NetSaint Dokumentation

Version 0.0.6
Letztes Update: 14. Juli 2001

Über NetSaint

- Was ist NetSaint?
- Systemanforderungen
- Lizensierung
- Geschichte
- Bekannte Probleme
- Danksagung
- Kommentare und Rückmeldungen
- Herunterladen der neuesten Version
- Andere Überwachungswerkzeuge

Release Notes

- Neues in dieser Version
- Change log

Installation von NetSaint

- Auspacken der Distribution
- Übersetzen der Quellen
- Installation von NetSaint
- Verzeichnisstruktur
- Installation der Weboberfläche
- Konfiguration der Authorisierung für die CGIs

Konfiguration von NetSaint

- Übersicht über die Konfiguration
- Optionen der Hauptkonfigurationsdatei
- Optionen der Hostkonfigurationsdatei
- Optionen der CGI Konfigurationsdatei
- Überprüfen der Konfiguration

NetSaint im laufenden Betrieb

NetSaint starten
NetSaint stoppen und neu starten

NetSaint Plugins

Standard plugins
Writing your own plugins

NetSaint Zusätze

Übersicht

cl_status	- Konsolenanwendung zur Anzeige des Status der überwachten Dienste
neat	- Web-basierte Administrations-Oberfläche für NetSaint
netsaint_mrtg	- MRTG Skripte für grafische Host und Diensteinformationen
netsaint_reports	- Reporting-Tool für Hosts und Services-Informationen im zeitlichen Zusammenhang
netsaint_statd	- Perl Daemon zum überwachen von entfernten Host-Informationen
nrpe	- Daemon und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
nrpep	- Service und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
nsa	- Web- und Datenbank-basiertes Administrations-Interface für NetSaint
nsca	- Daemon und Client-Programm um passive Check-Ergebnisse über ein Netzwerk zu senden
pscwatch	- Watchdog-Daemon der das Versenden der Check-Ergebnisse überwacht

Die Philosophie hinter NetSaint

Index
Erreichbarkeit und Status-Erkennung von Hosts
Netzwerk-Ausfälle
Benachrichtigungen
Plugin-Theorie
Überprüfungs-Zeitplan
Status-Typen
Definition von Zeit-Perioden

Advanced Topics

Event handlers
External commands

Indirect host and service checks
Passive service checks
Program modes
Redundant monitoring
Service check parallelization
Volatile services
Notification escalations
Distributed monitoring
Monitoring service and host clusters
Large-scale host and service monitoring

Developer Information

Index

Fun Things That Waste Time

Create a virtual network assistant that speaks!

Miscellaneous

Frequently Asked Questions (FAQs)
Securing NetSaint
Using macros in commands
NetSaint status levels
Information on the CGIs

Über NetSaint

Was ist NetSaint?

NetSaint ist eine Anwendung zur Überwachung von Netzwerkdiensten. Sie wurde entworfen für Linux, sollte aber unter den meisten anderen Unixen ebenso laufen. Einige Merkmale sind:

- Überwachung von Netzwerkdiensten (SMTP, POP3, HTTP, NNTP, PING, etc.)
- Überwachung von Rechnerressourcen (Prozessorlast, Plattenbelegung, etc.)
- Einfaches 'Plugin' Konzept, das den Nutzern ermöglicht, sehr einfach eigene Dienstüberwachungsfunktionen zu entwicklen
- Parallele Dienstüberwachung
- die Möglichkeit, eine Netzwerkhierarchie zu definieren mittels "parent hosts", Erkennung und Unterscheidung zwischen Rechnern die abgeschaltet oder nicht erreichbar sind
- Kontaktbenachrichtigungen, wenn Dienst oder Rechnerprobleme auftreten oder behoben werden (via email, Funkruf, oder benutzerdefinierte Methode)
- die Möglichkeit, Aktionen zu definieren die ablaufen, nachdem ein Ereignis eingetreten ist zur proaktiven Problemlösung
- Automatische Rotation der Log-Datei
- Unterstützung der redundanten Überwachung von Rechnern
- Optionale Weboberfläche zum Betrachten des momentaten Netzwerkstatus, Benachrichtigung und Problemliste, Logdatei, etc.

NetSaint ist *kein*...

- Programm, das für NT entwickelt wurde - das war es nie und wird es nie sein.
- SNMP Manager. Wenn du das brauchst, schau bitte woanders.

Systemanforderungen

Die einzige Anforderung um NetSaint zu Starten ist ein Rechner unter Linux (oder einer UNIX-Variante) und ein C-Compiler. Sehr wahrscheinlich wird auch TCP/IP konfiguriert sein, da die meisten Überwachungsfunktionen über das Netzwerk ausgeführt werden.

Es ist *nicht notwendig* die CGI-Skripte zu benutzen, die in der NetSaint-Software dabei sind. Aber um sie zu benutzen sollten folgende Programme installiert sein...

1. Ein Web-Server (vorzugsweise Apache)
2. Thomas Boutell's gd library version 1.6.3 oder höher (wird benötigt vom statusmap CGI)

Lizensierung

NetSaint ist freie Software und kann genutzt, kopiert, modifiziert (etc.) werden unter den Bedingungen der GNU General Public License Version 2 oder später. Information über die GPL Lizenz und Open Source Software ist zu finden unter www.opensource.org

Geschichte

- *Version 0.0.6b6 - 09/23/2000*
- Version 0.0.5 - 04/26/2000
- Version 0.0.4 - 09/02/1999
- Version 0.0.3 - 05/21/1999
- Version 0.0.2p1 - 04/18/1999
- Version 0.0.2 - 04/10/1999
- Version 0.0.1 - 03/14/1999

Bekannte Probleme

NetSaint ist noch kein ausgereiftes Programm, deshalb sind zwangsläufig noch viele Fehler in der Software. Die aktuelle Liste der bekannten Fehler und Probleme ist zu finden unter http://www.netsaint.org/bugs.html

Danksagung

Etliche Leute haben zu NetSaint beigetragen durch Fehlerberichte, Verbesserungsvorschläge, Plugin-Entwicklung usw. Eine Liste einiger dieser vielen Mitarbeiter an der Etnwicklung von NetSaint ist zu finden unter http://www.netsaint.org/contributors.html. Unglücklicherweise ist diese Liste ziemlich veraltet. Ich bekomme so viele Berichte, Patches, Vorschläge, Plugins usw., daß ich einfach nicht nachkomme...

Kommentare und Rückmeldungen

Ich entwickelte NetSaint für meine eigene Zwecke. Nachdem ich ganz zufrieden war entschied ich es zu veröffentlichen, damit auch andere es benutzen konnten. NetSaint ist freie Software, ich bekomme also kein Entgelt für die Stunden in denen ich an NetSaint arbeite. Um weitere Versionen zu ermöglichen bitte ich Dich mir entsprechendes Feedback zu geben. Ich muß wissen, was in der aktuellen Version nicht korrekt arbeitet und was Du Dir wünschst in zukünftigen Versionen. Für positive Rückmeldungen bin ich auch dankbar, denn sie helfen mir sicher zu sein, daß NetSaint benutzt wird und für Menschen arbeitet. Du kannst mich erreichen unter netsaint@netsaint.org

Herunterladen der neuesten Version

Auf folgenden Seiten können neue Versionen von NetSaint gefunden werden:

- http://www.netsaint.org
- http://www.freshmeat.net (appindex 923738250)

Andere Überwachungswerkzeuge

Falls Du es nicht wusstest, es gibt noch andere Netzwerküberwachungswerkzeuge neben NetSaint. Ich denke NetSaint ist ein ziemlich guter Wettbewerber, aber ich bin eindeutig nicht objektiv... Schau dir die Konkurrenz am besten selbst an - hier sind Links zu einigen:

- Angel Network Monitor
- Autostatus
- Big Brother
- Eclipse
- The Event Monitor Project
- MARS
- Mon
- Netup (French)
- NocMonitor
- NOCOL
- NodeWatch
- Over-CR
- PIKT
- RITW
- Spong
- Sysmon

Informationen zu den CGIs

Einleitung

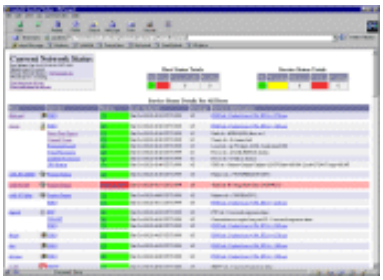
Dies ist eine kurze Beschreibung aller CGIs innerhalb von NetSaint und den verschiedenen Möglichkeiten die Ausgabe durch Parameter in der URL zu steuern. Die Zugriffsvoraussetzungen für die einzelnen CGIs werden ebenso behandelt.

Wichtig: In der Grundeinstellung ist es erforderlich am Webserver als Benutzer angemeldet zu sein und entsprechende Zugriffsrechte auf die CGIs zu haben. Sonst werden die angeforderten Informationen nicht angezeigt. Mehr Infos zur Konfiguration des Webservers und zur Konfiguration der Zugriffsrechte auf die CGIs sollten die Abschnitte Konfiguration des Webinterface und CGI Zugriffsrechte gelesen werden.

Inhalt

Status CGI
Status map CGI
Status world CGI (VRML)
Network outages CGI
Configuration CGI
Command CGI
Extended information CGI
Log file CGI
History CGI
Notifications CGI
Trends CGI

Status CGI




Dateiname: **status.cgi**

Beschreibung:
Dies ist das wichtigste CGI-Skript innerhalb von NetSaint. Es ermöglicht den aktuellen Status aller überwachten Hosts und Dienste anzuschauen. Das Status CGI-Skript kann zwei Hauptansichten dieser Information ausgeben - eine Statusübersicht aller Hostgruppen (oder einer bestimmten Hostgruppe) und eine Detailansicht aller Dienste (oder der Dienste eines bestimmten Hosts). Es können den Hosts nette Icons zugeordnet werden; diese Definition erfolgt in der erweiterten Hostinformation (extended host information) der CGI Konfigurationsdatei.

Zugriffsvoraussetzungen:

CGI Argumente:

Argument	Beschreibung
host=all	Damit wird eine Detailansicht der Stati aller durch NetSaint überwachten Dienste ausgegeben.
host=xxxx	Damit wird eine Detailansicht aller Stati des Hosts *xxxx* ausgegeben, *xxxx* ist dabei der Kurzname des Hosts wie er in der Hostkonfigurationsdatei definiert wurde.
hostgroup=all	Damit wird eine Übersicht aller durch NetSaint überwachten Dienste (und der zugeordneten Hosts), sortiert nach Hostgruppen, ausgegeben.
hostgroup=xxxx	Damit wird eine Übersicht aller Dienste (und der zugeordneten Hosts) die zu der Hostgruppe *xxxx* gehören ausgegeben. Dabei ist *xxxx* der Kurzname der Hostgruppe wie er in der Hostkonfigurationsdatei definiert wurde.

- Mit der *Berechtigung für alle Hosts* werden alle Hosts **und** alle Dienste angezeigt.
- Mit der *Berechtigung für alle Dienste* werden alle Dienste angezeigt.
- Ein *beglaubigter Kontakt* kann alle Hosts und Dienste sehen für die er als Kontakt definiert wurde.

columns=x	Diese Option kann nur in Verbindung mit dem Argument **hostgroup=all** verwendet werden. Damit ist es möglich, die Anzahl der Zeilen der Hostgruppen in der erzeugten Seite vorzugeben. Zum Beispiel wird mit der Angabe *hostgroup=all&columns=4* als Argument das Skript eine Übersicht mit vier Zeilen ausgegeben.
style=detail	Diese Option kann nur in Verbindung mit dem Argument **hostgroup** verwendet werden. Die Angabe diese Arguments veranlasst eine Detailansicht aller Dienste der Hosts innerhalb der angegebenen Hostgruppe. Wenn dieses Argument nicht angegeben wird, wird standardmäßig eine Statusübersicht ausgegeben.
nopopup	Diese Option unterdrückt die Anzeige des Host-Alarmfensters, das aufgemacht wird wenn ein oder mehrere überwachte Hosts abgeschaltet oder nicht erreichbar sind.

Status Map CGI

Dateiname: **statusmap.cgi**

Beschreibung:
Dieses CGI-Skript erzeugt dynamisch eine Karte aller Hosts die in NetSaint definiert wurden. Das Skript benutzt die Bibliothek gd library (Version 1.6.3 oder höher) von Thomas Boutell um Bilder im PNG-Format des definierten Netzwerklayouts zu erzeugen. Es können den Hosts nette Icons zugeordnet werden; diese Definition erfolgt in der erweiterten Hostinformation (extended host information) der CGI Konfigurationsdatei. Wenn dieses CGI-Skript nicht gefunden wird oder Fehler beim Übersetzen (Kompilieren) auftreten sollte diese Fragenliste (FAQ) gelesen werden.

Zugriffsvoraussetzungen:

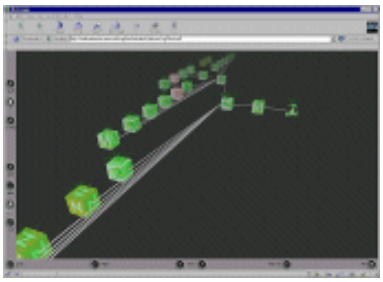
- Mit der *Berechtigung für alle Hosts* werden alle Hosts angezeigt.
- Ein *beglaubigter Kontakt* kann alle Hosts sehen für die er als Kontakt definiert wurde.

Bemerkung: Benutzer, die für einzelne Hosts nicht berechtigt sind sehen dann an dieser Position Knoten die als *unbekannt (unknown)* gekennzeichnet sind. Es ist klar, daß sie eigentlich überhaupt nichts sehen sollten, aber es macht ja keinen Sinn einen Plan zu erzeugen, in dem die Abhängigkeiten zwischen den Hosts nicht erkennbar sind...

CGI Argumente:

Argument	Beschreibung
host=all	Dieses Argument erzeugt einen Plan aller von NetSaint überwachten Hosts.
host=xxxx	Dieses Argument erzeugt einen Plan von Host *xxxx* und allen nachgeordneten Hosts (child hosts). Dabei ist *xxxx* der Kurzname des Hosts wie er in der Hostkonfigurationsdatei definiert wurde.
maxwidth=xxxx	Dieses Argument begrenzt die maximale Breite des Bildes auf *xxxx* Pixel.
maxheight=xxxx	Dieses Argument begrenzt die maximale Höhe des Bildes auf *xxxx* Pixel.
hspacing=xx	Dieses Argument setzt den horizontalen Abstand zwischen den Hostknoten auf *xx* Pixel.
vspacing=xx	Dieses Argument setzt den vertikalen Abstand zwischen den Hostknoten auf *xx* Pixel.
createimage	Veranlaßt das CGI-Skript ein PNG-Bild zu erzeugen anstatt HTML-Code mit imagemap Koordinaten.

Status World CGI (VRML)



Dateiname: **statuswrl.cgi**

Beschreibung:
Dieses CGI-Skript erzeugt dynamisch ein 3-D VRML Modell aller definierten Hosts im Netzwerk. Images can be used as texture maps on host objects defining extended host information entries in the CGI configuration file. Um die Ausgabe dieses CGI-Skript anschauen zu können ist ein VRML-Plugin erforderlich (z.B. Cortona, Cosmo Player oder WorldView).

Zugriffsvoraussetzungen:

- Mit der *Berechtigung für alle Hosts* werden alle Hosts angezeigt.
- Ein *beglaubigter Kontakt* kann alle Hosts sehen für die er als Kontakt definiert wurde.

Bemerkung: Benutzer, die für einzelne Hosts nicht berechtigt sind sehen dann an dieser Position Knoten die als *unbekannt (unknown)* gekennzeichnet sind. Es ist klar, daß sie eigentlich überhaupt nichts sehen sollten, aber es macht ja keinen Sinn einen Plan zu erzeugen, in dem die Abhängigkeiten zwischen den Hosts nicht erkennbar sind...

CGI Argumente:

Argument	Beschreibung
host=all	Dieses Argument erzeugt ein 3-D Modell aller von NetSaint überwachten Hosts.
host=xxxx	Dieses Argument erzeugt ein 3-D Modell von Host *xxxx* und allen nachgeordneten Hosts (child hosts). Dabei ist *xxxx* der Kurzname des Hosts wie er in der Hostkonfigurationsdatei definiert wurde.
notextures	This will prevent images from being texture mapped onto host objects.
notext	Diese Argument unterdrückt die Darstellung des Infotextes (Hostname und -status), der auf den Hostobjekten sichtbar ist.

Netzwerkausfälle CGI

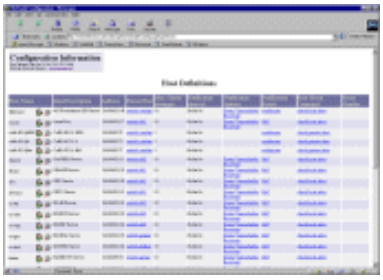
Dateiname: **outages.cgi**

Beschreibung:
Dieses CGI-Skript zeigt eine Liste der Ausfälle erzeugenden "Problemhosts" im Netzwerk. Das kann besonders in einem großen Netzwerk hilfreich sein um sehr schnell den Ursprung eines Problems festzustellen. Die Hosts werden sortiert nach der Schwere der ausgelösten Ausfälle. Mehr Information zu diesem CGI Skript ist hier erhältlich.

Zugriffsvoraussetzungen:

- Mit der *Berechtigung für alle Hosts* werden alle Hosts angezeigt.
- Ein *beglaubigter Kontakt* kann alle Hosts sehen für die er als Kontakt definiert wurde.

Configuration CGI



Dateiname: **config.cgi**

Beschreibung:
Dieses CGI-Skript zeigt die Definitionen der Hosts, Hostgruppen, Kontkte, Kontaktgruppen, Zeitperioden, Dienste und Kommandos die in der Hostkonfigurationsdatei definiert wurden.

CGI Argumente:

Argument	Beschreibung
type=xxxx	Dieses Argument erlaubt es den Typ der angezeigten Definitionen festzulegen. Gültige Werte sind "hosts", "hostgroups", "contacts", "contactgroups", "timeperiods", "commands", und "services".

Zugriffsvoraussetzungen:

- Die *Berechtigung für Konfigurationsinformation* ist erforderlich um die Definitionen für Kontakte, Kontaktgruppen, Hostgruppen, Zeitperioden und Kommandos anschauen zu können. Alle Host- und Dienstedefinitionen sind dann auch sichtbar.
- Mit der *Berechtigung für alle Hosts* werden alle Hosts **und** Dienste angezeigt.
- Mit der *Berechtigung für alle Dienste* werden alle Dienste angezeigt.
- Ein *beglaubigter Kontakt* kann alle Hosts und Dienste sehen für die er als Kontakt definiert wurde.

Command CGI

Dateiname: **cmd.cgi**

Beschreibung:
Dieses CGI-Skript erlaubt Befehle an den NetSaint-Prozeß zu senden. Obwohl dieses Skript verschiedene Argumente hat ist es besser, sie nicht zu benutzen. Die meisten werden zwischen den verschiedenen Versionen von NetSaint differieren. Das extended information CGI sollte als Startpunkt für das Erzeugen von Befehlen benutzt werden.

Zugriffsvoraussetzungen:

- Die Berechtigung *Berechtigung für Systembefehle* ist erforderlich um Befehle an NetSaint zu senden (Neustart, Herunterfahren, Modusänderungen, etc.).
- Mit der *Berechtigung für alle Hostbefehle* können Befehle für alle Hosts **und** Dienste ausgeführt werden.
- Mit der *Berechtigung für alle Dienstbefehle* können Befehle für alle Dienste ausgeführt werden.
- Ein *beglaubigter Kontakt* kann Befehle für alle Hosts und Dienste ausführen, für die er als Kontakt definiert wurde.

Bemerkungen:

- If you have chosen not to use authentication with the CGIs, this CGI will *not* allow anyone to issue commands to NetSaint. This is done for your own protection. I would suggest removing this CGI altogether if you decide not to use authentication with the CGIs.
- In order for the CGI to issue commands to NetSaint, you will have to set the proper file and directory permissions as described in this FAQ.

Extended Information CGI



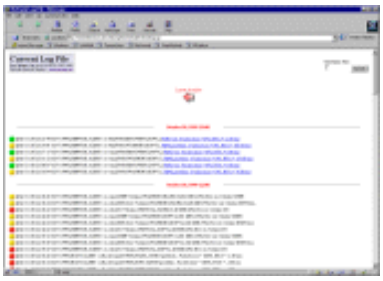

Dateiname: **extinfo.cgi**

Beschreibung:
This CGI allows you to view NetSaint process information, host and service state statistics, host and service comments, and more. It also serves as a launching point for sending commands to NetSaint via the command CGI. Although this CGI has several arguments, you would be better to leave them alone - they are likely to change between different releases of NetSaint. You can access this CGI by clicking on the 'Network Health' and 'Process Information' links on the side navigation bar, or by clicking on a host or service link in the output of the status CGI.

Zugriffsvoraussetzungen:

- You must be *authorized for system information* in order to view NetSaint process information.
- If you are *authorized for all hosts* you can view extended information for all hosts **and** services.
- If you are *authorized for all services* you can view extended information for all services.
- If you are an *authenticated contact* you can view extended information for all hosts and services for which you are a contact.

Log File CGI



Dateiname: **showlog.cgi**

Beschreibung:
Dieses CGI-Skript zeigt die Logdatei an. If you have log rotation enabled, you can browse notifications present in archived log files by using the navigational links near the top of the page.

Zugriffsvoraussetzungen:

- You must be *authorized for system information* in order to view the log file.

CGI Arguments:

Argument	Beschreibung
archive=x	This option allows you to browse notifications in the x^{th} latest log archive. A value of 0 will cause the current log file to be used, a value of 1 will cause the most recent archived log to be used, and so on...
oldestfirst	This option allows view notifications with older entries at the top of the page and newer entries at the bottom. The CGI will normally reverse the log file so that newer log entries show up at the top of the page while older ones are at the bottom.

History CGI

Dateiname: **history.cgi**

Beschreibung:

This CGI is used to display the history of problems with either a particular host or all hosts. The output is basically a subset of the information that is displayed by the log file CGI. You have the ability to filter the output to display only the specific types of problems you wish to see (i.e. hard and/or soft alerts, various types of service and host alerts, all types of alerts, etc.). If you have log rotation enabled, you can browse history information present in archived log files by using the navigational links near the top of the page.

Zugriffsvoraussetzungen:

- If you are *authorized for all hosts* you can view history information for all hosts **and** all services.
- If you are *authorized for all services* you can view history information for all services.
- If you are an *authenticated contact* you can view history information for all services and hosts for which you are a contact.

CGI Argumente:

Argument	Beschreibung
host=all	This will display the history of all hosts being monitored with NetSaint
host=xxxx	This will display the history of host *xxxx*, where *xxxx* is the short name of the host as defined in the host configuration file.
type=x	This option allows you to control which types of historical alerts are displayed. As *x* is a numerical value generated by the CGI, I would suggest using the dropdown box to select the type of alerts you want to view.
statetype=x	This option allows you to control whether soft or hard alerts (or both) are displayed. As *x* is a numerical value generated by the CGI, I would suggest using the dropdown box to select the type of alerts you want to view.
archive=x	This option allows you to browse the history information in the x^{th} latest log archive. A value of 0 will cause the current log file to be used, a value of 1 will cause the most recent archived log to be used, and so on...
oldestfirst	This option allows view history information with older entries at the top of the page and newer entries at the bottom. The CGI will normally reverse the log file so that newer log entries show up at the top of the page while older ones are at the bottom.

Notifications CGI

Dateiname: **notifications.cgi**

Beschreibung:
This CGI is used to display host and service notifications that have been sent to various contacts. The output is basically a subset of the information that is displayed by the log file CGI. You have the ability to filter the output to display only the specific types of notifications you wish to see (i.e. service notifications, host notifications, notifications sent to specific contacts, etc). If you have log rotation enabled, you can browse notifications present in archived log files by using the navigational links near the top of the page.

Zugriffsvoraussetzungen:

- If you are *authorized for all hosts* you can view notifications for all hosts **and** all services.
- If you are *authorized for all services* you can view notifications for all services.
- If you are an *authenticated contact* you can view notifications for all services and hosts for which you are a contact.

CGI Argumente:

Argument	Beschreibung
host=all	This will display all notifications that have been sent out for all hosts (and their associated services) being monitored with NetSaint
host=xxxx	This will display all notifications that have been sent out for host *xxxx* (and its associated services), where *xxxx* is the short name of the host as defined in the host configuration file.
contact=all	This will display all service and host notifications that have been sent out to all contacts.
contact=xxxx	This will display all service and host notifications that have been sent out to contact *xxxx*, where *xxxx* is the short name of the contact as defined in the host configuration file.
type=x	This option allows you to control which types of notifications are displayed. As *x* is a numerical value generated by the CGI, I would suggest using the dropdown box to select the types of notifications you want to view.
archive=x	This option allows you to browse notifications in the x^{th} latest log archive. A value of 0 will cause the current log file to be used, a value of 1 will cause the most recent archived log to be used, and so on...
oldestfirst	This option allows view notifications with older entries at the top of the page and newer entries at the bottom. The CGI will normally reverse the log file so that newer log entries show up at the top of the page while older ones are at the bottom.

Trends CGI

Dateiname: **trends.cgi**

Beschreibung:
This CGI is used to create a graph of host or service states over an arbitrary period of time. In order for this CGI to be of much use, you should enable log rotation and keep archived logs in the path specified by the log_archive_path directive.

Zugriffsvoraussetzungen:

- If you are *authorized for all hosts* you can view trends for all hosts **and** all services.
- If you are *authorized for all services* you can view trends for all services.
- If you are an *authenticated contact* you can view trends for all services and hosts for which you are a contact.

Authentizierung und Authorisierung in den CGIs

Bemerkung

In diesen Anleitungen wird stets davon ausgegangen, daß als Webserver Apache eingesetzt wird. Wenn ein anderer Webserver eingesetzt wird, müssen diese Anleitungen entsprechend abgeändert werden.

Definitionen

In diesen Anleitungen werden folgende Begriffe benutzt, hier eine kurze Erklärung wie diese Begriffe zu verstehen sind...

- Ein *beglaubigter (authenticated) Benutzer* ist jemand, der beim Webserver mit Name und Passwort angemeldet (authentiziert) wurde und der Zugriffsrechte auf die CGIs hat.
- Ein *beglaubigter (authenticated) Kontakt* ist ein beglaubigter Benutzer, dessen Benutzername dem Kurznamen einer Kontaktdefinition in der Hostkonfigurationsdatei entspricht.

Inhalt

Konfiguration der Webserver Authentizierung
Einrichtung beglaubigter (authentizierter) Benutzer
Aktivieren der Authentizierung/Authorisierung in den CGIs
Vorgegebene Berechtigungen für CGI Informationen
Vergabe zusätzlicher Berechtigungen für CGI Informationen
Berechtigungen auf abgesicherten Webservern

Konfiguration der Webserver Authentizierung

Der erste Schritt zur Konfiguration des Webservers für Authentizierung ist sicherzustellen, daß die Datei **httpd.conf** (oder **access.conf**) einen an '**AuthOverride AuthConfig**' Abschnitt für das NetSaint CGI-BIN Verzeichnis enthält. Falls nicht sollte dieser Abschnitt in die Datei **httpd.conf** (oder **access.conf**) eingefügt werden. Dabei ist zu beachten, daß der Webserver neu gestartet werden muß, um die Änderung wirksam werden zu lassen.

```
<Directory /usr/local/netsaint/sbin>
AllowOverride AuthConfig
order allow,deny
allow from all
Options ExecCGI
</Directory>
```

Wenn auch eine Authentizierung erfolgen soll um auf die HTML-Seiten für NetSaint zuzugreifen, muß folgender Abschnitt zur **httpd.conf** (oder **access.conf**) Datei hinzugefügt werden.

```
<Directory /usr/local/netsaint/share>
AllowOverride AuthConfig
order allow,deny
allow from all
</Directory>
```

Im zweiten Schritt ist eine Datei **.htaccess** im CGI Verzeichnis (und evtl. auch im HTML Verzeichnis) von NetSaint (meistens /usr/local/netsaint/sbin bzw. /usr/local/netsaint/share) anzulegen. Die Datei(en) sollte(n) ungefähr folgenden Inhalt haben...

```
AuthName "NetSaint Access"
AuthType Basic
AuthUserFile /usr/local/netsaint/etc/htpasswd.users
require valid-user
```

Einrichtung beglaubigter (authentizierter) Benutzer

Nachdem nun der Webserver so konfiguriert wurde, daß nur mit einer entsprechenden Berechtigung auf die CGIs zugegriffen werden kann, müssen nun die Benutzer eingerichtet werden denen dieser Zugriff erlaubt ist. Das wird erledigt mit dem **htpasswd** Befehl, der Teil von Apache ist.

Die Ausführung des folgenden Befehls wird eine neue Datei mit dem Namen *htpasswd.users* im Verzeichnis */usr/local/netsaint/etc* anlegen. Dabei wird ein Benutzer/Passwort Eintrag für *netsaintadmin* erzeugt. Die Eingabe eines Passworts ist notwendig. Diese Passwort wird wieder abgefragt, wenn *netsaintadmin* sich am Webserver anmeldet.

```
htpasswd -c /usr/local/netsaint/etc/htpasswd.users netsaintadmin
```

Es können weitere Benutzer angelegt werden, um jedem Zugriff auf die CGIs zu ermöglichen, der Zugriff haben sollte. Folgender Befehl kann benutzt werden, es ist lediglich <username> durch den eigentlichen, hinzuzufügenden Benutzernamen zu ersetzen. Die Option **-c** wird nicht mehr benutzt, da die Datei zuvor angelegt wurde und bereits existiert.

```
htpasswd /usr/local/netsaint/etc/htpasswd.users <username>
```

So, das war der erste Teil dessen was getan werden muß. Wird in einem Browser die URL für die NetSaint CGIs eingegeben, sollte nach einem Benutzername und Passwort gefragt werden. Falls es an dieser Stelle mit der Benutzerauthentizierung Probleme gibt, sollte die Webserver-Dokumentation zu Rate gezogen werden.

Aktivieren der Authentizierung/Authorisierung in den CGIs

Als nächstes muß nun sichergestellt werden, daß die Konfiguration der CGIs die Authentizierung undAuthorisierung in Abhängigkeit der Rechte des einzelnen Benutzers wirklich fordert. Dies wird erledigt durch das Setzen der use_authentication Variablen in der CGI Konfigurationsdatei auf einen Wert ungleich Null. Beispiel:

use_authentication=1

Die grundsätzliche Authentizierung/Authorisierung der CGIs ist hiermit eingrichtet und sollte funktionieren.

Vorgegebene (Default) Berechtigungen für CGI Informationen

Welche Berechtigungen haben Benutzer für die CGIs am Anfang, wenn die Authentizierung/Authorisierung aktiviert wird?

CGI Daten	Beglaubigter Kontakt*	Anderer beglaubigter Benutzer*
Host Status Information	Ja	Nein
Host Configuration Information	Ja	Nein
Host History	Ja	Nein
Host Notifications	Ja	Nein
Host Commands	Ja	Nein
Service Status Information	Ja	Nein
Service Configuration Information	Ja	Nein
Service History	Ja	Nein
Service Notifications	Ja	Nein
Service Commands	Ja	Nein
All Configuration Information	Nein	Nein
System/Process Information	Nein	Nein
System/Process Commands	Nein	Nein

*Beglaubigten Kontakten** werden folgende Berechtigungen für die **Dienste** gegeben, für die sie alsKontakte definiert wurden (nicht jedoch für Dienste, für die sie nicht als Kontakte definiert wurden)...

- die Berechtigung die Service Status Information anzuschauen
- die Berechtigung die Service Configuration Information anzuschauen
- die Berechtigung die Historie und Benachrichtigungen des Service anzuschauen
- die Berechtigung Service Commands auszugeben

Beglaubigten Kontakten[*] werden folgende Berechtigungen für die **Hosts** gegeben, für die sie alsKontakte definiert wurden (nicht jedoch für Hosts, für die sie nicht als Kontakte definiert wurden)...

- die Berechtigung die Host Status Information anzuschauen
- die Berechtigung die Host Konfiguration Information anzuschauen
- die Berechtigung die Historie und Benachrichtigungen des Hosts anzuschauen
- die Berechtigung Hostbefehle auszugeben
- die Berechtigung die Statusinformationen aller Dienste dieses Hosts anzuschauen
- die Berechtigung die die Konfigurationsinformationen aller Dienste dieses Hosts anzuschauen
- die Berechtigung die Historie und Benachrichtigungen aller Dienste dieses Hosts anzuschauen
- die Berechtigung Service Commandos für alle Dienste diese Hosts auszugeben

Es wichtig zu betonen , daß im vorgebenen Zustand (by default) **niemand** berechtigt ist folgendes zu tun...

- das Anschauen der unbearbeiteten Logdatei mit dem showlog CGI
- das Anschauen der NetSaint Processinformation mit dem extended information CGI
- Ausgeben von NetSaint Prozess commands mit dem command CGI
- das Anschauen der Hostgruppen, Kontakt, Kontaktgruppen, time period, und command - definitionen mit dem configuration CGI
-

Wenn diese Berechtigungen gewünscht werden, was zweifellos passieren wird, müssen sie an die entsprechenden Benutzer vergeben werden, was wiederum im folgenden Abschnitt beschrieben wird...

Vergabe zusätzlicher Berechtigungen für CGI Informationen

Es ist möglich *beglaubigten Kontakten* oder *beglaubigten Benutzern* Berechtigungen für weitere CGI-Informationen zu geben, indem sie entsprechenden Authorisierungsvariablen in der CGI Konfigurationsdatei zugeordnet werden. Es ist mir klar, daß die verfügbaren Optionen keine sehr speziellen Berechtigungen zulassen, aber es ist besser als gar nichts...

Zusätzliche Berechtigungen für Benutzer können vergeben werden durch Hinzufügen der Benutzer zu den folgenden Variablen in der CGI Konfigurationsdatei ...

- authorized_for_system_information
- authorized_for_system_commands
- authorized_for_configuration_information
- authorized_for_all_hosts
- authorized_for_all_host_commands
- authorized_for_all_services
- authorized_for_all_service_commands

Vorraussetzungen für CGI-Berechtigungen

Wenn sich nun Verwirrung darüber eingstellt hat, welche Berechtigungen für den Zugriff auf welche Informationen notwendig sind sollte einfach der Abschnitt ***Berechtigungsvoraussetzungen*** für CGIs gelesen werden.

Berechtigungen auf abgesicherten Webservern

Wenn der Webserver sich in einer sicheren Domäne (z. B. hinter einer Firewall) befindet oder SSL benutzt wird, kann ein Default-Benutzer definiert werden, der Zugriff auf die CGIs hat. Dies geschieht durch die Definition der default_user_name Option in der CGI Konfigurationsdatei. Durch die Definition diese Default-Benutzers kann auch solchen Benutzern der Zugriff auf CGIs gegeben die sich nicht am Webserver angemeldet (authentiziert) haben.. Das könnte erforderlich sein um die sog. basic web authentication zu vermeiden, da bei dieser Authentizierungsart Passwörter im Klartext übertragen werden.

Wichtig: Es sollte *nur dann* ein Default-Benutzername definiert werden, wenn der Webserver abgesichert ist und sichergestellt ist, daß jeder, der auf die CGIs Zugriff hat auch berechtigt und authentiziert ist. Wenn diese Variable definiert ist hat jeder, der nicht durch den Webserver authentiziert wurde, alle Rechte dieses Default-Benutzers!

Optionen der Hostkonfigurationsdatei

Bemerkung

Beim Anlegen oder Ändern von Konfigurationsdateien sollte folgendes beachtet werden:

1. Zeilen die mit dem '#' Zeichen beginnen werden als Kommentar betrachtet und deshalb nicht verarbeitet
2. Varibalenbezeichner müssen am Zeilenanfang beginnen - es ist kein Leerzeichen vor dem Bezeichner erlaubt
3. Bei Variablenbezeichnern wird die Groß- und Kleinschreibung unterschieden

Beispielkonfiguration

Eine beispielhafte Hauptkonfigurationsdatei kann mit dem 'make config' Befehl erzeugt werden. Der vorgegebene (default) Name der Hauptkonfigurationsdatei ist netsaint.cfg - zu finden ist diese Datei im NetSaint-Distributionsverzeichnis oder im etc/ Verzeichnis der Installation.

Beziehung zwischen den Daten

Um besser zu verstehen, wie Hosts, Hostgruppen, Kontakte, Kontaktgruppen, Dienste usw. zueinander in Beziehung stehen wurden ein paar Diagramme erstellt. Sie sind zu finden in dem Abschnitt Theorie der Operation der Dokumentation.

Inhalt

Hostdefinition
Host group definitions
Contact definitions
Contact group definitions
Command definitions
Service definitions
Time period definitions
Service escalation definitions
Hostgroup escalation definitions

Hostdefinition

Format: **host[<host_name>]=<host_alias>;<address>;<parent_hosts>;<host_check_command>;<max_attempts>;<notification_interval>;<notification_period>;<notify_recovery>;<notify_down>;<notify_unreachable>;<event_handler>**

Beispiel: **host[es-gra]=ES-GRA Server;192.168.0.1;;check-host-alive;3;120;24x7;1;1;1;**

Eine Hostdefinition wird benutzt um einen physikalischen Server, Workstation, Gerät oder ähnliches zu definieren, das sich im betreffenden Netzwerk befindet. Die verschiedenen Argumente einer Hostdefinition werden unten beschrieben.

<host_name>	Ein Kurzname um den Host zu identifizieren. Er wird benutzt in Hostgruppen und Dienstdefinitionen um den entsprechenden Host zu referenzieren. Hosts können mehrere Dienste (die überwacht werden) betreiben. Bei korrekter Benutzung enthält das $HOSTNAME$ macro diesen Kurznamen.
<host_alias>	Ein längerer Name oder eine Beschreibung um den Host zu identifizieren. Dieser Alias ist vorgesehen um einfacher einen bestimmten Host identifizieren zu können. Bei korrekter Benutzung enthält das $HOSTALIAS$ macro diesen Alias oder diese Beschreibung.
<address>	Die IP-Adresse des Hosts. Es kann der FQDN benutzt werden um eine Host zu identifizieren, aber wenn DNS nicht verfügbar ist kann dies Probleme verursachen. Bei korrekter Benutzung enthält das $HOSTADDRESS$ macro diese Adresse.
<parent_hosts>	This is a comma-delimited list of short names of the "parent" hosts for this particular host. Parent hosts are typically routers, switches, firewalls, etc. that lie between the monitoring host and a remote hosts. A router, switch, etc. which is closest to the remote host is considered to be that host's "parent". Read the "Determining Status and Reachability of Network Hosts" document in the theory of operation section for more information. If this host is on the same network segment as the host doing the monitoring (without any intermediate routers, etc.) the host is considered to be on the local network and will not have a parent host. Leave this value blank if the host does not have a parent host (i.e. it is on the same segment as the NetSaint host). The order in which you specify parent hosts has no effect on how things are monitored. However, the statusmap and statuswrl CGIs will use the first parent host that you specify as the primary parent for purposes of drawing only.
<host_check_command>	This is the *short name* of the command that should be used to check if the host is up or down. Typically, this command would try and ping the host to see if it is "alive". The command must return a status of OK (0) or NetSaint will assume the host is down. If you leave this argument blank, the host will not be checked - NetSaint will always assume the host is up. This is useful if you are monitoring printers or other devices that are frequently turned off.
<max_attempts>	This is the number of times that NetSaint will retry the host check command if it returns any state other than an OK state. Setting this value to 1 will cause NetSaint to generate an alert without retrying the host check again. Note: If you do not want to check the status of the host, you must still set this to a minimum value of 1. To bypass the host check, just leave the **<host_check_command>** option blank.
<notification_interval>	This is the number of "time units" to wait before re-notifying a contact that this server is *still* down or unreachable. Unless you've changed the **interval_length** value in the main configuration file from the default value of 60, this number will mean minutes. If you set this value to 0, NetSaint will *not* re-notify contacts about problems for this host - only one problem notification will be sent out.
<notification_period>	This is the short name of the time period during which notifications of events for this host can be sent out to contacts. If a host goes down, becomes unreachable, or recoveries during a time which is not covered by the time period, no notifications will be sent out. Read the "Time Periods" document in the theory of operation section for more information.
<notify_recovery>	This value determines whether or not notifications should be sent to any contacts if the host is in a RECOVERY state. Set this value to **1** if notifications should be sent out about recovery states, **0** if they *shouldn't*. Note: If a contact is configured to not receive notifications of host recoveries, they will not be notified, regardless of this setting.
<notify_down>	This value determines whether or not notifications should be sent to any contacts if the host is in a DOWN state. Set this value to **1** if notifications should be sent out when the host goes down, **0** if they *shouldn't*. Note: If a contact is configured to not receive notifications about hosts that go down, they will not be notified, regardless of this setting.
<notify_unreachable>	This value determines whether or not notifications should be sent to any contacts if the host is in aa UNREACHABLE state. Set this value to **1** if notifications should be sent out when the host becomes unreachable, **0** if they *shouldn't*. Note: If a contact is configured to not receive notifications about unreachable hosts, they will not be notified, regardless of this setting.
<event_handler>	This is the *short name* of the command that should be run whenever a change in the state of the host is detected (i.e. whenever it goes down or recovers). Read the documentation on event handlers for a more detailed explanation of how to write scripts for handling events. If you do not wish to define an event handler for the host, leave this option blank (as shown in the example above).

Host Group Definition

Format: **hostgroup[<group_name>]=<group_alias>;<contact_groups>;<hosts>**

Beispiel: **hostgroup[nt-servers]=All NT Servers;nt-admins;rosie,dev,liatris**

A host group definition is used to group one or more hosts together for the purposes of simplifying notifications. Each host that you define must be a member of at least one host group - even if it is the only host in that group. Hosts can be in more than one host group. When a host goes down, becomes unreachable, or recovers, NetSaint will find which host group(s) the host is a member of, get the contact group for each of those hostgroups, and notify all contacts associated with those contact groups. This may sound complex, but for most people it doesn't have to be. It does, however, allow for flexibility in determining who gets paged for what kind of problems. The different arguments to a host group definition are outlined below.

<group_name>	This is a short name used to identify the host group.
<group_alias>	This is a longer name or description used to identify the host group. It is provided in order to allow you to more easily identify a particular host group.
<contact_groups>	This is a list of the *short names* of the contact groups that should be notified whenever there are problems (or recoveries) with any of the hosts in this host group. Multiple contact groups should be separated by commas.
<hosts>	This is a list of the *short names* of hosts that should be included in this group. Multiple host names should be separated by commas.

Contact Definition

Format: **contact[<contact_name>]=<contact_alias>;<svc_notification_period>;<host_notification_period>;<svc_notify_recovery>;<svc_notify_critical>;<svc_notify_warning>;lt;host_notify_recovery>;<host_notify_down>;<host_notify_unreachable>;<service_notify_commands>;<host_notify_commands>;<email_address>;<pager>**

Beispiel: **contact[egalstad]=Ethan Galstad;24x7;24x7;1;1;1;1;1;1;notify-by-email,notify-by-epager;host-notify-by-epager;egalstad@nospam.extension.umn.edu;pagegalstad@pagenet.com**

A contact definition is used to identify someone who should be contacted in the event of a problem on your network. The different arguments to a contact definition are described below.

<contact_name>	This is the short name used to identify the contact. It is referenced in contact group definitions. Under the right circumstances, the $CONTACTNAME$ macro will contain this value.
<contact_alias>	This is a longer name or description for the contact. Under the rights circumstances, the $CONTACTALIAS$ macro will contain this value.
<svc_notification_period>	This is the short name of the time period during which the contact can be notified about service problems or recoveries. You can think of this as an "on call" time for service notifications for the contact. Read the "Time Periods" document in the theory of operation section of the documentation for more information on how this works and potential problems that may result from improper use.
<host_notification_period>	This is the short name of the time period during which the contact can be notified about host problems or recoveries. You can think of this as an "on call" time for host notifications for the contact. Read the "Time Periods" document in the theory of operation section of the documentation for more information on how this works and potential problems that may result from improper use.
<svc_notify_recovery>	This value determines whether or not the contact will be notified of service recoveries. Set this value to **1** if the contact should be notified, **0** if they shouldn't. Note: If a service is configured to not send out notifications upon recovery, contacts will not be notified about recoveries for that service, regardless of this setting.
<svc_notify_critical>	This value determines whether or not the contact will be notified if a service is in a critical state. Set this value to **1** if the contact should be notified of critical states, **0** if they shouldn't. Note: If a service is configured to not send out notifications for critical states, contacts will not be notified about critical states for that service, regardless of this setting.
<svc_notify_warning>	This value determines whether or not the contact will be notified if a service is in either a warning or an unknown state. Set this value to **1** if the contact should be notified of warning/unknown states, **0** if they shouldn't. Note: If a service is configured to not send out notifications for warning/unknown states, contacts will not be notified about warning/unknown states for that service, regardless of this setting.
<host_notify_recovery>	This value determines whether or not the contact will be notified if any host recovers. Set this value to **1** if the contact should be notified of hosts that recover, **0** if they shouldn't. Note: If a host is configured to not send out notifications for recoveries, contacts will not be notified when the host recovers, regardless of this setting.
<host_notify_down>	This value determines whether or not the contact will be notified if any host goes down. Set this value to **1** if the contact should be notified of hosts that go down, **0** if they shouldn't. Note: If a host is configured to not send out notifications for down states, contacts will not be notified when the host goes down, regardless of this setting.
<host_notify_unreachable>	This value determines whether or not the contact will be notified if any host becomes unreachable. Set this value to **1** if the contact should be notified of hosts that become unreachable, **0** if they shouldn't. Note: If a host is configured to not send out notifications for unreachable states, contacts will not be notified when the host becomes unreachable, regardless of this setting.
<service_notify_commands>	This is a list of the *short names* of the commands used to notify the contact of a *service* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified.
<host_notify_commands>	This is a list of the *short names* of the commands used to notify the contact of a *host* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified.
<email_address>	This is the email address for the contact. Depending on how you configure your notification commands, it can be used to send out an alert email to the contact. Under the right circumstances, the $CONTACTEMAIL$ macro will contain this value. fs
<pager>	This is the pager number for the contact. It can also be an email address to a pager gateway (i.e. pagejoe@pagenet.com). Depending on how you configure your notification commands, it can be used to send out an alert page to the contact. Under the right circumstances, the $CONTACTPAGER$ macro will contact this value.

Contact Group Definition

Format: **contactgroup[<group_name>]=<group_alias>;<contacts>**

Beispiel: **contactgroup[nt-admins]=NT Administrators;egalstad,jdoe**

A contact group definition is used to group one or more contacts together for the purpose of sending out alert/recovery notifications. When a host or service has a problem or recovers, NetSaint will find the appropriate contact groups to send notifications to, and notify all contacts in those contact groups. This may sound complex, but for most people it doesn't have to be. It does, however, allow for flexibility in determining who gets notified for particular events. The different arguments to a contact group definition are outlined below.

<group_name>	Ein Kurzname um die Kontaktgruppe zu identifizieren.
<group_alias>	This is a longer name or description used to identify the contact group.
<contacts>	This is a list of the *short names* of contacts that should be included in this group. Multiple contact names should be separated by commas.

Command Definition

Format: **command[<command_name>]=<command_line>**

Beispiel 1: **command[check-host-alive]=/usr/local/netsaint/libexec/check_ping $HOSTADDRESS$ 100 100 1000.0 1000.0**

Beispiel 2: **command[check_pop]=/usr/local/netsaint/libexec/check_pop $HOSTADDRESS$**

Beispiel 3: **command[check_disk]=/usr/local/netsaint/libexec/check_disk 85 95 $ARG1$**

A command definition is just that. It defines a command. Commands that can be defined include service checks, service notifications, service event handlers, host checks, host notifications, and host event handlers. Command definitions can contain macros, but you must make sure that you include only those macros that are "valid" for the circumstances when the command will be used. More information on what macros are available and when they are "valid" can be found here. The different arguments to a command definition are outlined below.

<command_name>	This is a short name used to identify the command. It is referenced in contact, host, and service definitions.
<command_line>	This is what is actually executed by NetSaint when the command is used for service or host checks, notifications, or event handlers. Before the command line is executed, all valid macros are replaced with their respective values. See the documentation on macros for determining when you can use different macros. Note that the command line is *not* surrounded in quotes.

Service Definition

Format: **service[<host>]=<description>;<volatile>;<check_period>;<max_attempts>;<check_interval>;<retry_interval>;<contactgroups>;<notification_interval>;<notification_period>;<notify_recovery>;<notify_critical>;<notify_warning>;<event_handler>;<check_command>**

Beispiel 1: **service[rosie]=FTP;0;24x7;3;5;1;nt-admins;120;24x7;1;1;1;;check_ftp**

Beispiel 2: **service[dev]=HTTP;0;24x7;3;5;1;nt-admins;240;24x7;1;1;1;;check_http2!192.168.0.2!/!88**

Beispiel 3: **service[real]=Zombie Processes;0;24x7;3;5;1;linux-admins;240;24x7;1;1;1;;check_procs!5!10!Z**

A service definition is used to identify a "service" that runs on a host. The term "service" is used very loosely. It can mean an actual service that runs on the host (POP, SMTP, HTTP, etc.) or some other type of metric associated with the host (response to a ping, number of logged in users, free disk space, etc.). The different arguments to a service definition are outlined below.

