siblingname> is not present, return editable properties of the check.

Otherwise return editable properties of the check template with given siblingName.

Reply

Example 14-55. Reply body schema

```
element template {
  element class { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
 }*
}
```


14.3.2.4.6. Method POST /checks/[id]/template/<siblingname>

Create a new check under given parent id using template <siblingname>.

Valid only for objects of class CheckTemplate (see method GET /subtree)

Request

Example 14-56. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created check is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.4.7. Method PUT /checks/[id]

Update a check with given properties

Request

Version of the check should be provided in the content-version header.

Example 14-57. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated check is returned in the content-version header of a successful 200 response.

14.3.2.4.8. Method DELETE /checks/[id]

Delete the check with given id (for netchecks) or make it inactive (for others).

Request

Version of the check should be provided in the content-version header.

Reply

If the check was made inactive the new version is returned in the content-version header of a successful 202 response.

14.3.2.5. Users

14.3.2.5.1. Method GET /users/[domain_id]

Returns a list of users under the given domain.

Reply

Example 14-58. Reply body schema

```
element users {
  element object {
    element class { text }
    & element id { text }
    & element version { text}
    & element name { text }
  }*
}
```

14.3.2.5.2. Method GET /users/[domain_id]/template

Reply

Example 14-59. Reply body schema

```
element template {
  element class { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
  }*
}
```

14.3.2.5.3. Method POST /users/[domain_id]

Create a new user under given domain.

Request

Example 14-60. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created user is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.5.4. Method PUT /users/[id]

Update a user with given properties

Request

Version of the user should be provided in the content-version header.

Example 14-61. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated user is returned in the content-version header of a successful 200 response.

14.3.2.5.5. Method DELETE /users/[id]

Delete the user with given id.

Request

Version of the user should be provided in the content-version header.

Reply

Returns 200 OK on success.

14.3.2.6. Enums

14.3.2.6.1. Method GET /enums/[type]

Returns a list of values for Enumeration of type <type>.

Reply

Example 14-62. Reply body schema

```
element enum {
 element enumvalue {
  element text { text }
  & element value { uint }
 }*
}
```

14.3.2.7. Alert paths

14.3.2.7.1. Method GET /alertpaths/[user_id]

Returns a list of alertpaths the given user.

Reply

Example 14-63. Reply body schema

```
element alertpaths {
  element object {
    element class { text }
    & element id { text }
    & element version { text}
    & element name { text }
  }*
}
```

14.3.2.7.2. Method GET /alertpaths/[user_id]/template

Reply

Example 14-64. Reply body schema

```
element alertpaths {
                element template {
  element class { text}
  & element property {
    element name { text }
    & element value { text }
    & element type { text }
  }*
}*
}
```

14.3.2.7.3. Method POST /alertpaths/[user_id]/template/[sibling_name]

Create a new alertpath under the given user.

Request

Example 14-65. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created alertpath is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.7.4. Method PUT /alertpaths/[id]

Update alertpath with given properties

Request

Version of the alertpath should be provided in the content-version header.

Example 14-66. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated alertpath is returned in the content-version header of a successful 200 response.

14.3.2.7.5. Method DELETE /alertpaths/[id]

Delete the alertpath with given id.

Request

Version of the user should be provided in the content-version header.

Reply

Returns 200 OK on success.

14.3.2.8. Users

14.3.2.8.1. Method GET /alertgroups/[domain_id]

Returns a list of alertgroups under the given domain.

Reply

Example 14-67. Reply body schema

```
element alertgroups {
  element object {
    element class { text }
    & element id { text }
    & element version { text}
    & element name { text }
  }*
}
```

14.3.2.8.2. Method GET /alertgroups/[domain_id]/template

Reply

Example 14-68. Reply body schema

```
                element alertgroups {
                element template {
                element class { text}
                & element property {
    element name { text }
    & element value { text }
    & element type { text }
  }*
}*
}
```

14.3.2.8.3. Method POST /alertgroups/[domain_id]

Create a new alertgroup under given domain.

Request

Example 14-69. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The URI to the created alertgroup is returned in the location header and version in the content-version header of a successful 201 response.

14.3.2.8.4. Method PUT /alertgroups/[id]

Update a alertgroup with given properties

Request

Version of the alertgroup should be provided in the content-version header.

Example 14-70. Request body schema

```
element class { text }
& element property {
  element name { text }
  & element value { text }
}*
```

Reply

The new version of the updated alertgroup is returned in the content-version header of a successful 200 response.

14.3.2.8.5. Method DELETE /alertgroups/[id]

Delete the alertgroup with given id.

Request

Version of the alertgroup should be provided in the content-version header.

Reply

Returns 200 OK on success.