<host>	This is the *short name* of the host that the service "runs" on or is associated with.
<description>	A description of the service, which may contain spaces, dashes, and colons (semicolons, apostrophes, and quotation marks should be avoided). No two services associated with the same host can have the same description.
<volatile>	This field is used to denote whether the service is "volatile". Services are normally *not* volatile. More information on volatile service and how they differ from normal services can be found here. Set this field to **1** to mark the service as being volatile, **0** to mark it as a normal service.
<check_period>	This is the short name of the time period that identifies when this service can be checked. Services checks are scheduled in such a way that they are only checked (or rechecked) during times that are valid within the specified service check time period. See the "Time Periods" documentation in the theory of operation section for more information on how time periods works and potentials problems with using them improperly.
<max_attempts>	This is the number of times that NetSaint will retry the service check if it returns any state other than an OK state. Setting this value to 1 will cause NetSaint to generate an alert (if the service check detected a problem) without retrying the service check again.
<check_interval>	This is the number of "time units" to wait before scheduling the next "regular" check of the service. "Regular" checks are those that occur when the service is in an OK state or when the service is in a non-OK state, but has already been rechecked **max_attempts** number of times. Unless you've changed the **interval_length** value in the main configuration file from the default value of 60, this number will mean minutes.
<retry_interval>	This is the number of "time units" to wait before scheduling a re-check of the service. Services are rescheduled at the retry interval when the have changed to a non-OK state. Once the service has been retried **max_attempts** times without a change in its status, it will revert to being scheduled at its "normal" rate as defined by the **check_interval** value. Unless you've changed the **interval_length** value in the main configuration file from the default value of 60, this number will mean minutes.
<contactgroups>	This is a comma-delimited list of the short names of contact groups that should be notified about problems or recoveries for this service. If a problem or recovery occurs for this service, NetSaint will attempt to notify all the contacts in each contact group (depending on the notification options that are set below).
<notification_interval>	This is the number of "time units" to wait before re-notifying a contact that this service is *still* at a non-OK state. Unless you've changed the **interval_length** value in the main configuration file from the default value of 60, this number will mean minutes. If you set this value to 0, NetSaint will *not* re-notify contacts about problems for this service - only one problem notification will be sent out.
<notification_period>	This is the short name of the time period that identifies when notifications about problems or recoveries for this service may be sent out. If a service problem or recovery occurs outside valid times within this time period, notifications will not be sent out. See the "Time Periods" documentation in the theory of operation section for more information on how time periods works and potentials problems with using them improperly.
<notify_recovery>	This value determines whether or not alert notifications will be generated if the service recovers from a non-OK state. Set this value to **1** if the service should generate alerts for recoveries, **0** if it shouldn't. Note: If a contact is configured to not receive recovery notifications, they will not be notified of any recoveries for this service, regardless of this setting.
<notify_critical>	This value determines whether or not alert notifications will be generated if the service is in a CRITICAL state. Set this value to **1** if the service should generate alerts for critical states, **0** if it shouldn't. Note: If a contact is configured to not receive critical notifications, they will not be notified of any critical states for this service, regardless of this setting.
<notify_warning>	This value determines whether or not alert notifications will be generated if the service is in a WARNING or UNKNOWN state. Set this value to **1** if the service should generate alerts for warning/unknown states, **0** if it shouldn't. Note: If a contact is configured to not receive warning/unknown notifications, they will not be notified of any warning/unknown states for this service, regardless of this setting.
<event_handler>	This is the *short name* of the command that should be run whenever a change in the status of the services is detected (i.e. whenever it goes down or recovers). Read the documentation on event handlers for a more detailed explanation of how to write scripts for handling events. If you do not wish to define an event handler for the service, leave this option blank (as shown in the examples above).
<check_command>	This is the command that NetSaint will run in order to check the status of the service. There are three command formats that can be used: **1. "Vanilla" Command:** The command name is just the name of command that was previously defined. Example 1 above shows this type of command. **2. Command w/ Arguments:** This is basically the same as the "vanilla" command style, but with command options separated by a **!** character. Example 2 above shows this type of command. Arguments are separated from the command name (and other arguments) with the **!** character. The command should be defined to make use of the $ARGx$ macros. In Example 2 above, $ARG1$ would resolve to **134.84.92.128**, $ARG2$ would resolve to /, and $ARG3$ would resolve to **88** for that particular service. Note: NetSaint will handle a maximum of sixteen command line arguments ($ARG1$ through $ARG16$). **3. "Raw" Command Line:** You may optionally specify an actual command line to be executed. To do so you must enclose the entire command line in double quotes. The outer double quotes will be stripped off before the command is actually executed. No macros are processed inside of raw command lines. Note: I haven't really tested this format too much, but it should work. Remember that the command must return a proper status level. See the documentation on writing plugins for numeric codes for each status level.

Time Period Definition

Format: **timeperiod[<timeperiod_name>]=<timeperiod_alias>;<sunday_ranges>;<monday_ranges>;<tuesday_ranges>;<wenesday_ranges>;<thursday_ranges>;<friday_ranges>;<saturday_ranges>;**

Example 1: **timeperiod[24x7]=All Day, Every Day;00:00-24:00;00:00-24:00;00:00-24:00;00:00-24:00;00:00-24:00;00:00-24:00;00:00-24:00**

Example 2: **timeperiod[workhours]="Normal" Working Hours;;09:00-17:00;09:00-17:00;09:00-17:00;09:00-17:00;09:00-17:00;**

Example 3: **timeperiod[none]=No Time Is A Good Time;;;;;;;**

Example 4: **timeperiod[nonworkhours]=Non-Work Hours;00:00-24:00;00:00-09:00,17:00-24:00;00:00-09:00,17:00-24:00;00:00-09:00,17:00-24:00;00:00-09:00,17:00-24:00;00:00-09:00,17:00-24:00;00:00-24:00**

A time period is a list of times during various days that are considered to be "valid" times for notifications and service checks. It consists one or more time periods for each day of the week that "rotate" once the week has come to an end. Exceptions to the normal weekly time range rotations are not suported.

<timeperiod_name>	This is a short name used to identify the time period.
<timeperiod_alias>	This is a longer name or description used to identify the time period.
<xday_ranges>	This is a comma-delimited list of time ranges that are "valid" times for a particular day of the week. Notice that there are seven different days for which you must define time ranges (Sunday through Saturday). Each time range is in the form of **HH:MM-HH:MM**, where hours are specified on a 24 hour clock. For example, **00:15-24:00** means 12:15am in the morning for this day until 12:20am midnight (a 23 hour, 45 minute total time range). If you leave a particular day's time range blank, it means that there are no "valid" times for that day.

Service Escalation Definition

Format: **serviceescalation[<host>;<description>]=<first_notification>-<last_notificiation>;<contact_groups>**

Beispiele: **serviceescalation[real;Zombie Processes]=3-5;linux-admins,managers**
serviceescalation[dev;HTTP]=6-0;nt-admins,managers,everyone

A service escalation definition is *completely optional* and is used to escalate notifications for a particular service. More information on how notification escalations work can be found here.

<host>	This is the *short name* of the host that the service "runs" on or is associated with.
<description>	A description of the service, which may contain spaces, dashes, and colons (semicolons, parentheses, and apostrophes are not allowed). No two services associated with the same host can have the same description.
<first_notification>	This is a number that identifies the *first* notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third escalation to go out.
<last_notification>	This is a number that identifies the *last* notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for the specified service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
<contact_groups>	This is a list of the *short names* of the contact groups that should be notified when a service notification is escalated. Multiple contact groups should be separated by commas.

Host Group Escalation Definition

Optionen der Hostkonfigurationsdatei

Format: **hostgroupescalation[<group_name>]=<first_notification>-<last_notificiation>;<contact_groups>**

Beispiele: **hostgroupescalation[nt-servers]=3-5;nt-admins,managers**
hostgroupescalation[nt-servers]=6-0;nt-admins,managers,everyone

A host group escalation definition is *completely optional* and is used to escalate notifications for hosts in a particular hostgroup. More information on how notification escalations work can be found here.

<group_name>	This is a short name used to identify the host group (as previously defined in a hostgroup definition) that the escalation should apply to.
<first_notification>	This is a number that identifies the *first* notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if a host in the hostgroup is down or unreachable long enough for a third escalation to go out.
<last_notification>	This is a number that identifies the *last* notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for any particular host in the specified hostgroup. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).
<contact_groups>	This is a list of the *short names* of the contact groups that should be notified when a host notification is escalated. Multiple contact groups should be separated by commas.

Author: Date: 00.00.00 00:00 Normal Receipt Requested Subject: ---------- Message Contents

Die Philosophie hinter NetSaint

Obwohl das generelle Konzept von NetSaint relativ einfach zu verstehen ist, sind die internen Strukturen von Netsaint manchmal etwas komplizierter.
Um diese Strukturen und den Source-Code von NetSaint besser zu verstehen wurde dieses Kapitel geschrieben. Es ist zwar noch nicht so detailliert, wie es sein sollte, dafür wird dieses Kapitel weiter ausgebaut.

Erreichbarkeit und Status-Erkennung von Hosts

Klicke hier um nachzulesen, wie NetSaint die Erreichbarkeit und den Status von Hosts ermittelt und was der "zentrale" NetSaint- Server ist.

Netzwerk-Ausfälle

Lese hier nach, wie NetSaint herausfindet, welche Hosts ausfälle in Deinem Netzwerk verursachen. Vor allem geht es dabei um die Arbeitsweise des CGIs, die diese Ausfälle erkennen, es bleibt aber bei einer Schnelleinführung.

Benachrichtigungen

Klicke hier um zu lesen, wie die NetSaint bei Ausfällen die entsprechenden Stellen benachrichtigt. Vor allem wird hier beschrieben, welche Benachrichtigungwege es gibt, wie sie vorab gefiltert werden müssen, bevor sie letztendlich zu den einzelnen Personen eingeleitet werden können.

PlugIn-Theorie

Klicke hier um nachzulesen, wie die PlugIns die Überprüfungen für den NetSaint-Server übernehmen.

Check-Zeitpläne

Hier geht es darum wie die Checks geplant werden und wie sich die einzelnen Zeitpläne unterscheiden, wenn die Checks letztendlich ausgeführt werden und ihre Ergebnisse mitteilen.

Status Typen

Klicke hier, um nachzulesen, was "softe" und "harte" Status-Typen sind, wann sie auftreten und welche Rolle sie in der Überwachungs-Logik spielen.

Zeit-Abstände

Klicke hier , wie die Definition von Zeitabständen die Überwachung von Diensten und Hosts sowie die Benachrichtigungen beeinflussen. Dieser Teil beschreibt vor allem die potentiellen Probleme, die auftauchen können, wenn man Zeit-Abstände definiert. Vor allem sollte man hier weiterlesen, wenn man Zeit-Abstände definiert, die nicht 24 Stunden, 7 Tage die Woche durchgehen.

Author: Date: 00.00.00 00:00 Normal Receipt Requested Subject: ---------- Message Contents

Bestimmung des Status und der Erreichbarkeit von Netzwerkhosts

---

Überwachung von Diensten abeschalteter oder unerreichbarer Hosts

Die Hauptaufgabe von NetSaint ist die Überwachung von Netzwerkdiensten die von physikalischen Rechnern oder anderen Geräten im Netzwerk bereitgestellt oder ausgeführt werden. Es ist offensichtlich, dass wenn ein Host oder ein Gerät im Netzwerk heruntergefahren wird auch sämtliche bereitgestellte Dienste heruntergefahren werden. Ebenso kann NetSaint Dienste eines Hosts der unerreichbar geworden ist nicht mehr überwachen.

NetSaint weiss um diese Tatsache und versucht dieses Szenario zu überprüfen wenn Probleme mit einem Dienst auftreten. Immer wenn ein Servicecheck (Dienstüberprüfung) ein Nicht-OK Resultat zurückliefert, wird NetSaint überprüfen, ob der Host, auf dem dieser Dienst bereitgestellt wird, überhaupt noch "lebt". Dies wird typischerweise durch das "Anpingen" und Auswerten der Antwort des entsprechenden Hosts gemacht. Wenn diese Überprüfung des Hosts auch ein Nicht-OK Resultat ergibt, geht NetSaint davon aus, dass ein Problem mit diesem Host vorliegt.
In dieser Situation wird NetSaint alle definierten Alarmbenachrichtigungen für Dienste die dieser Host bereitstellt "stillegen" und nur die entsprechenden Benachrichtigungen auslösen die definiert wurden für den Fall, dass dieser Host abgeschaltet oder unerreichbar wird. Wenn die Überprüfung des Hosts ein OK Resultat ergibt erkennt NetSaint, dass dieser Host "lebt" und wird Benachrichtigungen für den Dienst auslösen der nicht korrekt arbeitet.

Lokale Hosts

"Lokale" Hosts sind Hosts die sich im selben Netzwerksegment befinden wie der Host auf dem NetSaint läuft - keine Router oder Firewalls liegen dazwischen.
Bild 1 zeigt ein Beispielnetzwerk. Auf Host A wird NetSaint ausgeführt und überwacht alle anderen Hosts und Routers die im Bild dargestellt sind. Die Hosts B, C, D, E und F werden alle als "lokal" in Bezug auf Host A betrachtet.

Die <*parent_host*> Option in der Host Definition (host defintion) für einen "lokalen" Host sollte leergelassen werden, da lokale Hosts keine "Eltern" haben - deshalb sind sie ja lokal.

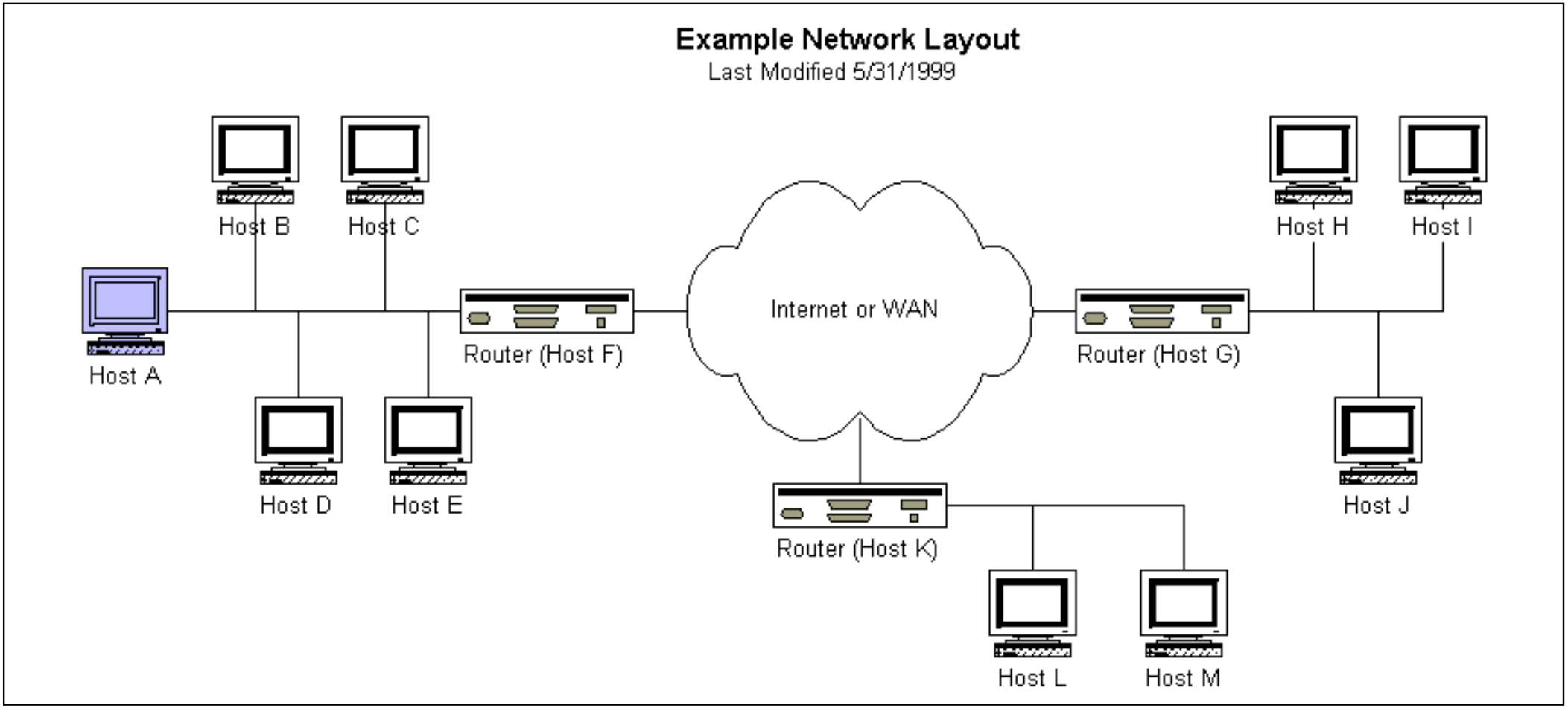
Überwachen Lokaler Hosts

Das Überprüfen von Hosts die sich im lokalen Netzwerk befinden ist recht einfach. Falls nicht jemand unbewusst (oder beabsichtigt) das Netzwerkkabel eines der Hosts aussteckt kann eigentlich nicht sehr viel schiefgehen in Bezug auf die Überprüfung der Netzwerkverbindung. Es gibt keine Routers oder

externe Netzwerke zwischen dem Host, der die Überprüfung durchführt und den anderen Hosts im lokalen Netzwerke.

Um zu überprüfen, ob ein lokaler Host "lebt", wird NetSaint einfach den dafür definierten Host Ürprüfungsbefehl (host check command) ausführen. Wenn der Befehl ein OK zurückliefert nimmt NetSaint an, dass der Host läuft, wenn der Befehl ein anderes Ergebnis zurückgibt nimmt NetSaiont an, der Host wäre abgeschaltet.

Bild 1.



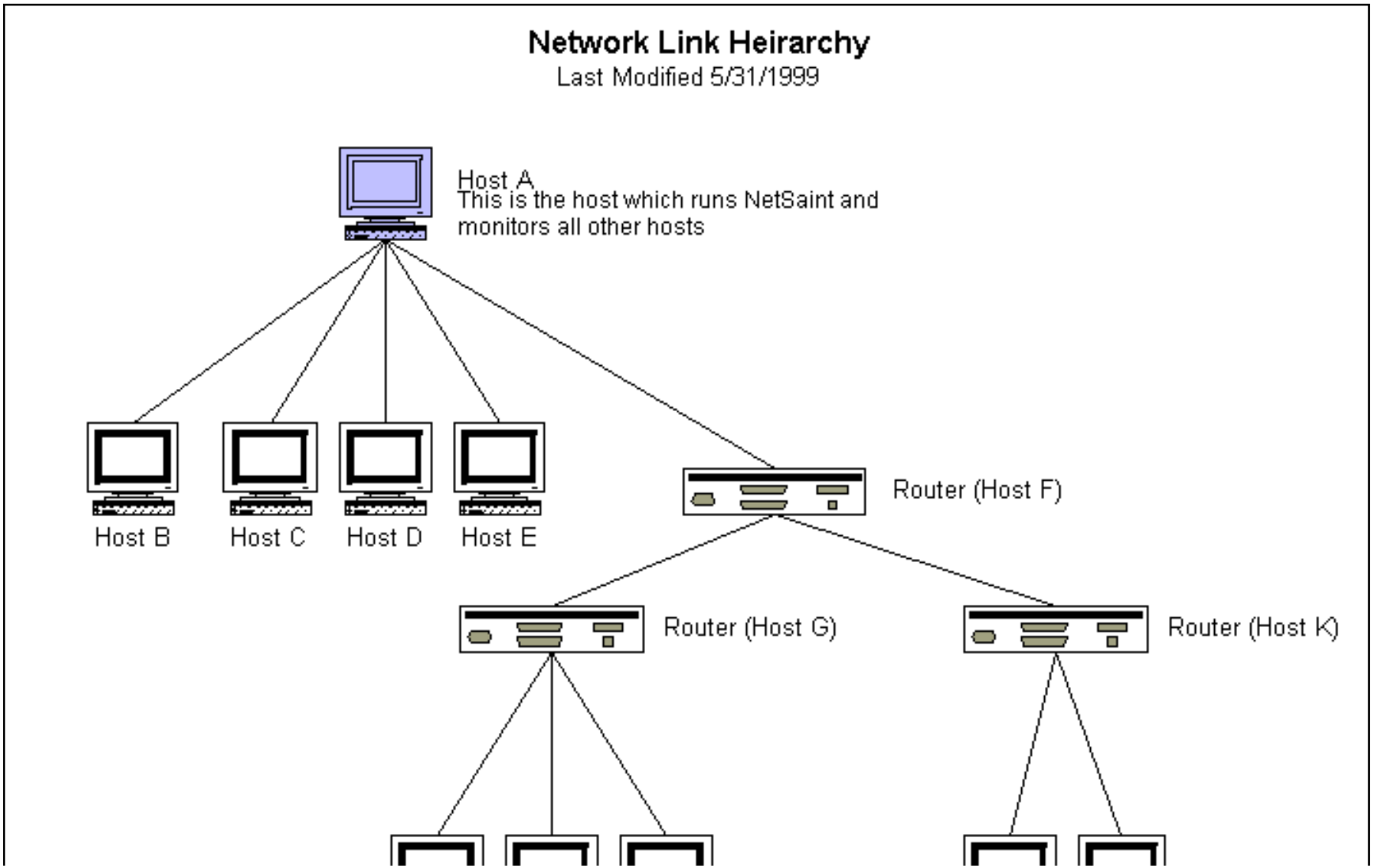
Enfernte Hosts

"Entfernte" Hosts sind Hosts die sich in einem anderen Netzwerksegment befinden als der Host auf dem NetSaint läuft. Im Bild oben, die Hosts G, H, I, J, K, L und M werden als "entfernte"Hosts in Bezug zu A betrachtet.

Zu bemerken ist, dass einige Hosts "weiter weg" als andere sind. Die Hosts H, I und J sind einen Sprung (hop) weiter entfernt von Host A als Host G (der Router). Aus dieser Beobachtung lässt sich eine Abhängigkeit zwischen den Hosts konstruieren wie sie im Bild 2 unten gezeigt ist. Dieses Baumdiagramm hilft bei der Konfiguration der einzelnen Hosts in NetSaint.

Die **<parent_host>** Option in der Host Definition (host defintion) für einen "entfernten" Host sollte der Kurzname des Hosts direkt über ihm im Baumdiagramm sein (wie unten dargestellt). So ist, zum Beispiel, der parent host für Host H der Host G. Der parent host für Host G ist Host F. Host F hat keinen parent host, denn er ist im selben Netzwerksegement wie Host A, es handelt sich um einen "lokalen" Host.

Bild 2.



Überwachung entfernter Hosts

Die Überprüfung des Status entfernter Hosts ist etwas komplizierter als die für lokale Hosts. Sollte NetSaint einen Service auf einem entfernten Host nicht überprüfen können, muss NetSaint herausfinden, ob der Host selbst down ist, oder "nur" das Netwerk nicht erreichbar, indem sich der Host befindet.
Zum Glück erlaubt dies (seit NetSaint 0.0.4) die **<parent_host>**-Option.

Sobald ein Host Ürprüfungsbefehl für einen entfernten Host einen "Nicht-Ok"-Status meldet, wird NetSaint den "Abhängigkeitsbaum" (siehe Bild 2) heraufgehen bis es an der Spitze oder bei dem Eltern-Host angekommen ist, der einen "OK"-Status meldet.
So kann NetSaint herausfinden, ob ein Problem mit einem Dienst nur an dessen Bereitstellung oder an dem Host liegt, der den Dienst hostet.

NetSaint Status Levels

Different status levels (also referred to as "states") for hosts and services are listed below. Some states are internal to NetSaint and cannot be generated by external plugins. Plugins are only capable of returning OK, UNKNOWN, WARNING, and CRITICAL states. See the documentation on writing plugins for more information

Service Status Levels

Status	Description
PENDING	This status level indicates that the service has not been checked yet. Pending status levels occur only after NetSaint is started and will disappear as services are checked.
OK	This status indicates that the service being monitored appears to be both running and functioning properly.
RECOVERED	This status indicates that the service is functioning properly at the moment, but that at the last check it was at either a warning, an unknown, or a critical status. In other words, it just came back up.
WARNING	This status indicates that the service being monitored appears to have some problems, but is still in a semi-functional state.
UNKNOWN	This status indicates that there was some sort of internal error with the plugin that prevented it from checking the status of a service. For the purposes of notification, unknown status levels are considered to be the same as warning status levels.
CRITICAL	This status indicates that either there is a big problem with the service being checked or that the service is completely unavailable.
UNREACHABLE	This status indicates that the service cannot be checked because the host that it is associated with is unreachable.
HOST DOWN	In the status CGI this indicates that the host associated with the service was down the last time the service was checked.

Host Status Levels

Status	Description
PENDING	This status level indicates that the status of the host is unknown, because no services associated with it have been checked yet. Pending status levels occur only when NetSaint is started, and will disappear as soon as at least one service associated with the host is checked.
UP	This status level indicates that the host appears to be up.
DOWN	This status level indicates that the host is down.

UNREACHABLE	This status level indicates that the host is unreachable because a host that it relied on (i.e. a parent or grandparent host) was down.

Plugin Development Guidelines

Plugin development for NetSaint has been moved over to SourceForge. The NetSaint plugin development project page can be found here.

The latest version of the plugin developers guide can be found at
http://netsaintplug.sourceforge.net/doc/developer-guidelines.html

---

Author: Date: 00.00.00 00:00 Normal Receipt Requested Subject: ---------- Message Contents

Netzwerk Ausfälle

Einleitung:

Das "Ausfall-CGI" wurde in NetSaint 0.0.6 hinzugefügt um die Ursache für Netzwerkausfälle zielgenauer bestimmen zu können. In kleinen Netzwerken ist dieses CGI vielleicht nicht sehr nützlich, aber dafür umso mehr in grösseren. Mit dem CGI kann ein Admin den wahren Grund für Netzwerkausfälle herauszufinden.

Allerdings sollte man beachten, dass das CGI nicht den *exakten* Grund ermitteln, aber zumindest den Host herausfinden kann, der ein Netzwerk unerreichbar macht und somit eine Vielzahl von Problemen nach sich zieht.
Den wahren Grund für den Ausfall zu ermitteln liegt dann in der Hand des Administrators.

Diagramme

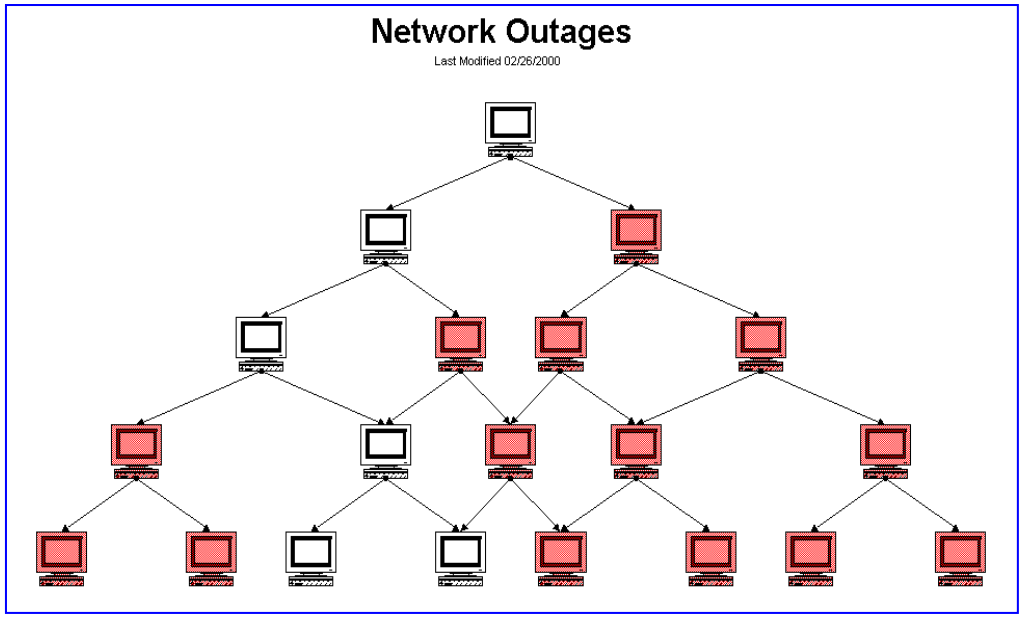
Die Diagramme etwas weiter unten sollen helfen zu verstehen, wie das Ausfall-CGI bei der Bestimmung des Grundes für einen Netzwerk-Ausfall vorgeht.
Um eine grössere Version des Diagramms zu bekommen, einfach auf das Bild klicken...

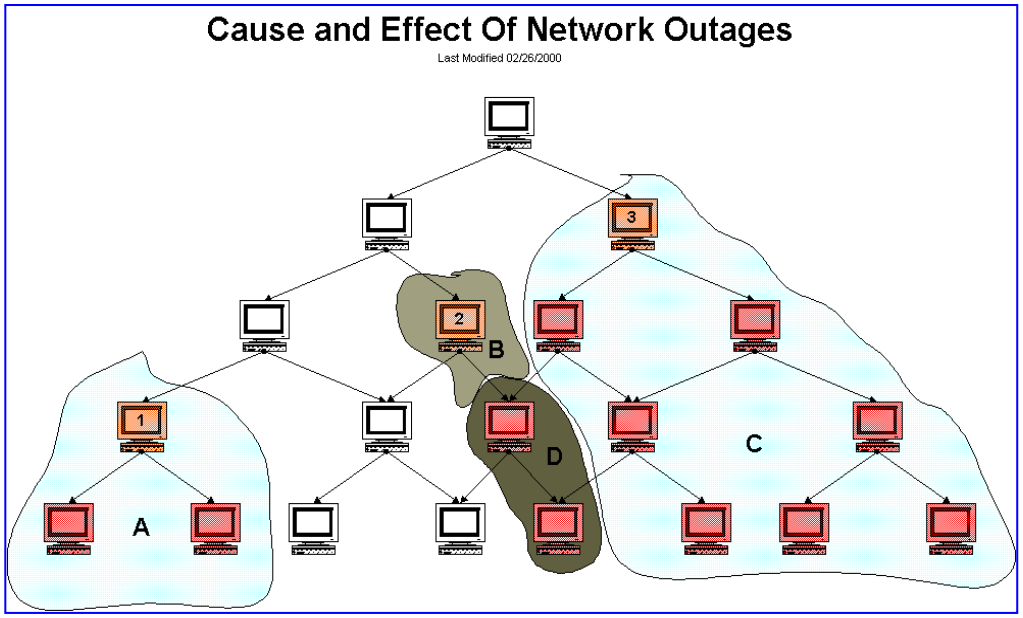
Diagram 1

Dieses Diagramm dient als Basis für unser Beispiel. Alle roten Hosts sind entweder aus der Sicht von NetSaint nicht erreichbar oder abgeschaltet.
Alle anderen Hosts sind ok.

Diagram 2

Dieses Diagramm zeigt Netzwerkausfälle (aus der Sicht von NetSaint) an und zeigt die verschiedenen Gruppen von Hosts, die durch diese Ausfälle betroffen sind.





Den Grund für einen Netzwerkausfall herausfinden

Aber wie findet das Ausfall-CGI nun den "Verursacher" eines solchen Problems heraus? *"Verursacher" müssen entweder abgeschaltet oder nicht erreichbar sein* ***und*** *der direkte Eltern-Host muss up sein*. Hosts, die in diese Kriterien erfüllen sind potentielle Verursacher.

Um herauszufinden, ob diese möglichen Verursacher wirklich ganze Netzwerk-Ausfälle nach sich ziehen, müssen noch ein paar Tests gemacht durchgeführt werden...

Falls *alle* der "Kind-Hosts" eines Verursachers nicht erreichbar oder abgeschaltet sind ist dieser Host ein Verursacher eines Netzwerk-Ausfalls. Erfüllt einer der "Kind-Hosts" dieses Kriterium allerdings nicht und meldet einen "Ok-Status", ist dieser Host *NICHT* der Verursacher eines Netzwerk-Ausfalls

Die Bestimmung der Folgen von Netzwerk-Ausfällen

Das Ausfall-CGI zeigt nicht nur welcher Host einen kompletten Netzwerk-Ausfall verursacht, sondern zeigt auch, wieviele Services und Hosts durch diesen Netzwerk-Ausfall betroffen sind.
Wie wird dieses bestimmt? Ein Blick auf das Diagramm 2 etwas weiter oben erklärt es...

Aus dem Diagramm ist klar ersichtlich, dass Host 1 zwei "Kind-Hosts" in der Domäne A blockiert. Host 2 ist nur dafür verantwortlich, dass er sich selbst in der Domäne B blockiert und Host 3 ist verantwortlich für die Blockierung von sieben anderen Hosts in der Domäne C.
Der Verursacher für die beiden nicht erreichbaren Hosts in Domäne ist allerdings unklar, da Domäne D einen "Eltern-Hosts" in Domäne C und einen in Domäne B aufweisen. Sollte einer der Hosts 2 oder 3 erreichbar sein, wäre die Domäne D nicht blockiert.

Die Anzahl der betroffenen Hosts, die jeder "Verursacher" mit sich bringt (der Verursacher ist in den Zahlen bereits eingerechnet):

- Host 1: 3 betroffene Hosts
- Host 2: 3 betroffene Hosts
- Host 3: 10 betroffene Hosts

Die Reihenfolge der Verursacher nach Dringlichkeits-Level

Das "Ausfall-CGI" zeigt alle problematischen Hosts, egal ob Sie ein Netzwerk-Ausfall nach sich ziehen oder nicht. Ausserdem zeit das CGI wie viele der problematischen Hosts (falls überhaupt) einen Netzwerk-Ausfall verursachen.

Um die problematischen Hosts wenigstens in einer sinnvollen Reihenfolge anzuzeigen, werden sie nach einer Art "Dringlichkeit" sortiert. Verursacht ein Host einen Netzwerk-Ausfall hat dieser natürlich eine höhere Dringlichkeit, als ein Host, der dieses nicht tut.
Die Dringlichkeit wird durch zwei Dinge bestimmt:

- Die Anzahl der nachfolgenden Hosts und nachfolgenden Dienste, die durch den Ausfall eines Hosts betroffen sind. Hosts werden dabei stärker bewertet als Dienste.
- Der aktuelle Code setzt die Gewichtung mit 4:1 fest (Hosts sind also vier Mal wichtiger als einzelne Dienste).

Angenommen das alle Hosts in Diagramm 2 die gleiche Anzahl von Diensten hosten, würde Host 3 die höchste Dringlichkeit bekommen, Hosts 1 und 2 wären auf dem gleichen Level.

Optionen der CGI Konfigurationsdatei

Bemerkung

Beim Anlegen oder Änder von Konfigurationsdateien sollte folgendes beachtet werden:

1. Zeilen die mit dem '#' Zeichen beginnen werden als Kommentar betrachtet und deshalb nicht verarbeitet
2. Varibalenbezeichner müssen am Zeilenanfang beginnen - es ist kein Leerzeichen vor dem Bezeichner erlaubt
3. Bei Variablenbezeichnern wird die Groß- und Kleinschreibung unterschieden

Beispielkonfiguration

Eine beispielhafte Hauptkonfigurationsdatei kann mit dem '**make config**' Befehl erzeugt werden. Der vorgegebene (default) Name der Hauptkonfigurationsdatei ist **netsaint.cfg** - zu finden ist diese Datei im NetSaint-Distributionsverzeichnis oder im etc/ Verzeichnis der Installation.

Inhalt

Speicherort der Hauptkonfigurationsdatei
Physikalischer HTML Pfad
URL HTML Pfad
Process check command
Authentication usage
Default user name
System/process information access
System/process command access
Configuration information access
Global host information access
Global host command access
Global service information access
Global service command access
Extended host information
Alert window suppression
CGI refresh rate
Maximum LIFO size
Audio alerts

Speicherort der Hauptkonfigurationsdatei

Format: **main_config_file=<file_name>**

Beispiel: **main_config_file=/usr/local/netsaint/etc/netsaint.cfg**

Hier wird der Speicherort der Hauptkonfigurationsdatei festgelegt. Die CGIs benötigen diese Angabe um Informationen über die Konfiguration, gegenwärtige Host- und Dienstestati usw. einholen zu können.

Physikalischer HTML Pfad

Format: **physical_html_path=<path>**

Beispiel: **physical_html_path=/usr/local/netsaint/share**

Dies ist der tatsächliche (physikalische) HTML Pfad, unter dem die HTML-Dateien für NetSaint auf der Workstation oder dem Server bereitgehalten werden. NetSaint geht davon aus, daß die Dokumentation und die Bilddateien (die von den CGIs benötigt werden) in den entsprechenden Unterverzeichnissen mit den Namen *docs/* und *images/* gespeichert sind.

URL HTML Pfad

Format: **url_html_path=<path>**

Beispiel: **url_html_path=/netsaint**

Dieser Wert sollte */netsaint* sein, wenn auf NetSaint über einen Webbrowser mit der URL **http://www.myhost.com/netsaint** ein Zugriff möglich sein soll. Im Wesentlichen ist dieser Wert also die "Pfadangabe" innerhalb der URL, mit der auf die NetSaint HTML-Seiten zugegriffen werden soll.

Process Check Command

Format: **process_check_command=<command_line>**

Beispiel: **process_check_command=/usr/local/netsaint/libexec/check_netsaint /usr/local/netsaint/var/status.log 5 '/usr/local/netsaint/bin/netsaint -d /usr/local/netsaint/etc/netsaint.cfg'**

Diese Variable definiert den Befehl, den die CGIs benutzen um den Status des NetSaint Prozesses zu überprüfen. Das gibt den CGIs (und auch den Benutzern) Hinweise darauf, ob NetSaint noch läuft oder nicht. Wenn die CGIs nicht ermitteln können, ob NetSaint noch läuft werden einige Funktionen wie die externen extended information und command CGI Befehle möglicherweise nicht verfügbar sein. Der hier definierte process check Befehl sollte den Richtlinien für Plugins genügen.

Bemerkung:

- Das check_netsaint Plugin ist ideal um den Status von NetSaint sowie die "Frische" der Daten in der Statuslogdatei zu überprüfen. Ich empfehle es sehr zu benutzen.
- If you are running a chroot'ed web server, you will have to place the plugin (or whatever you're using) in the **sbin/** subdirectory of your NetSaint installation.

Authentication Usage

Format: **use_authentication=<0/1>**

Beispiel: **use_authentication=1**

Diese Option legt fest, ob die Authentizierung und Authorisierung des Webservers benutzt werden soll um zu

ermitteln zu welchen Informationen und Befehlen Benutzer Zugriff haben. Ich empfehle sehr diese Funktionalitäaet für die CGIs zu benutzen. Wenn entschieden wird, die Authentizierung nicht zu benutzen sollte unbedingt das command CGI entfernt werden, um zu verhindern, daß unberechtigte Benutzer Befehle an NetSaint ausgeben können. Das CGI wird keine Befehle an NetSaint ausgeben wenn die Authentizierung ausgeschaltet ist, aber ich schlage vor es trotzdem zu entfernen um wirklich sicher zu gehen. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

- 0 = Authentizierung wird nicht benutzt
- 1 = Authentizierung wird benutzt (default)

Default User Name

Format: **default_user_name=<username>**

Beispiel: **default_user_name=guest**

Das Setzen dieser Variablen erlaubt einem definierten Benutzer Zugriff auf die CGIs. Das erlaubt Nutzern innerhalb einer sicheren Domäne (z.B. hinter einer Firewall) auf die CGIs zuzugreifen ohne sich vorher beim Webserver zu anzumelden. Diese Funktion kann auch benutzt werden um die Basisauthentizierung (basic authentication) zu vermeiden, wenn kein sicherer (verschlüsselter) Webserver verwendet wird, da in diesem Fall die Basiauthentizierung Passwörter im Klartext überträgt.

Wichtig: Es sollte *kein* Vorgabebenutzer definiert werden, solange kein sicherer Webserver benutzt wird und nicht sichergestellt ist, daß jeder, der hiermit Zugriff zu den CGIs hat auch in irgendeiner Weise dazu berechtigt ist! Wenn dieser Vorgabebenutzer definiert wurde hat jeder, der nicht als anderer Benutzer angemeldet wurde, alle Rechte dieses Benutzers!

System/Process Information Access

Format: **authorized_for_system_information=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_system_information=netsaintadmin,theboss**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die System und Prozessinformationen des extended information CGI anschauen können. Benutzer dieser Liste sind *nicht* automatisch berechtigt System- oder Prozessbefehle auszugeben. Sollen Benutzer berechtigt werden System- oder Prozessbefehle auszugeben, so müssen sie der Variablen authorized_for_system_commands hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

System/Process Command Access

Format: **authorized_for_system_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_system_commands=netsaintadmin**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die System und Prozessbefehle mit dem command CGI ausgeben können. Benutzer dieser Liste sind *nicht* automatisch berechtigt System- oder Prozessinformationen anzuschauen. Sollen Benutzer berechtigt werden auch System- oder Prozessbefehleinformationen anzuschauen, so müssen sie der Variablen

authorized_for_system_information hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Configuration Information Access

Format: **authorized_for_configuration_information=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_configuration_information=netsaintadmin**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die Informationen zur Konfiguration mit dem configuration CGI anschauen dürfen. Benutzer dieser Liste sind berechtigt Informationen zu allen konfigurierten Hosts, Hostgruppen, Diensten, Kontakten, Kontakgruppen, Zeitperioden und Befehlen anzuschauen. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Global Host Information Access

Format: **authorized_for_all_hosts=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_hosts=netsaintadmin,theboss**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die Status- und Konfigurationsinformationen aller Hosts anschauen dürfen. Benutzer dieser Liste sind automatisch auch berechtigt Informationen aller Dienste anzuschauen. Benutzer dieser Liste sind *nicht* automatisch berechtigt Befehle für alle Hosts oder Deinste auszugeben. Sollen Benutzer berechtigt werden Befehle für alle Hosts und Dienste auszugeben, so müssen sie der Variablen authorized_for_all_host_commands hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Global Host Command Access

Format: **authorized_for_all_host_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_host_commands=netsaintadmin**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die Befehle für alle Hosts mit dem command CGI ausgeben können. Benutzer dieser Liste sind automatisch auch berechtigt Befehle für alle Dienste auszugeben. Benutzer dieser Liste sind *nicht* automatisch berechtigt Status- oder Konfigurationsinformationen für alle Hosts oder Dienste anzuschauen. Sollen Benutzer berechtigt werden Status- oder Konfigrationsinformationen für alle Hosts oder Dienste anzuschauen, so müssen sie der Variablen authorized_for_all_hosts hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Global Service Information Access

Format: **authorized_for_all_services=<user1>,<user2>,<user3>,...<user*n*>**

Beispiel: **authorized_for_all_services=netsaintadmin,theboss**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die Status- und Konfigurationsinformationen aller Dienste anschauen dürfen. Benutzer dieser Liste sind *nicht* automatisch berechtigt die Status- und Konfigrationsinformationen aller Hosts anzuschauen. Ebenso sind sie *nicht*

automatisch berechtigt Befehle für alle Dienste auszugeben. Sollen Benutzer berechtigt werden Befehle für Dienste auszugeben, so müssen sie der Variablen authorized_for_all_service_commands hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Global Service Command Access

Format: **authorized_for_all_service_commands=<user1>,<user2>,<user3>,...<user*n*>**

Beispiele: **authorized_for_all_service_commands=netsaintadmin**

Dies ist eine durch Kommas getrennte Liste von Namen *berechtigter Benutzer (authenticated users)* die Befehle für alle Dienste mit dem command CGI ausgeben können. Benutzer dieser Liste sind *nicht* automatisch berechtigt Befehle für alle Hosts auszugeben. Ebenso sind sie *nicht* automatisch berechtigt die Status- und Konfigurationsinformationen aller Hosts anzuschauen. Sollen Benutzer berechtigt werden Status- oder Konfigrationsinformationen für alle Dienste anzuschauen, so müssen sie der Variablen authorized_for_all_services hinzugefügt werden. Weitere Informationen wie Berechtigungen und Authentizierungen für die CGIs konfiguriert werden sind hier zu finden.

Extended Host Information

Format: **hostextinfo[<host_name>]=<notes_url>;<icon_image>;<vrml_image>;<gd2_image>;<alt_tag>**

Beispiel: **hostextinfo[router3]=/hostinfo/router3.html;cat5000.gif;cat5000.jpg;cat5000.gd2;Cisco Catalyst 5000**

Erweiterte Hostinformationen werden benutzt, um die Ausgabe der status, statusmap, statuswrl, und extinfo CGIs netter aussehen zu lassen. Sie haben keinen Einfluß auf das Monitoring und sind optional.

<host_name>	Der Kurzname des Hosts, wie in der Hostkonfigurationsdatei definiert.
<notes_url>	Eine optionale URL, die mehr Information zu diesem Host zur Verfügung stellt. Wenn eine eine URL angegeben wird, ist ein Link sichtbaryou will see a link that says "Notes About This Host" in the extended information CGI (when you are viewing information about the specified host). Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. */cgi-bin/netsaint/*). This can be very useful if you want to make detailed information on the host, emergency contact methods, etc available to other support staff.
<icon_image>	Der Name einer GIF, PNG, oder JPG Grafik, die mit dem Host verknüpft wird. Diese Grafik wird angezeigt in den status und extended information CGIs. Die Grafik passt am besten wenn sie 40x40 Pixel groß ist. Grafiken für Hosts werden im **logos/** Unterverzeichnis des HTML-Verzeichnisses gesucht (z. B.. */usr/local/netsaint/share/images/logos*).
<vrml_image>	Der Name einer GIF, PNG, oder JPG Grafik, die mit dem Host verknüpft wird. Diese Grafik wird als Texturgrafik für den definierten Host statuswrl CGI benutzt. Ungleich der Grafik für +x+die *<icon_image>* Variable, sollte diese am besten *nicht* transparent sein. Falls doch, wird das Hostobjekt etwas ulkig aussehen. Grafiken für Hosts werden im **logos/** Unterverzeichnis des HTML-Verzeichnisses gesucht (z. B. */usr/local/netsaint/share/images/logos*).

<gd2_image>	Der Name einer Grafik im GD2-Format, die mit dem Host verknüpft wird. Diese Grafik wird benutzt in der Grafik, die vom statusmap CGI erzeugt wird. GD2 Grafiken können aus PNG Grafiken mit dem **pngtogd2** Werkzeug aus Thomas Boutell's gd library erzeugt werden. Die GD2 Grafiken sollten *unkomprimiert* vorliegen, um die CPU-Last zu reduzieren, wenn das statusmap CGI den Netzwerkplan erzeugt. Die Grafik passt am besten wenn sie 40x40 Pixel groß ist. Diese Variable kann unbelegt bleiben wenn die statusmap CGI nicht benutzt wird. Grafiken für Hosts werden im **logos/** Unterverzeichnis des HTML-Verzeichnisses gesucht (z. B. */usr/local/netsaint/share/images/logos*).
<alt_tag>	Eine optionale Zeichenkette, die im ALT tag des durch das <*icon_image*> Argument definierten Bildes benutzt wird. Das ALT tag wird in den status und statusmap CGIs benutzt.

Alert Window Suppression

Format: **suppress_alert_window=<0/1>**

Beispiel: **suppress_alert_window=1**

Diese Option legt fest, ob das Hostalarmfenster im status CGI permanent unterdrückt wird. Diese Fenster wird normalerweise angezeigt, wenn ein oder mehrere Hosts nicht erreichbar oder ausgeschaltet (down) sind.

- 0 = Alarmfenster wird nicht unterdrückt und ggf. angezeigt (Vorgabeeinstellung)
- 1 = Alarmfenster wird unterdrückt und nie angezeigt

CGI Refresh Rate

Format: **refresh_rate=<rate_in_seconds>**

Beispiel: **refresh_rate=90**

Diese Option legt die Anzahl der Sekunden zwischen den Auffrischungen Seiten der status, statusmap, und extinfo CGIs fest.

Maximum LIFO size

Format: **max_lifo_size=<size_in_bytes>**

Beispiel: **max_lifo_size=524288**

Diese Option legt die maximale Größe des Speichers (in Bytes) fest, der benutzt wird um die Ausgabe in den showlog, history, und notifications CGIs umzukehren (neueste Einträge oben). Wenn Warnungen ausgegeben werden, es sei nicht genug Speicher für die CGIs da um die Logdaten umzukehren sollte dieser Wert erhöht werden. Die vorgegebene Größe des Speichers, die das LIFO nutzen kann ist 512KB.

Audio Alerts

Formate: **host_unreachable_sound=<sound_file>**
host_down_sound=<sound_file>
service_critical_sound=<sound_file>
service_warning_sound=<sound_file>
service_unknown_sound=<sound_file>

Beispiele: **host_unreachable_sound=hostu.wav**
host_down_sound=hostd.wav
service_critical_sound=critical.wav
service_warning_sound=warning.wav
service_unknown_sound=unknown.wav

Diese Option legt fest, welche Audiodatei der Browser abspielt, falls beim Betrachten des status CGI Dienst- oder Hostprobleme auftreten. Wenn mehrere Probleme auftreten wird die Audiodatei für den kritischsten Problemtyp abgespielt. Der kritischste Problemtyp ist ein oder mehrere unerreichbare Hosts, während der unkritischste Problemtyp den Fall darstellt, daß ein oder mehrere Dienste in einem unbekannten (UNKNOWN) Zustand sind (siehe auch die Reihenfolge im obigen Beispiel). Audiodateien werden im **media/** Unterverzeichnis im HTML-Verzeichnis gesucht (z.B. */usr/local/netsaint/share/media*).

Optionen der Hauptkonfigurationsdatei

Bemerkung

Beim Anlegen oder Änder von Konfigurationsdateien sollte folgendes beachtet werden:

1. Zeilen die mit dem '#' Zeichen beginnen werden als Kommentar betrachtet und deshalb nicht verarbeitet
2. Varibalenbezeichner müssen am Zeilenanfang beginnen - es ist kein Leerzeichen vor dem Bezeichner erlaubt
3. Bei Variablenbezeichnern wird die Groß- und Kleinschreibung unterschieden

Beispielkonfiguration

Eine beispielhafte Hauptkonfigurationsdatei kann mit dem '**make config**' Befehl erzeugt werden. Der vorgegebene (default) Name der Hauptkonfigurationsdatei ist **netsaint.cfg** - zu finden ist diese Datei im NetSaint-Distributionsverzeichnis oder im etc/ Verzeichnis der Installation.

Inhalt

Logdatei
Hostkonfigurationsdatei
Resourcedatei
Statusdatei
Tempdatei

NetSaint Benutzer
NetSaint Gruppe

Programm Modus

Service check execution option
Passive service check acceptance option
Event handler option

Log rotation method
Log archive path

External command check option
External command check interval

External command file

Kommentardatei
Lock file

Statussicherungsoption
Statussicherungsdatei

Log severity level
Syslog logging option
Syslog severity level
Notification logging option
Service check retry logging option
Host retry logging option
Event handler logging option
Initial state logging option
External command logging option
Passive service check logging option

Global host event handler
Global service event handler

Inter-check sleep time
Inter-check delay method
Service interleave factor
Maximum concurrent service checks
Service reaper frequency
Timing interval length

Agressive host checking option

Service check timeout
Host check timeout
Event handler timeout
Notification timeout
Obsessive compulsive service processor timeout

Obsess over services option
Obsessive compulsive service processor command

Administrator email address

Administrator pager

Logdatei

Format: **log_file=<datei_name>**

Beispiel: **log_file=/usr/local/netsaint/var/netsaint.log**

Diese Variable legt das Verzeichnis fest, in das NetSaint die Hauptlogdatei ablegt. Das sollte die erste Variable sein, die in der Konfigurationsdatei definiert wird, da NetSaint versucht, Fehler die im Rest der Konfigurationsdatei gefunden werden in diese Datei zu schreiben. Diese Datei wird nie gelöscht, gekürzt oder rotiert. I suggest adding a cron job to do log rotations every month or so (häufiger wenn viele Alarme auftreten).

Hostkonfigurationsdatei

Format: **cfg_file=<datei_name>**

Beispiel: **cfg_file=/usr/local/netsaint/etc/hosts.cfg**

Diese Variable legt Name und Pfad der Hostkonfigurationsdatei fest, die NetSaint benutzen soll. Hostkonfigurationsdateien enthalten die Konfigurationsdaten für Hosts, Hostgruppen, Kontakte, Kontaktgruppen, Dienste, Befehle, etc. Diese Information kann auf mehrere Dateien aufgeteilt werden durch die Angabe mehrerer **cfg_file=** Definitionen.

Resourcedatei

Format: **resource_file=<datei_name>**

Beispiel: **resource_file=/usr/local/netsaint/etc/resource.cfg**

Diese Variable wird benutzt um eine optionale Resourcedatei festzulegen, die $USERn$ macro Definitionen enthalten kann. $USERn$ macros sind nützlich um Benutzernamen, Kennwörter und andere häufig vorkommende Bezeichner in Kommandodefinitionen (z.B. Verzeichnispfade) zu speichern. Die CGI's versuchen *nicht* die Resourcedatei zu lesen, daher kann die Dateiberechtigung auf Systemebene restriktiv (600 oder 660) eingestellt werden um sensible Informationen zu schützen. Es können mehrere Resourcedateien durch Hinzufügen mehrerer resource_file Definitionen benutzt werden - NetSaint wird alle verarbeiten. Als Beispiel kann die Datei resource.cfg im Hauptverzeichnis von NetSaint dienen, hier ist auch ersichtlich wie $USERn$ macros definiert werden.

Statusdatei

Format: **status_file=<datei_name>**

Beispiel: **status_file=/usr/local/netsaint/var/status.log**

Diese Datei benutzt NetSaint um den momentanen Status aller überwachten Dienste zu speichern. Die Stati aller Hosts in Verbindung mit den überwachten Diensten werden ebenfalls hier aufgezeichnet. Auf diese Datei greift das Status CGI-Skript zu, um den Status als Webreport auszugeben. Die CGIs

müssen deshalb Lesezugriff haben um korrekt zu arbeiten. Diese Datei wird gelöscht wenn NetSaint beendet wird und neu erzeugt, wenn NetSaint gestartet wird.

Tempdatei

Format: **temp_file=<datei_name>**

Beispiel: **temp_file=/usr/local/netsaint/var/netsaint.tmp**

In diese temporäre Datei leitet NetSaint die Standardausgaben und ggf. Fehlermeldungen der ausgeführten Plugins. Die Ausgaben der Plugins werden der Tempdatei entnommen und für die Anzeige im Stauts CGI sowie in Benachrichtigungsmacros (notification macros) benutzt. Die Tempdatei wird wieder gelöscht wenn das Plugin ausgeführt wurde. Die Tempdatei wird auch als Zwischenspeicher benutzt wenn NetSaint die Stautsdatei aktualisiert.

Bemerkung: Auf den meisten Systemen muß sich die Tempdatei im selben Dateisystem befinden wie die Statusdatei, die Logdatei, und die log file archive path.

NetSaint Benutzer

Format: **netsaint_user=<benutzername/UID>**

Beispiel: **netsaint_user=netsaint**

Diese Variable legt den tatsächlichen Benutzer fest als der der NetSaint Prozess ausgeführt wird. Nach dem anfänglichen Programmstart und bevor irgend etwas angezeigt wird verliert NetSaint seine ursprünglichen Rechte und wird als dieser Benutzer ausgeführt. Es kann ein Benutzername oder eine UID (UserID) festgelegt werden.

NetSaint Gruppe

Format: **netsaint_group=<gruppenname/GID>**

Beispiel: **netsaint_group=netsaint**

Diese Variable legt die tatsächliche Gruppe fest als die der NetSaint Prozess ausgeführt wird. Nach dem anfänglichen Programmstart und bevor irgend etwas angezeigt wird verliert NetSaint seine ursprünglichen Rechte und wird als diese Gruppe ausgeführt. Es kann ein Gruppenname oder eine GID (GroupID) festgelegt werden.

Programm Modus

Format: **program_mode=<a/s>**

Beispiel: **program_mode=a**

Diese Variabel legt den anfänglichen Programmodus fest wenn NetSaint gestartet oder neu gestartet wird. Weitere Information zu den Programmmodi ist hier zu finden. More information on program modes can be found here. Bemerkung: Wenn state retention eingeschaltet ist, wird NetSaint diese

Einstellung ignorieren und beim Start (oder Neustart) die letzte bekannte Einstellung dieser Option benutzen (wie in der state retention file abgespeichert). Wenn diese Option geändert werden soll, obwohl state retention eingeschaltet ist, muß der entsprechende externe Befehl (external command) benutzt werden oder die Änderung muß über das Webinterface erfolgen. Folgende Werte sind möglich:

- a = Aktiver Modus (default)
- s = Standby Modus

Service Check Execution Option

Format: **execute_service_checks=<0/1>**

Beispiel: **execute_service_checks=1**

Diese Option legt fest, ob NetSaint nach dem ersten Start (oder auch Neustart) Diensteüberprüfungen (Service checks) ausführen wird oder nicht. Wenn diese Option abgeschaltet ist, wird NetSaint keine Service Checks aktiv ausführen und in einer Art "Schlaf"modus bleiben (Es werden aber immer noch passive Service Checks akzeptiert, solange diese nicht abgeschaltet wurden). Diese Option wird meistens benutzt um einen Backup Überwachungsserver zu konfigurieren, wie im Abschnitt Redundanz in der Dokumentation beschrieben. Bemerkung: Wenn state retention eingeschaltet ist, wird NetSaint diese Einstellung ignorieren und beim Start (oder Neustart) die letzte bekannte Einstellung dieser Option benutzen (wie in der state retention file abgespeichert). Wenn diese Option geändert werden soll, obwohl state retention eingeschaltet ist, muß der entsprechende externe Befehl (external command) benutzt werden oder die Änderung muß über das Webinterface erfolgen. Folgende Werte sind möglich:

- 0 = Service Checks werden **nicht** ausgeführt
- 1 = Service Checks werden ausgeführt (default)

Passive Service Check Aktzeptanz Option

Format: **accept_passive_service_checks=<0/1>**

Beispiel: **accept_passive_service_checks=1**

Diese Option legt fest, ob NetSaint nach dem ersten Start (oder auch Neustart) passive Service Checks akzeptiert oder nicht. Wenn diese Option abgeschaltet ist wird NetSaint keine passive Service Checks akzeptieren. Bemerkung: Wenn state retention eingeschaltet ist, wird NetSaint diese Einstellung ignorieren und beim Start (oder Neustart) die letzte bekannte Einstellung dieser Option benutzen (wie in der state retention file abgespeichert). Wenn diese Option geändert werden soll, obwohl state retention eingeschaltet ist, muß der entsprechende externe Befehl (external command) benutzt werden oder die Änderung muß über das Webinterface erfolgen. Folgende Werte sind möglich:

- 0 = Passive service checks werden **nicht** akzeptiert
- 1 = Passive service checks werden akzeptiert(default)

Event Handler Option

Format: **enable_event_handlers=<0/1>**

Beispiel: **enable_event_handlers=1**

Diese Option legt fest, ob NetSaint Ereignisbehandlungsroutinen (event handlers) ausführen wird nach dem Start (oder Neustart). Wenn diese Option ausgeschaltet ist, wird NetSaint keine Host oder Service event handlers ausführen. Bemerkung: Wenn state retention eingeschaltet ist, wird NetSaint diese Einstellung ignorieren und beim Start (oder Neustart) die letzte bekannte Einstellung dieser Option benutzen (wie in der state retention file abgespeichert). Wenn diese Option geändert werden soll, obwohl state retention eingeschaltet ist, muß der entsprechende externe Befehl (external command) benutzt werden oder die Änderung muß über das Webinterface erfolgen. Folgende Werte sind möglich:

- 0 = event handlers sind ausgeschaltet
- 1 = event handlers sind eingeschaltet (default)

Log Rotation Method

Format: **log_rotation_method=<n/h/d/w/m>**

Beispiel: **log_rotation_method=d**

Diese Option legt den Rotationszyklus fest, den NetSaint auf die Logdatei anwendet. Folgende Werte sind möglich:

- n = Keine Rotation (keine Rotation der Logdatei - vorgegebener (default) Wert)
- h = Stündliche Rotation (Rotation der Logdatei am Ende jeder Stunde)
- d = Tägliche Rotation (Rotation der Logdatei an Mitternacht jeden Tages)
- w = Wöchentliche Rotation (Rotation der Logdatei an Mitternacht am Samstag)
- m = Monatliche Rotation (Rotation der Logdatei am letzten Tag des Monats)

Log Archiv Pfad

Format: **log_archive_path=<pfad>**

Beispiel: **log_archive_path=/usr/local/netsaint/var/archives/**

In diesem Verzeichnis legt NetSaint die Logdateien ab, die rotiert wurden. Diese Option wird ignoriert wenn die Logrotation nicht benutzt wird.

External Command Check Option

Format: **check_external_commands=<0/1>**

Beispiel: **check_external_commands=1**

Diese Option legt fest, ob NetSaint die Befehlsdatei (command file) auf auszuführende interne Befehle überprüft. Diese Option muß eingeschaltet werden, wenn beabsichtigt ist das command CGI Skript zu benutzen, um Befehle durch das Webinterface zu editieren. Programme von Dritten können ebenfalls Befehle durch Schreiben auf diese Befehlsdatei an NetSaint übergeben, vorausgesetzt die Berechtigungen für die Datei wurden richtig gesetzt (wie in der FAQ beschrieben). Weitere Information zu externen Befehlen ist hier zu finden.

- 0 = Externe Befehle werden nicht überprüft (default)
- 1 = Externe Befehle werden überprüft

External Command Check Interval

Format: **command_check_interval=<xxx>**

Beispiel: **command_check_interval=1**

Diese Option legt die Anzahl der Zeiteinheiten ("time units") fest, die NetSaint zwischen externen Befehlen (external command check) abwartet. Solange der Wert der Intervallänge (interval_length) (wie weiter unten beschrieben) unverändert auf 60 steht, ist dieses Zeiteinheit eine Minute. Jedesmal wenn NetSaint die externen Befehle überprüft werden alle Befehle in der command file eingelesen und verarbeitet, bevor NetSaint wieder andere Aufgaben ausführt. Weitere Information zu externen Befehlen ist hier zu finden.

External Command File

Format: **command_file=<dateiname>**

Beispiel: **command_file=/usr/local/netsaint/var/rw/netsaint.cmd**

Diese Datei benutzt NetSaint als externe Befehlsdatei. Das command CGI-Skript schreibt Befehle in diese Datei. Programme von Dritten können ebenfalls Befehle durch Schreiben auf diese Befehlsdatei an NetSaint übergeben wenn die Dateiberechtigungen wie hier beschrieben korrekt gesetzt wurden. Weitere Information zu externen Befehlen ist hier zu finden.

Kommentardatei

Format: **comment_file=<dateiname>**

Beispiel: **comment_file=/usr/local/netsaint/var/comment.log**

Diese Datei benutzt NetSaint um Service und Host Kommentare abzuspeichern. Kommentare können für Sevice und Hosts angeschaut werden durch das extended information CGI.

Lock File

Format: **lock_file=<dateiname>**

Beispiel: **lock_file=/tmp/netsaint.lock**

Diese Option gibt den Pfad zur Sperrdatei (lock file) an, die NetSaint erzeugt, wenn es als Dämon ausgeführt wird (Start mit "-d" als Argument in der Kommandozeile). Die Datei enthält die ID-Nummer des Prozesses (PID) von NetSaint.

Statussicherungsoption

Format: **retain_state_information=<0/1>**

Beispiel: **retain_state_information=1**

Diese Option legt fest, ob NetSaint die Statusinformationen für Hosts und Dienste bei einem Neustart des Programms sichert. Wenn diese Option eingeschaltet wird, sollte der Variablen state_retention_file ein Wert zugewiesen werden. Bei aktivierter Satusinformationssicherung wird NetSaint vor seiner Beendigung alle Statusinformationen für Hosts und Dienste in der definierten Datei abspeichern und auch diese Information wieder einlesen, wenn es (neu-) gestartet wird.

- 0 = Statusinformation werden nicht gesichert (default)
- 1 = Statusinformation werden gesichert

Statussicherungsdatei

Format: **state_retention_file=<dateiname>**

Beispiel: **state_retention_file=/usr/local/netsaint/var/status.sav**

In diese Datei speichert NetSaint die Statusinformation für Hosts und Dienste vor seiner Beendigung. Wenn NetSaint neu gestartet wird wird diese Information wieder eingelesen und alle Stati der Hosts und Dienste entsprechend vorinitialisiert, bevor irgendetwas angezeigt wird. Diese Datei wird nach dem Einlesevorgang beim Start (oder Neustart) von NetSaint wieder gelöscht. Um diese Sicherung der Statusinformation zu aktivieren, muß die Option retain_state_information eingeschaltet werden.

Log Severity Level

Format: **log_level=<1-2>**

Beispiel: **log_level=1**

Dieser Wert definiert den Grad der Heftigkeit der Servicenachrichten (service messages), der erforderlich ist um in der Hauptlogdatei aufgezeichnet zu werden. Folgende Werte sind möglich:

- 1 = Mitgeloggt werden Dienste in den Zuständen WARNING, UNKNOWN, oder CRITICAL.
- 2 = Mitgeloggt werden nur Dienste im Zustand CRITICAL.

Bemerkung:

> Dieser Wert sollte *immer* auf 1 gesetzt werden. Falls nicht, kann das Ergebnis variieren, da ich die Konsequenzen nicht bis ins letzte ausgetestet habe.

Syslog Logging Option

Format: **use_syslog=<0/1>**

Beispiel: **use_syslog=1**

Diese Option legt fest, ob Benachrichtigungen (messages) mitgeloggt werden oder nicht. Folgende Werte sind möglich:

- 0 = Benachrichtigungen werden nicht in der Systemlogdatei aufgezeichnet (mitgeloggt)
- 1 = Benachrichtigungen werden in der Systemlogdatei aufgezeichnet (mitgeloggt)

Syslog Severity Level

Format: **syslog_level=<1-2>**

Beispiel: **syslog_level=1**

Dieser Wert definiert den Grad der Heftigkeit der Dienstnachrichten (service messages), der erforderlich ist um in der Syslogdatei aufgezeichnet zu werden. Folgende Werte sind möglich:

- 1 = Mitgeloggt werden Dienste in den Zuständen WARNING, UNKNOWN, oder CRITICAL.
- 2 = Mitgeloggt werden nur Dienste im Zustand CRITICAL.

Bemerkung:

> Dieser Wert sollte *immer* auf 1 gesetzt werden. Falls nicht, kann das Ergebnis variieren, da ich die Konsequenzen nicht bis ins letzte ausgetestet habe.

Notification Logging Option

Format: **log_notifications=<0/1>**

Beispiel: **log_notifications=1**

Diese Option legt fest ob Benachrichtigungsbotschaften (notification messages) mitgeloggt werden oder nicht. Bei vielen Kontakten oder Dienstausfällen wird die Logdatei relativ schnell anwachsen. Mit dieser Option lässt sich verhindern, daß Benachrichtigungen aufgezeichnet werden.

- 0 = Benachrichtigungen werden nicht aufgezeichnet (mitgeloggt)
- 1 = Benachrichtigungen werden aufgezeichnet (mitgeloggt)

Service Check Retry Logging Option

Format: **log_service_retries=<0/1>**

Beispiel: **log_service_retries=1**

Diese Option legt fest, ob Wiederholungen der Dienstüberwachungen (service checks) aufgezeichnet werden. Service check Wiederholungen treten auf, wenn ein Service Check einen Nicht-OK Zustand

zurückliefert; und NetSaint konfiguriert wurde, den Service Check mehr als einmal auszuführen bevor ein Fehler angezeigt wird. Dienste in dieser Situation befinden sich in einem "Soft" Zustand. Das Aufzeichnen der Service Check Wiederholungen ist am ehesten nützlich beim Debuggen von NetSaint oder um service event handlers zu testen.

- 0 = Service Check Wiederholungen werden nicht aufgezeichnet
- 1 = Service Check Wiederholungen werden aufgezeichnet

Host Check Retry Logging Option

Format: **log_host_retries=<0/1>**

Beispiel: **log_host_retries=1**

Diese Option legt fest, ob Wiederholungen der Hostüberwachungen (host checks) aufgezeichnet werden. Das Aufzeichnen der Host Check Wiederholungen ist am ehesten nützlich beim Debuggen von NetSaint oder um service event handlers zu testen.

- 0 = Host Check Wiederholungen werden nicht aufgezeichnet
- 1 = Host Check Wiederholungen werden aufgezeichnet

Event Handler Logging Option

Format: **log_event_handlers=<0/1>**

Beispiel: **log_event_handlers=1**

Diese Option legt fest, ob Service und Host event handlers aufgezeichnet werden. Event handlers sind optionale Befehle, die ausgeführt werden können wenn ein Dienst oder ein Host den Zustand ändern. Das Aufzeichnen der event handlers ist am ehesten nützlich beim Debuggen von NetSaint oder beim ersten Ausprobieren von event handler Skripts.

- 0 = Event handlers werden nicht aufgezeichnet
- 1 = Event handlers werden aufgezeichnet

Initial States Logging Option

Format: **log_initial_states=<0/1>**

Beispiel: **log_initial_states=1**

Diese Option legt fest, ob NetSaint ein Aufzeichnen der anfänglichen Host und Service Zustände erzwingt, selbst wenn diese sich in einem OK-Zustand befinden. Diese Anfangszustände werden normalerweise nur aufgezeichnet, wenn es beim ersten Check ein Problem gibt. Das Einschalten dieser Option ist hilfreich, wenn eine Anwendung benutzt wird, die die Logdatei untersucht um eine Langzeitstatistik der Zustände der Host und Dienste auszugeben.

- 0 = Anfangszustände werden nicht aufgezeichnet (default)
- 1 = Anfangszustände werden aufgezeichnet

External Command Logging Option

Format: **log_external_commands=<0/1>**

Beispiel: **log_external_commands=1**

Diese Option legt fest, ob NetSaint externe Befehle aufzeichnet, die von der externen Befehlsdatei (external command file) kommen. Bemerkung: Diese Option steuert nicht die Aufzeichnung von passiven Service Checks (die ein Typ externer Befehle sind). Um die Aufzeichnung der passiven Service Checks ein- oder auszuschalten sollte die Option log_passive_service_checks benutzt werden.

- 0 = Externe Befehle werden nicht aufgezeichnet
- 1 = Externe Befehle werden aufgezeichnet (default)

Passive Service Check Logging Option

Format: **log_passive_service_checks=<0/1>**

Beispiel: **log_passive_service_checks=1**

Diese Option legt fest, ob NetSaint passive Service Checks aufzeichnet, die von der externen Befehlsdatei (external command file) kommen. Wenn eine verteilte Überwachungsumgebung eingerichtet werden soll oder eine große Anzahl regulärer passiver Service Checks geplant ist, kann diese Option ausgeschaltet werden um die Logdatei nicht zu groß werden zu lassen.

- 0 = Passive Service Checks werden nicht aufgezeichnet
- 1 = Passive Service Checks werden aufgezeichnet (default)

Global Host Event Handler Option

Format: **global_host_event_handler=<command>**

Beispiel: **global_host_event_handler=log-host-event-to-db**

Diese Option erlaubt die Definition eines Hostereignisbehandlungsbefehls (host event handler command), der bei jeder Hostzustandsänderung ausgeführt wird. Dieser globale event handler wird direkt vor dem event handler ausgeführt, der optional in der Hostdefinition festgelegt wurde. Das *command* Argument ist der Kurzname einer Befehlsdefinition in der Hostkonfigurationsdatei. Weitere Information zu Ereignisbehandlungsbefehlen (event handlers) ist hier zu finden.

Global Service Event Handler Option

Format: **global_service_event_handler=<command>**

Beispiel: **global_service_event_handler=log-service-event-to-db**

Diese Option erlaubt die Definition eines Dienstereignisbehandlungsbefehls (service event handler command), der bei jeder Dienstzustandsänderung ausgeführt wird. Dieser globale event handler wird direkt vor dem event handler ausgeführt, der optional in der Hostdefinition festgelegt wurde. Das *command* Argument ist der Kurzname einer Befehlsdefinition in der Hostkonfigurationsdatei. Weitere Information zu Ereignisbehandlungsbefehlen (event handlers) ist hier zu finden.

Inter-Check Sleep Time

Format: **sleep_time=<seconds>**

Beispiel: **sleep_time=1**

Diese Option legt die Anzahl der Sekunden fest, die NetSaint "schläft", bevor überprüft wird ob der nächste Service Check in der Warteschlange auszuführen ist. Zu beachten ist, daß NetSaint erst dann "schläft", nachdem es verspätete Service Checks aufgeholt hat.

Inter-Check Delay Method

Format: **inter_check_delay_method=<n/d/s/x.xx>**

Beispiel: **inter_check_delay_method=s**

Diese Option legt fest, wie Service Checks anfänglich in die Warteschlange eingereiht werden. Beim Benutzen der "smarten" (cleveren) Verzögerungsberechnung (Vorgabe) wird NetSaint einen druchschnittlichen Check Interval berechnen und anfängliche Checks über dieses Interval verteilen und damit helfen, Belastungsspitzen der CPU zu vermeiden. Keine Verzögerung zu benutzen wird prinzipiell *nicht* empfohlen, ausgenommen es soll die Funktionalität der Service Check Parallelisierung (service check parallelization) getestet werden. Wenn keine Verzögerung benutzt wird, werden alle Service Checks zur gleichen Zeit ausgeführt. Dies bedeutet hohe CPU-Belastungsspitzen, da alle Service Checks gleichzeitig ausgeführt werden. Weitere Informationen um abzuschätzen, wie die Inter-Check Verzögerung die Service Check-Warteschlange beeinflußt ist hier zu finden. Folgende Werte sind möglich:

- n = Keine Verzögerung - alle Service Checks werden sofort ausgeführt (zum gleichen Zeitpunkt!)
- d = Eine "dumme" Verzögerung von einer Sekunde zwischen den Checks wird benutzt
- s = Eine "smarte" Verzögerung wird berechnet und benutzt um die Service Checks gleichmäßig zu verteilen (default)
- x.xx = Benutzt wird die angebene Verzögerung mit x.xx Sekunden

Service Interleave Factor

Format: **service_interleave_factor=<s|*x*>**

Beispiel: **service_interleave_factor=s**

Diese Option legt fest, wie Dienstüberprüfungen (Service Checks) verschachtelt werden.

Verschachtelung (Interleaving) erlaubt eine gleichmäßigere Verteilung der Service Checks, reduzierte Last auf *entfernten* (remote) Hosts und im allgemeinen eine schnellere Erkennung von Hostproblemen. Mit der Einführung der Service Check Parallelisierung konnten entfernte Hosts bombadiert werden mit Checks falls Interleaving nicht implementiert war. Dies konnte zur Rückgabe falscher Resultate oder With the introduction of service check parallelization, remote hosts could get bombarded with checks if interleaving was not implemented. This could cause the service checks to fail or return incorrect results if the remote host was overloaded with processing other service check requests. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of NetSaint previous to 0.0.5 worked). Set this value to **s** (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. The best way to understand how interleaving works is to watch the status CGI (detailed view) when NetSaint is just starting. You should see that the service check results are spread out as they begin to appear. More information on how interleaving works can be found here.

- *x* = Ein Zahl größer oder gleich 1 die gibt den Interleavefaktor an. Der Interleavefaktor 1 bedeutet kein Interleaving auf die Service Checks anzuwenden.
- s = Benutze einen intelligenten ("smart") Interleavefaktor (default)

Maximum Concurrent Service Checks

Format: **max_concurrent_checks=<max_checks>**

Beispiel: **max_concurrent_checks=20**

Diese Option legt die maximale Anzahl der Dienstüberprüfungen (Service Checks) fest, die parallel zu einer bestimmten Zeit gleichzeitig ausgeführt werden können. Wird ein Wert von 1 für diese Option definiert, so wird eine parallelisierte Ausführung von jeglichen Service Checks verhindert. Diese Option muß entsprechend der verfügbaren Systemleistung der Maschine auf der NetSaint läuft, angepasst werden, da dieser Wert direkt die maximale Last beeinflusst, mit dem das System (Prozessorauslastung, Speicherbedarf, etc.) belastet wird. Weitere Informationen zur Abschätzung wie viele konkurrierende Checks zugelassen werden sollten ist hier zu finden.

Service Reaper Frequency

Format: **service_reaper_frequency=<frequency_in_seconds>**

Beispiel: **service_reaper_frequency=10**

Diese Option legt die Häufigkeit *in Sekunden* der Dienstauswertungsereignisse fest. "Auswertungsereignisse" verarbeiten die Ergebnisse der parallelisierten Dienstüberprüfungen (parallelized service checks), die beendet sind. Diese Ereignisse den Kern der Überwachungslogik in NetSaint.

Timing Interval Length

Format: **interval_length=<seconds>**

Beispiel: **interval_length=60**

Diese Option legt die Anzahl Sekunden fest, die ein "unit interval" dauert, welches benötigt wird für das Timing in der Warteschlange, Neubenachrichtigungen usw. "Unit intervals" werden in der Hostkonfigurationsdatei benutzt um festzulegen, wie oft ein Service Check durchgeführt muß oder wie oft Benachrichtigungen an einen Kontakt geschickt werden müssen usw.

Wichtig: Der vorgegebene Wert ist 60, was heißt, daß ein Einheitswert ("unit value") von 1 in der Hostkonfigurationsdatei 60 Sekunden (1 Minute) bedeutet. Ich habe andere Werte für diese Option nicht ausgetestet, so daß eine Änderung natürlich auf eigene Gefahr erfolgt!

Agressive Host Checking Option

Format: **use_agressive_host_checking=<0/1>**

Beispiel: **use_agressive_host_checking=0**

Ab der Release 0.0.4 geht NetSaint etwas intelligenter vor in der Art und Weise wie und wann Hostüberprüfungen durchgeführt werden. Generell erlaubt das Ausschalten dieser Option NetSaint intelligentere Entscheidungen zu treffen und die Hosts etwas schneller zu überprüfen. Das Einschalten der Option wird das Ausführen der Hostüberprüfungen etwas verlangsamen, aber dafür die Zuverlässigkeit etwas erhöhen. Für weitere Information, was diese Option in Einzelnen bewirkt kann der Quellcode der Datei **netsaint.c** nach der Zeichenkette "**use_agressive_host_checking**" durchsucht werden. Ich habe dort entsprechende Kommentare hinzugefügt. Solange NetSaint keine Probleme hat zu erkennen, daß ein Host sich erholt (recovered) hat, schlage ich vor diese Option **nicht** einzuschalten.

- 0 = Benutze keine agressive Hostüberprüfungen (default)
- 1 = Benutze agressive Hostüberprüfungen

Service Check Zeitlimit

Format: **service_check_timeout=<seconds>**

Beispiel: **service_check_timeout=60**

Diese Option legt die maximale Zeitdauer in Sekunden fest, die ein Service Check benötigen darf. Wenn der Check diese Zeit überschreitet wird er abgebrochen und ein CRITICAL Zustand wird zurückgegeben. Auch wird ein Zeitlimitfehler in der Logdatei festgehalten.

Host Check Zeitlimit

Format: **host_check_timeout=<seconds>**

Beispiel: **host_check_timeout=60**

Diese Option legt die maximale Zeitdauer in Sekunden fest, die ein Host Check benötigen darf. Wenn der Check diese Zeit überschreitet wird er abgebrochen und ein CRITICAL Zustand wird zurückgegeben und es wird auch davon ausgegangen, daß der Host DOWN ist. Auch wird ein

Zeitlimitfehler in der Logdatei festgehalten.

Event Handler Zeitlimit

Format: **event_handler_timeout=<seconds>**

Beispiel: **event_handler_timeout=60**

Diese Option legt die maximale Zeitdauer in Sekunden fest, die NetSaint einem event handlers zur Ausführung Zeit gibt. Wenn ein event handler dieses Zeitlimit überschreitet wird er abgebrochen und eine Warnung wird in der Logdatei verzeichnet.

Benachrichtigungszeitlimit

Format: **notification_timeout=<seconds>**

Beispiel: **notification_timeout=60**

Diese Option legt die maximale Zeitdauer in Sekunden fest, die NetSaint einem Benachrichtungsbefehl zur Ausführung Zeit gibt. Wenn ein Benachrichtungsbefehl (notification command) dieses Zeitlimit überschreitet wird er abgebrochen und eine Warnung wird in der Logdatei verzeichnet.

Obsessive Compulsive Service Processor Timeout

Format: **ocsp_timeout=<seconds>**

Beispiel: **ocsp_timeout=60**

Diese Option legt die maximale Zeitdauer in Sekunden fest, die NetSaint einem obsessive compulsive service processor command zur Ausführung Zeit gibt. Wenn ein Befehl (command) dieses Zeitlimit überschreitet wird er abgebrochen und eine Warnung wird in der Logdatei verzeichnet.

Obsess Over Services Option

Format: **obsess_over_services=<0/1>**

Beispiel: **obsess_over_services**

Diese Option legt fest, ob NetSaint Service Check Ergebnisse verfolgt (obsess over) und den definierten Befehl obsessive compulsive service processor command ausführt.This value determines whether or not NetSaint will "obsess" over service checks results and run the obsessive compulsive service processor command you define. I know - funny name, but it was all I could think of. Diese Option ist nützlich für die verteilte Überwachung (distributed monitoring). Wenn die verteilte Überwachung nicht benutzt wird sollte diese Option nicht eingeschaltet werden.

- 0 = Don't obsess over services (default)
- 1 = Obsess over services

Obsessive Compulsive Service Processor Command

Format: **ocsp_command=<command>**

Beispiel: **ocsp_command=obsessive_service_handler**

Diese Option ermöglicht es einen Befehl zu definieren, der nach *jeder* Dienstüberprüfung (Service Check) ausgeführt wird. Dieser Befehl wird ausgeführt nach jedem event handler oder Benachrichtungs- (notification) Befehl. Das *command* Argument ist der Kurzname einer Befehlsdefinition (command definition), die in der Hostkonfigurationsdatei definiert wurde. Diese Option ist nützlich für die verteilte Überwachung (distributed monitoring). Weitere Information zur verteilten Überwachung gibt es hier.

Administrator Email Address

Format: **admin_email=<email_address>**

Beispiel: **admin_email=root**

Diese Option legt die Emailadresse des Administrators der lokalen Maschine (d.h. der Maschine auf der NetSaint läuft) fest. Dieser Wert kann verwendet werden in Benachrichtigungsbefehlen (notification commands) durch das Benutzen des **$ADMINEMAIL$** macro.

Administrator Pager

Format: **admin_pager=<pager_number_or_pager_email_gateway>**

Beispiel: **admin_pager=pageroot@pagenet.com**

Diese Option legt die Pagernummer (oder Pager-Email Gateway) für den Administrator der lokalen Maschine (d.h. der Maschine auf der NetSaint läuft) fest. Die Pagernummer kann verwendet werden in Benachrichtigungsbefehlen (notification commands) durch das Benutzen des **$ADMINPAGER$**macro.

Using Macros In Commands

Macros

One of the features available in NetSaint is the ability to use macros in command defintions. Immediately prior to the execution of a command, NetSaint will replace all macros in the command with their corresponding values. This allows you to define a few generic commands to handle all your needs.

Macro Validity

Although macros can be used in all commands you define, not all macros may be "valid" in a particular type of command. For example, some macros may only be valid during service notification commands, whereas other may only be valid during host check commands. There are seven types of commands that NetSaint recognizes and treats differently. Six types are listed below, along with the macros that can be used with them. The seventh type of command is the ocsp command - any macros which are valid for service event handlers can be used with the ocsp command.

1. Service checks
2. Service notifications
3. Host checks
4. Host notifications
5. Service event handlers and/or a global service event handler
6. Host event handlers and/or a global host event handler

The table below lists all macros currently available in NetSaint, along with a brief description of each and the types of commands in which they are valid. If a macro is used in a command in which it is invalid, it is replaced with an empty string. It should be noted that macros consist of all uppercase characters and are enclosed in **$** characters.

Available Macros

Macro	Description	Service Checks	Service Notifications	Host Checks	Host Notifications	Service Event Handlers & Global Service Event Handler	Host Event Handlers & Global Host Event Handler
$CONTACTNAME$	Short name for the contact (i.e. "jdoe") that is being notified of a host or service problem	No	Yes	No	Yes	No	No
$CONTACTALIAS$	Long name/description for the contact (i.e. "John Doe") being notified	No	Yes	No	Yes	No	No

$CONTACTEMAIL$	Email address of the contact being notified	No	Yes	No	Yes	No	No
$CONTACTPAGER$	Pager number/address of the contact being notified	No	Yes	No	Yes	No	No
$HOSTNAME$	Short name for the host (i.e. "biglinuxbox"). During a service notification, this refers to the host associated with the service.	No	Yes	No	Yes	Yes	Yes
$HOSTALIAS$	Long name/description for the host (i.e. "Big Linux Server")	No	Yes	No	Yes	Yes	Yes
$HOSTADDRESS$	The IP address of the host	Yes	Yes	Yes	Yes	Yes	Yes
$HOSTSTATE$	The current state of the host ("UP", "DOWN", or "UNREACHABLE")	No	Yes	No	Yes	Yes	Yes
$ARGn$	The nth argument passed to the service check command. Read the documentation on service definitions for more info. NetSaint supports up to sixteen argument macros ($ARG1$ through $ARG16$).	Yes	No	No	No	No	No
$SERVICEDESC$	The long name/description of the service being monitored (i.e. "Main Website")	No	Yes	No	No	Yes	No
$SERVICESTATE$	The status of the service being monitored ("WARNING", "UNKNOWN", "CRITICAL", or "OK")	No	Yes	No	No	Yes	No
$OUTPUT$	The text output from the service or host check (i.e. "FTP ok - 1 second response time"). For service notifications and event handlers, this will contain the text output from the service check. For host notifications and event handlers, this will contain the text output from the host check.	No	Yes	No	Yes	Yes	Yes
$NOTIFICATIONTYPE$	Identifies the type of notification that is being sent ("PROBLEM", "RECOVERY", or "ACKNOWLEDGEMENT").	No	Yes	No	Yes	No	No
$DATETIME$	Date/time stamp	No	Yes	No	Yes	Yes	Yes
$ADMINEMAIL$	Email address for the local administrator (of the host doing the monitoring)	Yes	Yes	Yes	Yes	Yes	Yes

$ADMINPAGER$	Pager number/address for the local administrator	Yes	Yes	Yes	Yes	Yes	Yes
$STATETYPE$	The state type for the current service or host check ("HARD" or "SOFT"). Soft states occur when service or host checks return a non-OK state and are in the process of being retried. Hard states result when service or host checks have been checked a specified maximum number of times. Notifications are sent out only when hard state changes occur.	No	No	No	No	Yes	Yes
$SERVICEATTEMPT$	This refers to the number of the current service check retry. For instance, if this is the second time that the service is being rechecked, this will be the number two. Current attempt number is only useful when writing service event handlers for "soft" states that take a specific action based on the service retry number.	No	No	No	No	Yes	No
$HOSTATTEMPT$	This refers to the number of the current host check retry. For instance, if this is the second time that the host is being rechecked, this will be the number two. Current attempt number is only useful when writing host event handlers for "soft" states that take a specific action based on the host retry number.	No	No	No	No	No	Yes
$USERn$	The nth user-definable macro. User macros can be defined in one or more resource files . NetSaint supports up to thrity-two user macros ($USER1$ through $USER32$).	Yes	Yes	Yes	Yes	Yes	Yes

Event Handlers

Introduction

Event handlers are optional commands that are executed whenever a host or service state change occurs. An obvious use for event handlers (especially with services) is the ability for NetSaint to proactively fix problems before anyone is notified. Another use for event handlers is to log service or host events to an external database.

Event Handler Types

There are two main types of event handlers than can be defined - service event handlers and host event handlers. Event handler commands are (optionally) defined in each host and service definition. Because these event handlers are only associated with particular services or hosts, I will call these "local" event handlers. If a local event handler has been defined for a service or host, it will be executed when that host or service changes state.

You may also specify global event handlers that should be run for *every* host or service state change by using the global_host_event_handler and global_service_event_handler options in your main configuration file. Global event handlers are run immediately *prior* to running a local service or host event handler.

When Are Event Handler Commands Executed?

Service and host event handler commands are executed when a service or host:

- is in a "soft" error state
- initially goes into a "hard" error state
- recovers from a "soft" or "hard" error state

What are "soft" and "hard" states you ask? They are described here .

Event Handler Execution Order

Global event handlers are executed before any local event handlers that you have configured for specific hosts or services.

Writing Event Handler Commands

In most cases, event handler commands will be shell or perl scripts. At a minimum, the scripts should take the following macros as arguments:

Service event handler macros: **$SERVICESTATE$**, **$STATETYPE$**, **$SERVICEATTEMPT$**
Host event handler macros: **$HOSTSTATE$**, **$STATETYPE$**, **$HOSTATTEMPT$**

The scripts should examine the values of the arguments passed in and take any necessary action based upon those values. The best way to understand how event handlers should work is to see and example. Lucky for you, one is provided below. There are also some sample event handler scripts included in the **eventhandlers/** subdirectory of the NetSaint distribution. Some of these sample scripts demonstrate the use of external commands to implement redundant monitoring hosts.

Permissions For Event Handler Commands

Any event handler commands you configure will execute with the same permissions as the user under which NetSaint is running on your machine. This presents a problem with scripts that attempt to restart system services, as root privileges are generally required to do these sorts of tasks.

Ideally you should evaluate the types of event handlers you will be implementing and grant just enough permissions to the NetSaint user for executing the necessary system commands. I'll leave the details of how to do that up to you...

Debugging Event Handler Commands

When you are debugging event handler commands, I would highly recommend that you enable logging of service retries, host retries, and event handler commands. All of these logging options are configured in the main configuration file. Enabling logging for these options will allow you to see exactly when and why event handler commands are being executed.

When you're done debugging your event handler commands you'll probably want to disable logging of service and host retries. They can fill up your log file fast, but if you have enabled log rotation you might not care.

Service Event Handler Example

The example below assumes that you are monitoring the HTTP server on the local machine and have specified **restart-httpd** as the event handler command for the HTTP service definition. Also, I will be assuming that you have set the <max_attempts> option for the service to be a value of 4 or greater (i.e. the service is checked 4 times before it is considered to have a real problem).

First off, we must define the event handler as a command. Notice the macros that I am passing to the event handler command - these are important!

command[restart-httpd]=/usr/local/netsaint/restart-httpd $SERVICESTATE$ $STATETYPE$ $SERVICEATTEMPT$

Now, let's actually write the event handler script (this is the **/usr/local/netsaint/restart-httpd** file).

```
#!/bin/sh
#
# Event handler script for restarting the web server on the local machine
#
# Note: This script will only restart the web server if the service is
#       retried 3 times (in a "soft" state) or if the web service somehow
#       manages to fall into a "hard" error state.
#


# What state is the HTTP service in?
case "$1" in
OK)
        # The service just came back up, so don't do anything...
        ;;
WARNING)
        # We don't really care about warning states, since the service is probably still running...
        ;;
UNKNOWN)
        # We don't know what might be causing an unknown error, so don't do anything...
        ;;
CRITICAL)
        # Aha!  The HTTP service appears to have a problem - perhaps we should restart the
server...

        # Is this a "soft" or a "hard" state?
        case "$2" in

        # We're in a "soft" state, meaning that NetSaint is in the middle of retrying the
        # check before it turns into a "hard" state and contacts get notified...
        SOFT)

                # What check attempt are we on?  We don't want to restart the web server on the
first
                # check, because it may just be a fluke!
                case "$3" in
```

```
                # Wait until the check has been tried 3 times before restarting the web server.
                # If the check fails on the 4th time (after we restart the web server), the state
                # type will turn to "hard" and contacts will be notified of the problem.
                # Hopefully this will restart the web server successfully, so the 4th check will
                # result in a "soft" recovery.  If that happens no one gets notified because we
                # fixed the problem!
                3)
                        echo -n "Restarting HTTP service (3rd soft critical state)..."
                        # Call the init script to restart the HTTPD server
                        /etc/rc.d/init.d/httpd restart
                        ;;
                        esac
                ;;

        # The HTTP service somehow managed to turn into a hard error without getting fixed.
        # It should have been restarted by the code above, but for some reason it didn't.
        # Let's give it one last try, shall we?
        # Note: Contacts have already been notified of a problem with the service at this
        # point (unless you disabled notifications for this service)
        HARD)
                echo -n "Restarting HTTP service..."
                # Call the init script to restart the HTTPD server
                /etc/rc.d/init.d/httpd restart
                ;;
        esac
        ;;
esac
exit 0
```

The sample script provided above will attempt to restart the web server on the local machine in two different instances - after the HTTP service is being retried for the 3rd time (in an "soft" error state) and after the service falls into a "hard" state. The "hard" state situation shouldn't really occur, since the script should restart the service when its still in a "soft" state (i.e. the 3rd check retry), but its left as a fallback anyway.

It should be noted that the service event handler will only be execute the first time that the service falls into a "hard" state. This will prevent NetSaint from continuously executing the script to restart the web server when it is in a "hard" state.

State Types

Introduction

The current state of services and hosts is determined by two components: the status of the service or host and the *type* of state it is in. There are two state types in NetSaint - "soft" states and "hard" states. State types are a crucial part of NetSaint's monitoring logic. They are used to determine when event handlers are executed and when notifications are sent out.

Service and Host Check Retries

In order to prevent false alarms, NetSaint allows you to define how many times a service or host check will be retried before the service or host is considered to have a real problem. The maximum number of retries before a service or host check is considered to have a real problem is controlled by the <*max_attempts*> option in the service and host definitions, respectively. Depending on what attempt a service or host check is currently on determines what type of state it is is. There are a few exceptions to this in the service monitoring logic, but we'll ignore those for now. Let's take a look at the different service state types...

Soft States

Soft states occur for services and hosts in the following situations...

- When a service or host check results in a non-OK state and it has not yet been (re)checked the number of times specified by the <*max_attempts*> option in the service or host definition. Let's call this a soft error state...
- When a service or host recovers from a soft error state. This is considered to be a soft recovery.

Soft State Events

What happens when a service or host is in a soft error state or experiences a soft recovery?

- The soft error or recovery is logged if you enabled the log_service_retries or log_host_retries options in the main configuration file.
- Event handlers are executed (if you defined any) to handle the soft error or recovery for the service or host. (Before any event handler is executed, the **$STATETYPE$** macro is set to "**SOFT**").
- NetSaint does *not* send out notifications to any contacts because there is (or was) no "real" problem with the service or host.

As can be seen, the only important thing that really happens during a soft state is the execution of

event handlers. Using event handlers can be particularly useful if you want to try and proactively fix a problem before it turns into a hard state. More information on event handlers can be found here.

Hard States

Hard states occur for *services* in the following situations (hard host states are discussed later)...

- When a service check results in a non-OK state and it has been (re)checked the number of times specified by the <*max_attempts*> option in the service definition. This is a hard error state.
- When a service recovers from a hard error state. This is considered to be a hard recovery.
- When a service check results in a non-OK state and its corresponding host is either DOWN or UNREACHABLE. This is an exception to the general monitoring logic, but makes perfect sense. If the host isn't up why should we try and recheck the service?

Hard states occur for *hosts* in the following situations...

- When a host check results in a non-OK state and it has been (re)checked the number of times specified by the <*max_attempts*> option in the host definition. This is a hard error state.
- When a host recovers from a hard error state. This is considered to be a hard recovery.

Hard State Changes

Before I discuss what happens when a host or service is in a hard state, you need to know about hard state changes. Hard state changes occur when a service or host...

- changes from a hard OK state to a hard non-OK state
- changes from a hard non-OK state to a hard OK-state
- changes from a hard non-OK state of some kind to a hard non-OK state of another kind (i.e. from a hard WARNING state to a hard UNKNOWN state)

Hard State Events

What happens when a service or host is in a hard error state or experiences a hard recovery? Well, that depends on whether or not a hard state change (as described above) has occurred.

If a hard state change has occurred *and* the service or host is in a non-OK state the following things will occur..

- The hard service or host problem is logged.
- Event handlers are executed (if you defined any) to handle the hard problem for the service or host. (Before any event handler is executed, the **$STATETYPE$** macro is set to "**HARD**").
- Contacts will be notified of the service or host problem (if the notification logic allows it).

If a hard state change has occurred *and* the service or host is in an OK state the following things will occur..

- The hard service or host recovery is logged.
- Event handlers are executed (if you defined any) to handle the hard recovery for the service or host. (Before any event handler is executed, the **$STATETYPE$** macro is set to "**HARD**").
- Contacts will be notified of the service or host recovery (if the notification logic allows it).

If a hard state change has NOT occurred *and* the service or host is in a non-OK state the following things will occur..

- Contacts will be re-notified of the service or host problem (if the notification logic allows it).

If a hard state change has NOT occurred *and* the service or host is in an OK state nothing happens. This is because the service or host is in an OK state and was the last time it was checked as well.

Logic Diagrams

Soft and hard states can be a little difficult to understand. The logic diagrams found here may be of some help.

Author: Matthias Eichler at INTERNET Date: 27.12.00 21:17 Normal BCC: Dietmar Schurr at Whirlpool-DESCHPB2 TO: dietmar.schurr@gmx.de at Internet Subject: Neue =?iso-8859-1?q?=FCbersetzte=20Datei?= ---------- Message Contents

Benachrichtigungen

Einleitung:

Ich habe einige Fragen bekommen, wie die Benachrichtigungen genau funktionieren. Hoffentlich wird dieses Kapitel genau diese Fragen beantworten und erklären wer welche Benachrichtigungen bekommt und wie sie zu den Administratoren gelangen.

Index

Wann treten Benachrichtigung auf?
Wer wird benachrichtigt?
Welche Filter werden durchlaufen, bevor eine Benachrichtigung versandt wird?
Warum sind die Benachrichtigungs-Wege nicht direkt in NetSaint integriert?
Nützliche Links

Wann treten Benachrichtigungen auf?

Die Entscheidung eine Benachrichtigung auszusenden wird von der Dienste- und Host-Überprüfungs-Logik getroffen. Host- und Dienste-Benachrichtigungen treten in den folgenden Instanzen auf...

- Wenn eine harter Status-Veränderung auftritt. Weitere Informationen über den Status-Typ und harte Status-Veränderungen kann man hier nachlesen.
- Wenn ein Host oder Dienst weiterhin in einem harten "Nicht-Ok"-Status bleibt und die in der <*notification_interval*> (in der Host oder Dienste-Konfiguration) definierte Zeit seit der letzten Benachrichtigung (für diesen einen Dienst oder Host) verstrichen ist.

  Sollte Dir die Idee von wiederkehren Benachrichtigungen nicht gefallen, kannst Du einfach den <*notification_interval*>-Wert sehr hoch (z.B. auf 24 Stunden) setzen.

Wer wird benachrichtigt?

Jede Dienste-Definition hat eine <*Kontakt-Gruppen*>-Option, die angibt, welche Kontakt-Gruppen Benachrichtigungen über diesen Dienst erhalten.
Jede Kontakt-Gruppe can eine oder mehrere Personen enthalten. Wenn NetSaint eine Dienst-Benachrichtigung versendet, wird es jede Person in einer der Kontakt-Gruppen, die in der <*Kontakt-Gruppen*>-Option der Dienste-Definition angegeben wurden, benachrichtigen. NetSaint erkennt, falls

eine Person Mitglied in mehreren Kontakt-Gruppen ist, so dass es nicht zu doppelten Benachrichtigungen kommen kann.

Jeder Host kann zu einer oder mehreren Host- Gruppen gehören. Jede Host-Gruppe hat eine <*contact_groups*>-Option, die angibt, welche Kontakt-Gruppen eine Benachrichtigungen über einen Host in dieser Host-Gruppe erhalten.
NetSaint sendet eine Host-Benachrichtigung an alle Personen, die Mitglied einer der Gruppen, die Benachrichtigungen über diesen einen Host oder die ganze Host-Grupper erhalten sollen, sind. Vor dem Versenden entfernt NetSaint allerdings doppelte eMail-Adressen, damit mehrfache Benachrichtigungen nicht auftreten können.

Welche Filter werden durchlaufen, bevor eine Benachrichtigung versandt wird?

Nur das eine Host- oder Dienste-Benachrichtigung verschickt werden soll, heisst noch nicht, dass eine einzige Person benachrichtigt wird. Es gibt einige Filter, die potentielle Benachrichtigungen erst durchlaufen müssen, bevor sie als "wertvoll" genug erachtet werden, verschickt zu werden. Sogar wenn, Aber auch dann kann es sein, dass Benachrichtigungen an einzelne Personen nicht verschickt werden, wenn die Filter der Person einen versandt nicht erlauben.
Schauen wir uns mal die Filter, die durchlaufen werden etwas genauer an...

Benachrichtigungs-Logik

Einfach auf das Bild klicken, um eine Grafik der Filter zu erhalten, die durchlaufen werden müssen, bevor eine Benachrichtigung verschickt wird.

Programm-Modus Filter:

Der erst Filter, den eine Benachrichtigung durchlaufen muss, ist der Programm-Modus-Test. Sollte NetSaint im *STANDBY*-Modus laufen, wird **niemand benachrichtigt**. Sollte NetSaint im *ACTIVE*-Modus laufen, werden die Benachrichtigungenan den nächsten Filter weitergegeben...

Dienst- und Host-Filter:

Das erste Set an Dienst- oder Host-Filtern das durchlaufen werden muss, sind die Benachrichtigungs-Optionen.
Jede Dienste-Definition beinhaltet Optionen, um zu bestimmen, ob eine Benachrichtigung über warnende, kritische Meldungen und "Recoveries" verschickt werden kann.
Ganz ähnlich beinhaltet jede Host-Definition, die bestimmen, ob eine Benachrichtigung verschickt wird, wenn ein Host abgeschaltet wird, nicht mehr oder wieder erreichbar ist.
Falls eine Dienst- oder Host-Benachrichtigung diese Optionen nicht erfüllt, wird **niemand benachrichtigt**. Durchläuft eine Benachrichtigung diese Optionen, wird sie an den nächsten Filter weitergegeben...

Wichtig:
Benachrichtigungen über die Wiedererreichbarkeit von Diensten oder Hosts werden nur verschickt, wenn eine Benachrichtigung über das Ausfallen des Dienstes oder Hosts verschickt worden ist. Es macht wohl auch kaum Sinn, eine Benachrichtigung über die Wiedererreichbarkeit zu bekommen, wenn man nicht mal weiss, dass es ein Problem gegeben hat...

Das zweite Set von Hosts- und Dienste-Filtern die passiert werden müssen sind die Zeitperioden-Tests. Jede Host- oder Dienste-Definition hat eine <*notification_period*>-Option, die angibt, in welchen Zeitperioden Benachrichtigungen über den entsprechenden Host oder Dienst zulässig sind.
Sollte die Zeit zur der eine Benachrichtigung verschickt werden soll, nicht in eine zulässige Zeitperiode fallen, **wird niemand benachrichtigt**.
Fällt die Zeit in eine gültige Zeitperiode, wird sie an den nächsten Filter weitergegeben...

Wichtig:
Sollte eine Benachrichtigung den Zeitspannen-Filter nicht durchlaufen, wird NetSaint die nächste erneute Benachrichtigung über einen "Nicht-OK"-Status auf den erst möglichen Zeitpunkt in einer gültiger Zeitperiode legen. Dies hilft dabei sicher zu gehen, dass Personen über ein Problem so schnell wie möglich (oder auch erlaubt) benachrichtigt werden, wenn die zulässige Zeitperiode anfängt.

Das letzte Set von Host- oder Dienste-Filtern ist von zwei Dingen abhängig:

- eine Benachrichtigung über ein Problem mit dem Host oder Dienst wurde bereits verschickt und
- der Host oder Dienst ist seitdem in dem gleichen "Nicht-OK"-Status geblieben, indem die letzte Benachrichtigung verschickt worden ist.

Treffen diese Kriterien zu, überprüft NetSaint, ob die Zeit seit der letzten Benachrichtigung mit dem Wert, der mit der <*notification_interval*>-Option in der Host- oder Dienst-Definition angegeben wurde, übereinstimmt oder sogar überschreitet.
Falls nicht genügend Zeit vergangen ist, wird **niemand benachrichtigt**.
Falls genügend Zeit vergangen ist oder die beiden für diesen Filter erforderlichen Kriterien nicht übereinstimmen, würde die Benachrichtigung verschickt werden...
Ob sie letztendlich wirklich verschickt wird oder nicht, hängt von einem weiteren Set von Filtern ab...

Personen Filter:

An diesem Punkt hat die Benachrichtigung den Programm-Modus-Filter und alle Host- oder Dienst-Filter durchlaufen und NetSaint beginnt alle Leute, die es benachrichtigen soll, zu benachrichtigen. Bedeutet dies aber das jede ausgewählte Person eine Benachrichtigung? Nein! Jeder Kontakt hat sein eigenes Set von Filtern, dass die Benachrichtigung durchlaufen muss, bevor sie von jemandem empfangen werden kann.
Wichtig:

Personen Filter sind für jede Person spezifisch und beeinflussen die Benachrichtigung von anderen Personen nicht.

Die ersten Filter, die für jede Person durchlaufen werden müssen, sind die Benachrichtigungs-Optionen. Jede Kontakt-Definition enthält Optionen, die bestimmen, ob eine Benachrichtigung über warnende, kritische Meldungen und "Recoveries" verschickt werden kann oder nicht. Jede Kontakt-Definition enthält ausserdem Optionen, die bestimmen, ob eine Benachrichtigung verschickt wird, wenn ein Host abgeschaltet wird, nicht oder wieder erreichbar ist.
Wird eine Host- oder Dienst-Benachrichtigung von einem dieser Filter aufgehalten, **wird niemand benachrichtigt**.
Sollte sie die Filter durchlaufen, wird die Benachrichtigung an den nächsten Filter weitergegeben...

Wichtig:
Benachrichtigungen über die Wiedererreichbarkeit eines Hosts oder Dienstes werden nur verschickt, wenn auch eine Benachrichtigung über das Problem verschickt worden ist. Es macht keinen Sinn über das Ende eines Problems benachrichtigt zu werden, wenn man nicht wusste, dass es überhaupt ein Problem gegeben hat...

Der letzte Filter, der für jede zu benachrichtende Person durchlaufen werden muss ist der Zeitperioden-Test. Jede Kontakt-Definition hat eine <*notification_period*>-Option, die angibt in welchem Zeitraum die Person generell benachrichtigt werden darf.
Fällt die Zeit zu der die Benachrichtigung verschickt werden soll nicht in einen dieser Zeiträume, wird die Person **nicht benachrichtigt**. Sollte die Zeit der Benachrichtigung in einen gültigen Zeitraum fallen, wird die Benachrichtigung an die Person verschickt!

Warum sind die Benachrichtigungs-Wege nicht direkt in NetSaint integriert?

Ich habe einige Fragen erhalten, warum die einzelnen Methoden zur Benachrichtigung (paging, etc.) nicht direkt in den NetSaint-Code integriert wurden. Die Antwort ist einfach - es macht nicht wirklich Sinn. Der "Kern" von NetSaint ist nicht dafür "designed" ein "alles-in-einem"-Programm zu sein. Wären die Dienst-Überprüfungen direkt in den NetSaint-Code integriert, wäre es sehr schwer für den User neue Überprüfungen beizusteuern, bestehende Überprüfungen zu verändern. Bei der Benachrichtigung sieht es sehr ähnlich aus.
Es gibt tausend verschiedene Wege Benachrichtigungen abzuschicken und es gibt wirklich viele Programme da draussen, die diese nebensächliche Arbeit übernehmen. Warum sollte man also das Rad neu erfinden und sich selbst dabei begrenzen?
Es ist um vieles einfacher, einer externen Instanz (z.B. einem Skript oder einem völlig überfrachteten Messaging-System) diese Arbeit zu geben. Einige Messaging-Systeme, die Benachrichtigungen an Pager oder Handies verschicken können, sind etwas weiter unten aufgeführt.

Nützliche Links

Sollte es für Dich interessant sein alphanumerische Nachrichten an Deinen Pager oder Dein Handy via

eMail zu schicken, könnten die die nächsten Informationen ganz interessant sein. Hier sind ein paar Links zu den Webseiten von einigen Messaging-Dienst-Providern, die Infos darüber bereithalten, alphanumerische Nachrichten an Pager und Handies zu verschicken...

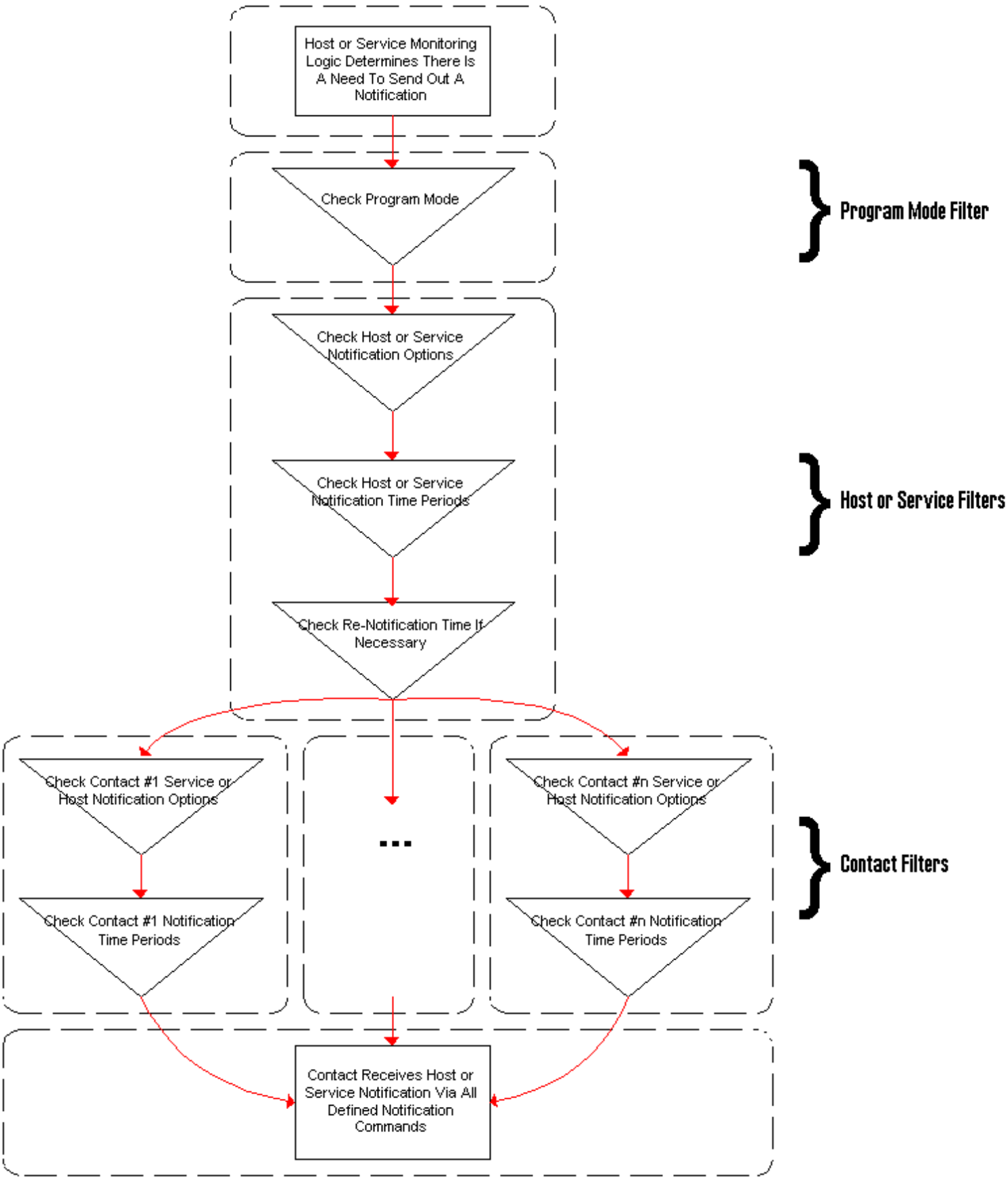
- AT&T Wireless
- PageNet
- SprintPCS (SMS phones)

Solltest Du nach einer Alternative suchen, um Nachrichten über eMail an Deinen Pager oder Dein Handy zu senden, solltest Du Dir diese Infos hier anschauen. Sie sollten in Zusammenarbeit mit NetSaint eingesetzt werden, um Benachrichtigungen über ein Modem zu versenden, wenn ein Problem auftreten sollte. Durch diesen Weg braucht man sich nicht auf den Weg von eMails zu verlassen, wenn Benachrichtigungen verschickt werden (denkt daran: eMail funktionieren vor allem bei Netzwerk-Problemen *nicht*). Ich habe diese Dienste selber zwar noch nicht ausprobiert, aber andere User berichten, dass sie durchaus funktionieren...

- Danpage (alphanumerische Pager-Software)
- QuickPage (alphanumerische Pager-Software)
- Sendpage (Paging-Software)
- SMS Client (Kommandozeilen-Tool, um Nachrichten aufs Handy oder auf einen Pager zu senden)

Zum Schluss gibt es noch einen Bereich in der Download-Bereich der NetSaint-Homepage wo man sich von anderen Usern geschrieben Benachrichtigungs-Skripts downloaden kann. Vielleicht wirst Du diese Skripts nützlich finden, da sie die ganze Drecksarbeit übernehmen, die gemacht werden muss, um alphanumerische Nachrichten zu verschicken...

Notification Filters



Program Modes

Introduction

The idea of program modes is quite simple, but you need to understand the difference between them before you start doing anything like implementing redundant monitoring hosts. There are two types of program modes - ***Active*** and ***Standby***...

Active Mode

This is the default program mode for NetSaint. While in *active* mode, NetSaint will monitor all services and hosts that you have defined in your host configuration file(s). When a problem arises with one of those services or hosts, NetSaint will send out notifications to all appropriate contacts. This is equivalent to how previous versions of NetSaint worked.

Standby Mode

While in *standby* mode, NetSaint will continue to monitor all services and hosts you defined in your host configuration file(s), but it **will not** send out notifications to any contacts when problems arise. This is particularly useful when implementing redudant monitoring hosts, and is equivalent to temporarily disabling notifications for all defined services and hosts. NetSaint will not send out notifications to any contacts until it returns to *active* mode.

Configuring The Initial Program Mode

By default, NetSaint will enter *active* mode when it (re)starts. If you wish to change the initial program mode to *standy*, you'll have to use the program_mode option in the main configuration file.

Changing Program Modes During Runtime

You can change the current program mode between *active* and *standby* by issuing an external command to NetSaint via the command file. Of course, this assumes that you have enabled external command checks. For more information on external commands, click here.

Two sample shell scripts (*enter_active_mode* and *enter_standy_mode*) are provided in the **sample-scripts/** subdirectory of the NetSaint distribution as examples of how to change the program mode during runtime. You will have to modify the scripts to match the location of your command file before you can use them.

Redundant Network Monitoring

Introduction

This section describes a few scenarios for implementing redundant monitoring hosts an various types of network layouts. With redundant hosts, you can maintain the ability to monitor your network when the primary host that runs NetSaint fails or when portions of your network become unreachable.

Note: If you are just learning how to use NetSaint, I would suggest not trying to implement redudancy until you have becoming familiar with the prerequisites I've laid out. Redundancy is a relatively complicated issue to understand, and even more difficult to implement properly.

Index

Prerequisites
Considerations
Sample scripts
Scenario 1 - Implementing redundancy on the same network segment
Scenario 2 - A simple way to implement redundancy across network segments
Scenario 3 - A smarter way to implement redundancy across network segments
Scenario 4 - Implementing multiple redundancy methods

Prerequisites

Before you can even think about implementing redundancy with NetSaint, you need to be familiar with the following...

- Implementing event handlers for hosts and services
- Issuing external commands to NetSaint via shell scripts
- Executing plugins on remote hosts
- Checking the status of the NetSaint process with the check_netsaint plugin

Considerations

There are a few things you need to understand before you jump into implementing redundancy...

First off, 0.0.5 is a first release of NetSaint where redundancy can actually be implemented in any kind of reasonable manner. It just so happened that all the pieces fell into place for accomodating this (event handlers, program modes, and external commands). Additional support for implementing redundancy will be incorporated into future versions of NetSaint, but I need your feedback!

Sample Scripts

All of the sample scripts that I use in this documentation can be found in the *eventhandlers/* subdirectory of the NetSaint distribution. You'll probably need to modify them to work on your system...

Scenario 1 - Implementing Redundancy On The Same Network Segment

Introduction

This is the easiest method of implementing redundant monitoring hosts on your network. However, this method only will only protect against a limited number of failures. More complex setups are necessary in order to provide better redundancy across different network segments.

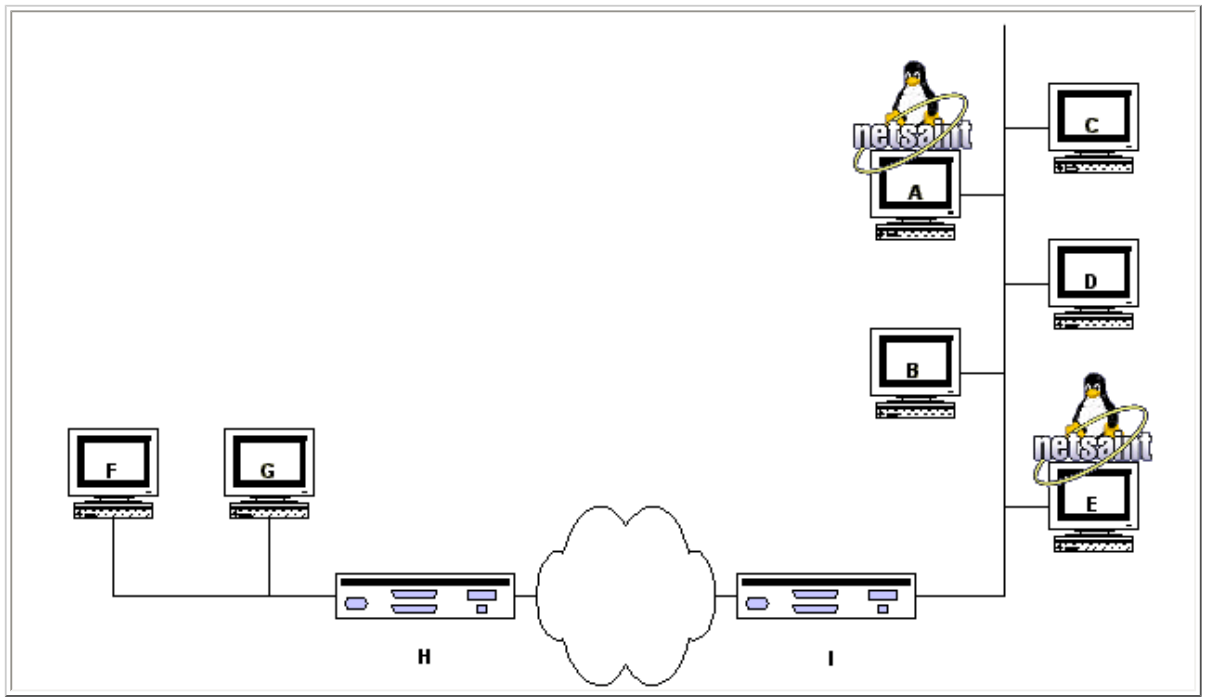
Goals

The goal of this type of redundancy implementation is for a "slave" host running NetSaint to take over the job of monitoring *the entire network* if:

1. The "master" host that runs NetSaint is down or..
2. The NetSaint process on the "master" host stops running for some reason

Network Layout Diagram

The diagram below shows a very simple network setup. For this scenario I will be assuming that hosts A and E are both running NetSaint and are monitoring all the hosts shown. Host A will be considered the "master" host and host E will be considered the "slave" host.



Initial Program Modes

First off, we need to define what program mode the master and slave hosts will be in when they start monitoring. This is done by using the program_mode option in the main configuration file. The master

host (host A) should have its initial program mode set to *active*, while the slave host (host B) should have its initial program mode set to *standby*. That was easy enough...

Initial Configuration

Next we need to consider the differences between the host configuration files on the master and slave hosts...

I will assume that you have the master host (host A) setup to monitor services on all hosts shown in the diagram above. The slave host (host E) should be setup to monitor the same services and hosts, with the following additions in the configuration file...

- The host definition for host A (in the host E configuration file) should have a host event handler defined. Lets say the name of the host event handler is handle-master-host-event.
- The configuration file on host E should have a service defined to check the status of the NetSaint process on host A. Lets assume that you define this service check to run the check_netsaint plugin on host A. This can be done by using one of the methods described in this FAQ.
- The service definition for the NetSaint process check on host A should have an event handler defined. Lets say the name of the service event handler is handle-master-proc-event.

It is important to note that host A (the master host) has no knowledge of host E (the slave host). In this scenario it simply doesn't need to. Of course you may be monitoring services on host E from host A, but that has nothing to do with the implementation of redundancy...

Event Handler Command Definitions

We need to stop for a minute and describe what the command definitions for the event handlers on the slave host look like. Here is an example...

```
command[handle-master-host-event]=/usr/local/netsaint/libexec/eventhandlers/handle-master-host-event $HOSTSTATE$ $STATETYPE$
command[handle-master-proc-event]=/usr/local/netsaint/libexec/eventhandlers/handle-master-proc-event $SERVICESTATE$ $STATETYPE$
```

This assumes that you have placed the event handler scripts in the */usr/local/netsaint/libexec/eventhandlers* directory. You may place them anywhere you wish, but you'll need to modify the examples I've given here.

Event Handler Scripts

Okay, now lets take a look at what the event handler scripts look like...

Host Event Handler (handle-master-host-event)

```
#!/bin/sh

# Only take action on hard host states...
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # The master host has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enter active mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_active_mode
                ;;
        UP)
```

Service Event Handler (handle-master-proc-event)

```
#!/bin/sh

# Only take action on hard service states...
case "$2" in
HARD)
        case "$1" in
        CRITICAL)
                # The master NetSaint process is not running!
                # We should now become the master host and
                # take over the responsibility of monitoring
                # the network, so enter active mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_active_mode
                ;;
        WARNING)
```

```
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # enter standby mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_standby_mode
                ;;
        esac
        ;;
esac
exit 0
```

```
        UNKNOWN)
                # The master NetSaint process may or may not
                # be running.. We won't do anything here, but
                # to be on the safe side you may decide you
                # want the slave host to become the master in
                # these situations...
                ;;
        OK)
                # The master NetSaint process running again!
                # We should go back to being the slave host,
                # so enter standby mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_standby_mode
                ;;
        esac
        ;;
esac
exit 0
```

What This Does For Us

When things first start out, host A (the master host) is in *active* mode. This means that it monitors all services and sends out notifications if there are problems or recoveries. Host E (the slave host) is in *standby* mode, which means that it will monitor all services but will *not* send out any notifications.

The NetSaint process on host E becomes the master host when...

- Host A goes down (the *handle-master-host-event* host event handler is executed).
- The NetSaint process on host A is not running (the *handle-master-proc-event* service event handler is executed).

When the NetSaint process on host E has entered active mode, it will be able to send out notifications about any service or host problems or recoveries. At this point host E has effectively taken over the responsibility of monitoring the network!

The NetSaint process on host E returns to being the slave host when...

- Host A has recovers (the *handle-master-host-event* host event handler is executed).
- The NetSaint process on host A recovers (the *handle-master-proc-event* service event handler is executed).

When the NetSaint process on host E has entered standby mode, it will not send out notifications about any service or host problems or recoveries. At this point host E has handed over the responsibilities of monitoring the network back to host A. Everything is now as it was when we first started!

Time Lags

Redundancy in NetSaint is by no means perfect. One of the more obvious problems is the lag time between the master host failing and the slave host taking over. This is affected by the following...

- The time between a failure of the master host and the first time the slave host detects a problem
- The time needed to verify that the master host really does have a problem (using service or host check retries on the slave host)
- The time between the execution of the event handler and the next time that NetSaint checks for external commands

You can minimize this lag by...

- Ensuring that the NetSaint process on host E (re)checks one or more services at a high frequency. This is done by using the *check_interval* and *retry_interval* arguments in each service definition.
- Ensuring that the number of host rechecks for host A (on host E) allow for fast detection of host problems. This is done by using the *max_attempts* argument in the host definition.
- Increase the frequency of external command checks on host E. This is done by modifying the command_check_interval option in the main configuration file.

When NetSaint recovers on the host A, there is also some lag time before host E returns to standby mode. This is affected by the following...

- The time between a recovery of host A and the time the NetSaint process on host E detects the recovery
- The time between the execution of the event handler on host B and the next time the NetSaint process on host E checks for external commands

The exact lag times between the transfer of monitoring responsibilities will vary depending on how many services you have defined, the interval at which services are checked, and a lot of pure chance. At any rate, its definitely better than nothing...

Special Cases

Here is one thing you should be aware of... If host A goes down, host E will switch to *active* mode and take over the responsibilities of monitoring. When host A recovers, host E will switch to *standby* mode. If - when host A recovers - the NetSaint process on host A does not start up properly, there will be a period of time when neither host is monitoring the network! Fortunately, the service check logic in NetSaint accounts for this. The next time the NetSaint process on host E checks the status of the NetSaint process on host A, it will find that it is not running. Host E will then switch back to *active* mode and take over all responsibilities of monitoring.

The exact amount of time that neither host is monitoring the network is hard to determine. Obviously, this period can be minimized by increasing the frequency of service checks (on host E) of the NetSaint process on host A. The rest is up to pure chance, but the total "blackout" time shouldn't be too bad...

Scenario 2 - A Simple Way To Implement Redundancy Across Network Segments

Introduction

If you're monitoring hosts that reside on different network segments, you're going to need a more substantial redundancy model that described in scenario 1. The following example is more complex than that in the first scenario, but the logic behind it should become clear if you study it closely enough.

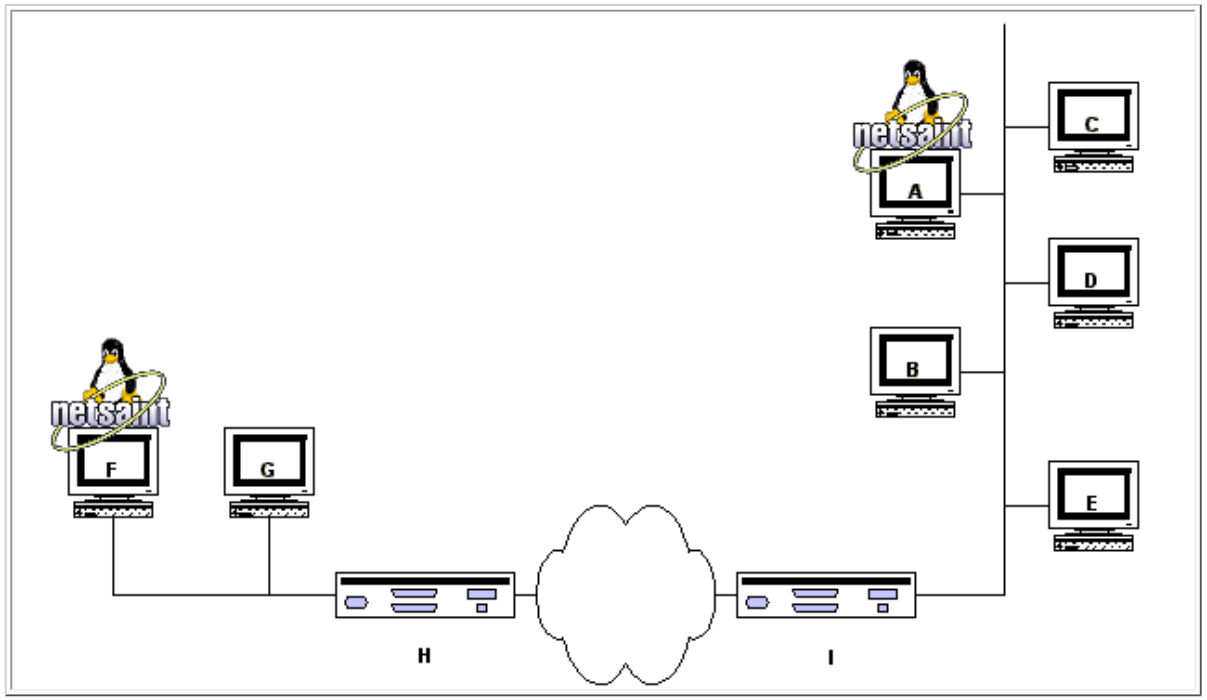
Goals

The goal of this type of redundancy implementation is for a "slave" host running NetSaint to take over the job of monitoring *the entire network* if:

1. The "master" host that runs NetSaint is down or unreachable or...
2. The NetSaint process on the "master" host stops running for some reason

Network Layout Diagram

The diagram below shows a relatively simple network setup with host on two network segments. For this scenario I will be assuming that hosts A and F are both running NetSaint and are monitoring all the hosts shown. Host A will be considered the "master" host and host F will be considered the "slave" host. Nodes H and I are routers that lie between the two network segments.



Initial Program Modes

For this example, the master host (host A) should have its initial program mode set to *active*, while the slave host (host F) should have its initial program mode set to *standby*.

Initial Configuration

Next we need to consider the differences between the host configuration files on the master and slave hosts...

I will assume that you have the master host (host A) setup to monitor services on all hosts shown in the diagram above. The slave host (host F) should be setup to monitor the same services and hosts, with the following additions in the configuration file...

- The host definition for host A (in the host F configuration file) should have a host event handler defined. Lets say the name of the host event handler is handle-master-host-event.
- The configuration file on host F should have a service defined to check the status of the NetSaint process on host A. Lets assume that you define this service check to run the check_netsaint plugin on host A. This can be done by using one of the methods described in this FAQ.
- The service definition for the NetSaint process check on host A should have an event handler defined. Lets say the name of the service event handler is handle-master-proc-event.
- The host definitions for both host H and I should have event handlers defined. Lets say the name of the host event handler in both definitions is handle-router-event

It is important to note that host A (the master host) has no knowledge of host F (the slave host). In this scenario it simply doesn't need to. Of course you may be monitoring services on host F from host A, but that has nothing to do with the implementation of redundancy...

Event Handler Command Definitions

We need to stop for a minute and describe what the command definitions for the event handlers on the slave host look like. Here is an example...

```
command[handle-master-host-event]=/usr/local/netsaint/libexec/eventhandlers/handle-master-host-event $HOSTSTATE$ $STATETYPE$
command[handle-master-proc-event]=/usr/local/netsaint/libexec/eventhandlers/handle-master-proc-event $SERVICESTATE$ $STATETYPE$
command[handle-router-event]=/usr/local/netsaint/libexec/eventhandlers/handle-router-event $HOSTSTATE$ $STATETYPE$
```

This assumes that you have placed the event handler scripts in the */usr/local/netsaint/libexec/eventhandlers* directory. You may place them anywhere you wish, but you'll need to modify the examples I've given here.

Event Handler Scripts

Okay, now lets take a look at what the event handler scripts look like...

Host Event Handler (handle-master-host-event)

```
#!/bin/sh

# Only take action on hard host states...
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # The master host has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enter active mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_active_mode
                ;;
        UP)
                # The master host has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # enter standby mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_standby_mode
                ;;
        esac
        ;;
esac
exit 0
```

Service Event Handler (handle-master-proc-event)

```
#!/bin/sh

# Only take action on hard service states...
case "$2" in
HARD)

        case "$1" in

        CRITICAL)
                # The master NetSaint process is not running!
                # We should now become the master host and
                # take over the responsibility of monitoring
                # the network, so enter active mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_active_mode
                ;;

        WARNING)
                ;;
        UNKNOWN)
                ;;
                # The master NetSaint process may or may not
                # be running.. We won't do anything here, but
                # to be on the safe side you may decide you
                # want the slave host to become the master in
                # these situations...

        OK)
                # The master NetSaint process running again!
                # We should go back to being the slave host,
                # so enter standby mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_standby_mode
                ;;
        esac
        ;;
esac
exit 0
```

Host Event Handler (handle-router-event)

```
#!/bin/sh

# Only take action on hard host states...
case "$2" in
HARD)
        case "$1" in
        DOWN)
                # The router has gone down!
                # We should now become the master host and take
                # over the responsibilities of monitoring the
                # network, so enter active mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_active_mode
                ;;
        UP)
                # The router has recovered!
                # We should go back to being the slave host and
                # let the master host do the monitoring, so
                # enter standby mode...
                /usr/local/netsaint/libexec/eventhandlers/enter_standby_mode
                ;;
        esac
        ;;
esac
exit 0
```

What This Does For Us

When things first start out, host A (the master host) is in *active* mode. This means that it monitors all services and sends out notifications if there are problems or recoveries. Host F (the slave host) is in *standby* mode, which means that it will monitor all services but will *not* send out any notifications.

The NetSaint process on host F becomes the master host when...

- Host A goes down (the *handle-master-host-event* host event handler is executed).
- The NetSaint process on host A is not running (the *handle-master-proc-event* service event handler is executed). If either router H or I goes down (the *handle-router-event* host event handler is executed).

When the NetSaint process on host F has entered active mode, it will be able to send out notifications about any service or host problems or recoveries. At this point host F has effectively taken over the responsibility of monitoring the network!

The NetSaint process on host F returns to being the slave host when...

- Host A has recovers (the *handle-master-host-event* host event handler is executed).
- The NetSaint process on host A recovers (the *handle-master-proc-event* service event handler is executed). If either router H or I recovers (the *handle-router-event* host event handler is executed).

When the NetSaint process on host F has entered standby mode, it will not send out notifications about any service or host problems or recoveries. At this point host F has handed over the responsibilities of monitoring the network back to host A. Everything is now as it was when we first started!

Shortcomings

This simple example has some shortcomings that you should be aware of. Note that when one of the routers goes down, the NetSaint process on host F acts as if the NetSaint process on host A is no longer running. This may or may not be the case. If the process on host A *is* running, you'll get potentially bogus notifications being sent out from both NetSaint processes...

As an example, lets say that router H goes down and severs the connection between the two network segments, but everything else is okay. From the view of the NetSaint process on host F, all hosts beyond router H (hosts A, B, C, D, E, and I) are unreachable. At the same time, the NetSaint process on host A (which is on the other side of router H) thinks that all hosts beyond router H (hosts F and G) are unreachable. Both NetSaint processes see that router H is down, but that's the only thing they agree on. This might lead to an enormous amount of bogus notifications being sent out to you. You could potentially get two notifications about router H being down (one from each process) and one notification about every other host on the network being unreachable!

Scenario 3 - A Smarter Way To Implement Redundancy Across Network Segments

Introduction

This is basically just an improvement in the redundancy logic described above in scenario 2. What we will do is make both monitoring hosts aware of each other. In scenario 2, the slave host (host F) knew about the master host (host A), but the master was unaware of the slave. In this scenario both the slave and master hosts will be aware of each other, and will use that information to make better decisions on how to take over or adjust monitoring responsibilities.

Goals

We have several goals with this redundancy scenario...

The "slave" host running NetSaint should take over the job of monitoring *the entire network* if:

1. The NetSaint process on the "master" host stops running for some reason
2. The "master" host that runs NetSaint is down
3. The "master" host becomes unreachable due to one or both of the routers going down and the "master" host was last known to be either down or unreachable

The "slave" host running NetSaint should take over the job of monitoring *only its local network segment* if:

1. The "master" host becomes unreachable due to one or both of the routers going down and the "master" host was last known to be up

The "master" host running NetSaint should *stop* monitoring the entire network and change to monitoring *only its local network segment* if:

1. The "slave" host becomes unreachable due to one or both of the routers going down and the "slave" host was last known to be up

Network Layout Diagram

See network diagram for scenario 2 - its the same...

Initial Program Modes

The master host (host A) should have its initial program mode set to *active*, while the slave host (host F) should have its initial program mode set to *standby*. This is the same setup as described in scenario 2.

Initial Configuration

Scenario 4 - Implementing Multiple Redundancy Methods

If you've got a large, complex network and are paranoid about ensuring that NetSaint monitors everything, you'll probably want to look into implementing multiple redundancy methods. This basically involves combining the redundancy methods described in scenarios 1 and 3 to create a pool of monitoring hosts that are all aware of each other's state and can take over all or part of the network monitoring responsibilites if necessary. If you found the concepts presented in scenario 3 difficult to understand, you should be aware that the complexity of configuration files and event handler scripts will grow exponentially as you add additional monitoring hosts to a multiple redundancy setup.

Since there are endless possibilities for implementing multiple redundancy methods, I won't try to discuss them here. If you decide to implement mixed redundancy methods on your network be prepared to spend a *lot* of time analyzing your network structure, its critical failure points (i.e. routers, firewalls, etc.), the location of monitoring hosts, and what should happen at each monitoring host in the event of a problem. When implementing multiple redundancy methods you cannot simply create event handler scripts based on the state of routers, etc. - you must also take into account the state of other monitoring hosts on the local network segment and (possibly) on other segments.

External Commands

Introduction

NetSaint can process commands from external applications (including CGIs - see the command CGI for an example) and alter various aspects of its monitoring functions based on the commands it receives.

Enabling External Commands

By default, NetSaint does not check for or process any external commands. If you want to enable external command processing, you'll have to do the following...

- Enable external command checking with the check_external_commands option
- Set the frequency of command checks with the command_check_interval option
- Specify the location of the command file with the command_file option

Note: If external applications or CGIs will be issuing commands to NetSaint, you will have to grant the user that those processes run as permission to write to the command file. An outline of how to configure proper permissions for the command file can be found here.

When Does NetSaint Check For External Commands?

- At regular intervals specified by the command_check_interval option in the main configuration file
- Immediately after event handlers are executed. This is in addtion to the regular cycle of external command checks and is done to provide immediate action if an event handler submits commands to NetSaint.

Using External Commands

External commands can be used to accomplish a variety of things while NetSaint is running. Example of what can be done include changing program modes, temporarily disabling notifications for services and hosts, temporarily disabling service checks, forcing immediate service checks, adding comments to hosts and services, etc.

External Command Examples

Some example scripts that can be used to issue commands to NetSaint can be found in the *eventhandlers/* subdirectory of the NetSaint distribution. You may have to modify the scripts to accomodate for differences in system command syntaxes, file and directory locations, etc.

Command Format

External commands that are written to the command file have the following format...

[*time*] *command_id;command_arguments*

...where *time* is the time (in *time_t* format) that the external application or CGI committed the external command to the command file. The various commands that are available, along with their *command_id* and a description of their *command_arguments*, can be found in the table below.

Implemented Commands

This is a description of the external commands which have been implemented in NetSaint thus far. More commands will be added in future releases. Note that all time arguments should be specified in *time_t* format (seconds since the UNIX epoch).

Command ID	Command Arguments	Command Description
ADD_HOST_COMMENT	<host_name>;<persistent>;<author>;<comment>	This command is used to associate a comment with the specified host. The *author* argument generally contains the name of the person who entered the comment. The actual comment should not contain any semi-colons. The persistent flag determines whether or not the comment will survive program restarts (1=save comment across program restarts, 0=delete comment on restart).

ADD_SVC_COMMENT	<host_name>;<service_description>;<persistent>;<author>;<comment>	This command is used to associate a comment with the specified host. Note that both the host name and service description are required. The *author* argument generally contains the name of the person who entered the comment. The actual comment should not contain any semi-colons. The persistent flag determines whether or not the comment will survive program restarts (1=save comment across program restarts, 0=delete comment on restart).
DEL_HOST_COMMENT	<comment_id>	This is used to delete a comment having a ID matching *comment_id* for the specified host.
DEL_ALL_HOST_COMMENTS	<host_name>	This is used to delete all comments associated with the specified host.
DEL_SVC_COMMENT	<comment_id>	This is used to delete a comment having a ID matching *comment_id* for the specified service.
DEL_ALL_SVC_COMMENTS	<host_name>;<service_description>	This is used to delete all comments associated with the specified service. Note that both the host name and service description are required.
DELAY_HOST_NOTIFICATION	<host_name>;<next_notification_time>	This will delay the next notification about this host until the time specified by the *next_notification_time* argument. This will have no effect if the host state changes before the next notification is scheduled to be sent out.
DELAY_SVC_NOTIFICATION	<host_name>;<service_description>;<next_notification_time>	This will delay the next notification about this service until the time specified by the *next_notification_time* argument. Note that both the host name and service description are required. This will have no effect if the service state changes before the next notification is scheduled to be sent out. This *does not* delay notifications about the host.
SCHEDULE_SVC_CHECK	<host_name>;<service_description>;<next_check_time>	This will reschedule the next check of the specified service for the time specified by the *next_check_time* argument. Note that both the host name and service description are required.
SCHEDULE_HOST_SVC_CHECKS	<host_name><next_check_time>	This will reschedule the next check of all services on the specified host for the time specified by the *next_check_time* argument.

ENABLE_SVC_CHECK	<host_name>;<service_description>	This will re-enable checks of the specified service. Note that both the host name and service description are required.
DISABLE_SVC_CHECK	<host_name>;<service_description>	This will temporarily disable checks of the specified service. Service checks are automatically re-enabled when NetSaint restarts. Issuing this command will have the side effect of temporarily preventing notifications from being sent out for the service. It *does not* prevent notifications about the host from being sent out.
ENABLE_SVC_NOTIFICATIONS	<host_name>;<service_description>	This is used to re-enable notifications for the specified service. Note that both the host name and service description are required.
DISABLE_SVC_NOTIFICATIONS	<host_name>;<service_description>	This is used to temporarily disable notifications from being sent out about the specified service. Notifications are automatically re-enabled when NetSaint restarts. Note that both the host name and service description are required. This *does not* disable notifications for the host.
ENABLE_HOST_SVC_NOTIFICATIONS	<host_name>	This is used to re-enable notifications for all services on the specified host. This *does not* enable notifications for the host.
DISABLE_HOST_SVC_NOTIFICATIONS	<host_name>	This is used to temporarily disable notifications for all services on the specified host. This *does not* disable notifications for the host.
ENABLE_HOST_SVC_CHECKS	<host_name>	This will re-enable checks of all services on the specified host. If one or more services were in a non-OK state when they were disabled, contacts may receive notifications if the service(s) recover after the checks are re-enabled.

DISABLE_HOST_SVC_CHECKS	<host_name>	This will temporarily disable checks of all services on the specified host. Service checks are automatically re-enabled when NetSaint restarts. Issuing this command will have the side effect of temporarily preventing notifications from being sent out for any of the affected services. It *does not* prevent notifications about the host from being sent out.
ENABLE_HOST_NOTIFICATIONS	<host_name>	This will temporarily disable notifications for this host. Note that this *does not* enable notifications for the services associated with this host.
DISABLE_HOST_NOTIFICATIONS	<host_name>	This will temporarily disable notifications for this host. Notifications are automatically re-enabled when NetSaint restarts. Note that this *does not* disable notifications for the services associated with this host.
ENABLE_ALL_NOTIFICATIONS_BEYOND_HOST	<host_name>	This will enable notifications for all hosts and services "beyond" the host specified by the *host_name* argument (from the view of NetSaint). This command is most often used in conjunction with redundant monitoring hosts.
DISABLE_ALL_NOTIFICATIONS_BEYOND_HOST	<host_name>	This will temporarily disable notifications for all hosts and services "beyond" the host specified by the *host_name* argument (from the view of NetSaint). Notifications are automatically re-enabled when NetSaint restarts. This command is most often used in conjunction with redundant monitoring hosts.
ENTER_STANDBY_MODE	<execution_time>	This will change the current program mode to *Standby* at the time specified by the *execution time* argument.
ENTER_ACTIVE_MODE	<execution_time>	This will change the current program mode to *Active* at the time specified by the *execution time* argument.
SHUTDOWN_PROGRAM	<execution_time>	This will cause NetSaint to shutdown at the time specified by the *execution_time* argument. Note: NetSaint cannot be restarted via the web interface once it has been shutdown.

RESTART_PROGRAM	<execution_time>	This will cause NetSaint to flush all configuration state information, re-read all the config files, and restart monitoring at the time specified by the *execution_time* argument
PROCESS_SERVICE_CHECK_RESULT	<host_name>;<service_description>;<return_code>;<plugin_output>	This command is used to submit check results for a particular service to NetSaint. These "passive" checks are acted upon in the same manner as normal "active" checks. More information on passive service checks can be found here.
SAVE_STATE_INFORMATION	<execution_time>	This will force NetSaint to dump current state information for all services and hosts to the file specified by the state_retention_file variable. You must enable the retain_state_information option for this to work.
READ_STATE_INFORMATION	<execution_time>	This will force NetSaint to read previously saved state information for all services and hosts from the file specified by the state_retention_file variable. You must enable the retain_state_information option for this to work.
START_EXECUTING_SVC_CHECKS		This is used to resume the execution of service checks. The execution of service checks may have been stopped at an earlier time by either receiving a *STOP_EXECUTING_SVC_CHECKS* command, or by setting the execute_service_checks option in the main config file to 0. Most often used when implementing redundant monitoring hosts.
STOP_EXECUTING_SVC_CHECKS		This is used to stop the execution of service checks. When service checks are not being executed, NetSaint will not keep requeuing checks for a later time, but will not actually execute any checks. This essentially puts NetSaint into a "sleep" mode, as far as monitoring is concerned. Most often used when implementing redundant monitoring hosts.

START_ACCEPTING_PASSIVE_SVC_CHECKS		This is used to resume the acceptance of passive service checks for all services. The acceptance of passive service checks may have been stopped at an earlier time by either receiving a *STOP_ACCEPTING_PASSIVE_SVC_CHECKS* command, or by setting the accept_passive_service_checks option in the main config file to 0. If passive checks have been disabled for specific services using the *DISABLE_PASSIVE_SVC_CHECKS* command, passive checks will *not* be accepted for those services, but will for all others.
STOP_ACCEPTING_PASSIVE_SVC_CHECKS		This is used to disable the acceptance of passive service checks for all services.
ENABLE_PASSIVE_SVC_CHECKS	<host_name>;<service_description>	This is used to resume the acceptance of passive service checks for a specific service. The acceptance of passive checks may have been disabled for a service at an earlier time by receiving a *DISABLE_PASSIVE_SVC_CHECKS* command. If passive checks have been disabled for all services either by using the *STOP_ACCEPTING_PASSIVE_SVC_CHECKS* command or by setting the accept_passive_service_checks option in the main config file to 0, passive checks will *not* be accepted for this service.
DISABLE_PASSIVE_SVC_CHECKS	<host_name>;<service_description>	This is used to disable the acceptance of passive service checks for a specific service.

External Command File Permissions

Introduction

One of the most common problems people have seems to be with setting proper permissions for the external command file. Since the command file is deleted by NetSaint every time it processes the external commands, we need to set the proper permission on the */usr/local/netsaint/var/rw* **directory** (or whatever the path portion of the command_file directive in your main configuration file is set to). I'll show you how to do this. Note: You must be *root* in order to do some of these steps...

Users and Groups

First, find the user that your web server process is running as. On many systems this is the user *nobody*, although it will vary depending on what OS/distribution you are running. You'll also need to know what user Netsaint is effectively running as - this is usually the user *netsaint*.

Next, create a new group called '**nscmd**'. On RedHat Linux you can use the following command to add a new group (other systems may differ):

```
/usr/sbin/groupadd nscmd
```

Next, add the web server user (*nobody*) and the NetSaint user (*netsaint*) to the newly created group with the following commands:

```
/usr/sbin/usermod -G nscmd netsaint
/usr/sbin/usermod -G nscmd nobody
```

Creating the directory

Next, create the directory where the command file should be stored. By default, this is */usr/local/netsaint/var/rw*, although it can be changed by modifying the path specified in thecommand_file directory.

```
mkdir /usr/local/netsaint/var/rw
```

Setting directory permissions

Next, change the ownership of the directory that will be used to hold the command file...

```
chown netsaint.nscmd /usr/local/netsaint/var/rw
```

Make sure the NetSaint user has full permissions on the directory...

chmod o+rwx /usr/local/netsaint/var/rw

Make sure the group we created has read and write permissions on the directory. Also, make sure to specify the 's' flag.

chmod g+rws /usr/local/netsaint/var/rw

Verifying the permissions

Check the permissions on the rw/ subdirectory by running '**ls -al /usr/local/netsaint/var**'. You should see something similiar to the following:

```
drwxrws---   2 netsaint nscmd     1024 Aug 11 16:30 rw
```

Note that the user *netsaint* is the owner of the directory and the group *nscmd* is the group owner of the directory. The *netsaint* user has **rwx** permissions and group *nscmd* has **rw** permissions on the directory. The **s** flag indicates that whenever a someone creates a new file (or modifies an existing file) in the rw subdirectory, the permissions on the file will be set so that group *nscmd* is the group owner of that file. That's what we want...

Notes...

If you supplied the **--with-command-grp=*somegroup*** option when running the configure script, you can create the directory to hold the command file and set the proper permissions automatically by running '**make install-commandmode**'.

Passive Service Checks

Introduction

Beginning with release 0.0.6, NetSaint can now process service check results that are submitted by external applications. Service checks which are performed and submitted to NetSaint by external apps are called *passive* checks. Passive checks can be contrasted with *active* checks, which are service checks that have been initiated by NetSaint.

Why The Need For Passive Checks?

Passive checks are useful for monitoring services that are:

- located behind a firewall, and can therefore not be checked actively from the host running NetSaint
- asynchronous in nature and can therefore not be actively checked in a reliable manner (e.g. SNMP traps, security alerts, etc.)

How Do Passive Checks Work?

The only real difference between active and passive checks is that active checks are initiated by NetSaint, while passive checks are performed by external applications. Once an external application has performed a service check (either actively or by having received an synchronous event like an SNMP trap or security alert), it submits the results of the service "check" to NetSaint through the external command file.

The next time NetSaint processes the contents of the external command file, it will place the results of all passive service checks into a queue for later processing. The same queue that is used for storing results from active checks is also used to store the results from passive checks.

NetSaint will periodically execute a service reaper event and scan the service check result queue. Each service check result, regardless of whether the check was active or passive, is processed in the same manner. The service check logic is exactly the same for both types of checks. This provides a seamless method for handling both active and passive service check results.

How Do External Apps Submit Service Check Results?

External applications can submit service check results to NetSaint by writing a PROCESS_SERVICE_CHECK_RESULT external command to the external command file.

The format of the command is as follows:

[<timestamp>]
PROCESS_SERVICE_CHECK_RESULT;<host_name>;<description>;<return_code>;<plugin_output>

where...

- *timestamp* is the time in time_t format (seconds since the UNIX epoch) that the service check was perfomed (or submitted). Please note the single space after the right bracket.
- *host_name* is the short name of the host associated with the service in the service definition
- *description* is the description of the service as specified in the service definition
- *return_code* is the return code of the check (0=OK, 1=WARNING, 2=CRITICAL, -1=UNKNOWN)
- *plugin_output* is the text output of the service check (i.e. the plugin output)

Note that in order to submit service checks to NetSaint, a service must have already been defined in the host configuration file! NetSaint will ignore all check results for services that had not been configured before it was last (re)started.

An example shell script of how to submit passive service check results to NetSaint can be found in the documentation on volatile services.

Submitting Passive Service Check Results From Remote Hosts

If an application that resides on the same host as NetSaint is sending passive service check results, it can simply write the results directly to the external command file as outlined above. However, applications on remote hosts can't do this so easily. In order to allow remote hosts to send passive service check results to the host that runs NetSaint, I've developed the nsca addon. The addon consists of a daemon that runs on the NetSaint hosts and a client that is executed from remote hosts. The daemon will listen for connections from remote clients, perform some basic validation on the results being submitted, and then write the check results directly into the external command file (as described above). More information on the nsca addon can be found here...

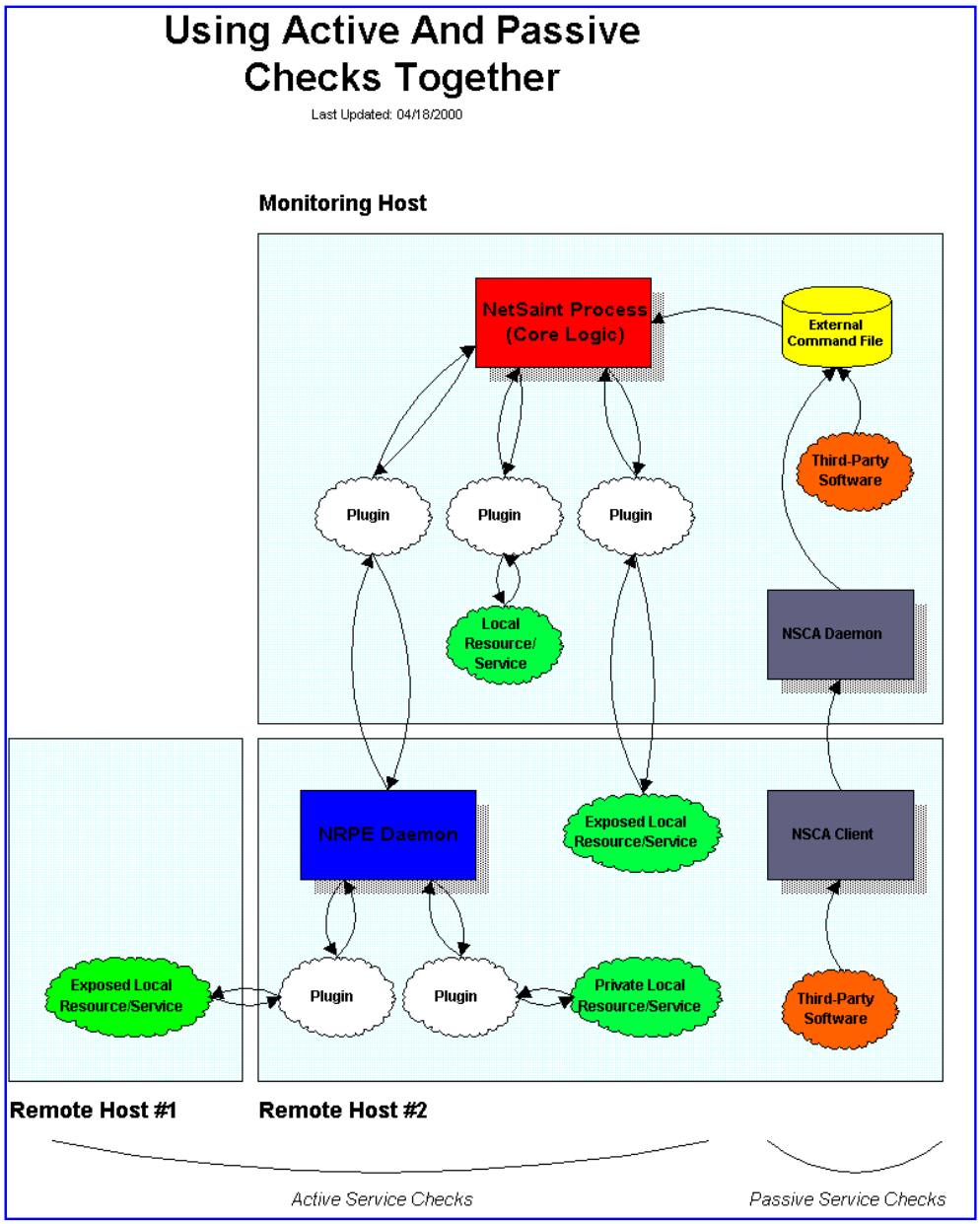
Using Both Active And Passive Service Checks

Unless you're implementing a distributed monitoring environment with the central server accepting only passive service checks (and not performing any active checks), you'll probably be using both types of checks in your setup. As mentioned before, active checks are more suited for services that lend themselves to periodic checks (availability of an FTP or web server, etc), whereas passive checks are better off at handling asynchronous events that occur at variable intervals (security alerts, etc.).

The image below gives a visual representation of how active and passive service checks can both be used to monitor network resources (click on the image for a larger version).

The orange bubbles on the right side of the image are third-party applications that submit passive check results to NetSaint's external command file. One of the applications resides on the same host as NetSaint, so it can write directly to the command file. The other application resides on a remote host and makes used of the nsca client program and daemon to transfer the passive check results to NetSaint.

The items on the left side of the image represent active service checks that NetSaint is performing. I've shown how the checks can be made for local resources (disk usage, etc.), "exposed" resources on remote hosts (web server, FTP server, etc.), and "private" resources on remote hosts (remote host disk usage, processor load, etc.). In this example, the private resources on the remote hosts are actually checked by making use of the nrpe addon, which facilitates the execution of plugins on remote hosts.


Using Active And Passive Checks Together
Last Updated: 04/18/2000
Monitoring Host
NetSaint Process (Core Logic)
External Command File
Third-Party Software
Plugin
Plugin
Plugin
Local Resource/ Service
NSCA Daemon
NRPE Daemon
Exposed Local Resource/Service
NSCA Client
Exposed Local Resource/Service
Plugin
Plugin
Private Local Resource/Service
Third-Party Software
Remote Host #1
Remote Host #2
Active Service Checks
Passive Service Checks

Volatile Services

Introduction

Beginning with release 0.0.6 of NetSaint, service definitions have been extended to allow for a distinction between "normal" services and "volatile" services. The <*volatile*> option in each service definition allows you to specify whether a specific service is volatile or not. For most people, the majority of all monitored services will be non-volatile (i.e. "normal"). However, volatile services can be very useful when used properly...

What Are They Useful For?

Volatile services are useful for monitoring...

- things that automatically reset themselves to an "OK" state each time they are checked
- events such as security alerts which require attention every time there is a problem (and not just the first time)

What's So Special About Volatile Services?

Volatile services differ from "normal" services in three important ways. *Each time* they are checked when they are in a hard non-OK state, and the check returns a non-OK state (i.e. no state change has occurred)...

- the non-OK service state is logged
- contacts are notified about the problem (if that's what should be done)
- the event handler for the service is run (if one has been defined)

These events normally only occur for services when they are in a non-OK state and a hard state change has just occurred. In other words, they only happen the first time that a service goes into a non-OK state. If future checks of the service result in the same non-OK state, no hard state change occurs and none of the events mentioned take place again.

The Power Of Two

If you combine the features of volatile services and passive service checks, you can do some very useful things. Examples of this include handling SNMP traps, security alerts, etc.

How about an example... Let's say you're running Psionic Software's PortSentry product (which is free, by the way) to detect port scans on your machine and automatically firewall potential intruders. If you want to let NetSaint know about port scans, you could do the following..

In NetSaint:

- Configure a service called *Port Scans* and associate it with the host that PortSentry is running on.
- Set the <*max_attempts*> option in the service definition to 1. This will tell NetSaint to immediate force the service into a hard state when a non-OK state is reported.
- Set the <*check_time*> option in the service definition to a timeperiod that contains *no* valid time ranges. This will prevent NetSaint from ever actively checking the service. Even though the service check will get scheduled, it will never actually be checked.

In PortSentry:

- Edit your PortSentry configuration file (portsentry.conf), define a command for the **KILL_RUN_CMD** directive as follows:

  KILL_RUN_CMD="/usr/local/netsaint/libexec/eventhandlers/submit_check_result <*host_name*> 'Port Scans' 2 'Port scan from host $TARGET$ on port $PORT$. Host has been firewalled.'"

  Make sure to replace <*host_name*> with the short name of the host that the service is associated with.

Create a shell script in the */usr/local/netsaint/libexec/eventhandlers* directory named *submit_check_result*. The contents of the shell script should be something similiar to the following...

```
#!/bin/sh

# Write a command to the NetSaint command file to cause
# it to process a service check result

echocmd="/bin/echo"

CommandFile="/usr/local/netsaint/var/rw/netsaint.cmd"

# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

Note that if you are running PortSentry as root, you will have to make additions to the script to reset file ownership and permissions so that NetSaint and the CGIs can read/modify the command file. Details on permissions/ownership of the command file can be found here.

So what happens when PortSentry detects a port scan on the machine?

- It blocks the host (this is a function of the PortSentry software)
- It executes the *submit_check_result* shell script to send the security alert info to NetSaint
- NetSaint reads the command file, recognized the port scan entry as a passive service check
- NetSaint processes the results of the service by logging the CRITICAL state, sending notifications to contacts (if configured to do so), and executes the event handler for the *Port Scans* service (if one is defined)

Time Periods

or...
"Is This a Good Time?"

Introduction

With the release 0.0.4 the notion of time periods was introduced. Time periods allow you to have greater control over when service checks may be run, when host and service notifications may be sent out, and when contacts may receive notifications. With this newly added power come some potential problems, as I will describe later. I was initially very hesitant to introduce time periods because of these snafus. I'll leave it up to you to decide what it right for your particular situation...

How Time Periods Work With Service Checks

Previous to release 0.0.4, NetSaint would monitor all services that you had defined 24 hours a day, 7 days a week. While this is fine for most services that need monitoring, it doesn't work out so well for others. For instance, do you really need to monitor printers all the time when they're really only used during normal business hours? Perhaps you have development servers which you would prefer to have up, but aren't "mission critical" and therefore don't have to be monitored for problems over the weekend. Time period definitions now allow you to have more control over when such services may be checked...

The <*check_period*> argument of each service definition allows you to specify a time period that tells NetSaint when the service can be checked. When NetSaint attempts to reschedule a service check, it will make sure that the next check falls within a valid time range within the defined time period. If it doesn't, NetSaint will adjust the next service check time to coincide with the next "valid" time in the specified time period. This means that the service may not get checked again for another hour, day, or week, etc.

Potential Problems With Service Checks

If you use time periods which do not cover a 24x7 range, you *will* run into problems, especially if a service (or its corresponding host) is down when the check is delayed until the next valid time in the time period. Here are some of those problems...

1. Contacts will not get re-notified of problems with a service until the next service check can be run.
2. If a service recovers during a time that has been excluded from the check period, contacts will not be notified of the recovery.
3. The status of the service will appear unchanged (in the status log and CGI) until it can be

checked next.

4. If all services associated with a particular host are on the same check time period, host problems or recoveries will not be recognized until one of the services can be checked (and therefore notifications may be delayed or not get sent out at all).

Limiting the service check period to anything other than a 24 hour a day, 7 days a week basis can cause a lot of problems. Well, not really problems so much as annoyances and inaccuracies... Unless you have good reason to do so, I would *strongly* suggest that you set the <*check_period*> argument of each service definition to a "24x7" type of time period.

How Time Periods Work With Contact Notifications

Probably the best use of time periods is to control when notifications can be sent out to contacts. By using the <*svc_notification_period*> and <*host_notification_period*> arguments in contact definitions, you're able to essentially define an "on call" period for each contact. Note that you can specify different time periods for host and service notifications. This is helpful if you want host notifications to go out to the contact any day of the week, but only have service notifications get sent to the contact on weekdays. It should be noted that these two notification periods should cover *any time* that the contact can be notified. You can control notification times for specific services and hosts on a one-by-one basis as follows...

By setting the <*notification_period*> argument of the host definition, you can control when NetSaint is allowed to send notifications out regarding problems or recoveries for that host. When a host notification is about to get sent out, NetSaint will make sure that the current time is within a valid range in the <*notification_period*> time period. If it is a valid time, then NetSaint will attempt to notify each contact of the host problem. Some contacts may not receive the host notification if their <*host_notification_period*> does not allow for host notifications at that time. If the time is *not* valid within the <*notification_period*> defined for the host, NetSaint will not send the notification out to *any* contacts. A logic diagram outlining the basic decisions NetSaint makes when sending out host notifications can be found here.

You can control notification times for services in a similiar manner to host notification times. By setting the <*notification_period*> argument of the service definition, you can control when NetSaint is allowed to send notifications out regarding problems or recoveries for that service. When a service notification is about to get sent out, NetSaint will make sure that the current time is within a valid range in the <*notification_period*> time period. If it is a valid time, then NetSaint will attempt to notify each contact of the service problem. Some contacts may not receive the service notification if their <*svc_notification_period*> does not allow for service notifications at that time. If the time is *not* valid within the <*notification_period*> defined for the service, NetSaint will not send the notification out to *any* contacts. A logic diagram outlining the basic decisions NetSaint makes when sending out service notifications can be found here.

Potential Problems With Contact Notifications

There aren't really any major problems that you'll run into with using time periods to create custom

contact notification times. You do, however, need to be aware that contacts may not always be notified of a service or host problem or recovery. If the time isn't right for both the host or service notification period and the contact notification period, the notification won't go through. Once you weigh the potential problems of time-restricted notifications against your needs, you should be able to come up with a configuration that works well for your situation.

Conclusion

Time periods allow you to have greater control of how NetSaint performs its monitoring and notification functions, but can lead to problems. If you are unsure of what type of time periods to implement, or if you are having problems with your current implementation, I would suggest using "24x7" time periods (where all times are valid for each day of the week). Feel free to contact me if you have questions or are running into problems.

T-Online

Private Homepage

Das Dokument konnte nicht gefunden werden.

Bitte überprüfen Sie Ihren URL

Hinweis :

Bei Dateinamen wird auf Groß- und Kleinschreibung unterschieden

We are sorry, but the private homepage server could not find the file you asked for.
Please check the URL to ensure that the path is correct.

404

Weiter mit

Häufig gestellte Fragen (FAQs)

Inhalt

- Probleme beim Übersetzen (Compilieren) von NetSaint
- Probleme beim Übersetzen der statusmap und/oder trends CGIs ...oder.. "Wo kann ich die statusmap und trends CGIs finden?"
- "NetSaint process may not be running" Warnung in den CGIs
- Hosts werden fälschlicherweise als DOWN gelistet und/oder Dienste haben den Status "HOST DOWN"
- When hosts go down, I get notification about services instead of hosts and the service notifications contain incorrect data
- Can I monitor a host without defining any services for it?
- How can I change the timeout values for service checks?
- "Return code of *x* is out of bounds" errors

- Debugging "unknown variable" errors during configuration verification or runtime
- Running multiple instances of NetSaint on the same machine
- Missing data in the CGIs or errors about improper authorization
- Wo kann ich die traceroute und daemonchk CGIs finden?
- Requiring users to authenticate before accessing web interface
- Displaying pretty host icons
- Errors commiting commands via the command CGI
- NetSaint shuts down with warnings about permissions on the command file
- Monitoring remote host information
- Monitoring Windows NT servers
- Monitoring Novell servers
- Sending SNMP traps to management hosts
- Logging events to an external database
- Troubleshooting problems with NetSaint

Ich habe Probleme mit dem Übersetzen von Netsaint - Was kann ich tun?

Das Kompilieren (Übersetzen) von NetSaint auf verschiedenen Betriebssystemen scheint kein Problem mehr zu sein, soweit nicht einige String-Funktionen fehlen...

Wenn Fehlermeldungen zu den Funktionen **strncat()**, **strncpy()** , oder **snprintf()** auftreten, ist sehr wahrscheinlich die glibc Bibliothek nicht auf dem System installiert. Dies scheint meistens auf HP-UX und Solaris der Fall zu sein. Ich habe versucht mit diesen Funktionen in NetSaint und den CGIs mögliche Bufferüberläufe zu vermeiden, sie sind also überall im Code verteilt. Wenn die glibc Bibliothek aus irgendeinem Grund nicht installiert werden soll, gilt es einen andern Weg zu finden alles zu kompilieren. Wenn nur die snprintf() Funktion fehlt, kann die Datei snprintf.c von http://www.ijs.si/software/snprintf/ heruntergeladen werden und den Makefiles hinzugefügt werden, so daß sie beim Compilieren miteinbezogen wird.

Ich kann die statusmap und/oder trends CGIs nicht finden oder habe Probleme beim Kompilieren...

Wenn alle CGIs kompiliert wurden, aber das statusmap CGI oder trends CGI nicht zu finden sind, wurde wahrscheinlich Thomas Boutell's gd library nicht korrekt auf dem System installiert. Die **gd** library (und so auch die statusmap CGI) benötigt auch die **zlib** und **png** libraries. Erforderlich ist die Version 1.6.3 oder höher der gd library, da das CGI eine PNG Grafik des Netzwerklayouts erzeugt.

Wenn also das statusmap CGI nicht kompiliert wurde, sollte sichergestellt werden, daß die gd library auf dem System installiert ist. Dann ist '**make clean**' und das configure script mit den folgenden Optionen neu auszuführen:

./configure --with-gd-lib=LIBDIR --with-gd-inc=INCDIR [other options...]

Ersetze LIBDIR mit dem Verzeichnis in dem die gd library installiert ist (normalerweise /usr/lib oder /usr/local/lib) und ersetze INCDIR mit dem Verzeichnis in dem die Headerdateien für die gd library installiert sind (normalerweise /usr/include oder /usr/local/include).

Nachdem das configure script neu ausgeführt wurde, müssen die CGIs neu kompiliert und an den richtigen Ort installiert werden.

Wenn es viele Schwierigkeiten macht, die Parameter dem configure script bekanntzugeben und die gd libraries nicht erkannt werden (und RedHat Linux benutzt wird), empfehle ich die Installation der Version **1.8.1-2** der *gd* und *gd-devel* RPMs von www.rpmfind.net - diese Bibliothek scheint ohne besondere Argumente vom configure script erkannt zu werden.

"NetSaint process may not be running" Warnung in den CGIs

Wenn irrtümliche Meldungen über einen angeblich nicht laufenden NetSaint Prozess beim Betrachten der CGIs erscheinen, ist wahrscheinlich einer der folgenden Punkte der Grund hierfür:

1. Es wurde kein Befehl (command) zur Statusüberprüfung des NetSaint Prozesses definiert. Dies wird definiert durch die Angabe eines Wertes für die process_check_command Variable in der CGI-Konfigurationsdatei.

2. Wenn ein Befehl definiert wurde gibt er vielleicht nicht korrekten Exitcode zurück. Der Befehl muß denselben Regeln gehorchen wie die anderen Plugins (mehr Infos sind unter plugin guidelines zu finden): ein Rückgabewert von 0 bedeutet, daß NetSaint läuft; andere Werte weisen darauf hin, daß NetSaint nicht läuft oder sich in einem schlechten (degraded) Zustand befindet.

3. Wenn das check_netsaint Plugin benutzt wird, sollte die Richtigkeit der an das Plugin zu übergebenden Argumente überprüft werden. Das erste Argument ist der vollständige Pfad zur Statusdatei, das zweite Argument ist die Anzahl der Minuten, die die Statusdatei höchstens "alt" sein darf und das dritte Argument ist eine Zeichenkette, die der von dem Befehl *ps* zurückgegebenen Zeile für den NetSaint Prozeß entspricht. Probieren sie *ps axuw | grep netsaint* aus, um herauszufinden wie diese Zeichenkette aussehen sollte - ein häufiges Beispiel einer passenden Zeichenkette ist *"/usr/local/netsaint/bin/netsaint -d /usr/local/netsaint/etc/netsaint.cfg"*

4. Wenn das check_netsaint Plugin zur Statusüberprüfung benutzt wird, sollte sichergestellt werden, daß das Plugin wie gewünscht funktioniert. Dazu kann einfach das check_netsaint Plugin von der Kommandozeile ausgeführt und die Rückgabewerte geprüft werden. Wenn das Plugin meldet, daß der NetSaint Prozeß nicht gefunden werden kann oder die Nachricht "Could not open pipe" zurückgegeben wird, kann es erforderlich sein die PS_RAW_COMMAND Definition in der Datei common/config.h der Plugin Distribution zu editieren um die Syntax des **ps** Befehls für das entsprechende System anzupassen. Zum Beispiel sollte sie unter FreeBSD entweder *"/bin/ps -ao 'state user ppid args'"* oder *"/bin/ps -axo 'state user ppid command'"* (es scheint unterschiedlich zu sein) lauten. Nachdem die PS_RAW_COMMAND Definition geändert wurde, müssen die Plugins neu kompiliert werden. Dann ist am besten, gleich das neu kompilierte check_netsaint Plugin zu testen.

Die CGIs werden keine Befehle ausführen solange sie denken daß der NetSaint Prozeß nicht läuft. Das dient in erster Linie dazu, unbeabsichtigt eingegebene mehrfache shutdown/restart Befehle zu verhindern, die eh nicht ausgeführt werden, solange NetSaint irgendwann in der Zukunft gestartet wird.

Hosts werden fälschlicherweise als DOWN gelistet und/oder Dienste haben den Status "HOST DOWN"

Das scheint eines der größten Probleme für neue Benutzer zu sein. In 99.9% aller Fälle passiert dies wegen einer fehlerhaften Befehlsdefinition des host check Befehls in der Hostkonfigurationsdatei (host definition).

Eine Hauptursache für dieses Problem war eine Syntaxänderung der Kommandozeilenargumente des check_ping Plugin. Es sollte sichergestellt werden, daß der Hostüberprüfungsbefehl (host check command) die richtige Syntax für die benutzte Version des check_ping Plugin benutzt. Ob der Befehl richtig funktioniert kann überprüft werden, indem er einfach von der Kommandozeile ausgeführt wird. Neuere Versionen des check_ping Plugin benötigen einen **-p** Parameter um die Anzahl der zu sendenden Pakete zu definieren. Ältere Versionen des Plugin benötigen diesen Parameter nicht - und hier liegt das Problem. Es ist also die Definition des Hostüberprüfungsbefehl (host check command) auf korrekte Syntax zu überprüfen. Beispiel:

command[check-host-alive]=/usr/local/netsaint/libexec/check_ping $HOSTADDRESS$ 100 100 1000.0 1000.0 -p 1

Wichtig! Nur die Tatsache, daß es einen Dienst gibt, der die ping-Statistik für einen Host überwacht bedeutet *nicht* , daß der aktuelle Hoststatus überprüft wird. Der Status eines Hosts wird nur dann überprüft, wenn eine Dienstüberprüfung (service check) einen Nicht-OK-Zustand zurückgibt oder der Host zuvor unten (down) war und die Dienstüberprüfung einen OK-Zustand zurückgibt.

Einige Symptome eines unrichtigen Hostüberprüfungsbefehls (host check commands) sind:

1. Hosts werden fälschlicherweise als DOWN aufgelistet
2. Dienste die den Status "HOST DOWN" haben, obwohl der Host auf dem sie arbeiten UP ist
3. Wechselnde Alarme/Benachrichtigungen über Hostprobleme und Wiederherstellungen (recoveries)

When hosts go down, I get notification about services instead of hosts and the service notifications contain incorrect data

Several people have reported this problem and I spent hours trying to find the problem until I realized it wasn't a bug in the code. If you get service notifications when you should be getting host notifications (and the service notifications you get seem to contain bogus data), check your contact definitions in the host config file. They are most likely incorrect.

Make sure that you are not using the same notification command for service and host notification commands. Service and host notifications are very different and make use of macros which are not transferrable between each type. Look at the sample host config file provided with NetSaint to see what the contact definitions look like and how the service and host notification commands differ. If you're wondering what macros can be used in either type of notification, look at this table.

Can I monitor a host without defining any services for it?

No. You must define at least one service for each host you want to monitor. NetSaint is primarily geared towards monitoring services - hosts are really only checked when there are problems or recoveries with services.

How can I change the timeout values for service checks?

First you need to identify where the timeout is occurring. Most plugins time out after 10 seconds of not being able to contact a service (FTP, HTTP, etc). If the plugins are timing out after a short period of time, increase the timeout value for the plugin (use an appropriate command line argument).

In addition to plugins having timeouts, NetSaint enforces its own timeout value on all service checks that run. By default, this is set to 30 seconds. If the plugin executes for more than 30 seconds, NetSaint will automatically kill it off and return a critical error for that service. If you see entries in the log file that say a service check timed out, this may be your problem. You can adjust the maximum timeout value for service checks by using the service_check_timeout directive.

As a side note, there are also directives for setting the maximum timeout for host checks, notifications, event handlers, and the ocsp command .

"Return code of *x* is out of bounds" errors

If the plugin output for a host or service check give a "(Return code of *x* is out of bounds)" error it usually means one of two things:

1. The plugin you're using to perform the host or service check is not returning the proper return code when it exits (as described in the plugin developer guidelines)

2. The path to the plugin is invalid (i.e. the binary or script does not exist). This is most likely the case if you get errors about a return code of **127** being out of bounds. If this is the error you're getting, check your command definitions and make sure the path to all executables is correct (and that they're actually installed on your system).

Debugging "unknown variable" errors during configuration file verification or runtime

When trying to run NetSaint or verify your configuration file data using the **-v** argument, NetSaint may print out a message like "Error in configuration file 'xxxxxxx.cfg' - Line 34 (Unknown variable)". A few simple checks will usually resolve this problem...

1. Make sure you are passing the path to the main configuration file and *not* the host configuration file on the command line. Many people have made this mistake. The correct syntax would be as follows (modified for your system, of course):

   ./netsaint -v /usr/local/netsaint/etc/netsaint.cfg

2. Make sure that you don't have any invalid variables defined in your configuration file. Notice that the error message will contain a reference to the name of the configuration file and the line number on which the error was encountered. Make sure that all comment lines contain a

pound sign (#) in the first character of the line. If you're not sure about what variables are valid, check the documentation for the main and host configuration files.

3. Make sure all variable identifiers are in lower case. Example:

   "admin_email=someaddress@somedomain.com"instead of
   "ADMIN_EMAIL=somedomain@nowhere.com"

How do I run multiple instances on NetSaint on the same machine?

You can run multiple instances of NetSaint on the same machine, if you ensure that the following variables are unique for each instance of NetSaint...

- Main configuration file
- Temp file
- Status file
- External command file
- Log file
- Log archive path
- Lock file

If you are using the web interface, you will have to setup separate directories to hold the CGIs for each instance of NetSaint and create appropriate script aliases in your web server configuration file. This is necessary because CGI configuration file must be unique for each setup of CGIs, as it contains a reference to which main configuration file the CGIs should read.

One last thing you should check is your init script (if you're using one). The init script should start, stop, restart, and reload all copies of NetSaint (if that's what you want it to do).

When I access the CGIs I don't see everything I should or I get authorzation errors...

If you believe you are unable to see all the information in the CGIs or if you are getting authorization errors, you probably haven't configured the web server to require authentication or haven't setup authorzation correctly. See the documentation on authentication and authorization in the CGIs here.

Wo kann ich die traceroute und daemonchk CGIs finden?

Die *traceroute* und *daemonchk* CGIs sind nun im contrib/ Unterverzeichnis der NetSaint Installation enthalten.

How do I requre users to authenticate before accessing the web interface?

See the documentation on authentication and authorization in the CGIs here.

How do I get those pretty pretty host icons to display in my CGIs?

If you want to associate images with particular hosts for use in the status, status map, status world, and extended information CGIs, you must define extended host information entries in your CGI configuration file.

I'm getting errors when attempting to commit commands to NetSaint via the command CGI

If you are getting '**Could not open command file *somefile* for update**' errors when attempting to commit commands to NetSaint via the command CGI, the most likely problem is with directory and/or file permissions. Here is what you can do to fix it...

1. Make sure you've created the directory to hold the command file as outlined here.

2. Make sure you restart your web server so that it inherits the new group permissions you just assigned

NetSaint shuts down with warnings about permissions on the command file

If NetSaint is shutting itself down after it processes external commands and you get warnings in the log file about incorrect permissions on the command file, make sure to read the directions found here .

How do I monitor remote host information?

Several people have asked how to use various plugins that check information on the local host to report information from remote hosts. Various methods for doing this are described below..

If you need to actually execute a plugin on a remote host and get the results back, you can use one of the following methods...

- Use the check_by_ssh "plugin" to execute a plugin on a remote host. The **check_by_ssh** plugin is basically a wrapper for executing a plugin on a remote host using SSH. You must have SSH installed and configured properly in order to use this.
- Use the nrpe addon to accomplish this. The plugins and the nrpe daemon reside on the remote host. The check_nrpe plugin (included with the nrpe package) sends a request to the nrpe daemon to execute the plugin on the remote host and then grabs the results for NetSaint.
- Use the nrpep addon. This addon works in a similiar manner to the nrpe package, but it encrypts the transmitted data, runs as a service from inetd, and makes use of the TCP Wrappers package for access control.
- Use **rsh** to execute the plugin remotely, although I guess I wouldn't recommend this..

If all you need is to check disk space, etc. on a remote host, you can use one of the methods below...

- Use one of the plugins included with the netsaint_statd addon for NetSaint. The addon, written by Charlie Cook, includes a Perl daemon which runs on the remote host and four plugins which are used to gather the remote host information from the daemon. The daemon is designed to run on Linux, IRIX, HP-UX, SunOS, and OSF/1 systems. Modifying the code

for other systems should be fairly easy. More on the **netsaint_statd** plugin can be found here.
- Use the check_overcr plugin to query information from a remote host. The remote host must be running Eric Molitor's Over-CR collector in order for this to work.
- Use the check_snmp plugin to check the value of various OIDs on the remote host. You must have SNMP services installed and running on the remote host in order to do this.

Wie kann ich Windows NT Server überwachen?

Ja, NT Server können mit NetSaint überwacht werden. Es gibt derzeit drei grundsätzliche Ansätze dies zu tun...

- Mittels SNMP
- Mittels dem NTSTAT addon (service und plugin)
- Mittels dem NSSERVICER addon (service und plugins)

SNMP

Die gute Nachricht ist, daß NT reichlich Performancedaten zu Überwachung anbietet. Die schlechte ist, daß dies nicht einfach ist. Das beste ist es wohl, SNMP-Dienste auf allen NT-Kisten zu installieren. Ian Cass hat eine FAQ geschrieben was zu tun ist unter http://elton.dev.knowledge.com/snmpfaq.html

Um NT Leistungsdaten anzuzeigen, muß der SNMP-Dienst auf allen zu überwachenden Servern laufen. Auch müssen alle erforderlichen Leistungs-MIBs für die zu überwachenden Dienste installiert werden. Ich glaube, dass die im NT Resource Kit oder in Server Admin Packages zu finden sind. Zur Not kann auch die Microsoft Seite nach den Ausdrücken **SNMP** und **MIB** durchsucht werden, vielleicht findet sich da was...

You can search the MRTG mailing list archives for more information on configuring NT servers to expose various performance counters via SNMP. I know this has been discussed in the past, as many people are graphing various NT performance statistics using MRTG. In fact, somebody from Microsoft is actually doing it - you can find their web page at http://snmpboy.rte.microsoft.com/ .

Once you've actually got the SNMP stuff working, you can use the check_snmp plugin to query your NT servers and generate alarms.

NTSTAT Addon

A while back I wrote an NT service (ntstat) and corresponding plugin (check_ntstat) that can be used to monitor basic information about NT servers. The plugin and service are capable of monitoring CPU utilization (overall or individually on up to four processors), physical memory usage, paged memory usage, and disk usage. The service has been reported to work on NT 4 servers, as well as Windows 2000 servers.

The ntstat service and check_ntstat plugin can both be found at http://www.netsaint.org/download/alpha/ . Don't let the directory name scare you. Several people (myself included) have been using the service and plugin for some time without any problems.

NSSERVICER Addon

Jan Christian Kaldestad and Hallstein Lohne have written the nsservicer addon for monitoring NT servers. The addon is similiar to the ntstat package and includes a service that runs on your NT servers and several plugins that run from the NetSaint host. The nsservicer addon is capable of monitoring the event log, disk usage, process usage, and other info.

You can find the addon at http://www.netsaint.org/download/contrib/addons/.

I plan on merging the functionality of the ntstat and nsservicer addons into one package late this summer. When an updated package is ready for testing, it will be announced on the netsaint-announce mailing list.

How can I monitor Novell Netware servers?

You can monitor basic stats on your Novell server like disk usage, user connections, LRU sitting time, cache buffers, long term cache hits, and processor load by using the *check_nwstat* plugin (which is included in the main plugin distribution). In order for the plugin to work, you have to install and run James Drew's MRTGEXT NLM on your Novell server. The NLM can be obtained here.

Can NetSaint send SNMP traps to management hosts?

Yes, but not directly. NetSaint relies on plugins to handle the gathering of service and host information and event handler scripts to handle events that occur with services and hosts. If you want to have NetSaint send an SNMP trap to a management host in the event that a particular service has a problem, you will have to write a service event handler script and add it to the **event_handler** option of the service definition. If you have the UCD-SNMP package installed on your host, you could have the script call the **snmptrap** command to actually send a trap message, depending on what type of service event occurred. Look at the example event handler script to get a better idea of how to write a script.

Can NetSaint log host and service events to an external database?

Not directly, but this can be done fairly easily. You'll probably want to define global host and service event handlers to do this. The global event handlers could call a script which inserts the appropriate event information into a database of your choosing. This would allow you to run queries and generate more detailed reports than what are available in the CGIs.

Something isn't working properly - How can I track down the problem?

I've worked in tech support for a few years and have spent my share of time on a helpdesk. Most people are vague when they report a problem and have no desire whatsoever to try and track down the problem - they just want you to fix it **now**. I hope you are not that type of person. NetSaint is relatively new and is probably chock full of bugs, so things will not always work properly. If you suspect that either the service check or notification routines are not working, here are a few things you can do to try and track down the problem...

This first thing you should do is verify your configuration data by running NetSaint with the **-v** option. Example:

/usr/local/netsaint/bin/netsaint -v /usr/local/netsaint/etc/netsaint.cfg

If no errors are found, proceed to the next steps. If NetSaint reports some error, go back and fix your configuration files.

The next step will take more time, but will give you more information on what is going on inside of NetSaint. When I first developed NetSaint I added a lot of debugging code to help me track down problems. I still use that code when I add new features or track down bugs myself. Here is how to use the debugging code...

Reconfigure NetSaint and enable one or more debug options as follows, replacing the "--enable-DEBUGx" with one or more of the values from the table below:

./configure --prefix=/your/netsaint/directory --enable-DEBUGx

Debugging Options

Debug Option	Description
--enable-DEBUG0	Used to trace function calls. A **lot** of messages will be printed out if you uncomment this option, but it very useful to trace what functions are being called. Note that not all functions will print an exit message if code within the function causes an early exit (before reaching the end of the function).
--enable-DEBUG1	Used to print out informational messages about variable settings. Most useful when trying to debug the configuration data as it is being read or verified.
--enable-DEBUG2	Used to print out warning messages, usually when configuration data is being read or verified.
--enable-DEBUG3	Used to print out informational messages during host and service checks. Good to use if you suspect problems are occuring during service checks.

--enable-DEBUG4	Used to print out informational messages during host and service notifications. Good to use if you suspect problems are occurring during the notification events.

Recompile NetSaint.

Verify your configuration data again - you'll see a lot more information this time if you have enabled the **DEBUG1** option. Try redirecting output to a file so that you can view or print it at a later time.

If you have defined either the **DEBUG3** or **DEBUG4** options, run NetSaint as a foreground process and start monitoring your services. Example:

/usr/local/netsaint/bin/netsaint /usr/local/netsaint/etc/netsaint.cfg

Kill NetSaint at an approprate point (i.e. after a service check fails) and look through the output. It should help you track down where the problem is occurring. You may want to redirect the output to a file to make it easier to review it. Some code tweaking may be necessary on your part in order to fix things. Let me know if you have to make any such alterations so I can include the fix in future releases.

If you are unable to determine or fix the problem on your own, email me the following items (give me some warning if you're planning on sending a large attachment):

1. The version of NetSaint you are running
2. A description of what is going wrong and what you suspect is the problem
3. The OS/distribution/version/architecture you're running NetSaint on (i.e. RedHat Linux 6.1 for Intel)
4. Your configuration files (**netsaint.cfg** and **hosts.cfg**)
5. Output from the program run (with debugging options on)

NetSaint Plugins

Wo gibt es Plugins

Bemerkung: Die Plugin-Entwicklung für NetSaint wurde zu SourceForge verlagert. Die NetSaint Plugin-Entwicklungsseite (wo immer die neuesten Versionen der Plugins zu finden sind) ist hier zuhause: http://netsaintplug.sourceforge.net/.

Zusätzlich zu den Plugins die vom SourceForge Projekt verteilt werden sind noch andere, von NetSaint Benutzern beigetragene Plugins im Contribdownloadbereich der NetSaint Seite erhältlich: http://www.netsaint.org/download/contrib/plugins/

Befehlsdefinitionsbeispiel für Dienste

Wer nach Beispielen sucht, wie man Befehle für Host- oder Dienstüberprüfungen definiert, kann in der alten Plugin-Dokumentation hier nachschauen. Es ist zu beachten, dass diese Dokumentation ziemlich alt ist und eventuell Fehler enthält, da sich die Plugins manchmal geändert haben.

NetSaint Plugins

Note: This document is quite outdated. Plugin development for NetSaint has been moved off to SourceForge. The NetSaint plugin development project page can be found here.

The following is a basic description of the some of the plugins that are available for use with NetSaint. If you have further questions, try running the plugins manually using the "manual execution" examples I've provided.

Other Resources

If you are confused about what the macros are all about, read up on them here. If you have questions about configuring services to actually make use these plugin examples, read up on the configuration documentation here. If you read the documentation and still can't figure things out, feel free to email me and I'll give you a hand.

Documented Plugins

TCP port plugin (check_tcp)
UDP port plugin (check_udp)
SMTP plugin (check_smtp)
POP3 plugin (check_pop)
FTP plugin (check_ftp)
NNTP plugin (check_nntp)
HTTP plugin (check_http)
Time plugin (check_time)
Ping plugin (check_ping)
DNS plugin (check_dns)
SSH plugin (check_ssh)
SNMP plugin (check_snmp)
Disk space plugin (check_disk)
Current users plugin (check_users)
Process plugin (check_procs)
Processor load plugin (check_load)
HP printer plugin (check_hpjd)
MRTG traffic plugin (check_mrtgtraf)
MRTG generic plugin (check_mrtg)
Novell server statistics plugin (check_nwstat)
Over-CR collector plugin (check_overcr)

Other Undocumented Plugins

These are some other plugins which are included in the core plugin distribution, but are not documented because of a lack of time on my part. Help for each plugin is usually available by running the plugin with no command line arguments.

- **Dummy plugin** (check_dummy)
- **FPing plugin** (check_fping)
- **Game server** plugin (check_game)
- **IMAP plugin** (check_imap)
- **LDAP plugin** (check_ldap)
- **MySQL plugin** (check_mysql)
- **REAL server plugin** (check_real)
- **Reply plugin** (check_reply)
- **Breezecom wireless signal strength plugin** (check_breeze.pl)
- **WaveLAN wireless signal strength plugin** (check_wave.pl)
- **FlexLM license manager plugin** (check_flexlm.pl)
- **IRCD server plugin** (check_ircd.pl)
- **NFS plugin** (check_nfs.pl)
- **NTP plugin** (check_ntp.pl)
- **SMB share disk space plugin**

Process image size plugin (check_vsz) (check_disk_smb.pl)
Swap usage plugin (check_swap)
Oracle database server plugin (check_oracle)
PostgresQL database plugin (check_pgsql)
Log file pattern detector plugin (check_log)
UPS plugin (check_ups)
SSH plugin executor (check_by_ssh)
NetSaint process plugin (check_netsaint)

TCP Port Plugin (check_tcp)

Command Line Format:	**check_tcp <host_address> [-p port] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_tcp 192.168.0.2 -p 23**
Command Definition Example:	**command[check_tcp]=/usr/local/netsaint/libexec/check_tcp $HOSTADDRESS$ -p $ARG1$**

This plugin is fairly simple - it just checks to see if it can connect to the specified host on the specified port number. A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

UDP Port Plugin (check_udp)

Command Line Format:	**check_udp <host_address> [-p port] [-s send] [-e expect] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_udp 192.168.0.2 -p 20796 -s "Hello, Mr. Server" -e "Hi there, Mr. Client"**
Command Definition Example:	**command[check_udp]=/usr/local/netsaint/libexec/check_udp $HOSTADDRESS$ -p $ARG1$ -s $ARG2$ -e $ARG3$**

This plugin will attempt to connect to the specified UDP port on the given host. The plugin will send the string specified by the *send* argument upon making a connection. The plugin will expect to recieve a response from the server, which should include the substring specified by the *expect* argument. If the plugin does not receive a response that contains the substring specified by the *expect* argument, it will

return a critical status. A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

SMTP Plugin (check_smtp)

Command Line Format:	**check_smtp <host_address> [-p port] [-e expect] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_smtp 192.168.0.2**
Command Definition Example:	**command[check_smtp]=/usr/local/netsaint/libexec/check_smtp $HOSTADDRESS$**

This plugin will check to see if it can connect to the SMTP port on the specified host. The plugin will look for the string specified by the *expect* argument in the first line of the response from the host (default is "220"). Specifying an optional port number on the command line will override the default port (25). A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

POP3 Plugin (check_pop)

Command Line Format:	**check_pop <host_address> [-p port] [-e expect] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_pop 192.168.0.2**
Command Definition Example:	**command[check_pop]=/usr/local/netsaint/libexec/check_pop $HOSTADDRESS$**

This plugin will check to see if it can connect to the POP3 port on the specified host. The plugin will look for the string specified by the *expect* argument in the first line of the response from the host (default is "+OK"). Specifying an optional port number on the command line will override the default port (110). A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

FTP Plugin (check_ftp)

Command Line Format:	**check_ftp <host_address> [-p port] [-e expect] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_ftp 192.168.0.2**
Command Definition Example:	**command[check_ftp]=/usr/local/netsaint/libexec/check_ftp $HOSTADDRESS$**

This plugin will check to see if it can connect to the FTP port on the specified host. The plugin will look for the string specified by the *expect* argument in the first line of the response from the host (default is "220"). Specifying an optional port number on the command line will override the default port (21). A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

NNTP Plugin (check_nntp)

Command Line Format:	**check_nntp <host_address> [-p port] [-e expect] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_nntp 192.168.0.2**
Command Definition Example:	**command[check_nntp]=/usr/local/netsaint/libexec/check_nntp $HOSTADDRESS$**

This plugin will check to see if it can connect to the NNTP port on the specified host. The plugin will look for the string specified by the *expect* argument in the first line of the response from the host (default is "220"). Specifying an optional port number on the command line will override the default port (119). A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option.

HTTP Plugin (check_http)

Command Line Format:	**check_http <host_address> [-e expect] [-u url] [-p port] [-hn host_name] [-wt warn_time] [-ct crit_time] [-to to_sec] [-nohtml]**
Manual Execution Example:	**check_http 192.168.0.1**
Command Definition Example:	**command[check_http]=/usr/local/netsaint/libexec/check_http $HOSTADDRESS$**

This plugin will check to see if it can connect to the HTTP port on the specified host and retrieve the specified URL. If no URL is specified on the command line, the plugin will fetch the root document. The plugin looks for a "HTTP/1." message from the host or whatever you specify for the *expect* argument. Specifying an optional port number on the command line will override the default port (80). Specifying the optional *host_name* argument will cause the plugin to send a "Host: [host_name]" header string to the HTTP server immediately after the GET request. This is useful when trying to monitor virtual servers that use host headers. A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds). By default, this plugin produces HTML output which provides a link to host/url that you specify on the command line. If you want to suppress the HTML output, use the *nohtml* argument.

Time Plugin (check_time)

Command Line Format:	**check_time <host_address> [-p port] [-wd warn_diff] [-cd crit_diff] [-wt warn_time] [-ct crit_time] [-to to_sec]**
Manual Execution Example:	**check_ttime 192.168.0.2 -wd 300 -cd 600**
Command Definition Example:	**command[check_time]=/usr/local/netsaint/libexec/check_tcp $HOSTADDRESS$ -wd 300 -cd 600**

This plugin will attempt to check the time on a remote host. Specifying an optional port number on the command line will override the default (37). The plugin will return a critical status if the time difference in seconds between the remote and local hosts exceeds the value of the *crit_diff* argument (if the -cd option is supplied). It will return a warning status if the time difference exceeds the value of the *warn_diff* argument (assuming the -wd option is supplied). A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

Ping Plugin (check_ping)

Command Line Format:	**check_ping <host_address> <wpl> <cpl> <wrta> <crta> [-p packets] [-nohtml]**
Manual Execution Example:	**check_ping 192.168.0.1 40 100 100.0 1000.0**
Command Definition Example:	**command[check_ping]=/usr/local/netsaint/libexec/check_ping $HOSTADDRESS$ $ARG1$ $ARG2$ $ARG3$ $ARG4$**

This plugin will check to see if it can ping the specified host. It will also check the packet loss and round trip average and compare that against the warning and critical threshold levels specified for each. The <wpl> and <cpl> arguments are the warning and critical thresholds for percent packet loss, respectively. Likewise, the <wrta> and <crta> arguments are the warning and critical thresholds for round trip average (in milliseconds). The optional *packets* argument allows you to control how many ICMP ECHO packets are sent to the specified host (default is 5). By default, this plugin produces HTML output which provides a link to a traceroute CGI, which allows you to run a traceroute to the specified host via the web interface. If you want to suppress the HTML output, use the *nohtml* argument.

DNS Plugin (check_dns)

Command Line Format:	**check_dns <host_query> [dns_server]**
Manual Execution Example:	**check_dns www.onepermanentdomain.com 192.168.0.1**
Command Definition Example:	**command[check_dns]=/usr/local/netsaint/libexec/check_dns www.onepermanentdomain.com $HOSTADDRESS$**

This plugin will check to see if it can resolve the host or domain name specified by the <host_query> option. If you don't want to use the default DNS servers specified in /etc/resolv.conf you can specify a different one by supplying it as the second argument. The most useful purpose of this plugin is to check the status of one of your DNS servers. If you want to monitor one of your DNS servers, supply a well known host/domain name as the first argument and the address of your DNS server as the second argument. The "well known host/domain name" should be something that is widely know and is should always resolve to a valid IP address - try picking the name of a big search engine or corporate website.

Notes:

- This plugin uses the **nslookup** command to do the actual DNS lookup.

SSH Plugin (check_ssh)

Command Line Format:	**check_ssh <host_address> [port]**
Manual Execution Example:	**check_ssh 192.168.0.1**
Command Definition Example:	**command[check_ssh]=/usr/local/netsaint/libexec/check_ssh $HOSTADDRESS$ $ARG1$**

This plugin will attempt to connect to the SSH server on the port number specified by the *port* argument (default is port 22). Upon successfully contacting the SSH server and receiving a valid response, the plugin will display the protocol and server version information. If the plugin receives and invalid response it will display the message returned from the server and generate a warning state.

Notes:

- This plugin was contributed by Remi Paulmier.

SNMP Plugin (check_snmp)

Command Line Format:	**check_snmp <host_address> [-c community] [-o object_id] [-e eval_method] [-wv warn_value] [-cv crit_value] [-l label] [-r rate]**
Manual Execution Examples:	**check_snmp 192.168.0.1 -o ip.ipOutNoRoutes.0 -e GT -wv 5000 -cv 10000 -l "IP Packets Out W/ No Route" -r "total packets"**
Command Definition Example:	**command[check_ssnmp]=/usr/local/netsaint/libexec/check_snmp $HOSTADDRESS$ -c $ARG1$ -o $ARG2$ -e $ARG3$ -wv $ARG4$ -cv $ARG5$ -l $ARG6$ -r $ARG7$**

This plugin will attempt to obtain the value of the objectID (specified by the *object_id* argument) from the SNMP source. The value can then be evaluated with a variety of methods (listed in the table below) against optionally specified warning and critical thresholds.

Evalution Methods

Method	Description
PR	The plugin will check only see see if some sort of value was present in the server response

GT	The plugin will return a non-OK state if the data receieved is greater than either the warning or the critical threshold values
LT	The plugin will return a non-OK state if the data receieved is less than either the warning or the critical threshold values
GTE	The plugin will return a non-OK state if the data receieved is greater than or equal to either the warning or the critical threshold values
LTE	The plugin will return a non-OK state if the data receieved is less than or equal to either the warning or the critical threshold values
EQ	The plugin will return a non-OK state if the data receieved is equal to either the warning or the critical threshold values
NE	The plugin will return a non-OK state if the data receieved is not equal to either the warning or the critical threshold values

Notes:

- This plugin used the **snmpget** command distributed in the UCD-SNMP package. If you don't have it installed on your system you will need to get it from here before you can use the plugin.
- For all evalutations methods other than **PR**, it is expected that the data being evaluated is an unsigned integer.

Disk Space Plugin (check_disk)

Command Line Format:	**check_disk <wusp> <cusp> <file_system>**
Manual Execution Example:	**check_disk 85 95 /dev/hda1**
Command Definition Example:	**command[check_disk]=/usr/local/netsaint/libexec/check_disk 85 95 $ARG1$**

This plugin will check the free disk space on a specific file system. If used disk space percentage exceeds the *<cusp>* threshold value, a critical state results. If the used disk space percentage exceeds the *<wusp>* threshold value, a warning state results. The *<file_system>* argument should be in the form of */dev/hda1*, */dev/hdb2*, etc.

Notes:

- This plugin uses the **df** command to do the actual disk space check.
- This plugin is UNIX-specific, in that you cannot check the free disk space on NT or Novell servers, etc.
- This plugin can only check disk space on the host that is doing the monitoring. You can use **rsh**, **ssh**, or similiar methods to check space on remote hosts, but the netsaint_statd remote perl daemon will do the job just as well and make your life easier.

Current Users Plugin (check_users)

Command Line Format:	**check_users <wusers> <cusers>**
Manual Execution Example:	**check_users 50 75**
Command Definition Example:	**command[check_users]=/usr/local/netsaint/libexec/check_users $ARG1$ $ARG2$**

This plugin will check the number of currently logged in users. If the number of logged in users exceeds the <*cusers*> threshold value, a critical state results. If it exceeds the <*wusers*> threshold value, a warning state results.

Notes:

- This plugin uses the **who** command to do the actual check of logged in users.
- This plugin is UNIX-specific, in that you cannot check the number of logged in users on NT or Novell servers, etc.
- This plugin can only check users on the host that is doing the monitoring. You can use **rsh**, **ssh**, or similiar methods to check users on remote hosts, but the netsaint_statd remote perl daemon will do the job just as well and make your life easier.

Process Plugin (check_procs)

Command Line Format:	**check_procs <wprocs> <cprocs> [process_flags]**
Manual Execution Example:	**check_procs 5 10 ZT**
Command Definition Example:	**command[check_procs]=/usr/local/netsaint/libexec/check_procs $ARG1$ $ARG2$ $ARG3$**

This plugin will check the number of processes on the current machine. Optional process flags include R (Running), S (Sleeping), Z (Zombie), T (Stopped or Traced), and D (Uninterruptible Sleep). If no process flags are specified, the plugin will count all types of processes. The example above will check for processed that are in either a zombie state or are stopped/traced.

Notes:

- This plugin uses the **ps** command to do the actual check of processes.
- This plugin is UNIX-specific, in that you cannot check processes on NT or Novell servers, etc.
- This plugin can only check processes on the host that is doing the monitoring. You can use **rsh**, **ssh**, or similiar methods to check processes on remote hosts, but the netsaint_statd remote perl daemon will do the job just as well and make your life easier.

Processor Load Plugin (check_load)

Command Line Format:	**check_load <wload1> <cload1> <wload5> <cload5> <wload15> <cload15>**
Manual Execution Example:	**check_load 95 100 90 95 80 90**
Command Definition Example:	**command[check_load]=/usr/local/netsaint/libexec/check_load $ARG1$ $ARG2$ $ARG3$ $ARG4$ $ARG5$ $ARG6$**

This plugin will check the load average on the local machine over 1, 5, and 15 minute time periods using the data found in the **/proc/loadavg** file. A critical status is returned if the 1, 5, or 15 minute load averages exceed the *<cload1>*, *<cload5>*, or *<cload15>* thresholds specified. A warning status is returned if the 1, 5, or 15 minute load averages exceed the *<wload1>*, *<wload5>*, or *<wload15>* thresholds specified.

Notes:

- This plugin was contributed by Felipe Gustavo de Almeida.
- This plugin is UNIX-specific, in that you cannot check processor loads on NT or Novell servers, etc.
- This plugin can only check processor load on the host that is doing the monitoring. You can use **rsh**, **ssh**, or similiar methods to check processor load on remote hosts, but the netsaint_statd remote perl daemon will do the job just as well and make your life easier.

HP Printer Plugin (check_hpjd)

Command Line Format:	**check_hpjd <address> [community]**
Manual Execution Example:	**check_hpjd 192.168.0.1**
Command Definition Example:	**command[check_hpdj]=/usr/local/netsaint/libexec/check_hpdj $HOSTADDRESS$ $ARG1$**

This plugin will check the status of an HP printer that has a JetDirect© card installed. This plugin will return a critical status when the printer is turned off, a warning status when it has a paper jam, is offline, out of paper, low on toner, etc. Specifying an optional [community] argument on the command line will override the default SNMP community used in the communication with the printer (public).

My guess is that you don't want to get alert emails or pages everytime a printer jams or gets turned off at the end of the day, right? If so, read the FAQ on monitoring printers.

Notes:

- This plugin works *only* on HP brand printers with JetDirect© cards installed, and it may not work on all of them.
- This plugin used the **snmpget** command distributed in the UCD-SNMP package. If you don't have it installed on your system you will need to get it from here before you can use the plugin.
- The idea (and some code) for this plugin came from Jim Trocki's printer alert script in his *mon* program.
- The JetDirect© card must have the TCP/IP protocol stack enabled and configured. External JetDirect© devices must have TCP/IP enabled on each port they want to monitor.
- *JetDirect* is copyrighted by Hewlett-Packard. Don't sue me please. :-)

MRTG Traffic Plugin (check_mrtgtraf)

Command Line Format:	**check_mrtgtraf <log_file> <expire_minutes> <AVG|MAX> <iwl> <icl> <owl> <ocl>**
Manual Execution Example:	**check_mrtgtraf /home/httpd/html/mrtg/router1.log 10 AVG 1000000 1500000 1000000 1500000**
Command Definition Example:	**command[check_mrtgtraf]=/usr/local/netsaint/libexec/check_mrtgtraf $ARG1$ 10 AVG $ARG2$ $ARG3$ $ARG4$ $ARG5$**

This plugin will check a traffic log file generated by MRTG and generate alerts if the incoming or outgoing rates (in Bytes/sec) exceed the specified thresholds. If the newest entry in the log file is more than <*expire_minutes*> old, the plugin will return a warning level. Specifying *AVG* or *MAX* as the third argument will control whether the plugin looks at average or maximum values in the log file (default is average). If the incoming or outgoing rates exceed the <*iwl*> or <*owl*> warning thresholds, respectively, a warning status is returned. If the rates exceed the <*icl*> or <*ocl*> critical thresholds, a critical status is returned. Command line thresholds are in Bytes/sec.

Notes:

- This plugin requires MRTG to do the actual traffic rate monitoring. You can download MRTG from http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html.
- The traffic rates reported by the plugin are slightly different from those reported by MRTG - I'll look into this.

MRTG Generic Plugin (check_mrtg)

Command Line Format:	**check_mrtg <log_file> <expire_minutes> <AVG|MAX> <column> <vwl> <vcl> <label> [rate]**
Manual Execution Example:	**check_mrtg /home/httpd/html/mrtg/router1.log 10 AVG 1 1000000 1500000 In Bytes/Sec**
Command Definition Example:	**command[check_mrtg]=/usr/local/netsaint/libexec/check_mrtg $ARG1$ 10 AVG $ARG2$ $ARG3$ $ARG4$ $ARG5$ $ARG6$**

This plugin will check a log file generated by MRTG and generate alerts if the value of the specified variable exceeds the specified thresholds. If the newest entry in the log file is more than *<expire_minutes>* old, the plugin will return a warning level. Specifying *AVG* or *MAX* as the third command line argument will control whether the plugins looks at the average or maximum value of the variable in the log file (default is average). This plugin will only check one of the two possible variables recorded by MRTG. If you want to monitor the first variable, specify **1** as the *<column>* argument. If you want to monitor the second variable, specify **2** as the argument. If the value of the specified variable exceeds the *<vcl>* threshold, a critical status is returned. If it exceeds the *<vwl>* threshold, a warning status is returned. The *<label>* argument is used in the output to identify what type of data is being monitored. Examples include **Connections**, **"User Connections"**, **"Processor Utilization"**, **"Traffic In"**, etc. The optional [rate] argument is used to give the variable value som meaning. Examples include **%**, **Packets/Sec**, **Bytes/Sec**, **"Errors Per Second"**, etc.

Notes:

- This plugin requires MRTG to do the actual monitoring. You can download MRTG from http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html.

Process Image Size Plugin (check_vsz)

Command Line Format:	**check_vsz <wsize> <csize> [command_name]**
Manual Execution Example:	**check_vsz 100000 150000 betaprogram**

Command Definition Example:	**command[check_vsz]=/usr/local/netsaint/libexec/check_vsz $ARG1$ $ARG2$ $ARG3$**

This plugin will check for processes whose total image size (in bytes) exceeds the warning or critical thresholds given on the command line (<*wsize*> and <*csize*>, respectively). With no [command_name] specified, every command that shows up in the **ps** command is evaluated. Otherwise, only jobs with names matching the [command_name] argument are examined. This program is particularly useful if you have to run a piece of commercial software that has a potential memory leak and you want to watch its memory usage carefully.

Notes:

- This plugin was contributed by Karl DeBisschop.
- Used in conjuction with an approriate service event handler, you could use this plugin to automatically kill and restart a program which is consuming memory.

Swap Usage Plugin (check_swap)

Command Line Format:	**check_swap <wswap> <cswap>**
Manual Execution Example:	**check_swap 100000 120000**
Command Definition Example:	**command[check_swap]=/usr/local/netsaint/libexec/check_swap $ARG1$ $ARG2$**

This plugin will check all the swap partitions on the local machine and return a warning or critical status if the percent of swap usage is above the <*wswap*> or <*cswap*> thresholds.

Notes:

- This plugin was contributed by Karl DeBisschop.

Novell Server Statistics Plugin (check_nwstat)

Command Line Format:	**check_nwstat <host_address> [-p port] [-v variable] [-wv warn_value] [-cv crit_value] [-to to_sec]**

Manual Execution Example:	**check_nwstat 192.168.1.5 -v LOAD5 -wv 80 -cv 95**
Command Definition Example:	**command[check_nwstat]=/usr/local/netsaint/libexec/check_nwstat $HOSTADDRESS$ -v $ARG1$ -wv $ARG2$ -cv $ARG3$**

This plugin allows you to monitor disk usage, connections, cache buffers, and LRU sitting time on your Novell servers. The plugin obtains server information by talking to the MRTGEXT NLM (distributed with James Drews' MRTG extension - see below) on the Novell server. The default port used to communicate with the server NLM is 9999. If the value for a given variable is higher than the specified critical threshold (or possibly lower - see note below), a critical status is returned. If the value is highter than the specified warning threshold (or possibly lower - see note below), a warning status is returned. Only one variable can be checked at a time. Valid variables are listed below. A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

Variable	Description
LOAD1	1 minute load average
LOAD5	5 minute load average
LOAD15	15 minute load average
CONNS	Number of licensed connections
LTCH	Percentage of long term cache hits
CBUFF	Number of total cache buffers
CDBUFF	Number of dirty cache buffers
LRUM	LRU sitting time in minutes
VPF<volume>	Percent free space on volume <volume>
VKF<volume>	KB of free space on volume <volume>

Notes:

- Both the critical and warning thresholds are optional. If neither is specified, the plugin returns an ok status (except in the case of a communication or configuration error).
- The critical thresholds should be **lower** than the warning thresholds for volume free space, cache buffers, and LRU sitting time because a lower value for these variables is worse than a higher number!
- This plugin requires that **MRTGEXT.NLM** (distributed in James Drews' MRTG Extensions for Netware package) be running on the Novell servers you wish to monitor. You don't have to use MRTG to use this plugin - just run MRTGEXT.NLM on your servers.

Over-CR Collector Plugin (check_overcr)

Command Line Format:	**check_overcr <host_address> [-p port] [-v variable] [-wv warn_value] [-cv crit_value] [-to to_sec]**
Manual Execution Example:	**check_overcr 192.168.1.5 -v LOAD5 -wv 80 -cv 95**
Command Definition Example:	**command[check_overcr]=/usr/local/netsaint/libexec/check_overcr $HOSTADDRESS$ -v $ARG1$ -wv $ARG2$ -cv $ARG3$**

This plugin allows you to monitor active network connections, uptime, running processes, disk usage, and processor load on remote servers. The plugin obtains server information by talking to Over-CR collector that runs on the remote server (see note below). The default port used to communicate with the Over-CR collector is 2000. If the value for a given variable is higher than the specified critical threshold (or possibly lower - see note below), a critical status is returned. If the value is highter than the specified warning threshold (or possibly lower - see note below), a warning status is returned. Only one variable can be checked at a time. Valid variables are listed below. A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

Variable	Description
LOAD1	1 minute load average
LOAD5	5 minute load average
LOAD15	15 minute load average
DPU<filesys>	Percent *used* disk space on file system **filesys**
PROC<process>	Number of running processes with a name of **process**
NET<port>	Number of active TCP/IP connection on port **port**
UPTIME	System uptime in seconds

Notes:

- Both the critical and warning thresholds are optional. If neither is specified, the plugin returns an ok status (except in the case of a communication or configuration error).
- The critical thresholds should be **lower** than the warning thresholds for UPTIME, since a lower uptime value is worse than a higher one!
- This plugin requires that Eric Molitor's Over-CR collector be running on the servers you wish to monitor.

Oracle Database Server Plugin (check_oracle)

Command Line Format:	**check_oracle <host_address>**

Manual Execution Example:	**check_oracle 192.168.0.2**
Command Definition Example:	**command[check_oracle]=/usr/local/netsaint/libexec/check_oracle $HOSTADDRESS$**

This plugin will attempt to see if an Oracle database server on the specified host can be contacted. The plugin returns a critical status if the database server cannot be reached.

Notes:

- This plugin was contributed by Jorge Sanchez
- This plugin requires the **tnsping** program distributed with Oracle SQL*Net software.
- This plugin is a shell script and does not need to be compiled

PostgresQL Database Plugin (check_pgsql)

Command Line Format:	**check_pgsql [twarn] [tcrit] [host] [port] [db] [user] [password]**
Manual Execution Example:	**check_pgsql 120 3600 192.168.0.1 5432 mydatabase**
Command Definition Example:	**command[check_pgsql]=/usr/local/netsaint/libexec/check_pgsql 120 3600 $HOSTADDRESS$ $ARG1$ $ARG2$ $ARG3$ $ARG4$**

This plugin will attempt to connect to the specified postgresQL database on the host. Connection refusals and timeouts result in a critical status. If the connection time exceeds the *tcrit* value in seconds, a critical status results. If the connection time exceeds the *twarn* value in seconds, a warning status results. All other errors result in an unknown status.

All arguments to this plugin are optional. The defaults are equivalent to **"check_pgsql 120 3600 localhost 5432"** and assume that the user that runs the plugin can connect to the database without a password.

Security Tip: If you use the [username] and [password] arguments in a command or service definition, you should take steps to ensure that this information does not end up getting displayed in the HTML pages that NetSaint generates!

Notes:

- This plugin was contributed by Karl DeBisschop

- This plugin requires that the PostgresQL libraries be installed on your machine in order to be compiled
- This plugin requires that the backend for remote machines use TCP/IP (start postmaster with the -i option)

Log File Pattern Detector Plugin (check_log)

Command Line Format:	**check_log <log_file> <old_log_file> <pattern>**
Manual Execution Example:	**check_oracle /var/log/messages /usr/local/netsaint/var/check_log.old.loginfailure 'LOGIN FAILURE'**
Command Definition Example:	**command[check_log]=/usr/local/netsaint/libexec/check_log $ARG1$ $ARG2$**

This plugin will scan a log file (specified by the *log_file* option) for a specific pattern (specified by the *pattern* option). Successive calls to the plugin script will only report **new** pattern matches in the log file, since an copy of the log file from the previous run is saved to *old_log_file*. The plugin returns a critical status if the log file cannot be located. The first time the plugin is executed it will initialize the data it needs and return with an ok status. Successive executions will return an OK status if no pattern matches are detected in the changes to the original log file. If one or more pattern matches are found, the plugin will return a CRITICAL status and print a string in the following format: *(x) last_entry*, where *x* is the total number of matches found and *last_entry* is the last matching entry from the log file.

Notes:

- This plugin is a shell script and does not need to be compiled
- This plugin is very "expensive" as far as disk space is concerned, because it keeps an old copy of the original log file for each pattern that you want to check. I am aware of this and know this should be done better - I wrote this simply as an example. A long term solution is a tie-in with a log watcher like SWATCH.

UPS Plugin (check_ups)

Command Line Format:	**check_ups <host_address> [-p port] [-u ups] [-v variable] [-wv warn_value] [-cv crit_value] [-to to_sec]**
Manual Execution Example:	**check_ups 192.168.0.3 -u mybigups -v BATTPCT -wv 80 -cv 40**

Command Definition Example: **command[check_ups]=/usr/local/netsaint/libexec/check_ups $HOSTADDRESS$ -u $ARG1$ -v $ARG2$ -wv $ARG3$ -cv $ARG4$**

This plugin attempts to determine the status of an UPS (Uninterruptible Power Supply) on a remote host (or the local host) that is being monitored with Russel Kroll's "Smart UPS Tools" package. If the UPS is online or calibrating, the plugin will return an OK state. If the battery is on it will return a WARNING state. If the UPS is off or has a low battery the plugin will return a CRITICAL state. You may also specify a variable to check (such as temperature, utility voltage, battery load, etc.) as well as warning and critical thresholds for the value of that variable. If the remote host has multiple UPS that are being monitored, you will have to use the *ups* option to specify which UPS to check. A critical status is returned if the host cannot be contacted within *crit_time* seconds (if the -ct option is supplied) and a warning status is returned if the host cannot be contacted within *warn_time* seconds (if the -wt option is supplied). A critical status is returned if the plugin cannot contact the host within the timeout period specified by the *to_sec* option (default is 10 seconds).

Variable	Description
UTILITY	The *difference* (absolute value) between 120.0 VAC and the voltage that is being supplied to the UPS via the utility line
BATTPCT	Percent of battery charge remaining
LOADPCT	Percent load being put on the UPS by the electronic devices attached to it
TEMP	The temperature (in degrees Farenheit) of the UPS

Notes:

- If the UPS being monitored does not support one or more of the variables that the plugin monitors, those variables will not be checked.
- The critical thresholds should be **lower** than the warning thresholds for BATTPCT, since a lower battery reserve is worse than a higher one!
- This plugin requires that the UPSD daemon distributed with Russel Kroll's "Smart UPS Tools" be installed on the remote host. If you don't have the package installed on your system, you can download it from http://www.exploits.org/~rkroll/smartupstools.

SSH Plugin Executor (check_by_ssh)

Command Line Format: **check_by_ssh <user> <host> <command>**

Manual Execution Example: **check_by_ssh**

Command Definition Example:	**command[check_by_ssh]=/usr/local/netsaint/libexec/check_by_ssh $ARG1$ $ARG2$ $ARG3$**

This is not so much a plugin as it is a plugin wrapper. It basically allows you to execute plugins on a remote host by using SSH. The SSH username on the remote host is specified with the *user* argument, the address of the remote host is specified by the *host* argument, and the command that should be executed on the remote host is specified by the *command* argument.

Notes:

- This plugin was contributed by Karl DeBisschop
- You must have SSH installed and configured properly in order to use this plugin
- The plugins that you want to execute on the remote host must be installed on the remote host!

NetSaint Process Plugin (check_netsaint)

Command Line Format:	**check_netsaint <status_log> <expire_minutes> <process_string>**
Manual Execution Example:	**check_netsaint /usr/local/netsaint/var/status.log 5 "/usr/local/netsaint/bin/netsaint -d /usr/local/netsaint/etc/netsaint.cfg"**
Command Definition Example:	**command[check_netsaint]=/usr/local/netsaint/libexec/check_netsaint $ARG1$ $ARG2$ $ARG3$**

This plugin is used to check the status of the NetSaint process on the local host. The plugin will check the contents of the status log (specified by the *status_log* argument) and make sure that the most recent entry is no older than the number of minutes specified by the *expire_minutes* argument. If the status log is older than this value, a warning state results. The plugin will also use the **ps** command to search for a running process that matches the *process_string* argument. If the plugin cannot locate a match of the process string, it assumes that NetSaint is not running and returns a critical state.

Notes:

- This plugin can be used by the CGIs to check the status of the NetSaint process. This is done by using the process_check_command option in the CGI configuration file.
- This plugin can be used when implementing redundant monitoring hosts, although it must be executed on the remote hosts as explained in this FAQ.

Die Konfiguration von NetSaint

Übersicht über die Konfiguration

Die Konfiguration von NetSaint erfolgt durch das Editieren von drei Dateien - die Hauptkonfigurationsdatei, die Hostkonfigurationsdatei, und die CGI-Konfigurationsdatei.

Hauptkonfigurationsdatei

Die Dokumentation zur Hauptkonfigurationsdatei ist hier zu finden. Eine Beispieldatei wird automatisch erzeugt beim Ausführen des **configure** Skripts vor dem Übersetzen des Quellcodes. Zu finden ist sie in entweder im Verzeichnis der Distribution oder im /etc Unterverzeichnis der Installation. Werden die Beispieldateien mit dem Befehl **make install-config** installiert, so wird eine Beispieldatei im Verzeichnis für Einstellungen (in der Regel /usr/local/netsaint/etc) zu finden sein. Der vorgegebene (default) Name der Hauptkonfigurationsdatei ist **netsaint.cfg**.

Hostkonfigurationsdatei

Die Dokumentation zur Hostkonfigurationsdatei ist hier zu finden. Eine Beispieldatei wird automatisch erzeugt beim Ausführen des **configure** Skripts vor dem Übersetzen des Quellcodes. Zu finden ist sie in entweder im Verzeichnis der Distribution oder im /etc Unterverzeichnis der Installation. Werden die Beispieldateien mit dem Befehl **make install-config** installiert, so wird eine Beispieldatei im Verzeichnis für Einstellungen (in der Regel /usr/local/netsaint/etc) zu finden sein. Der vorgegebene (default) Name der Hostkonfigurationsdatei ist **hosts.cfg**. In der "Host"konfigurationsdatei werden Hosts, Hostgruppen, Kontakte, Kontaktgruppen, Befehle, Zeitperioden und Dienste definiert..

CGI-Konfigurationsdatei

Die Dokumentation zur CGI-Konfigurationsdatei ist hier zu finden. Eine Beispieldatei wird automatisch erzeugt beim Ausführen des **configure** Skripts vor dem Übersetzen des Quellcodes. Zu finden ist sie in entweder im Verzeichnis der Distribution oder im /etc Unterverzeichnis der Installation. Werden die Beispieldateien mit dem Befehl **make install-config** installiert, so wird eine Beispieldatei im selben Verzeichnis wie die Haupt- und Hostkonfigurationsdatei (in der Regel /usr/local/netsaint/etc) zu finden sein. Der vorgegebene (default) Name der CGI-Konfigurationsdatei ist **nscgi.cfg**.

Wie geht's nun weiter

Nachdem NetSaint nach Belieben konfiguriert wurde, muß, bevor irgendetwas überwacht werden kann, diese Konfiguration überprüft werden.

Installation von NetSaint

Auspacken der Distribution

Um die NetSaint Distribution auszupacken, bitte folgende zwei Befehle in der Shell eingeben:

gunzip netsaint-0.0.6.tar.gz
tar xf netsaint-0.0.6.tar

Wenn die ZIP Version der Distribution heruntergeladen wurde bitte folgendes eingeben:

unzip netsaint-0.0.6.zip

Wenn diese Befehle ausgeführt wurden, sollte ein **netsaint-0.0.6** Verzeichnis im aktuellen Verezeichnis angelegt worden sein. In diesem Verzeichnis sind alle Dateien vorhanden, die die NetSaint Core Distribution enthält.

Übersetzen der Quellen

Erzeuge das Hauptverzeichnis, in das NetSaint wie folgt installiert werden soll...

mkdir /usr/local/netsaint

Ausführen des configure Skripts um Variablen zu initalisieren und das Makefile zu erzeugen...

./configure --prefix=*prefix* --with-cgiurl=*cgiurl* --with-htmurl=*htmurl* --with-netsaint-user=*someuser* --with-netsaint-grp=*somegroup*

- *prefix* wird ersetzt mit dem Verzeichnis, das im vorhergehenden Schritt erzeugt wurde (Vorgabe ist */usr/local/netsaint*)
- *cgiurl* wird ersetzt mit der URL, mit der auf die CGIs zugegriffen werden soll (Vorgabe ist */cgi-bin/netsaint*). Bitte KEINEN Schrägstrich ans Ende der URL anhängen.
- *htmurl* wird ersetzt mit der URL, mit der auf die HTML-Seiten der Weboberfläche und der Dokumentation zugegriffen werdensoll (Vorgabe ist */netsaint/*)
- *someuser* wird ersetzt mit dem Namen des Benutzers dessen Rechte die zu installierenden Dateien haben sollen (Vorgabe ist *netsaint*)
- *somegroup* wird ersetzt mit dem Namen der Gruppe deren Rechte die zu installierenden Dateien haben sollen(Vorgabe ist *netsaint*)

WICHTIG: Das *--prefix* Argument des Konfigurationskritps ist sehr wichtig, da es festlegt unter welchem Verzeichnis alles installiert wird. Wenn dieses Argument nicht angegeben wird, wird das Skript */usr/local/netsaint* als Zielverzeichnis benutzen. Es sollte sichergestellt werden, daß dieses

Verzeichnis bereits im System existiert bevor versucht wird alles zu installieren.

Das Kompilieren (Übersetzen) von NetSaint und den CGIs kann nun mit dem folgenden Befehl erfolgen:

make all

Installation der Binaries (Uebersetzte Programme) und der HTML-Dateien

Die Installation der Binaries und der HTML-Dateien (Dokumentation und Hauptseite) erfolgt dann mit dem folgenden Befehl:

make install

Erzeugung und Installation der Beispielkonfigurationsdateien

Beispielhafte Haupt, Host, Resource, und CGI (main, host, resource, und CGI) -konfigurationsdateien werden automatisch vom configure Skript im Hauptverzeichnis der Distribution angelegt.

Diese Beispieldateien können mit dem folgenden Befehl installiert werden:

make install-config

Installation eines Autostart-Skripts (Init Script)

Falls gewünscht, kann mit dem folgenden Befehl eine Init-Skript im Verzeichnis */etc/rc.d/init.d/netsaint* erstellt werden:

make install-init

...oder falls geplant ist NetSaint als Dämon auszuführen (was empfohlen wird), kann ein Init-Skript für einen Dämon im Verzeichnis */etc/rc.d/init.d/netsaint* mit folgendem Befehl erstellt werden:

make install-daemoninit

Verzeichnisstruktur

Mit dem folgenden Befehl ist nun in das Installationsverzeichnis von NetSaint zu wechseln...

cd /usr/local/netsaint

Es sollten fünf verschiedene Unterverzeichnis zu sehen sein. Eine kurze Beschreibung des Inhalts dieser Unterverzeichnisse gibt die untenstehende Tabelle.

Unterverzeichnis	Inhalt
bin/	NetSaint-Hauptprogram
etc/	Main und host -konfigurationsdateien (netsaint.cfg und hosts.cfg)
eventhandlers/	Enthält Beispielskripte, die in Ereignisbehandlungsroutinen (event handlers) eingesetzt werden können. Es sind auch Beispielskripte zur Implementierung redundanter Überwachung (redundant monitoring) vorhanden.
sbin/	CGI Programme und Konfigurationsdatei (nscgi.cfg)
share/	HTML-Dateien und Bilder für web interface and documentation
var/	Leeres Verzeichnis für Logdateien

Bemerkung:

1. Die vorgegebene Hostkonfigurationsdatei hosts.cfg, die vom configure Skript erzeugt wurde erwartet, daß sich alle Plugins im Unterverzeichnis **libexec/** der NetSaint befinden. Dieses Verzeichnis wird nicht vom Installationsskript angelegt, sondern wird erzeugt vom Installationsskript das bei den Plugins dabei ist. Plugins sind zu bekommen unter http://www.netsaint.org/download.

Was jetzt zu tun ist

Okay, das Übersetzen und Installieren von NetSaint ist nun erledigt. Jetzt sollte die Konfiguration von NetSaint durchgeführt werden, bevor es gestartet werden kann. Wahrscheinlich soll die Weboberfläche genutzt werden, deshalb sind auch die Anleitungen zur Installation der Weboberfläche und Konfiguration der Webauthentizierung, etc zu lesen.

Installation derWeboberfläche

Bemerkung

In diesen Anleitungen wird stets davon ausgegangen, daß als Webserver Apache eingesetzt wird. Wenn ein anderer Webserver eingesetzt wird, müssen diese Anleitungen entsprechend abgeändert werden.

Konfigurieren von Aliases für die HTML-Dateien und CGIs

Um auf die HTML-Dateien und die CGIs über das Web zugreifen zu können muß die Apache Konfiguration wie folgt abgeändert werden...

Die folgende Zeile in der Datei **httpd.conf** muß hinzugefügt werden (bitte anpassen auf die Verzeichnisstruktur der jeweiligen Installation)...

Alias /netsaint/ /usr/local/netsaint/share/

Dies ermöglicht es mit einer URL in der Form **http://yourmachine/netsaint/** auf die HTML-Weboberfläche und die Dokumentation zuzugreifen. Der Alias sollte denselben Wert haben der beim configure-Skript dem Argument **--with-htmurl** gegeben wurde (Vorgabe ist */netsaint/*).

Für die NetSaint CGIs muß ebenfalls ein Alias erzeugt werden. Die Default-Installation erwartet sie unter **http://yourmachine/cgi-bin/netsaint/**, diese URL kann aber auch mit der **--with-cgiurl** Option im configure Skript abgeändert werden. In jedem Fall ist ungefähr folgendes zur Konfigurationsdatei **httpd.conf** hinzuzufügen (auch hier wieder ist evtl eine Anpassung an das tatsächliche Verzeichnis notwendig)...

ScriptAlias /cgi-bin/netsaint/ /usr/local/netsaint/sbin/

Wichtig: Der ScriptAlias Eintrag für die NetSaint CGIs muß **vor** dem Standardeintrag '**ScriptAlias /cgi-bin/ /some...where../**' , der ja bereits in der Konfigurationsdatei vorhanden ist, stehen. Wenn das nicht der Fall ist, ist der Zugriff auf die CGIs sehr wahrscheinlich unmöglich.

Nachdem die Apache Konfigurationsdatei geändert wurde, ist es erforderlich Apache neu zu starten um die Änderungen wirksam werden zu lassen. Das kann normalerweise (unter Linux) geschehen mit einem Befehl wie diesem...

/etc/rc.d/init.d/httpd restart

Nachdem der Webserver neu gestartet wurde ist nur noch eine Kleinigkeit zu überprüfen. Und zwar

ist sicherzustellen, daß in der CGI Konfigurationsdatei (nscgi.cfg) im **etc/** Unterverzeichnis die Variable **main_config_file** auf den korrekten Pfad der Hauptkonfigurationsdatei zeigt. Die CGIs sollten dies wissen, um die Logdateien (status log, history log, etc.) zu finden.

Es sollte nicht vergessen werden zu prüfen, ob diese Änderungen im Apache wirksam sind. Es sollte möglich sein mit dem Browser die URL **http://yourmachine/netsaint** aufzurufen und die Weboberfläche von NetSaint zu sehen. Die CGIs können noch keine Informationen anzeigen, aber dies wird sich ändern, sobald die Konfigration des Webservers durchgeführt und NetSaint gestartet wurde.

Was jetzt zu tun ist

Nachdem die Weboberfläche korrekt konfiguriert wurde ist die Authentizierung des Webservers einzurichten um auf die CGIs zuzugreifen und die Konfiguration der Benutzerauthorisierung durchzuführen. Einzelheiten wie dies geschieht sind hier zu finden.

Überprüfen der NetSaint Konfiguration

Überprüfen der Konfiguration von der Befehlszeile

Nachdem alle notwendigen Daten in die Konfigurationsdateien eingegeben wurden, ist es Zeit einen Korrektheitscheck vorzunehmen. Jeder macht von Zeit zu Zeit Fehler, deshalb ist es am besten zu überprüfen, was eingegeben wurde. NetSaint führt selbst einen Sicherheitscheck durch, bevor die Überwachung tatsächlich beginnt. Aber es gibt die Möglichkeit diese Überprüfung manuell durchzuführen, bevor man versucht NetSaint zu starten. Um das zu tun, muß lediglich NetSaint mit der Option **-v** in der Befehlszeile gestartet werden, wie im folgenden dargestellt...

./netsaint -v <main_config_file>

Anzumerken ist, daß als zweites Argument der Name (mit Pfadangabe) der *Hauptkonfigurationsdatei* und *nicht* der Hostkonfigurationsdatei angegeben werden muß. NetSaint liest dann die Hauptkonfigurationsdatei und wird ausgehend von der Option **cfg_file** die Hostkonfigurationsdatei ermitteln und überprüfen.

Beim Check überprüfte Beziehungen

Während des "Pre-Flight"-Checks überprüft NetSaint, ob alle für das Monitoring notwendigen Beziehungen korrekt definiert wurden. Dienste, Hosts, Hostgruppen, Kontakte, Kontaktgruppen und Zeitperioden stehen zueinander in Beziehung und müssen korrekt definiert sein. Hier ist eine Liste der grundlegenden Beziehungen die NetSaint versucht zu überprüfen, bevor das Monitoring gestartet wird...

1. Prüfe, ob alle Kontakte Mitglied mindestens einer Kontaktgruppe sind.
2. Prüfe, ob alle Kontakte die in einer Kontaktgruppe definiert wurden gültig sind.
3. Prüfe, ob alle Hosts Mitglied mindestens einer Hostgruppe sind.
4. Prüfe, ob alle Hosts die in einer Hostgruppe definiert wurden gültig sind.
5. Prüfe, ob alle Hosts mindestens einen angegliederten Dienst haben.
6. Prüfe, ob alle Befehle in den Dienst- und Hostchecks gültig sind.
7. Prüfe, ob alle Befehle in den Dienst- und Hostereignisbehandlungsroutinen (event handlers) gültig sind.
8. Prüfe, ob alle Befehle in den Dienst- und Hostbenachrichtigungen gültig sind.
9. Prüfe, ob alle Benachrichtigungszeitperioden, die für Dienste, Hosts und Kontakte definiert wurden, gültig sind.
10. Prüfe, ob alle Dienstcheckzeitperioden, die fuer Dienste definiert wurden, gültig sind.

Behebung eventueller Konfigurationsfehler

Ob nun wesentliche Daten vergessen wurden oder die Konfiguration nur leichte Fehler enthält,

NetSaint gibt in jedem Fall eine Warnung oder Fehlermeldung aus die auch Hinweise mitliefert, wo das Problem steckt. Fehlermeldungen geben generell die Zeile der Konfigurationsdatei aus, in der das Problem zu sein scheint. Bei Fehlern beendet NetSaint meistens den "Pre-Flight"-Check und gibt nur den ersten entdeckten Fehler aus. Dies ist so beabsichtigt, um zu vermeiden, daß ein Fehler mehrere Folgefehlermeldungen aus der restlichen Konfiguration auslöst und ausgibt. Wenn Fehlermeldungen auftreten, muß erst die entsprechende Konfigurationsdatei editiert und damit der Fehler behoben werden, dann sollte der Check wiederholt werden. Warnungen können *generell* ohne Gefahr ignoriert werden, das sie nur Empfehlungen sind.

Wie geht's nun weiter

Nachdem die Konfigurationsdateien überprüft wurden und alle Fehler beseitigt wurden, kann man sich mit gutem Grund sicher sein, daß sich NetSaint starten läßt und die definierten Dienst und Hosts überwachen wird. Weiter mit dem Starten von NetSaint!

Author: Matthias Eichler at INTERNET Date: 19.12.00 18:56 Normal BCC: Dietmar Schurr at Whirlpool-DESCHPB2 TO: dietmar.schurr@gmx.de at Internet Subject: ERGEBNISSE VON HEUTE ---------- Message Contents

NetSaint starten

ACHTUNG: Bevor NetSaint das erste Mal gestartet werden kann, muss es richtig konfiguriert sein, und einige Plugins installiert haben.
Die Plugins werden unabhängig von NetSaint selbst installiert, sind aber notwendig, um überhaupt einen Service monitoren zu können.
Sie sind erhältlich auf der Download-Seite von http://www.netsaint.org .
("Wie?!? Konfigurieren?!? --> werft einen Blick auf die UEBERSCHRIFT VON DIETMAR und UEBERSCHRIFT VON DIETMAR Konfigurations-Dateien, sowie auf UEBERSCHRIFT VON DIETMAR)

Die verschiedenen Wege NetSaint zu starten

Es gibt gundsätzlich vier verschiedene Wege NetSaint zu starten:

1. Manuell, als Programm im Vordergrund (allerdings nur sinnvoll zum testen und debuggen)
2. Manuell, als Programm im Hintergrund
3. Manuell, als Daemon
4. Automatisch beim booten der Maschine

Ok, werfen wir einen Blick auf diese vier Wege nach Rom...

NetSaint manuell als Programm im Vordergrund starten

Dies wird Deine Wahl sein, wenn Du die Debugging-Optionen aktiviert (und anschliessend NetSaint neu kompiliert) hast und das Programm testen willst. NetSaint als Programm im Vordergrund laufen zu lassen macht hier besonders Sinn, da man so alle Check- und Benachrichtigungs-Prozesse auf der Konsole beobachten kann.
Um NetSaint im Vordergrund zu Starten, sollte man es wie folgt aufrufen...

./netsaint <Haupt-Konfigurationsdatei>

Beachte, dass Du dabei die *Haupt-Konfigurationsdatei* mit dem kompletten Pfad- und Dateinamen angeben musst. Wird hier der Name der Host-Konfigurationsdatei übergeben, wird NetSaint nichts tun, ausser eine Fehlermeldung auszugeben.

Um NetSaint zu stoppen, kann man jederzeit STRG-C drücken. Natürlich sollte man beim debuggen

darüber nachdenken, die Konsolen- Ausgabe in eine Datei umzuleiten, um sich die Meldungen später genauer ansehen zu können...

NetSaint manuell als Programm im Hintergrund starten

Um NetSaint als Programm in den Hintergrund zu schicken, muss man nur das Und-Zeichen (Ampersand) an die Kommandozeile anhängen...

./netsaint <Haupt-Konfigurationsdatei> &

Beachte, dass Du dabei die *Haupt-Konfigurationsdatei* mit dem kompletten Pfad- und Dateinamen angeben musst. Wird hier der Name der Host-Konfigurationsdatei übergeben, wird NetSaint nichts tun, ausser eine Fehlermeldung auszugeben.

NetSaint manuell als Daemon starten

Seit NetSaint 0.0.6 ist der Daemon-Betrieb von eigentlich stabil.
Um Netsaint als Daemon laufen zu lassen, muss man das Programm nur mit dem Schalter **-d** in der Komandozeile aufrufen...

./netsaint -d <Haupt-Konfigurationsdatei>

NetSaint automatisch beim Booten der Maschine starten

Sobald Du NetSaint getestet hast und Dir sicher bist, dass es nicht abstürzt und die Konfiguration korrekt ist, macht es durchaus Sinn NetSaint automatisch beim Booten der Maschine starten zu lassen. Der Vorteil liegt auf der Hand: Nach einem Neustart des Rechners muss man nicht NetSaint per Hand starten...
Unter Linux (und den meisten anderen UNIXen) muss hierfür unter **/etc/rc.d/init.d/** ein Start-Skript angelegt werden. Anschliessend muss ein Link zum Verzeichnis des Runlevels gesetzt werden, in dem NetSaint gestartet werden soll. (Die Leute die sich hierüber Gedanken machen, werden schon wissen, was ich meine...;-)

Ein einfaches Start-Skript (heisst **init-script**) wird in das NetSaint-Hauptverzeichnis geschrieben, sobald das ./configure-Skript aufgrufen wurde. Du kannst dieses Beispiel-Skript unter /etc/rc.d/init.d mit dem Befehl '**make install-init**' installieren lassen. Dies ist ausserdem in der Installations-Beschreibung nachzulesen. Falls Du NetSaint automatisch beim Booten als Daemon starten lassen willst, solltest Du den Befehl '**make install-daemoninit**' benutzen.

Das mitgelieferte Start-Skript sollte ohne Probleme unter Linux laufen. Fü FreeBSD, Solaris, etc. sollte ein paar Modifizierungen nötig sein...

NetSaint stoppen und neu starten

Wie NetSaint gestoppt und neu gestartet wird, findest Du in dieser Beschreibung.

Author: Matthias Eichler at INTERNET Date: 19.12.00 18:56 Normal BCC: Dietmar Schurr at Whirlpool-DESCHPB2 TO: dietmar.schurr@gmx.de at Internet Subject: ERGEBNISSE VON HEUTE ---------- Message Contents

NetSaint stoppen und neu starten

Während dem Betrieb von NetSaint kann es durchaus vorkommen, dass das Programm bzw. der Daemon gestoppt oder die Konfiguration durch einen Neustart neu eingelesen werden soll. Dieses Kapitel beschreibt genau diese Vorgehensweise...

ACHTUNG: Bevor NetSaint neu gestartet wird, sollte man sicher gehen, dass die neue Haupt bzw. Host-Konfigurationsdatei korrekt ist. NetSaint kann die Konfiguration mit dem Schalter -v beim Programmaufruf überprüfen.
Sollte NetSaint beim Neustart ein Problem mit einer der Konfigurationsdateien feststellen, wird dieser Fehler im Logfile festgehalten und das Programm beendet sich selbst.

NetSaint stoppen und neu starten mit dem Start-Skript

Falls Du das mitgelieferte Start-Skript in Deinem /etc/rc.d/init.d Verzeichnis installiert hast, kannst Du NetSaint sehr einfach stoppen und auch wieder neu starten.
Falls Du das Skript nicht installiert hast, kannst Du den folgenden Teil überspringen und weiterlesen, wie man NetSaint manuell anhält.
Ich gehe jetzt einfach mal davon aus, dass das Skript bei Dir **netsaint** heisst, wie in den Beispielen hier auch...

Gewünschte Aktion	Kommando	Beschreibung
NetSaint anhalten	**/etc/rc.d/init.d/netsaint stop**	Dieser Befehl hält NetSaint an und löscht das aktuelle Status-Logfile.
NetSaint neu starten	**/etc/rc.d/init.d/netsaint restart**	Dieser Befehl hält NetSaint an, löscht das aktuelle Status-Logfile, und startet NetSaint anschliessend wieder.
Konfigurationsdateien neu laden	**/etc/rc.d/init.d/netsaint reload**	Dieser Befehl sendet dem NetSaint-Prozess das SIGHUP-Signal, was diesen veranlasst seine geladene Konfiguration zu vergessen, die Konfigurationsdateien neu einzulesen, und den Dienst wieder aufzunehmen.

NetSaint anzuhalten, neu zu starten, bzw. die Konfiguration neu zu laden sind mit dem Start-Skript recht einfach, so dass ich diesen Weg immer empfehlen würde.

Manuelles stoppen und neu starten von NetSaint

Falls Du kein Skript benutzt um NetSaint zu starten, sind einige Dinge per Hand zu tun...
Als erstes muss rausgefunden werden, unter welchen Prozess-ID NetSaint läuft. Anschliessend musst Du den *kill*-Befehl benutzen, um das Programm bzw. den Daemon zu beenden, neu zu starten bzw. die Konfigurationen neu einlesen zu lassen. Eine genauere Anleitung folgt etwas weiter unten...

Die Prozess-ID ausfindig machen

Ganz am Anfang musst Du die Prozess-ID von NetSaint kennen. Diese findet man relativ einfach mit dem folgenden Befehl heraus:

ps axu | grep netsaint

Der Befehl sollte dann eine Ausgabe bringen, die ungefähr wie diese aussieht:

```
netsaint  6808  0.0  0.7   840   352  p3 S    13:44   0:00 grep netsaint
netsaint 11149  0.2  1.0   868   488  ?  S   Feb 27   6:33 ./netsaint
netsaint.cfg
```

Hier erkennst Du, dass NetSaint vom user **netsaint** gestartet wurde und unter der Prozess-ID **11149** läuft.

NetSaint stoppen

Um NetSaint nun zu stoppen, muss der *kill*-Befehl wie folgt benutzt werden...

kill 11149

Natürlich sollte man **11149** mit der aktuellen Prozess-ID unter der NetSaint jetzt gerade bei Dir läuft ersetzen.

NetSaint neu starten

Wenn die Konfigurationen geändert wurden, willst Du wahrscheinlich, dass NetSaint neu startet und und seine neue Konfigurations-Dateien neu einliest.
Falls Du den Source-Code von NetSaint geändert und die NetSaint-Binary-Datei neu kompiliert hast, sollte man diese Methode *NICHT* anwenden. Statt dessen, sollte man einfach NetSaint hart stoppen und anschliessend komplett neu starten.
NetSaint lädt sich bei der hier beschriebenen Art und Weise nicht komplett neu, sondern 'vergisst' nur seine Konfiguration und liest diese neu ein. Um NetSaint hierzu zu veranlassen, muss man dem Prozess das **SIGHUP**-Signal schicken.
Wenn wir davon ausgehen, dass NetSaint mit der ID **11149** läuft (das entnehmen wie dem Beispiel oben), benutzen wir folgendes Kommand:

kill -HUP 11149

Aber nicht vergessen **11149** mit der aktuellen Prozess-ID von NetSaint zu ersetzen.

Author: Matthias Eichler at INTERNET Date: 21.12.00 00:35 Normal BCC: Dietmar Schurr at Whirlpool-DESCHPB2 TO: dietmar.schurr@gmx.de at Internet Subject: Ergebnisse von heute ---------- ---------- Message Contents

NetSaint Zusätze

In diesem Kapitel werden die Verschiedenen Zusätze, die für NetSaint verfügbar sind beschrieben. Diese und weitere Zusätze sind erhältlich im Downloadbereich der NetSaint Webseite (www.netsaint.org).

Index

cl_status	- Konsolenanwendung zur Anzeige des Status der überwachten Dienste
neat	- Web-basierte Administrations-Oberfläche für NetSaint
netsaint_mrtg	- MRTG Skripte für grafische Host und Diensteinformationen
netsaint_reports	- Reporting-Tool für Hosts und Services-Informationen im zeitlichen Zusammenhang
netsaint_statd	- Perl Daemon zum überwachen von entfernten (nehmen wir doch den englischen Begriff, "remote"...ok?!?) Host-Informationen
nrpe	- Daemon und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
nrpep	- Service und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
nsa	- Web- und Datenbank-basiertes Administrations-Interface für NetSaint
nsca	- Daemon und Client-Programm um Überprüfungs-Ergebnisse über ein Netzwerk zu anderen NetSaint-Servern zu senden
pscwatch	- Watchdog-Daemon der das Versenden der Check-Ergebnisse überwacht

cl_status	- Konsolenanwendung zur Anzeige des Status der überwachten Dienste
Autor:	Adam Bowen
Beschreibung:	Das Programm wurde geschrieben, um den Status der überwachten Dienste auf der Konsole anzeigen zu lassen. Es benutzt ncurses um so viele Status-Zeilen wie möglich anzuzeigen, abhängig von der Bildschirm-Größe. Ausserdem lässt es die Konsolle piepen und blinken, sobald ein überwachter Service versagt. Die Konfiguration erlaubt unterschiedliche Einstellungen, wie oft cl_status das NetSaint Status-Logfile ausliest.

neat	- Web-basierte Administrations-Oberfläche für NetSaint
Autor:	Jason Blakey

Beschreibung: NetSaint Easy Administration Tool (NEAT) ist eine in Perl geschriebene und Web-basierte Oberfläche für NetSaint. Es erlaut das ändern, löschen und neuanlegen von Einträgen in der Host-Konfiguration. Ausserdem kann man über NEAT nach einer Änderung der Konfiguration NetSaint neu starten lassen. Anders als nsa, benötigt es allerdings keine Datenbank um die Daten zu speichern.

netsaint_mrtg - MRTG Skripte für grafische Host und Diensteinformationen

Autor: Richard Mayhew

Überblick: MRTG erlaubt es einem graphische Auswertungen von der Status-Informationen von überwachten Hosts und Services zu erzeugen.

Dateien:

mrtghost_total.pl	- Ein Perl Skript, das die Gesamt-Anzahl der Hosts ermittelt, die up und down sind.
mrtgsvc_total.pl	- Ein Perl Skript, das die Gesamt-Anzahl der Services ermittelt, die up und down sind.
mrtgsvchost_total.pl	- Ein Perl Skript, das die Gesamt-Anzahl der Services ermittelt, die auf einem bestimmten Host up und down sind.
mrtgsvctyp_total.pl	- Ein Perl Skript, das die Gesamt-Anzahl eines Service-Types ermittelt, die up und down sind.

Beschreibung: Das "e;netsaint_mrtg"e;-Paket beinhaltet zwei Skripte, die es MRTG erlaubt grafische Darstellung aus den NetSaint-Status-Informationen zu erzeugen.
Die Skripts scannen das Status-Logfile von NetSaint und erkennen die Gesamt-Anzahl von Hosts bzw. Services, die ok sind oder ein Problem aufweisen.
Eine Erklärung wie die Skripts in MRTG zu implementieren sind, findet man unter **README.mrtg**.

Wichtig:

- Um die Grafiken von netsaint-mrtg erzeugen zu lassen, muss MRTG installiert sein.

netsaint_statd - Perl Daemon zum überwachen von entfernten Host-Informationen

Autor: Nick Reinking

Überblick: Erlaubt einem die Auslastung der Festplatte, die Durchschnitts-Last, die Prozesse und User auf einer entfernten Maschine zu überwachen.

Dateien:

netsaint_statd	- Perl Daemon, der auf der Remote-Maschine läuft.
check_disk.pl	- Ein Perl PlugIn, das von NetSaint ausgeführt wird, um Informationen über die Auslastung einer einzelnen Festplatte auf der Remote-Maschine zu sammeln.
check_all_disks.pl	- Ein Perl PlugIn, das von NetSaint ausgeführt wird, um Informationen über die Auslastung aller Festplatten auf der Remote-Maschine zu sammeln.
check_users.pl	- Ein Perl PlugIn, das von NetSaint ausgeführt wird, um Informationen über die Anzahl der User auf der Remote-Maschine zu sammeln.
check_procs.pl	- Ein Perl PlugIn, das von NetSaint ausgeführt wird, um Informationen über die Prozesse auf der Remote-Maschine zu sammeln.
check_load.pl	- Ein Perl PlugIn, das von NetSaint ausgeführt wird, um Informationen über die Last der Remote-Maschine zu sammeln.
Changelog	- Die letzten Änderungen von netsaint_statd
README	- Kommando-Optionen und Parameter (für die hosts.cfg)

Beschreibung: netsaint_statd ist ein Daemon, der es einem NetSaint-Server mit einem korrespondierenden Skript erlaubt Informationen wie Prozess-, und User-Anzahl, sowie Festplatten-Benutzung und Last-Informationen auf der Maschine, auf der der Daemon läft, zu sammeln.
Der Daemon verarbeitet die gesammelten Informationen nicht selbst.
Statt dessen sammelt er nur die Informationen und reicht sie an den "rufenden" NetSaint-Server weiter, damit dieser sie interpretieren kann.

Das Daemon-Skript ist so geschrieben, dass es leicht auf andere Betriebssysteme zu portieren ist (...so lange auf der Maschine Perl installiert ist ;-).
Deswegen sollte es auch ein leichtes sein, andere Informationen zu sammeln, indem sie der entschrenden Kommando-Liste von *netsaint_statd* hinzugefügt werden.
Mittlerweile läuft mittlerweile problemlos unter HP-UX, Linux, Solaris/SunOS, IRIX, OSF1, FreeBSD, AIX, OpenBSD und NEXTSTEP. Vor allem kann *netsaint_statd* deshalb so schnell neue Informationen sammeln, indem es einfach dem Skript hinzugefügt werden.
Das Einzige, was *netsaint_statd* benötigt, sind die Standard-UNIX-Tools, wie *ps*, *df*, etc.

Damit nicht jeder Internet-User Einblick bekommt, wieviel Speicherplatz auf einer Maschine belegt ist, kann *netsaint_statd* den Zugriff von fremden NetSaint-Servern unterbindet.
In einer kurzen Liste werden die IP's eingetragen, den *netsaint_statd* überhaupt "zuhört".

Wichtig:

- Man muss immer die erste Zeile in der entsprechenden Datei ändern, damit das Skript die Perl-Binary zu finden.

nrpe	- Service und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
Autor:	Me
Überblick:	Erlaubt es einem, PlugIns auf einer Remote-Maschine in einer einfachen und transparenten Art und Weise auszuführen.
Dateien:	check_nrpe - Dieses PlugIn wird benötigt, um die Ausführungs-Anfrage an den nrpe-Agenten auf der Remote-Maschine zu übergeben. nrpe - Dies ist der Agent, der auf der Remote-Maschine läuft und die Ausführungs-Anfragen entegegenzunehmen. nrpe.cfg - Konfigurationsdatei für den nrpe-Agenten auf der Remote-Maschine.
Beschreibung:	Dieser Zusatz wurde geschrieben, um PlugIns auf einer Remote-Maschine auszuführen. Das *check_nrpe*-PlugIn läft auf dem NetSaint-Server und sendet Ausführungs-Anfragen an den nrpe-Agenten auf einer Remote-Maschine. Der nrpe-Agent wird beim Empfang der Anfrage ein entsprechendes PlugIn auf der Remote- Maschine ausführen und den Code, den das PlugIn zurückgibt an das *check-nrpe* zurücksenden, als hätte das PlugIn den Wert selbst erhalten. Der nrpe-Agent kann entweder als Stand-Alone-Daemon oder unter dem Super-Daemon inetd laufen.
Wichtig:	• Läuft der Agent im Stand-Alone-Mode, überprüft dieser die Anfrage, indem er die IP des fragenden Servers mit einer Liste in seiner Konfigurationsdatei vergleicht. • Läuft er unter inetd kann der "TCP-Wrapper" benutzt werden, um den Zugriff auf den Agenten zu schützen.

nrpep	- Service und PlugIn zum ausführen von NetSaint PlugIns auf entfernten Maschinen
Autor:	Adam Jacob
Überblick:	Erlaubt es einem, PlugIns auf einer Remote-Maschine in einer einfachen und transparenten Art und Weise auszuführen.
Beschreibung:	Das "NetSaint Remote Plugin Executor/Perl" (NRPEP) wurde als Ersatz für die netsaint_statd und nrpe Zusätze geschrieben. Obwohl dieser Zusatz ähnlich funktioniert wie nrpe, wurde es in Perl geschrieben und benutzt TripleDES-Verschlüsselung, um die Daten während dem senden durchs Netz zu schützen. Es sollter unter inetd eingesetzt werden, so dass der "TCP Wrapper" den Zugang zu nrpep kontrolliert.

Wichtig:

- *nrpep* benötigt zwei Perl-Module: **Crypt-TripleDES-0.24** und **Digest-MD5-2.09**

nsa	- Web- und Datenbank-basiertes Administrations-Interface für NetSaint

Autor: Daniel Burke

Beschreibung: Daniel Burke hat diesen exzellenten Zusatz - "NetSaint Administrator" - geschrieben, um dem Anspruch nach einer User-freundlichen Konfiguration von NetSaint gerecht zu werden.
Das Programm erlaubt es die Konfigurationsdateien (für Hosts-, Services-, Kontakt-, Zeit-Konfigurationen, etc.) vie einer Web-Oberfläche zu editieren. Die Konfigurationsdaten werden dabei in einer MySQL-Datenbank gespeichert und dann im richtigen Format in eine Text-Datei geschrieben.
Die NSA-CGI's können NetSaint auch mit der Option **-v** aufrufen, um die Konfiguration zu überprüfen.
Es ist ein wirklich enormes Tool für jeden der entweder Text-Konfigurationsdateien hasst, oder einfach einen einfacheren Weg sucht, um die Daten zu administrieren.

Wichtig:

- Um *nsa* laufen zu lassen, muss man MySQL v2.22.25 oder neuer, Perl5 mit DBI und DBD installiert haben. Ausserdem ist es doch recht hilfreich zu wissen wie Datenbanken und Tabellen in MySQL angelegt und gelöscht werden.

nsca	- Daemon und Client-Programm um Überprüfungs-Ergebnisse über ein Netzwerk zu anderen NetSaint-Servern zu senden

Autor: Me

Überblick: Dieser Zusatz erlaubt es einem NetSaint-Server die Überprüfungs von entfernten NetSaint-Servern entgegenzunehmen und zu verarbeiten.

Dateien:

nsca	- Der nsca-Daemon, der auf dem zentralen NetSaint-Server läuft und die von den Remote-NetSaint-Server gelieferten Überprüfungsergebnisse entgegennimmt
nsca.cfg	- Die Konfigurationsdatei für den nsca-Daemon
send_nsca	- Das Client-Programm, das auf dem Remote-NetSaint-Server ausgeführt wird, um seine Ergebnisse an den nsca-Daemon auf dem zentralen NetSaint-Server weiterzureichen
send_nsca.cfg	- Konfigurationsdatei für den send_nsca-Client

Beschreibung: Dieses Zusatz-Programm erlaubt es einem passive Überprüfungs-Ergebnisse von einem Remote-NetSaint-Server an einen zentralen NetSaint-Server zu senden, der diese Ergebnisse dann auswertet und behandelt.
Der Client kann als Stand-Alone-Programm oder als Teil eines ganzen NetSaint-Servers (dieser muss mit dem ocsp-Befehl eine verteilte Struktur aufzubauen) laufen.

Wichtig:

- Seit der ersten Beta-Version benutzten sowohl nsca-Client und -Server als einen elementaren Bestandteil eine XOR-Operation um die Daten auf Ihrem Weg durch das Netzwerk zu "verschlüsseln". Dies ist zwar nicht gerade der "sicherste" Weg, wurde aber implementiert, um dem Konzept Rechnung zu tragen, das der nsca-Daemon den vom Client kommenden Daten vertrauen kann.
  Hoffentlich wird bald eine starke Verschlüsselung auf Basis von privaten Keys (ähnlich dem PGP-Verfahren) in das Programm integriert.
  Vorraussetzung hierfür ist allerdings, dass das Feature anschliessend unter allen NetSaint-Plattformen (ja, auch NT, Novell, etc.) läft.

pscwatch	- Watchdog-Daemon der das Versenden der Check-Ergebnisse überwacht
Autor:	Me
Überblick:	Überwacht den zuverlässigen und regelmässigen Versand der passiven Überwachungs-Ergebnisse des nsca-Clients.
Beschreibung:	Dieser NetSaint-Zusatz stellt sicher, dass die passiven Service-Überwachungen regelmässigen Abständen ausgeführt werden und Ihre Ergebnisse an NetSaint verschicken. Dieser Zusatz wurde geschrieben, um auf dem zentralen NetSaint-Server zu laufen, sobald man eine verteilte Überwachungs-Struktur aufbaut.

Distributed Monitoring

Introduction

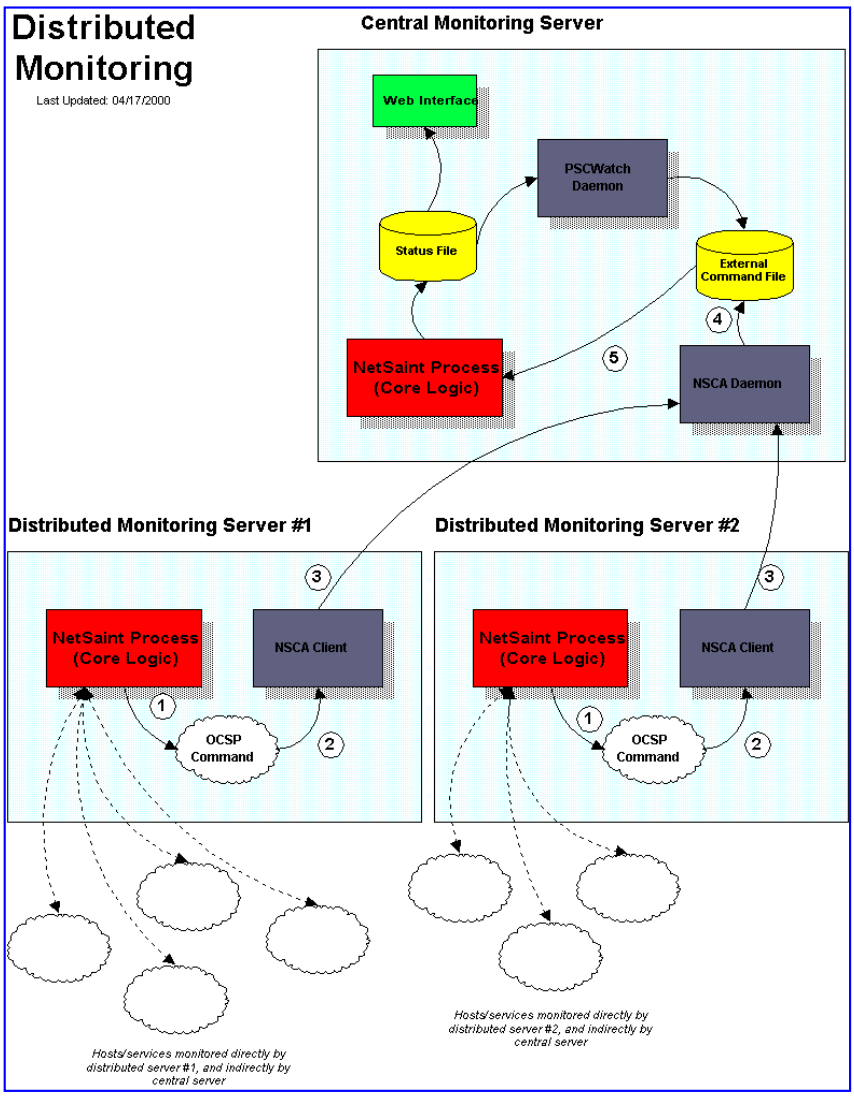
Beginning with release 0.0.6, NetSaint can *optionally* be configured to support distributed monitoring of network services and resources. I'll try to briefly explan how this can be accomplished...

Goals

The goal in the distributed monitoring environment that I will describe is to offload the overhead (CPU usage, etc.) of performing service checks from a "central" server onto one or more "distributed" servers. Most small to medium sized shops will not have a real need for setting up such an environment. However, when you want to start monitoring hundreds or even thousands of *hosts* (and several times that many services) using NetSaint, this becomes quite important.

Reference Diagram

The diagram below should help give you a general idea of how distributed monitoring works with NetSaint. I'll be referring to the items shown in the diagram as I explain things...



Central Server vs. Distributed Servers

When setting up a distributed monitoring environment with NetSaint, there are differences in the way the central and

distributed servers are configured. I'll show you how to configure both types of servers and explain what effects the changes being made have on the overall monitoring. For starters, lets describe the purpose of the different types of servers...

The function of a *distributed server* is to actively perform checks all the services you define for a "cluster" of hosts. I use the term "cluster" loosely - it basically just mean an arbitrary group of hosts on your network. Depending on your network layout, you may have several cluters at one physical location, or each cluster may be separated by a WAN, its own firewall, etc. The important thing to remember to that for each cluster of hosts (however you define that), there is one distributed server that runs NetSaint and monitors the services on the hosts in the cluster. A distributed server is usually a bare-bones installation of NetSaint. It doesn't have to have the web interface installed, send out notifications, run event handler scripts, or do anything other than execute service checks if you don't want it to. More detailed information on configuring a distributed server comes later...

The purpose of the *central server* is to simply listen for service check results from one or more distributed servers. Even though services are actively checked from the central server, the active checks are only performed at long intervals (as will be described later), so lets just say that the central server only accepts passive check for now. Since the central server is obtaining passive service check results from one or more distributed servers, it serves as the focal point for all monitoring logic (i.e. it sends out notifications, runs event handler scripts, determines host states, has the web interface installed, etc).

Obtaining Service Check Information From Distributed Monitors

Okay, before we go jumping into configuration detail we need to know how to send the service check results from the distributed servers to the central server. I've already discussed how to submit passive check results to NetSaint from same host that NetSaint is running on (as described in the documentation on passive checks), but I haven't given any info on how to submit passive check results from other hosts.

In order to facilitate the submission of passive check results to a remote host, I've written the nsca addon. The addon consists of two pieces. The first is a client program (send_nsca) which is run from a remote host and is used to send the service check results to another server. The second piece is the nsca daemon (nsca) which either runs as a standalone daemon or under inetd and listens for connections from client programs. Upon receiving service check information from a client, the daemon will sumbit the check information to NetSaint (on the central server) by inserting a *PROCESS_SVC_CHECK_RESULT* command into the external command file, along with the check results. The next time NetSaint checks for external commands, it will find the passive service check information that was sent from the distributed server and process it. Easy, huh?

Distributed Server Configuration

So how exactly is NetSaint configured on a distributed server? Basically, its just a bare-bones installation. You don't need to install the web interface or have notifications sent out from the server, as this will all be handled by the central server.

Key configuration changes:

- Only those services and hosts which are being monitored directly by the distributed server are defined in the host configuration file.
- The distributed server has its initial program mode set to *STANDBY*. This will prevent any notifications from being sent out by the server.
- The distributed server is configured to obsess over services.
- The distributed server has an ocsp command defined (as described below).

In order to make everything come together and work properly, we want the distributed server to report the results of *all* service checks to NetSaint. We could use event handlers to report *changes* in the state of a service, but that just doesn't cut it. In order to force the distributed server to report all service check results, you must enabled the obsess_over_services option in the main configuration file and provide a ocsp_command to be run after every service check. We will use the ocsp command to send the results of all service checks to the central server, making use of the send_nsca client and nsca daemon (as described above) to handle the tranmission.

In order to accomplish this, you'll need to define an ocsp command like this:

ocsp_command=submit_check_result

The command definition for the *submit_check_result* command looks something like this:

command[submit_check_result]=/usr/local/netsaint/libexec/eventhandlers/submit_check_result $HOSTNAME$ '$SERVICEDESC$' $SERVICESTATE$ '$OUTPUT$'

The *submit_check_result* shell scripts looks something like this (replace *central_server* with the IP address of the central server):

```
        #!/bin/sh

        # Arguments:
        #  $1 = host_name (Short name of host that the service is
        #       associated with)
        #  $2 = svc_description (Description of the service)
        #  $3 = state_string (A string representing the status of
        #       the given service - "OK", "WARNING", "CRITICAL"
        #       or "UNKNOWN")
        #  $4 = plugin_output (A text string that should be used
        #       as the plugin output for the service checks)
        #

        # Convert the state string to the corresponding return code
        return_code=-1

        case "$3" in
            OK)
                        return_code=0
                    ;;
                WARNING)
                    return_code=1
                        ;;
                CRITICAL)
                    return_code=2
                        ;;
                UNKNOWN)
                    return_code=-1
                        ;;
        esac

        # pipe the service check info into the send_nsca program, which
        # in turn transmits the data to the nsca daemon on the central
        # monitoring server

        /bin/echo -e "$1\t$2\t$return_code\t$4\n" | /usr/local/netsaint/bin/send_nsca
central_server -c /usr/local/netsaint/var/send_nsca.cfg
```

The script above assumes that you have the send_nsca program and it configuration file (send_nsca.cfg) located in the */usr/local/netsaint/bin/* and */usr/local/netsaint/var/* directories, respectively.

That's it! We've sucessfully configured a remote host running NetSaint to act as a distributed monitoring server. Let's go over exactly what happens with the distributed server and how it sends service check results to NetSaint (the steps outlined below correspond to the numbers in the reference diagram above):

1. After the distributed server finishes executing a service check, it executes the command you defined by the ocsp_command variable. In our example, this is the */usr/local/netsaint/libexec/eventhandlers/submit_check_result* script. Note that the definition for the *submit_check_result* command passed four pieces of information to the script: the name of the host the service is associated with, the service description, the return code from the service check, and the plugin output from the service check.
2. The *submit_check_result* script pipes the service check information (host name, description, return code, and output) to the *send_nsca* client program.
3. The *send_nsca* program transmits the service check information to the *nsca* daemon on the central monitoring server.
4. The *nsca* daemon on the central server takes the service check information and writes it to the external command file for later pickup by NetSaint.
5. The NetSaint process on the central server reads the external command file and processes the passive service check information that originated from the distributed monitoring server.

Central Server Configuration

We've looked at hot distributed monitoring servers should be configured, so let's turn to the central server. For all intensive purposes, the central is configured as you would normally configure a standalone server. It is setup with:

- The web interface (optional, but recommended)
- Notifications (optional, but recommended)
- Event handlers (optional)
- Active service checks enabled (required)
- External command checks enabled (required)
- Passive service checks enabled (required)

There are two other very important things that you need to keep in mind when configuring the central server:

- The central server must have service definitions for *all services* that are being monitored by all the distributed servers. NetSaint will ignore passive check results if they do not correspond to a service that has been defined.
- The normal *check_interval* argument for each service definition should be set to a long time interval (i.e. 24 hours or a week).

It is important that you set the *check_interval* argument for each service definition to a long interval. This will ensure that active service checks account for only a minimal load on the central server. We don't want to disable service checks, as it will be necessary to sometimes force NetSaint to actively check services (as discussed below).

That's it! Easy, huh?

Problems With Passive Checks

For all intensive purposes we can say that the central server is relying solely on passive checks for monitoring. While it does perform active checks of all services, it only does so at very long intervals, so lets disregard that fact. The main problem with relying completely on passive checks for monitoring is the fact that NetSaint must rely on something else to provide the monitoring data. What if the remote host that is sending in passive check results goes down or becomes unreachable? If NetSaint isn't actively checking the services on the host, how will it know that there is a problem?

We can protect against this type of problem by using another addon to monitoring incoming passive check results...

Watchdog Daemon

In order to protect against situations where remote hosts may stop sending passive service checks into the central monitoring server, I've developed the pscwatch daemon. The daemon's sole purpose in life is to ensure that service checks are being either performed actively by the central server or being provided passively be distributed servers on a regular basis.

If the *pscwatch* daemon detects that a service check has not been performed within a given threshold of time, it will send a

command to NetSaint via the external command file telling it to schedule an immediate active check of the service. When NetSaint performs an active check of the service, it will be able to tell if there is a real problem or not. Problem solved.

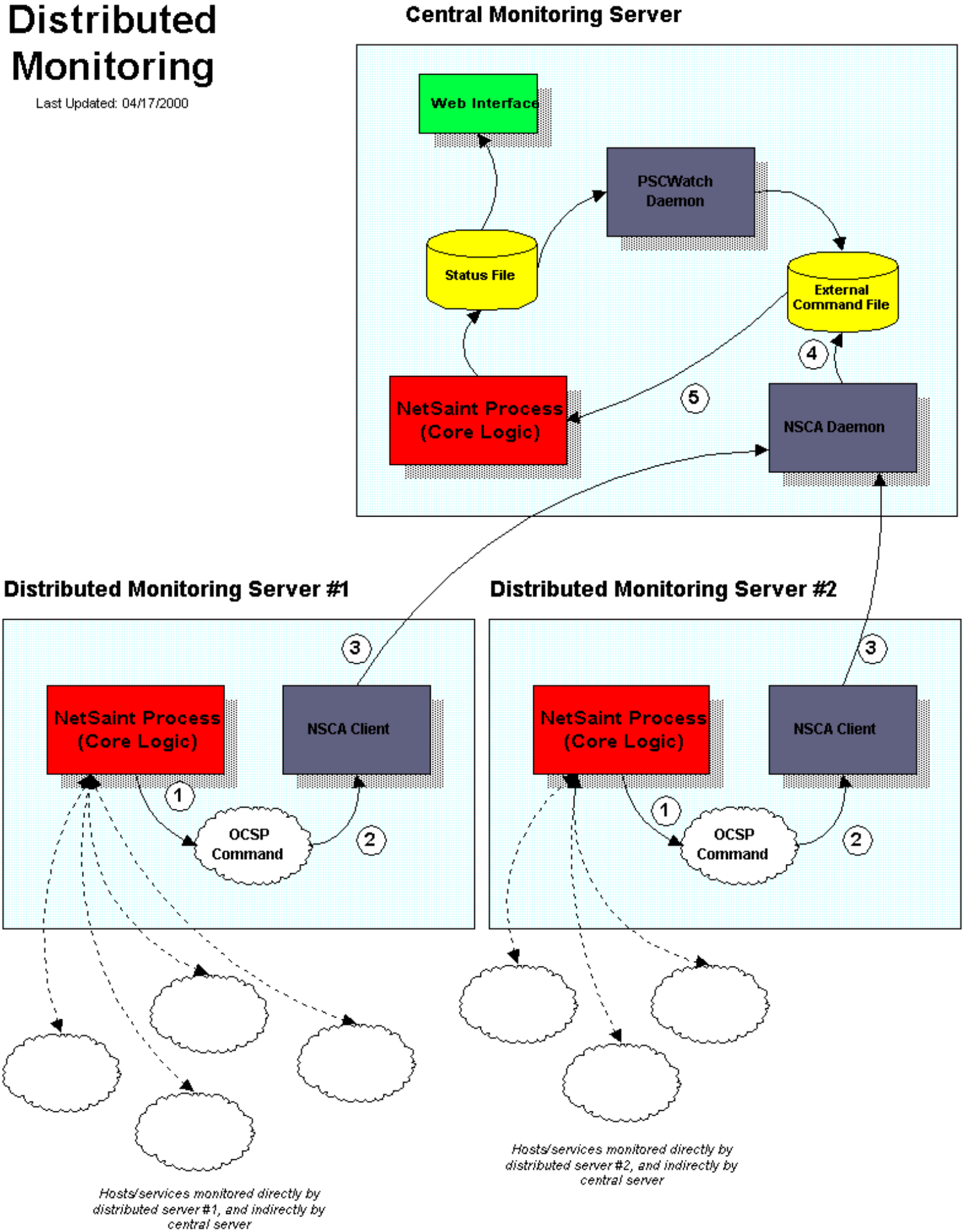
Note: If service checks are disabled, NetSaint will refuse to actively perform a service check. This is the reason why we don't want to disable active checks on the central server. Instead, we just set the normal check interval for all services to a very long time period.

Combining Distributed Monitoring With Redundancy

Nothing here yet...

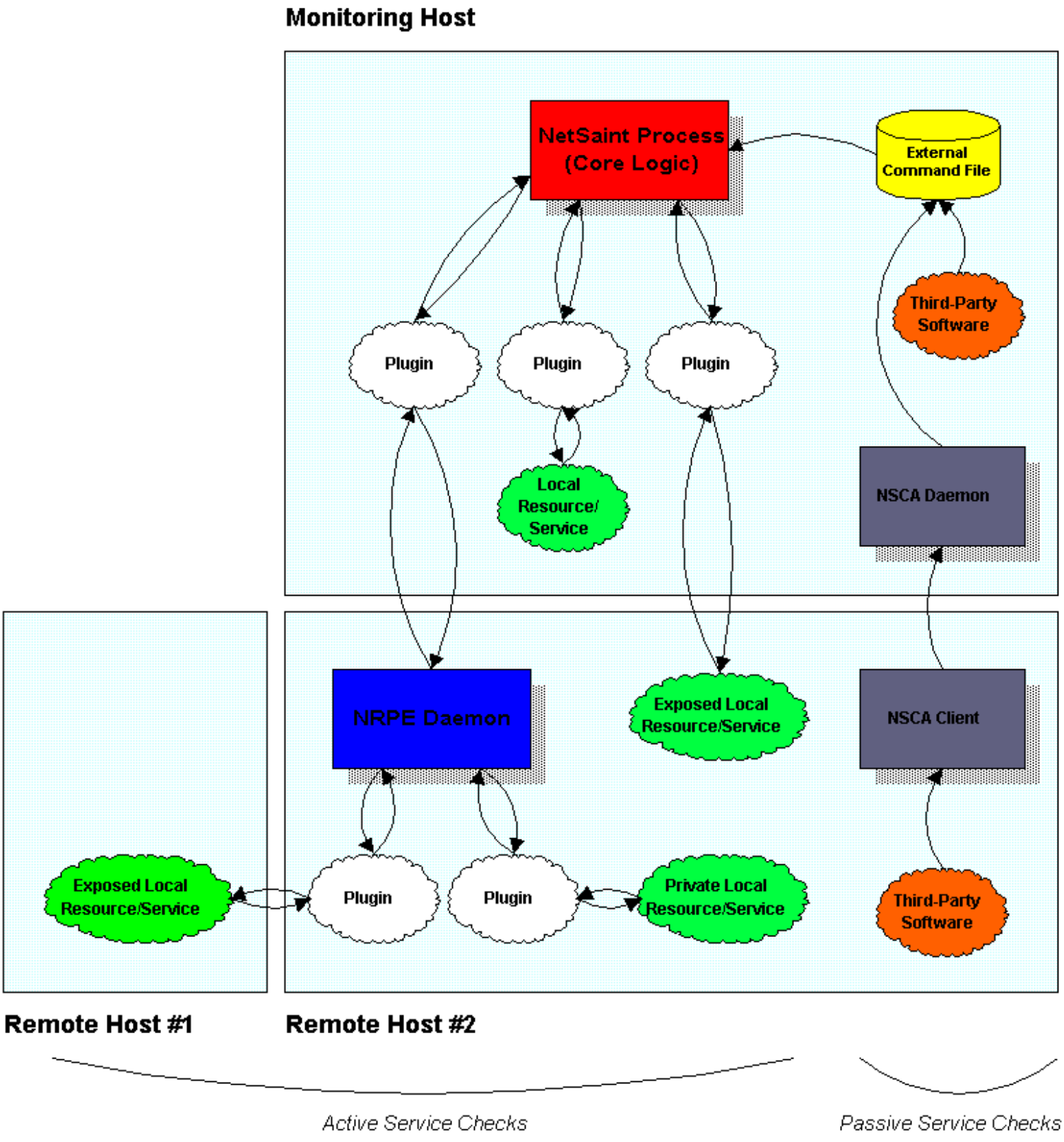
Distributed Monitoring

Last Updated: 04/17/2000



Using Active And Passive Checks Together

Last Updated: 04/18/2000



Service Check Parallelization

Introduction

Beginning with release 0.0.5, the ability to execute service checks in parallel was built into NetSaint. This documentation will attempt to explain in detail what that means and how it affects services that you have defined.

Changes In Service Check Logic

In order to facilitate parallelized service checks, the service check logic has been changed from that of version 0.0.4 and earlier. These earlier versions of NetSaint executed one service check at a time and processed the results from the check before moving onto the next service.

Beginning with version 0.0.5, the service check logic has been broken up into two distinct parts - *execution of service checks* and *processing of service check results* (also called service "reaper" events).

How The Parallelization Works

Before I can explain how the service check parallelization works, you first have to understand a bit about how NetSaint schedules events. All internal events in NetSaint (i.e. log file rotations, external command checks, service checks, etc.) are placed in an event queue. Each item in the event queue has a time at which it is scheduled to be executed. NetSaint does its best to ensure that all events get executed when they should, although events may fall behing schedule if NetSaint is busy doing other things.

Service checks are one type of event that get scheduled in NetSaint's event queue. When it comes time for a service check to be executed, NetSaint will kick off another process (using a call to fork()) to go out and run the service check (i.e. a plugin of some sort). NetSaint does *not*, however, wait for the service check to finish! Instead, NetSaint will immediately go back to servicing other events that reside in the event queue...

So what happens when the service check finishes executing? Well, the process that was started by NetSaint to run the service check sends a message back to NetSaint containing the results of the service check. It is then up to NetSaint to check for and process the results of that service check when it gets a chance.

In order for NetSaint to actually do any monitoring, it much process the results of service checks that have finished executing. This is done via a service check "reaper" process. Service "reapers" are another type of event that get scheduled in NetSaint's event queue. The frequency of these "reaper" events is determined by the service_reaper_frequency option in the main configuration file. When a

"reaper" event is executed, it will check for any messages that contain the result of service checks that have finished executing. These service check results are then handled by the core service monitoring logic. From there NetSaint determines whether or not hosts should be checked, notifications should be sent out, etc. When the service check results have been processed, NetSaint will reschedule the next check of the service and place it in the event queue for later execution. That completes the service check/monitoring cycle!

For those of you who really want to know, but haven't looked at the code, NetSaint uses message queues to handle communication between NetSaint and the process that actually runs the service check...

Potential Gotchas...

You should realize that there are potential drawbacks to having service checks parallelized. Since more than one service check may be running at the same time, they have may interfere with one another. You'll have to evaluate what types of service checks you're running and take appropriate steps to guard against any unfriendly outcomes. This is particularly important if you have more than one service check that accesses any hardware (like a modem). Also, if two or more service checks connect to daemon on a remote host to check some information, make sure that daemon can handle multiple simultaneous connections.

Fortunately, there are some things you can do to protect against problems with having some types of service checks "collide"...

1. The easiest thing you can do to prevent service check collisions to to use the service_interleave_factor variable. Interleaving services will help to reduce the load imposed upon remote hosts by service checks. Set the variable to use "smart" interleave factor calculation and then adjust it manually if you find it necessary to do so.
2. The second thing you can do is to set the max_attempts argument in each service definition to something greater than one. If the service check does happen to collide with another running check, NetSaint will retry the service check *max_attempts-1* times before notifying anyone of a problem.
3. You could try is to implement some kind of "back-off and retry" logic in the actual service check code, although you may find it difficult or too time-consuming
4. If all else fails you can effectively prevent service checks from being parallelized by setting the max_concurrent_checks option to 1. This will allow only one service to be checked at a time, so it isn't a spectacular solution. If there is enough demand, I will add an option to the service definitions which will allow you to specify on a per-service basis whether or not a service check can be parallelized. If there isn't enough demand, I won't...

One other thing to note is the effect that parallelization of service checks can have on system resources on the machine that runs NetSaint. Running a lot of service checks in parallel can be taxing on the CPU and memory. The inter_check_delay_method will attempt to minimize the load imposed on your machine by spreading the checks out evenly over time (if you use the "smart" method), but it isn't a surefire solution. In order to have some control over how many service checks can be run at any

given time, use the max_concurrent_checks variable. You'll have to tweak this value based on the total number of services you check, the system resources you have available (CPU speed, memory, etc.), and other processes which are running on your machine. For more information on how to tweak the *max_concurrent_checks* variable for your setup, read the documentation on check scheduling.

What Isn't Parallelized

It is important to remember that only the *execution* of service checks has been parallelized. There is good reason for this - other things cannot be parallelized in a very safe or sane manner. In particular, event handlers, contact notifications, processing of service checks, and host checks are *not* parallelized. Here's why...

Event handlers are not parallelized because of what they are designed to do. Much of the power of event handlers comes from the ability to do proactive problem resultion. An example of this is restarting the web server when the HTTP service on the local machine is detected as being down. In order to prevent more than one event handler from trying to "fix" problems in parallel (without any knowledge of what each other is doing), I have decided to not parallelize them.

Contact notifications are not parallelized because of potential notification methods you may be using. If, for example, a contact notification uses a modem to dial out and send a message to your pager, it requires exclusive access to the modem while the notification is in progress. If two or more such notifications were being executed in parallel, all but one would fail because the others could not get access to the modem. There are ways to get around this, like providing some kind of "back-off and retry" method in the notification script, but I've decided not to rely on users having implemented this type of feature in their scripts. One quick note - if you have service checks which use a modem, make sure that any notification scripts that dial out have some method of retrying access to the modem. This is necessary because a service check may be running at the same time a notification is!

Processing of service check results has not been parallelized. This has been done to prevent situations where multiple notifications about host problems or recoveries may be sent out if a host goes down, becomes unreachable, or recovers.

Service Check Planung

Inhalt

Einleitung
Konfigurationsoptionen
Initial scheduling
Inter-check delay
Service interleaving
Max concurrent service checks
Time restraints
Normal scheduling
Scheduling during problems
Host checks
Scheduling delays
Scheduling example

Einleitung

Ich habe viele Fragen erhalten betreffend der Planung von Service Checks (Überprüfung von Diensten) in bestimmten Situationen und auch darüber, wie die Planung von der Ausführung abweicht und die Ergebnisse verarbeitet werden. Ich versuche im folgenden Abschnitt dies zu erläutern...
I've gotten a lot of questions regarding how service checks are scheduled in certain situations, along with how the scheduling differs from when the checks are actually executed and their results are processed. I'll try to go into a little more detail on how this all works...

Konfigurationsoptionen

Before we begin, there are several configuration options that affect how service checks are scheduled, executed, and processed. For starters, each service definition contains three options that determine when and how each specific service check is scheduled and executed. Those three options include:

- *check_interval*
- *retry_interval*
- *check_period*

There are also four configuration options in the main configuration file that affect service checks. These include:

- *inter_check_delay_method*
- *service_interleave_factor*
- *max_concurrent_checks*
- *service_reaper_frequency*

We'll go into more detail on how all these options affect service check scheduling as we progress. First off, let's see how services are initially scheduled when NetSaint first starts or restarts...

Initial Scheduling

When NetSaint (re)starts, it will attempt to schedule the initial check of all services in a manner that will minimize the load imposed on the local and remote hosts. This is done by spacing the initial service checks out, as well as interleaving them. The spacing of service checks (also known as the inter-check delay) is used to minimize/equalize the load on the local host running NetSaint and the interleaving is used to minimize/equalize load imposed on remote hosts. Both the inter-check delay and interleave functions are discussed below.

Even though service checks are initially scheduled to balance the load on both the local and remote hosts, things will eventually give in to the ensuing chaos and be a bit random. Reasons for this include the fact that services are not all checked at the same interval, some services take longer to execute than others, host and/or service problems can alter the timing of one or more service checks, etc. At least we try to get things off to a good start. Hopefully the initial scheduling will keep the load on the local and remote hosts fairly balanced as time goes by...

Note: If you want to view the initial service check scheduling information, start NetSaint using the **-s** command line option. Doing so will display basic scheduling information (inter-check delay, interleave factor, first and last service check time, etc) and will create a new status log that shows the exact time that all services are initially scheduled. Because this option will overwrite the status log, you should not use it when another copy of NetSaint is running. NetSaint does *not* start monitoring anything when this argument is used.

Inter-Check Delay

As mentioned before, NetSaint attempts to equalize the load placed on the machine that is running NetSaint by equally spacing out initial service checks. The spacing between consecutive service checks is called the inter-check delay. By giving a value to the inter_check_delay_method variable in the main config file, you can modify how this delay is calculated. I will discuss how the "smart" calculation works, as this is the setting you will want to use for normal operation.

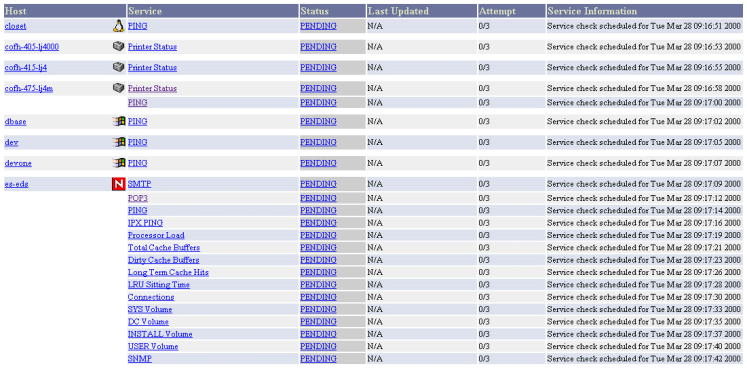
When using the "smart" setting of the *inter_check_delay_method* variable, NetSaint will calculate an inter-check delay value by using the following calculation:

inter-check delay = (total normal check interval for all services) / (total number of services)^2

Let's take an example. Say you have 1,000 services that each have a normal check interval of 5 minutes (obviously some services are going to be checked at different intervals, but let's look at an easy case...). The total check interal time for all services is 5,000 (1,000 * 5). That means that the average check interval for each service is 5 minutes (5,000 / 1,000). Give that information, we realize that (on average) we need to re-check 1,000 services every 5 minutes. This means that we should use an inter-check delay of 0.005 minutes (0.3 seconds) when spacing out the initial service checks. By spacing each service check out by 0.3 seconds, we can somewhat guarantee that NetSaint is scheduling and/or executing 3 new service checks every second. By spacing the checks out evenly over time like this, we can hope that the load on the local server that is running NetSaint remains somewhat balanced.

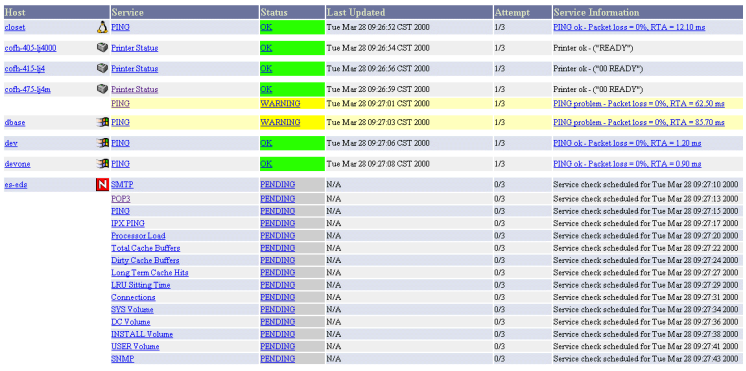
The following two images show some output from the status CGI after NetSaint has been started and demonstrate how the inter-check delay works. For these examples, the inter-check delay was approximately 2.3 seconds (there were a total of 113 services with an average check interval of about 4.3 minutes). The first image shows the inital scheduling of service checks and the second image shows how NetSaint executes service checks (the interleave_factor option was set to 1 for this example, so checks are not interleaved). Click on either image for a larger version.

Image 1. Initial scheduling of service checks (non-interleaved)



Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:51 2000
coffs-405-lj4000	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:53 2000
coffs-415-lj4	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:55 2000
coffs-475-lj4m	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:58 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:00 2000
dbase	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:02 2000
dev	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:05 2000
devone	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:07 2000
se-sds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:09 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:12 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:14 2000
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:16 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:19 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:21 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:23 2000
	Long Term Cache Hits	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:26 2000
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:28 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:30 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:33 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:35 2000
	INSTALL Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:37 2000
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:40 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:42 2000

Image 2. Non-interleaved execution of checks



Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	OK	Tue Mar 28 09:26:52 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 12.10 ms
coffs-405-lj4000	Printer Status	OK	Tue Mar 28 09:26:54 CST 2000	1/3	Printer ok - ("READY")
coffs-415-lj4	Printer Status	OK	Tue Mar 28 09:26:56 CST 2000	1/3	Printer ok - ("00 READY")
coffs-475-lj4m	Printer Status	OK	Tue Mar 28 09:26:59 CST 2000	1/3	Printer ok - ("00 READY")
	PING	WARNING	Tue Mar 28 09:27:01 CST 2000	1/3	PING problem - Packet loss = 0%, RTA = 62.50 ms
dbase	PING	WARNING	Tue Mar 28 09:27:03 CST 2000	1/3	PING problem - Packet loss = 0%, RTA = 85.70 ms
dev	PING	OK	Tue Mar 28 09:27:06 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 1.20 ms
devone	PING	OK	Tue Mar 28 09:27:08 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 0.90 ms
se-sds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:10 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:13 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:15 2000
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:17 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:20 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:22 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:24 2000
	Long Term Cache Hits	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:27 2000
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:29 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:31 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:34 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:36 2000
	INSTALL Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:38 2000
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:41 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:43 2000

Service Interleaving

As discussed above, the inter-check delay helps to equalize the load that NetSaint imposes on the local host. What about remote hosts? Is it necessary to equalize load on remote hosts? Why? Yes, it is important and yes, NetSaint can help out with this. Equalizing load on remote hosts is especially important with the advent of service check parallelization. If you monitor a large number of services on a remote host and the checks were not spread out, the remote host might think that it was the victim of a SYN attack if there were a lot of open connections on the same port. Plus, attempting to equalize the load on hosts is just a nice thing to do...

By giving a value to the service_interleave_factor variable in the main config file, you can modify how the interleave factor is calculated. I will discuss how the "smart" calculation works, as this will probably be the setting you will want to use for normal operation. You can, however, use a pre-set interleave factor instead of having NetSaint calculate one for you. Also of note, if you use an interleave factor of 1, service check interleaving is basically disabled.

When using the "smart" setting of the *service_interleave_factor* variable, NetSaint will calculate an interleave factor by using the following

calculation:

interleave factor = ceil (total number of services / total number of hosts)

Let's take an example. Say you have a total of 1,000 services and 150 hosts that you monitor. NetSaint would calculate the interleave factor to be 7. This means that when NetSaint schedules initial service checks it will schedule the first one it finds, skip the next 6, schedule the next one, and so on... This process will keep repeating until all service checks have been scheduled. Since services are sorted (and thus scheduled) by the name of the host they are associated with, this will help with minimizing/equalizing the load placed upon remote hosts.

The following two images show some output from the status CGI after NetSaint has been started and demonstrate how the interleaving works. For these examples, the inter-check delay was approximately 2.3 seconds and the interleave factor was 5 (there were a total of 113 services and 28 hosts). The first image shows the inital scheduling of service checks with interleaving and the second image shows how NetSaint executes service checks. Notice the differences between these two images and images 1 and 2 above. Click on either image for a larger version.

Image 3. Initial scheduling of service checks (interleaved)

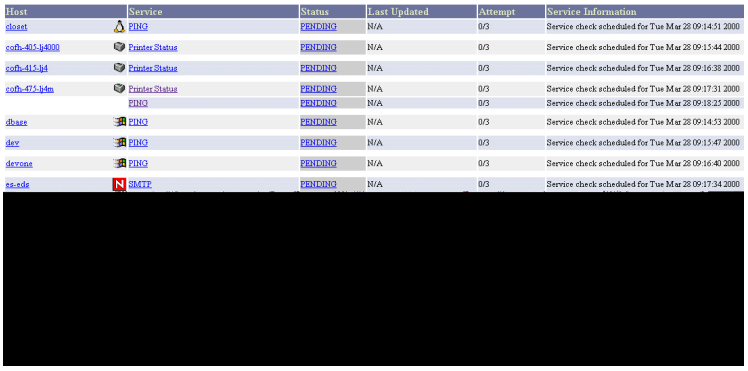


Image 4. Interleaved execution of checks

Maximum Concurrent Service Checks

In order to prevent NetSaint from consuming all of your CPU resources, you can restrict the maximum number of concurrent service checks that can be running at any given time. This is controlled by using the max_concurrent_checks option in the main config file.

The good thing about this setting is that you can regulate NetSaint's CPU usage. The down side is that service checks may fall behind if this value is set too low. When it comes time to execute a service check, NetSaint will make sure that no more than *x* service checks are either being executed or waiting to have their results processed (where *x* is the number of checks you specified for the *max_concurrent_checks* option). If that limit has been reached, NetSaint will postpone the execution of any pending checks until some of the previous checks have completed. So how does one determine a reasonable value for the *max_concurrent_checks* option?

First off, you need to know the following things...

- The inter-check delay that NetSaint uses to initially schedule service checks (use the **-s** command line argument to check this)
- The frequency (in seconds) of service reaper events, as specified by the service_reaper_frequency variable in the main config file.
- A general idea of the average time that service checks actually take to execute (most plugins timeout after 10 seconds, so the average is probably going to be lower)

Next, use the following calculation to determine a reasonable value for the maximum number of concurrent checks that are allowed...

max. concurrent checks = ceil(max(service reaper frequency , average check execution time) / inter-check delay)

The calculated number should provide a reasonable starting point for the *max_concurrent_checks* variable. You may have to increase this value a bit if service checks are still falling behind schedule or decrease it if NetSaint is hogging too much CPU time.

Let's say you are monitoring 875 services, each with an average check interval of 2 minutes. That means that your inter-check delay is going to be 0.137 seconds. If you set the service reaper frequency to be 10 seconds, you can calculate a rough value for the max. number of concurrent checks as follows (I'll assume that the average execution time for service checks is less than 10 seconds) ...

max. concurrent checks = ceil(10 / 0.137)

In this case, the calculated value is going to be 73. This makes sense because (on average) NetSaint are going to be executing just over 7 new service checks per second and it only processes service check results every 10 seconds. That means at given time there will be a just over 70 service checks that are either being executed or waiting to have their results processed. In this case, I would probably recommend bumping the max. concurrent checks value up to 80, since there will be delays when NetSaint processes service check results and does its other work. Obviously, you're going to have test and tweak things a bit to get everything running smoothly on your system, but hopefully this provided some general guidelines...

Time Restraints

The *check_period* option determines the time period during which NetSaint can run checks of the service. Regardless of what status a particular service is in, if the time that it is actually executed is not a vaid time within the time period that has been specified, the check will *not* be executed. Instead, NetSaint will reschedule the service check for the next valid time in the time period. If the check can be run (e.g. the time is valid within the time period), the service check is executed.

Note: Even though a service check may not be able to be executed at a given time, NetSaint may still *schedule* it to be run at that time. This is most likely to happen during the initial scheduling of services, although it may happen in other instances as well. This does *not* mean that NetSaint will execute the check! When it comes time to actually *execute* a service check, NetSaint will verify that the check can be run at the current time. If it cannot, NetSaint will not execute the service check, but will instead just reschedule it for a later time. Don't let this one throw you confuse you! The scheduling and execution of service checks are two distinctly different (although related) things.

Normal Scheduling

In an ideal world you wouldn't have network problems. But if that were the case, you wouldn't need a network monitoring tool. Anyway, when things are running smoothly and a service is in an OK state, we'll call that "normal". Service checks are normally scheduled at the frequency specified by the *check_interval* option. That's it. Simple, huh?

Scheduling During Problems

So what happens when there are problems with a service? Well, one of the things that happens is the service check scheduling changes. If you've configured the *max_attempts* option of the service definition to be something greater than 1, NetSaint will recheck the service before deciding that a real problem exists. While the service is being rechecked (up to *max_attempts* times) it is considered to be in a "soft" state (as described here) and the service checks are rescheduled at a frequency determined by the *retry_interval* option.

If NetSaint rechecks the service *max_attempts* times and it is still in a non-OK state, NetSaint will put the service into a "hard" state, send out notifications to contacts (if applicable), and start rescheduling future checks of the service at a frequency determined by the *check_interval* option.

As always, there are exceptions to the rules. When a service check results in a non-OK state, NetSaint will check the host that the service is associated with to determine whether or not is up (see the note below for info on how this is done). If the host is not up (i.e. it is either down or unreachable), NetSaint will immediately put the service into a hard non-OK state and it will reset the current attempt number to 1. Since the service is in a hard non-OK state, the service check will be rescheduled at the normal frequency specified by the *check_interval* option instead of the *retry_interval* option.

Host Checks

Unlike service checks, host checks are *not* scheduled on a regular basis. Instead they are run on demand, as NetSaint sees a need. This is a common question asked by users, so it needs to be clarified.

One instance where NetSaint checks the status of a host is when a service check results in a non-OK status. NetSaint checks the host to decide whether or not the host is up, down, or unreachable. If the first host check returns a non-OK state, NetSaint will keep pounding out checks of the host until either (a) the maximum number of host checks (specified by the *max_attempts* option in the host definition) is reached or (b) a host check results in an OK state.

Also of note - when NetSaint is check the status of a host, it holds off on doing anything else (executing new service checks, processing other service check results, etc). This can slow things down a bit and cause pending service checks to be delayed for a while, but it is necessary to determine the status of the host before NetSaint can take any further action on the service(s) that are having problems.

Scheduling Delays

It should be noted that service check scheduling and execution is done on a best effort basis. Individual service checks are considered to be

low priority events in NetSaint, so they can get delayed if high priority events need to be executed. Examples of high priority events include log file rotations, external command checks, and service reaper events. Additionally, host checks will slow down the execution and processing of service checks.

Scheduling Example

The scheduling of service checks, their execution, and the processing of their results can be a bit difficult to understand, so let's look at a simple example. Look at the diagram below - I'll refer to it as I explain how things are done.

Image 5.

First off, the $\mathbf{X}_n$ events are service reaper events that are scheduled at a frequency specified by the service_reaper_frequency option in the main config file. Service reaper events do the work of gathering and processing service check results. They serve as the core logic for NetSaint, kicking off host checks, event handlers and notifications as necessary.

For the example here, a service has been scheduled to be executed at time **A**. However, NetSaint got behind in its event queue, so the check was not actually executed until time **B**. The service check finished executing at time **C**, so the difference between points **C** and **B** is the actual amount of time that the check was running.

The results of the service check are not processed immediately after the check is done executing. Instead, the results are saved for later processing by a service reaper event. The next service reaper event occurs at time **D**, so that is approximately the time that the results are processed (the actual time may be later than **D** since other service check results may be processed before this one).

At the time that the service reaper event processes the service check results, it will reschedule the next service check and place it into NetSaint's event queue. We'll assume that the service check resulted in an OK status, so the next check at time **E** is scheduled after the originally scheduled check time by a length of time specified by the *check_interval* option. Note that the service is *not* rescheduled based off the time that it was actually executed! There is one exception to this (isn't there always?) - if the time that the service check is actually executed (point **B**) occurs after the next service check time (point **E**), NetSaint will compensate by adjusting the next check time. This is done to ensure that NetSaint doesn't go nuts trying to keep up with service checks if it comes under heavy load. Besides, what's the point of scheduling something in the past...?

Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:51 2000
cofh-405-lj4000	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:53 2000
cofh-415-lj4	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:55 2000
cofh-475-lj4m	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:58 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:00 2000
dbase	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:02 2000
dev	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:05 2000
devone	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:07 2000
es-eds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:09 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:12 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:14 2000
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:16 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:19 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:21 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:23 2000
	Long Term Cache Hits	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:26 2000
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:28 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:30 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:33 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:35 2000
	INSTALL Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:37 2000
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:40 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:42 2000

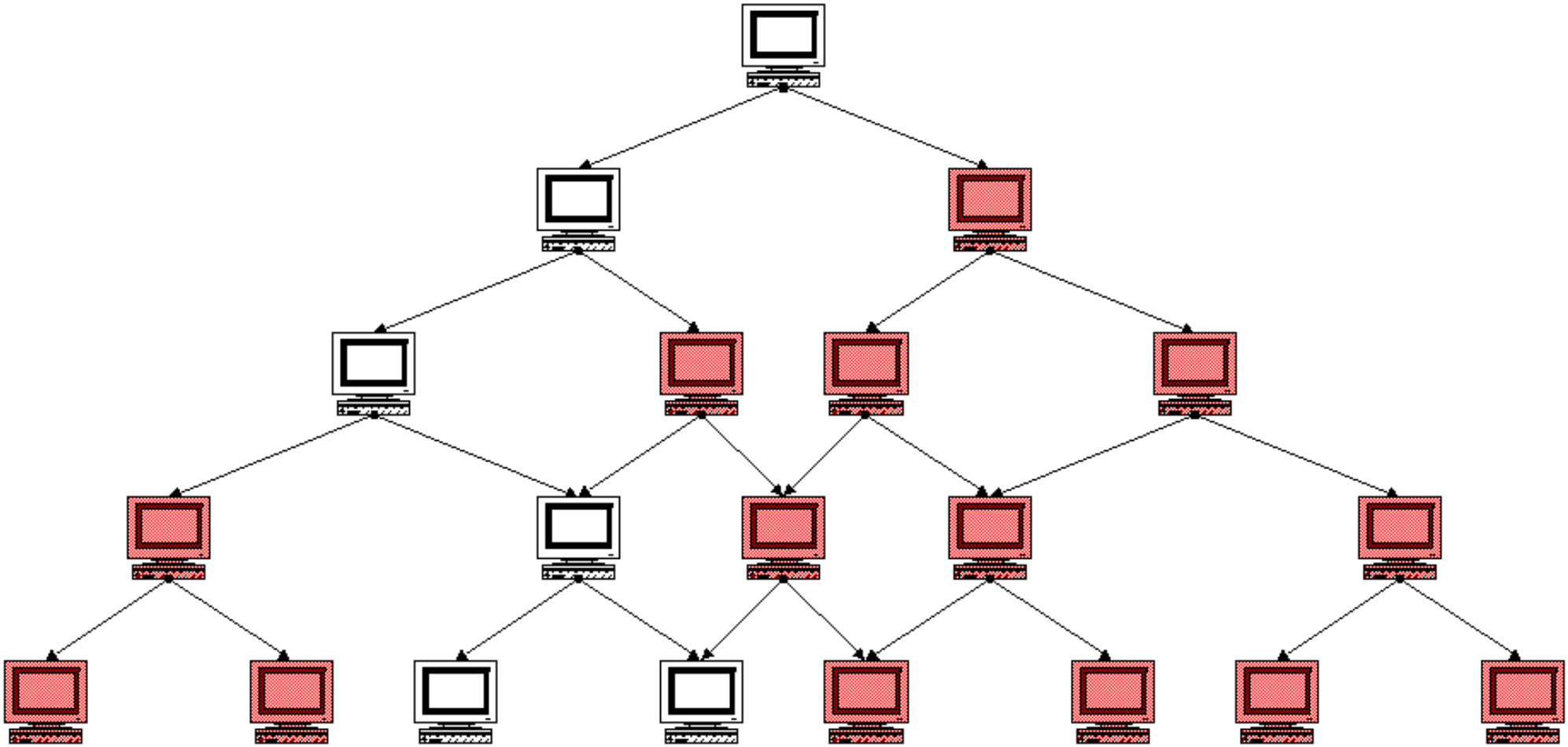
Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	OK	Tue Mar 28 09:26:52 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 12.10 ms
cofh-405-lj4000	Printer Status	OK	Tue Mar 28 09:26:54 CST 2000	1/3	Printer ok - ("READY")
cofh-415-lj4	Printer Status	OK	Tue Mar 28 09:26:56 CST 2000	1/3	Printer ok - ("00 READY")
cofh-475-lj4m	Printer Status	OK	Tue Mar 28 09:26:59 CST 2000	1/3	Printer ok - ("00 READY")
	PING	WARNING	Tue Mar 28 09:27:01 CST 2000	1/3	PING problem - Packet loss = 0%, RTA = 62.50 ms
dbase	PING	WARNING	Tue Mar 28 09:27:03 CST 2000	1/3	PING problem - Packet loss = 0%, RTA = 85.70 ms
dev	PING	OK	Tue Mar 28 09:27:06 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 1.20 ms
devone	PING	OK	Tue Mar 28 09:27:08 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 0.90 ms
es-eds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:10 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:13 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:15 2000
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:17 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:20 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:22 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:24 2000
	Long Term Cache Hits	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:27 2000
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:29 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:31 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:34 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:36 2000
	INSTALL Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:38 2000
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:41 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:27:43 2000

Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:51 2000
cofh-405-lj4000	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:44 2000
cofh-415-lj4	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:38 2000
cofh-475-lj4m	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:31 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:18:25 2000
dbase	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:53 2000
dev	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:47 2000
devone	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:40 2000
es-eds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:34 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:18:27 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:55 2000
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:49 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:42 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:36 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:18:29 2000
	Long Term Cache Hits	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:58 2000
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:51 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:45 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:38 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:18:32 2000
	INSTALL Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:00 2000
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:54 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:47 2000

Host	Service	Status	Last Updated	Attempt	Service Information
closet	PING	OK	Tue Mar 28 09:13:30 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 0.70 ms
cofh-405-lj4000	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:23 2000
cofh-415-lj4	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:17 2000
cofh-475-lj4m	Printer Status	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:10 2000
	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:04 2000
dbase	PING	OK	Tue Mar 28 09:13:32 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 0.30 ms
dev	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:26 2000
devone	PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:19 2000
es-eds	SMTP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:13 2000
	POP3	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:06 2000
	PING	OK	Tue Mar 28 09:13:34 CST 2000	1/3	PING ok - Packet loss = 0%, RTA = 0.20 ms
	IPX PING	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:28 2000
	Processor Load	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:21 2000
	Total Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:15 2000
	Dirty Cache Buffers	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:08 2000
	Long Term Cache Hits	OK	Tue Mar 28 09:13:37 CST 2000	1/3	Long term cache hits = 99%
	LRU Sitting Time	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:30 2000
	Connections	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:24 2000
	SYS Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:16:17 2000
	DC Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:17:11 2000
	INSTALL Volume	OK	Tue Mar 28 09:13:39 CST 2000	1/3	12012 MB (69%) free on volume INSTALL
	USER Volume	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:14:33 2000
	SNMP	PENDING	N/A	0/3	Service check scheduled for Tue Mar 28 09:15:26 2000

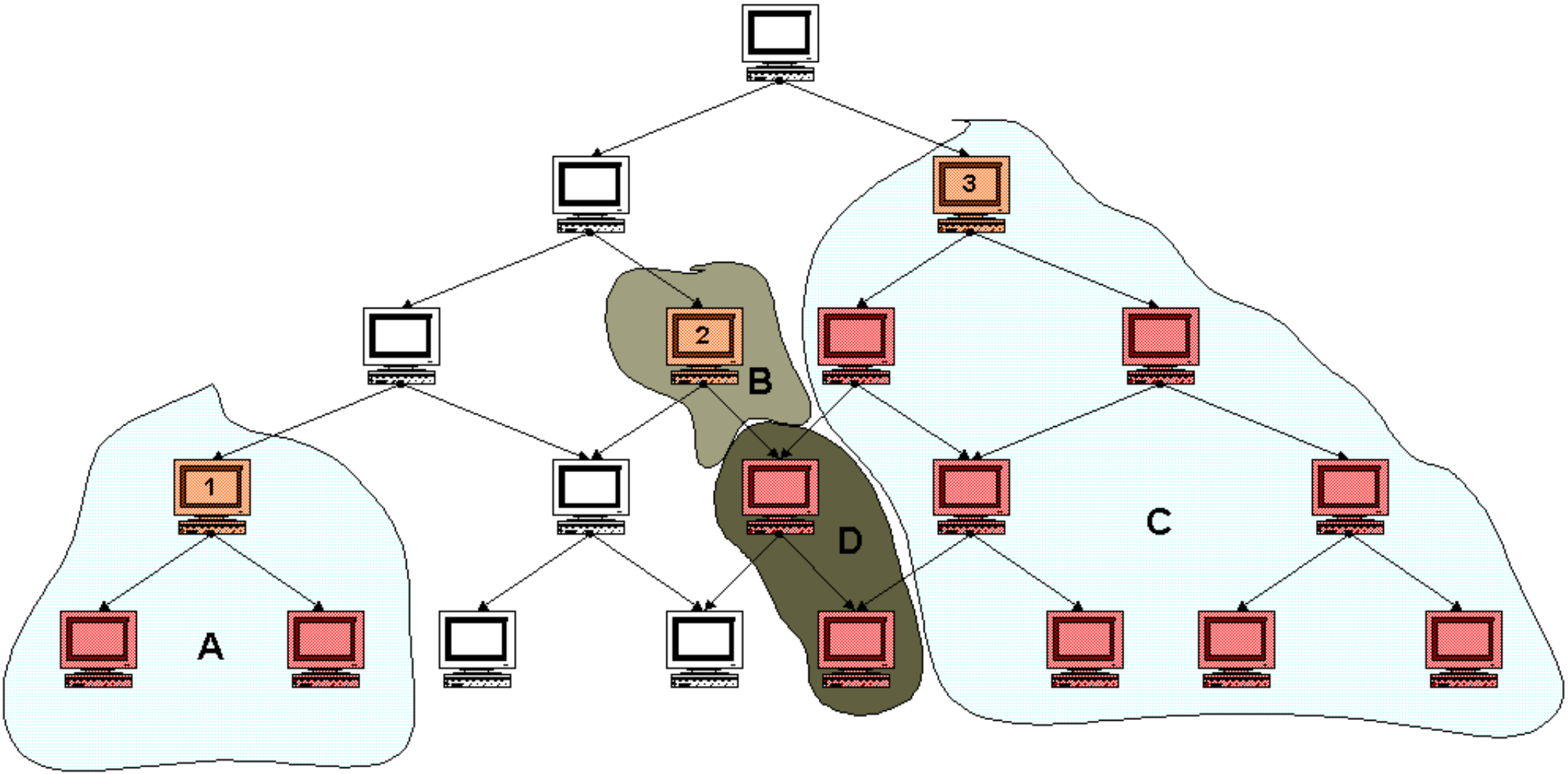
Network Outages

Last Modified 02/26/2000



Cause and Effect Of Network Outages

Last Modified 02/26/2000

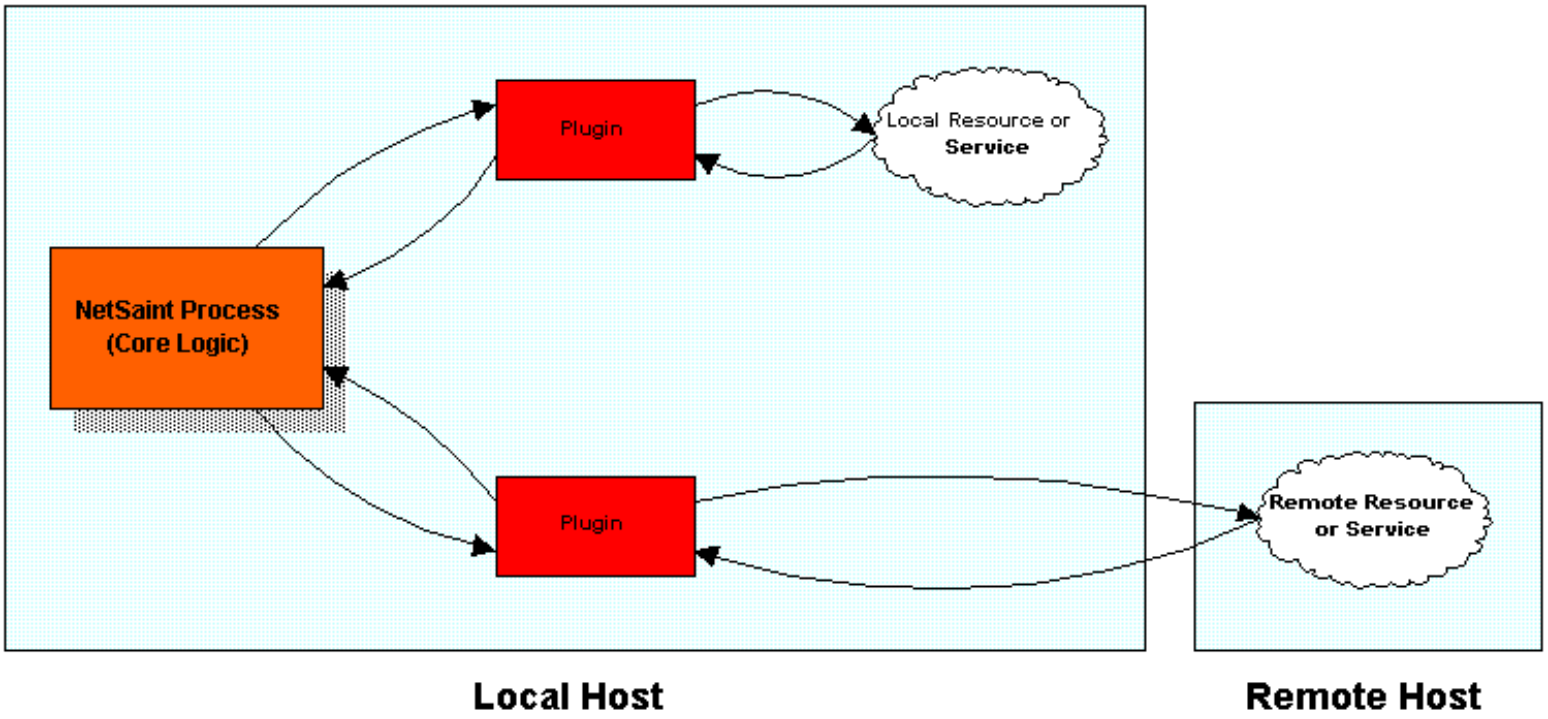


Plugin Theory

Introduction

Unlike many other monitoring tools, NetSaint does not include any internal mechanisms for checking the status of services, hosts, etc. Instead, NetSaint relies on external programs (called plugins) to do the all the dirty work. NetSaint will execute a plugin whenever there is a need to check a service or host that is being monitored. The plugin does *something* (notice the very general term) to perform the check and then simply returns the results to NetSaint. NetSaint will process the results that it receives from the plugin and take any necessary actions (running event handlers, sending out notifications, etc).

The image below show how plugins are separated fromt the core program logic in NetSaint. NetSaint executes the plugins which then check local or remote resources or services of some type. When the plugins have finished checking the resource or service, they simply pass the results of the check back to NetSaint for processing. A more complex diagram on how plugins work can be found in the documentation on passive service checks.



The Upside

The good thing about the plugin architecture is that you can monitor just about anything you can think of. If you can automate the process of checking something, you can monitor it with NetSaint. There are already a lot of plugins that have been created in order to monitor basic resources such as processor load, disk usage, ping rates, etc. If you want to monitor something else, take a look at the documentation on writing plugins and roll your own. Its simple!

The Downside

The only real downside to the plugin architecture is the fact that NetSaint has absolutely no idea what it is that you're monitoring. You could be monitoring network traffic statistics, data error rates, room temperate, CPU voltage, fan speed, processor load, disk space, or the ability of your super-fantastic toaster to properly brown your bread in the morning... As such, NetSaint cannot produce graphs of changes to the exact values of resources you're monitoring over time. It can only track changes in the *state* of those resources. Only the plugins themselves know exactly what they're monitoring and how to perform checks...

Using Plugins For Service Checks

The correlation between plugins and service checks should be fairly obvious. When NetSaint needs to check the status of a particular

service that you have defined, it will execute the plugin you specified in the *<check_command>* argument of the service definition. The plugin will check the status of the service or resource you specify and return the results to NetSaint.

Using Plugins For Host Checks

Using plugins to check the status of hosts may be a bit more difficult to understand. In each host definition you use the *<host_check_command>* argument to specify a plugin that should be executed to check the status of the host. Host checks are not performed on a regular basis - they are executed only as needed, usually when there are problems with one or more services that are associated with the host.

Host checks can use the same plugins as service checks. The only real difference is the important of the plugin results. If a plugin that is used for a host check results in a non-OK status, NetSaint will believe that the host is down.

In most situations, you'll want to use a plugin which checks to see if the host can be pinged, as this is the most common method of telling whether or not a host is up. However, if you were monitoring some kind of super-fantastic toaster, you might want to use a plugin that would check to see if the heating elements turned on when the handle was pushed down. That would give a decent indication as to whether or not the toaster was "alive".

Writing NetSaint Plugins

Plugin development for NetSaint has been moved over to SourceForge. The NetSaint plugin development project page can be found here.

The latest version of the plugin developers guide can be found at
http://netsaintplug.sourceforge.net/doc/developer-guidelines.html

---

Notification Escalations

Introduction

Beginning with release 0.0.6, NetSaint supports *optional* escalation of contact notifications for specific services or hosts within specific hostgroups. I'll explain quickly how they work, although they should be fairly self-explanatory...

Service Notification Escalations

Escalation of service notifications is accomplished by defining service escalation definitions in the host config file. Service escalation definitions are used to escalate notifications for a particular service.

Host Notification Escalations

Escalation of host notifications is accomplished by defining hostgroup escalation definitions in the host config file. Hostgroup escalation definitions are used to escalate host notifications for all hosts in a particular hostgroup. The examples I provide below all use service escalation definitions, but hostgroup escalations work the same way (except for the fact that they are used for host notifications and not service notifications).

When Are Notifications Escalated?

Notifications are escalated *if and only if* one or more escalation definitions matches the current notification that is being sent out. If a host or service notification *does not* have any valid escalation definitions that applies to it, the contact group(s) specified in either the host group or service definition will be used for the notification. Look at the example below:

service[dev]=HTTP;0;24x7;3;5;1;nt-admins;240;24x7;1;1;1;;check_http
serviceescalation[dev;HTTP]=3-5;nt-admins,managers
serviceescalation[dev;HTTP]=6-10;nt-admins,managers,everyone

Notice that there are "holes" in the notification escalation definitions. In particular, notifications 1 and 2 are not handled by the escalations, nor are any notifications beyond 10. For the first and second notification, as well as all notifications beyond the tenth one, the *default* contact groups specified in the service definition are used. In the example above, this would mean that the *nt-admins* contact group would be the only group that was notified during these "holes".

Contact Groups

When defining notification escalations, it is important to keep in mind that any contact groups that were members of "lower" escalations (i.e. those with lower notification number ranges) should also be included in "higher" escalation definitions. This should be done to ensure that anyone who gets notified of a problem *continues* to get notified as the problem is escalated. Example:

service[dev]=HTTP;0;24x7;3;5;1;nt-admins;240;24x7;1;1;1;;check_http
serviceescalation[dev;HTTP]=3-5;nt-admins,managers
serviceescalation[dev;HTTP]=6-0;nt-admins,managers,everyone

The default contact group for the service 'HTTP' on host 'dev' is the group named *nt-admins*. The first (or "lowest") escalation level includes both the *nt-admins* and *managers* contact groups. The last (or "highest") escalation level includes the *nt-admins*, *managers*, and *everyone* contact groups. Notice that the *nt-admins* contact group is included in both escalation definitions. This is done so that they continue to get paged if there are still problems after the first two service notifications are sent out. The *managers* contact group first appears in the "lower" escalation definition - they are first notified when the third problem notification gets sent out. We want the *managers* group to continue to be notified if the problem continues past five notifications, so they are also included in the "higher" escalation definition.

Overlapping Escalation Ranges

Notification escalation definitions can have notification ranges that overlap. Take the following example:

serviceescalation[dev;HTTP]=3-5;nt-admins,managers
serviceescalation[dev;HTTP]=4-0;on-call-support

In the example above:

- The *nt-admins* and *managers* contact groups get notified on the third notification
- All three contact groups get notified on the fourth and fifth notifications
- Only the *on-call-support* contact group gets notified on the sixth (or higher) notification

Recovery Notifications

Recovery notifications are slightly different than problem notifications when it comes to escalations. Take the following example:

serviceescalation[dev;HTTP]=3-5;nt-admins,managers
serviceescalation[dev;HTTP]=4-0;on-call-support

If, after three problem notifications, a recovery notification is sent out for the service, who gets notified? The recovery is actually the fourth notification that gets sent out. However, the escalation code is smart enough to realize that only those people who were notified about the problem on the third notification should be notified about the recovery. In this case, the *nt-admins* and *managers*

contact groups would be notified of the recovery.

Neues in der Version 0.0.6

Hier einige Bereiche, die geändert oder hinzugefügt wurden seit der Version 0.0.5...

Neue Features

1. **Mehrfache Eltern (Parent) Hosts**. Es können nun mehrere Elternhosts in jeder Hostdefinition angegeben werden. Die Reihenfolge, in der die Elternhosts definiert werden hat keinerlei Einfluß auf das Monitoring. Allerdings werden die statusmap und statuswrl CGIs den ersten definierten Elternhost als primären Host zum Zwecke der Darstellung in den Zeichnungen der Hostpläne benutzen.

2. **Passive Dienstüberprüfungen (Service Checks)**. Vor der Version 0.0.6 war der einzige Weg den Status eines beliebigen Dienstes zu überprüfen für NetSaint die aktive Dienstüberprüfung, d.h. die Dienstüberprüfung (Service Check) selbst wurde ausgeführt. In der Version 0.0.6 kann NetSaint auf Resultate von Dienstüberprüfungen externer Programme zugreifen. Externe Programme können Resultate an NetSaint durch den neu hinzugefügten Befehl PROCESS_SERVICE_CHECK_RESULT external command übertragen. NetSaint behandelt und reagiert auf passive Dienstüberprüfungen in der gleichen Art und Weise wie auf 'herkömmliche' aktive Dienstüberprüfungen. Weitere Informationen wie passive Dienstüberprüfungen funktionieren sind hier zu finden.

3. **Flüchtige Dienste (Volatile Services)**. Dienstdefinitionen wurden so erweitert, daß zwischen normalen Diensten und den neuen "flüchtigen" Diensten unterschieden wird. Flüchitge Dienste unterscheiden sich von herkömmlichen Diensten dadurch, daß sie mitgeloggt werden, eine Benachrichtigung auslösen und eine Ereignisbehandlungsroutine gestartet wird *jedesmal*, wenn sie in einem harten, Nicht-OK-Zustand sind und die Dienstüberprüfung den Dienst im selben Nicht-OK-Zustand vorfindet. Flüchtige Dienste sind besonders nützlich für das Überwachen von asynchronen Ereignissen wie SNMP traps (Fallen?) und Sicherheitsalarme. Weitere Informationen wie flüchtige Dienste funktionieren sind hier zu finden.

4. **Benachrichtigungseskalationen (Notification Escalations)**. Zwei neue Definitionstypen wurden der Hostkonfigurationsdatei hinzugefügt um die *optionale* Eskalation der Dienst- und Hostbenachrichtigungen zu unterstützen. Die zwei neue Definitionen sind die Diensteskalation (service escalations) und die Hostgruppeneskalation (hostgroup escalations). Weitere Informationen wie Benachrichtigungsverschärfungen funktionieren sind hier zu finden.

5. **Verteiltes Monitoring**. NetSaint kann nun für verteiltes Monitoring im Netzwerk konfiguriert

werden. Mehr Einzelheiten wie verteiltes Monitoring funktioniert sind hier zu finden.

6. **Netzwerkausfälle**. Ein neues CGI-Skript für Netzwerkausfälle (network outages CGI) wurde hinzugefügt, um die Ursache von Netzwerkausfällen (aus der Sicht von NetSaint) genauer lokalisieren zu können. Weitere Informationen wie das neue CGI-Skript funktioniert ist hier zu finden.

7. **Trends CGI**. Ein neues Trends CGI-Skript wurde hinzugefügt. Es bietet die Möglichkeit eine Grafik der vergangenen Zustände eines beliebigen Hosts oder Dienstes über einen willkürlich gewählten Zeitraum auszugeben. Um vernünftige Ergebnisse zu liefern, geht das CGI-Skript davon aus, daß die Logrotation eingeschaltet wurde und die historischen Logdateien in dem Verzeichnis sind, das in der Variablen log_archive_path definiert wurden.

8. **Sortierung im Status CGI**. Diese Funktion wurde schon seit einiger Zeit gefordert und endlich habe ich es geschafft etwas dafür zu tun. Dienstresultate im Status-CGI (Detailansicht) können nun sortiert werden nach Hostname, Dienstbeschreibung, Zustand, Anzahl der Versuche und Zeit der letzten Überprüfung. Diese Sortierreihenfolge kann auch umgedreht werden. Um eine solche Sortierung durchzuführen muß nur auf die Pfeile im Kopf der Tabelle geklickt werden.

9. **Audioalarm im Status CGI**. Um eine hörbare Benachrichtigung von Problemen zu bekommen kann die Variable audio_alerts in der CGI Konfigurationsdatei gesetzt werden. Es ist möglich verschiedene Klänge für Dienste die sich in einem kritischen, warnenden oder unbekannten Zustand befinden zu definieren. Ebenso auch für Hosts in unerreichbarem oder heruntergefahrenem Zustand. Wenn Klänge für mehrere Alarmzustände definiert wurden spielt NetSaint nur den Klang, der dem kritischsten Problem entspricht.

10. **CGI-Skript benutzen nun Stylesheets**. Jeder hat so sein Idee wie die CGIs aussehen sollten. Deshalb habe ich die meisten Formatanweisungen der CGIs in Stylesheets ausgelagert. Jedes CGI-Skript hat sein eigenes Stylesheet, das nach Belieben verändert werden kann. Es wird mindesten ein 3.0 Browser benötigt um die Stylesheets benutzen zu können - die Ausgabe sieht reichlich langweilig aus ohne jegliche Styledefinitionen. Nebenbei bemerkt, Netscape und IE scheinen beide eine entsetzliche Unterstützung der Stylesheets zu haben wenn es um Tabellen geht. Netscape ist wahrscheinlich noch schlechter als IE, aber beide haben ihre Probleme...

11. **Statuserhaltung nach Neustarts**. Dienst- und Hoststatusinformationen können zwischen Programmstarts gesichert werden. Das ist hilfreich wenn beim Neustart von NetSaint schon Probleme bekannt sind und die dadurch bedingten Benachrichtigungen nicht nochmals verschickt werden sollen. Diese Option erhält Statusinformationen, Pluginausgaben, Zeit der letzten Benachrichtigung und Statusstatistik für Hosts und Dienste. Um diese Informationen

zwischen Programmstarts zu speichern, muß die Variable Statuserhaltung (retain_state_information) gesetzt werden und eine Datei in der Variablen Statuserhaltungsdatei (state_retention_file) definiert werden.

12. **Loggen der anfänglichen Stati**. Anfängliche (initial) Host- und Dienstestati können mitgeloggt werden wenn es erforderlich erscheint. Das ist hilfreich, wenn eine Anwendung benutzt wird, die die Logdatei durchsucht um eine Langzeitstatistik für Hosts und Dienste zu erstellen. Normalerweise werden Zustände (Stati) nur mitgeloggt wenn ein Problem auftritt oder wieder verschwindet (recovery). Das Loggen der Anfangszustände kann aktiviert werden mit der Variablen log_initial_states in der Hauptkonfigurationsdatei.

13. **Bestätigung von Problemen**. Benutzer können nun mit dem extinfo CGI Host- und Dienstprobleme 'bestätigen'. Solche Bestätigungen sind nur möglich, wenn ein Host- oder Dienstproblem erkannt wurde und mindestens eine Benachrichtigung verschickt wurde. Nachdem ein Problem bestätigt wurde, wird dem betroffenen Dienst oder Host ein Kommentar hinzugefügt, eine Bestätigungsbenachrichtigung wird verschickt und alle weiteren Benachrichtigungen werden unterdrückt bis sich der Status des Hosts (oder Dienstes) wieder ändert.

14. **Befehltimeouts**. Zeitüberschreitungen (Timeouts) für Befehle können nun global für Dienstüberprüfungen, Hostüberprüfungen, Ereignisbehandlungsroutinen und Benachrichtigungen definiert werden. Die Werte dafür sind in den Variablen service_check_timeout, host_check_timeout, event_handler_timeout, und command_timeout in der Hauptkonfigurationsdatei anzugeben.

15. **Makroänderungen**. Diese Änderung ist wichtig. This one is important. I've changed the $SERVICESTATE$ And $HOSTSTATE$ macros to reflect the actual state of the service or host during recoveries, instead of setting the macro equal to "RECOVERY". For service recoveries the $SERVICESTATE$ macro is set to "OK" and for host recoveries the $HOSTSTATE$ macro is set to "UP". This was an inconsistency which had been annoying me for a long time, so I decided to change it and be done with it. Make sure to modify any event handlers you have that use the state macros! Also, a new macro ($NOTIFICATIONTYPE$) has been introduced, which can be used to identify what type of notification is being sent out. Values for the macro include "PROBLEM", "RECOVERY", and "ACKNOWLEDGEMENT". The $SUMMARY$ macro has been removed - at some point it stopped working and I just decided to kill it off. The $OUTPUT$ macro can now be used in host notifications as well as service notifications. When the $OUTPUT$ macro is used in host notifications, it will contain the text returned from the host check command. Lastly, user macros ($USERn$) can now be defined in an optional resource file and can be used to hide information like usernames and passwords, or to store information commonly used in command definitions (like directory paths). Weiter Informationen zu Makros sind hier zu finden.

16. **Änderung des Pfades der CGI-Konfigurationsdatei**. Die CGI-Skripte erwarten die CGI-Konfigurationsdatei (nscgi.cfg) im selben Verzeichnis wie die Haupt- und Hostkonfigurationsdatei *(normalerweise /usr/local/netsaint/etc)*. Dies wurde geändert um die Dinge einheitlicher zu gestalten und die Erzeugung vom RPMs zu vereinfachen.

17. **Enwicklerdokumentation**. Ich habe einen neuen Abschnitt zur Dokumentation hinzugefügt für Entwickler die Schnittstellen zu anderen Produkten entwickeln oder die internen Fähigkeiten von NetSaint voll ausschöpfen (die bislang noch nicht durch Konfigurationsdateien beeinflusst werden können) wollen. Dokumentation ist vorhanden zu den Formaten der verschiedenen Dateien die NetSaint benutzt, ebenso wie zu internen Funktionen mit denen die Fähigkeit NetSaint's Konfigurationsdaten zu lesen oder zu speichern erweitert werden kann. Ich werde diese Information aktuell halten über die verschiedenen Versionen von NetSaint hinweg. Die Entwicklerdokumentation (in englischer Sprache) ist hier zu finden.

18. **Interne Verbesserungen**. Eine Menge interner Quellcode im Hauptprogramm und in den CGIs wurde überarbeitet. Benutzer werden keinen Unterschied erkennen, aber es vereinfacht die Arbeit mit dem Code. Einige Änderungen betreffen statische Buffer in den Datenstrukturen, die nun dynamisch zugewiesenen Speicher benutzen; eine Verbesserung des internen Loggingcodes und vom Kern und den CGIs gemeinsam benutzten Datenstrukturen und Funktionen.

19. **Die übliche Fehlereliminierung**. Wäre eine neue Version fertig ohne die bekannten Fehler der vorherigen Version auszumerzen?

NetSaint Developer Documentation

Version 0.0.6
Last Updated: July 30th, 2000

Plugin Development

- Plugin theory
- Guidelines for plugin development

Standard File Formats

- Status file
- Comment file
- State retention file

External Data API

- Overview
- External status data (XSD) overview
- External retention data (XRD) overview
- External comment data (XCD) overview
- External extended data (XED) overview
- External object data (XOD) overview

Status File Format

Introduction

In order to give external applications (such as the CGIs) access to the current host and service status information in NetSaint, all status information is saved to the file specified by the status_file option in the main config file. External applications can read the contents of this file to determine the current status of any monitored host or service. External applications *should not* write anything to the status file. NetSaint does not read the status file to determine current service and host information - it is simply provided as a means for third-party apps to access the internal status information in an easy manner.

File Format

The status file contains three types of entries: a program entry, one or more host status entries, and one or more service status entries. The format for each type of entry it described below.

Program Entry Format:

[<timestamp>] PROGRAM;<start_time>;<daemon_mode>;<program_mode>;<last_mode_change>;<last_command_check>;<last_log_rotation>;<executing_service_checks>;<accept_passive_service_checks>;<enable_event_handlers>;<obsess_over_services>

where...

- ***timestamp*** is the time in time_t format (seconds since UNIX epoch) that the program entry was last updated.
- ***start_time*** is the time in time_t format (seconds since UNIX epoch) that NetSaint was last (re)started.
- ***daemon_mode*** in an integer that indicates whether or not NetSaint is running as a daemon. If this value is 1, NetSaint is running in daemon mode. If this value is 0, NetSaint is running as a normal (foreground or background) process.
- ***program_mode*** a string which identifies what program mode NetSaint is currently in. If this string is "ACTIVE", NetSaint is in active mode. If this string is "STANDBY", NetSaint is in standby mode.
- ***last_mode_change*** is the time in time_t format (seconds since UNIX epoch) when the last program mode change occurred.
- ***last_command_check*** is the time in time_t format (seconds since UNIX epoch) that NetSaint last checked for external commands. A value of zero means that NetSaint has not checked for external commands since it was last (re)started.
- ***last_log_rotation*** is the time in time_t format (seconds since UNIX epoch) that NetSaint last rotated the main log file. A value of zero means that the log file has not been rotated since NetSaint was last (re)started.
- ***execute_service_checks*** in an integer that indicates whether or not NetSaint is actively executing service checks. Values: 0=checks are *not* being executed, 1=checks are being executed.
- ***accept_passive_service_checks*** in an integer that indicates whether or not NetSaint is accepting passive service checks. Values: 0=passive service checks are *not* being accepted, 1=passive checks are being accepted.
- ***enable_event_handlers*** in an integer that indicates whether or not host and service event handlers are enabled. Values: 0=event handlers are *not* enabled, 1=event handlers are enabled.
- ***obsess_over_services*** in an integer that indicates whether or not is running "obsessing" over service check results and running a obsessive service check processor command. Values: 0=Netsiant is *not* obsessing, 1=NetSaint is obsessing.

Host Status Format:

[<timestamp>] HOST; <host_name>;<state>;<last_state_change>;<problem_has_been_acknowledged>;<time_up>;<time_down>;<time_unreachable>;<last_notification>;<current_notification_number>;<notifications_enabled>;<event_handler_enabled>;<checks_enabled>;<plugin_output>

where...

- ***timestamp*** is the time in time_t format (seconds since UNIX epoch) that the host was last checked (or its current state was assumed).
- ***host_name*** is the short name of the host (as defined in the host configuration file) that the state information corresponds to.
- ***state*** is a string that indicates the current state of the host. Values include "PENDING", "UP", "DOWN", and "UNREACHABLE".
- ***last_state_change*** is the time in time_t format (seconds since UNIX epoch) that the host last experienced a hard state change.
- ***problem_has_been_acknowledged*** is an integer indicating whether or not this host problem has been acknowledged. If the host is UP, or it is DOWN or UNREACHABLE and has not been acknowledged, this is set to 0. If this host is DOWN or UNREACHABLE and the problem has been acknowledged, this is set to 1.
- ***time_up*** is the number of seconds (since monitoring began) that the host has been in an UP state.
- ***time_down*** is the number of seconds (since monitoring began) that the host has been in a DOWN state.
- ***time_unreachable*** is the number of seconds (since monitoring began) that the host has been in an UNREACHABLE state.
- ***last_notification*** is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the last notification for this host was sent out. If no notifications have been sent out (or if the host is UP), this value is set to zero.
- ***current_notification_number*** is an integer representing the number of notifications that have been sent out about this host problem. If no notifications have been sent out since the host last changed state (of if it is in an UP state), this value is set to zero.
- ***notifications_enabled*** is an integer that indicates whether or not notifications for this host are enabled. Values: 0=notifications are *not* enabled, 1=notifications are enabled.
- ***event_handler_enabled*** is an integer that indicates whether or not the event handler for this host are enabled. Values: 0=event handler is *not* enabled, 1=event handler is enabled.
- ***checks_enabled*** is an integer that indicates whether or not checks this host are enabled. Values: 0=checks are *not* enabled, 1=checks are enabled.
- ***plugin_output*** is the output from the last host check (text)

Service Status Format:

[<timestamp>] SERVICE;
<host_name>;<svc_description>;<state>;<current_attempt>/<max_attempts>;<state_type>;<next_check>;<check_type>;<checks_enabled>;<passive_checks_accepted>;<last_state_change>;<problem_has_been_acknowledged>;<last_hard_state>;<time_ok>;<time_unknown>;<time_warning>;<time_critical>;<last_notification>;<current_notification_number>;<notifications_enabled>;<check_latency>;<execution_time>;<plugin_output>

where...

- ***timestamp*** is the time in time_t format (seconds since UNIX epoch) that the service was last checked.
- ***host_name*** is the short name of the host that this service is associated with.
- ***svc_description*** is the description of the service (as defined in the host configuration file) that the state information corresponds to. Together, the *host_name* and *svc_description* fields uniquely identify a service definition.
- ***state*** is string indicating the current state of the service. Values include "OK", "UNKNOWN", "WARNING", "CRITICAL", "RECOVERY", "UNREACHABLE", and "HOST DOWN". A value of "RECOVERY" indicates that the service is in an OK state, but just recovered from a non-OK state. Values of "UNREACHABLE" and "HOST DOWN" indicate that the host that the service is associated with is either down or unreachable.
- ***current_attempt*** is an integer representing the current service check attempt number. This value will be set to 1 if the host that the service is associated with is either down or unreachable.
- ***max_attempts*** is an integer representing the maximum number of check attempts for this service.
- ***state_type*** is a string indicating what type of state the service is currently in. Values include "SOFT" and "HARD".
- ***next_check*** is the time in time_t format (seconds since UNIX epoch) that the service is next scheduled to be checked.
- ***check_type*** is a string indicating what type of service check this was. Values include "ACTIVE" and "PASSIVE".
- ***checks_enabled*** is an integer representing whether or not checks for this service are enabled. Values: 0=checks are *not* enabled, 1=checks are enabled.
- ***accept_passive_checks*** is an integer representing whether or not passive checks are being accepted for this service. Values: 0=passive checks are *not* being accepted, 1=passive checks are being accepted.
- ***event_handler_enabled*** is an integer representing whether or not the event handler for this service is enabled. Values: 0=event handler is *not* enabled, 1=event handler is enabled.
- ***passive_checks_accepted*** is an integer representing whether or not passive checks are being accepted for this service. If this value is 1, they are being accepted. If this value is 0, passive checks are not being accepted.
- ***last_state_change*** is the time in time_t format (seconds since UNIX epoch) that the service last had a hard state change.
- ***problem_has_been_acknowledged*** is an integer indicating whether or not this service problem has been acknowledged. If the service is in an OK state, or it is in a non-OK state and has not been acknowledged, this is set to 0. If this service is in a non-0K state and the problem has been acknowledged, this is set to 1.
- ***last_hard_state*** is a string that indicates the last hard state of the service. Values include "OK", "UNKNOWN", "WARNING", and "CRITICAL".
- ***time_ok*** is the number of seconds that the service has been in an OK state.
- ***time_warning*** is the number of seconds that the service has been in a WARNING state.
- ***time_unknown*** is the number of seconds that the service has been in an UNKNOWN state.
- ***time_critical*** is the number of seconds that the service has been in a CRITICAL state.
- ***last_notification*** is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the last notification for this service was sent out. If no notifications have been sent out or if the service is currently in an OK state, this value is set to zero.
- ***current_notification_number*** is an integer representing the number of notifications that have been sent out about this service problem. If no notifications have been sent out since the service last changed state (of if it is in an OK state), this value is set to zero.
- ***notifications_enabled*** is an integer that indicates whether or not notifications for this service have been enabled. Values: 0=notifications are *not* enabled, 1=notifications are enabled.
- ***check_latency*** is an integer indicating the number of seconds that the service check lagged behind its scheduled execution time (actual time of execution - scheduled time of execution = latency)
- ***execution_time*** is an integer indicating the number of seconds that this service check took to execute
- ***plugin_output* is the output from the last service check (text)**

Comment File Format

Introduction

In order to help share information among administrators, techs, etc., NetSaint allows comments to be added to all hosts and services that are being monitored. The comments are stored in the file specified by the comment_file directive in the main configuration file.

It should be noted that NetSaint "cleans" the comment file each time it restarts. During the cleaning process, NetSaint will remove all comments that are not marked as being persistent or that do not correspond to valid hosts or services that you have defined, and it will re-number all comment IDs.

Adding Comments

If you wish to use or write an external application that adds comments to hosts or services, you should *not* write comments directly to the comment file. Instead, use the ***ADD_SVC_COMMENT*** and ***ADD_HOST_COMMENT*** external commands. The commands should be written to the external command file. NetSaint will periodically scan the external command file and process any commands it finds in there.

Deleting Comments

Similiarly, if you want to delete one or more comments from the command file, use the ***DEL_SVC_COMMENT***, ***DEL_HOST_COMMENT***, ***DEL_ALL_SVC_COMMENTS***, or ***DEL_ALL_HOST_COMMENTS*** external commands. Do *not* modify the contents of the comment file yourself!

File Format

The comment file contains two types of entries: host comments and service comments. The format for each type of comment it described below.

Host Comment Format:

[<timestamp>] HOST_COMMENT;<id>;<host_name>;<persistent>;<author>;<comment>

where...

- *timestamp* is the time in time_t format (seconds since UNIX epoch) that the comment was entered by the user.
- *id* is a comment identification number which is unique among other host and service comments. This number is generated by NetSaint and cannot be specified by the user.

- *host_name* is the short name of the host (as defined in the host configuration file) that the comment is associated with.
- *persistent* is a flag which indicated whether the comment is persistent or not. Persistent comments survive program restarts, while non-persistent comments are deleted when NetSaint is restarted. A value of 0 indicates that the comment is non-persistent, while a value of 1 indicates that it is persistent.
- *author* is a text field that contains the name of the person who entered the comment.
- *comment* is a text field that contains the actual comment.

Service Comment Format:

[<timestamp>]
SERVICE_COMMENT;<id>;<host_name>;<svc_description>;<persistent>;<author>;<comment>

where...

- *timestamp* is the time in time_t format (seconds since UNIX epoch) that the comment was entered by the user.
- *id* is a comment identification number which is unique among other host and service comments. This number is generated by NetSaint and cannot be specified by the user.
- *host_name* is the short name of the host that the service is associated with.
- *svc_description*is the description of the service (as defined in the host configuration file) that the comment is associated with. Together the *host_name* and *svc_description* uniquely identiry a particular service.
- *persistent* is a flag which indicated whether the comment is persistent or not. Persistent comments survive program restarts, while non-persistent comments are deleted when NetSaint is restarted. A value of 0 indicates that the comment is non-persistent, while a value of 1 indicates that it is persistent.
- *author* is a text field that contains the name of the person who entered the comment.
- *comment* is a text field that contains the actual comment.

State Retention File Format

Introduction

In order to preserve host and service state information (current status, state time statistics, etc.) between program restarts, users can opt to enable the state retention feature by using the retain_state_information option in the main config file. If this option is enabled, state retention information is stored in the file specified by the state_retention_file directive in the main configuration file. Immediately before shutting down (or restarting) NetSaint will write all current host and service state information to the retention file. Upong restarting, NetSaint will read the information stored in the retention file, initialize host and service information, and delete the file.

At any time while NetSaint is running, you can have it save service and host state information, by using the *SAVE_STATE_INFORMATION* external command. You can also force NetSaint to read in previously save state information by using the *READ_STATE_INFORMATION* command, although this is not recommend, as the current state information that NetSaint has will be replaced with whatever is stored in the state retention file.

It should be noted that NetSaint will only save state information for service and hosts that have been checked at the time the file is written. Also, NetSaint will only save the last hard state for the host or service.

File Format

The state retention file contains four types of entries: a creation timestamp, program state information, host state information and service state information. The format for each type of entry it described below.

Creation Time Format:

CREATED: <timestamp>

where...

- *timestamp* is the time in time_t format (seconds since UNIX epoch) that the state information was saved.

Program Information Format:

PROGRAM: <program_mode>;<execute_service_checks>;<accept_passive_service_checks>;<enable_event_handlers>;<obsess_over_services>

where...

- *program_mode* is an integer that represents the last program mode that NetSaint was in. Values: 0=standby mode, 1=active mode.
- *execute_service_checks* is an integer indicating whether or not service checks were being executed when NetSaint was running. Values: 0=checks were *not* being executed, 1=checks were being executed.
- *accept_passive_service_checks* is an integer indicating whether or not passive service checks were being accepted when NetSaint was running. Values: 0=passive checks were *not* being accepted, 1=passive checks were being accepted.
- *enable_event_handlers* is an integer indicating whether or not host and service event handlers were enabled when NetSaint was running. Values: 0=event handlers were *not* enabled, 1=event handlers were enabled.
- *obsess_over_services* is an integer indicating whether or not NetSaint was obsessing over service checks when it was running. Values: 0=NetSaint was *not* obsessing, 1=NetSaint was obsessing.

Host Information Format:

HOST: <host_name>;<state>;<last_check>;<checks_enabled>;<time_up>;<time_down>;<time_unreachable>;<last_notification>;<current_notification_number>;<current_notification_number>;<notifications_enabled>;<event_handler_enabled>;<problem_has_been_acknowledged>;<plugin_output>

where...

- *host_name* is the short name of the host (as defined in the host configuration file) that the state information corresponds to.
- *state* is an integer corresponding to the state of the host (UP, DOWN, or UNREACHABLE). See the *base/netsaint.h* file for the integer values of different states.
- *last_check* is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the host status was last checked.
- *checks_enabled* is an integer indicating whether or not checks of this host have been enabled. Values: 0=checks have been disabled, 1=checks are enabled.
- *time_up* is the number of seconds that the host has been in an UP state.
- *time_down* is the number of seconds that the host has been in a DOWN state.
- *time_unreachable* is the number of seconds that the host has been in an UNREACHABLE state.
- *last_notification* is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the last notification for this host was sent out. If no notifications have been sent out, this value is set to zero.
- *current_notification_number* is an integer representing the number of notifications that have been sent out about this host problem. If no notifications have been sent out since the host last changed state (of if it is in an UP state), this value is set to zero.
- *notifications_enabled* is an integer that indicates whether or not notifications for this host have been enabled. Values: 0=notifications have been disabled, 1=notifications are enabled.
- *event_handler_enabled* is an integer indicating whether or not the event handler for this host has been enabled. Value: 0=event handler has been disabled, 1=event handler is enabled.
- *problem_has_been_acknowledged* is an integer indicating whether or not this host problem has been acknowledged. If the host is UP, or it is DOWN or UNREACHABLE and has not been acknowledged, this is set to 0. If this host is DOWN or UNREACHABLE and the problem has been acknowledged, this is set to 1.
- *plugin_output* is the output from the last host check (text)

Service Information Format:

SERVICE:
<host_name>;<svc_description>;<state>;<last_check>;<time_ok>;<time_warning>;<time_unknown>;<time_critical>;<last_notification>;<current_notification_number>;<notifications_enabled>;<checks_enabled>;<accept_passive_checks>;<event_handler_enabled>;<problem_has_been_acknowledged>;<plugin_output>

where...

- *host_name* is the short name of the host that this service is associated with.
- *svc_description* is the description of the service (as defined in the host configuration file) that the state information corresponds to. Together, the *host_name* and *svc_description* fields uniquely identify a service definition.
- *state* is an integer corresponding to the state of the state (OK, WARNING, UNKNOWN, or CRITICAL). See the *base/netsaint.h* file for the exact values of different states.
- *last_check* is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the service status was last checked.
- *time_ok* is the number of seconds that the service has been in an OK state.
- *time_warning* is the number of seconds that the service has been in a WARNING state.
- *time_unknown* is the number of seconds that the service has been in an UNKNOWN state.
- *time_critical* is the number of seconds that the service has been in a CRITICAL state.
- *last_notification* is a timestamp in time_t format (number of seconds since UNIX epoch) that indicates when the last notification for this service was sent out. If no notifications have been sent out, this value is set to zero.

- *current_notification_number* is an integer representing the number of notifications that have been sent out about this host problem. If no notifications have been sent out since the host last changed state (of if it is in an UP state), this value is set to zero.
- *notifications_enabled* is an integer that indicates whether or not notifications for this service have been enabled. Values: 0=notifications have been disabled, 1=notifications are enabled.
- *checks_enabled* is an integer that indicates whether or not checks of this service have been enabled. Values: 0=checks have been disabled, 1=checks are enabled.
- *accept_passive_checks* is an integer representing whether or not passive checks are being accepted for this service. If this value is 1, they are being accepted. If this value is 0, passive checks are not being accepted.
- *event_handler_enabled* is an integer indicating whether or not the event handler for this service has been enabled. Value: 0=event handler has been disabled, 1=event handler is enabled.
- *problem_has_been_acknowledged* is an integer indicating whether or not this service problem has been acknowledged. If the service is in an OK state, or it is in a non-OK state and has not been acknowledged, this is set to 0. If this service is in a non-0K state and the problem has been acknowledged, this is set to 1.
- *plugin_output* is the output from the last service check (text)

External Data API Overview

Introduction

Ever since the first release of NetSaint, I've been asked if I had plans to support MySQL, PostgresQL, Oracle, or a slew of other data sources. In order to support alternate data stores for things like configuration data, status data, etc. I decided to change a lot of the underlying code in both the core program and CGIs to allow developers to add their own data I/O routines...

One of the major changes that was made in version 0.0.6 was a complete rewrite of the data I/O routines in both the core program and the CGIs. These revisions resulted in the creation of an abstraction layer that separated implementation-specific code for accessing data sources from the data processing routines present in the core and CGIs. The end result of this is a set of external data APIs that allows developers to easily write code to support different types of data sources in the core program and CGIs.

NetSaint 0.0.7 will make use of the new external data APIs to optionally provide native MySQL database support for status, retention, comment, extended, and object data. Developers should be able to write their own routines for providing support for other data sources (PostgresQL and Oracle databases, LDAP servers, etc.) fairly easily.

The reset of this documentation is provided to give developers an overview of how the APIs work and how to write their own code to support other data sources than those already natively provided by NetSaint.

External data types

With the exception of standard logging (to the log file or syslog facility), there are five different types of external data that the core program and/or the CGIs use:

1. **Object data** - This consists of object definitions (host, services, contact, contact groups, commands, etc) that are used by both the core program and CGIs. Basically everything that can be defined in the standard host config file(s)...
2. **Comment data** - This consists of host and service comments which are processed by the core program and available for display in the CGIs. By default, comments are stored in file specified by the comment_file directive.
3. **Extended data** - This consists of optional information that is used by the CGIs when displaying information about specific services and host. Currently, only extended data for hosts is supported. One example of extended data for a host is the graphics associated with it (i.e. icons). By default, extended information can be specified for particular hosts by using hostextinfo definitions in the CGI config file.
4. **Status data** - This consists of current program, host, and service status information which is

made available by the core program and used by the CGIs. By default, status data is stored in the file specified by the status_file directive.

5. **Retention data** - This consists of saved program, host, and service status information that is used by the core program and should be retained across program restarts. By default, retention data is stored in the file specified by the state_retention_file directive.

Access to external data

The following table outlines what type of access the core program and CGIs have to each type of external data:

	Core Program	CGIs
Object data	Read	Read
Comment data	Read/Write	Read
Extended data	-	Read
Status data	Write	Read
Retention data	Read/Write	-

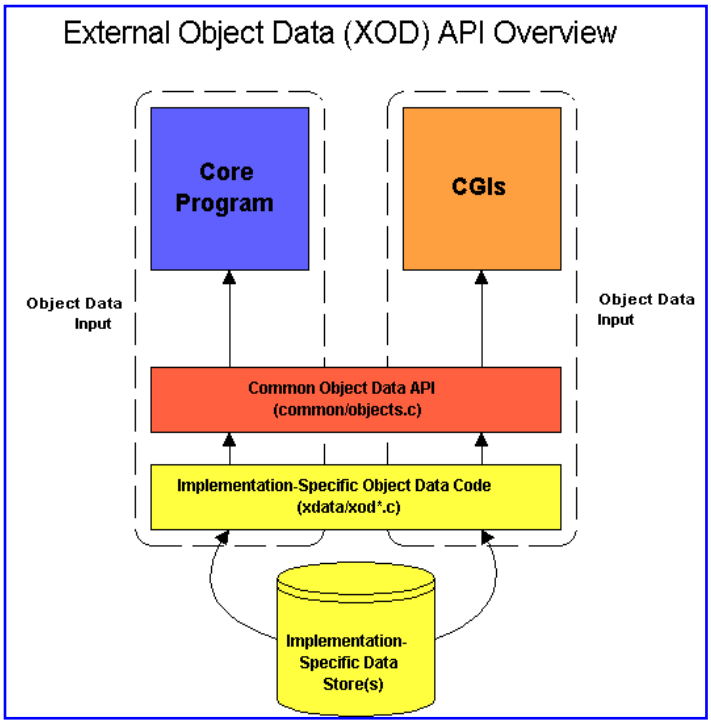
API details

For more information on how the various types of external data APIs work, as well as information on writing your own I/O routines for external data in the core program and CGIs, click on one of the links below...

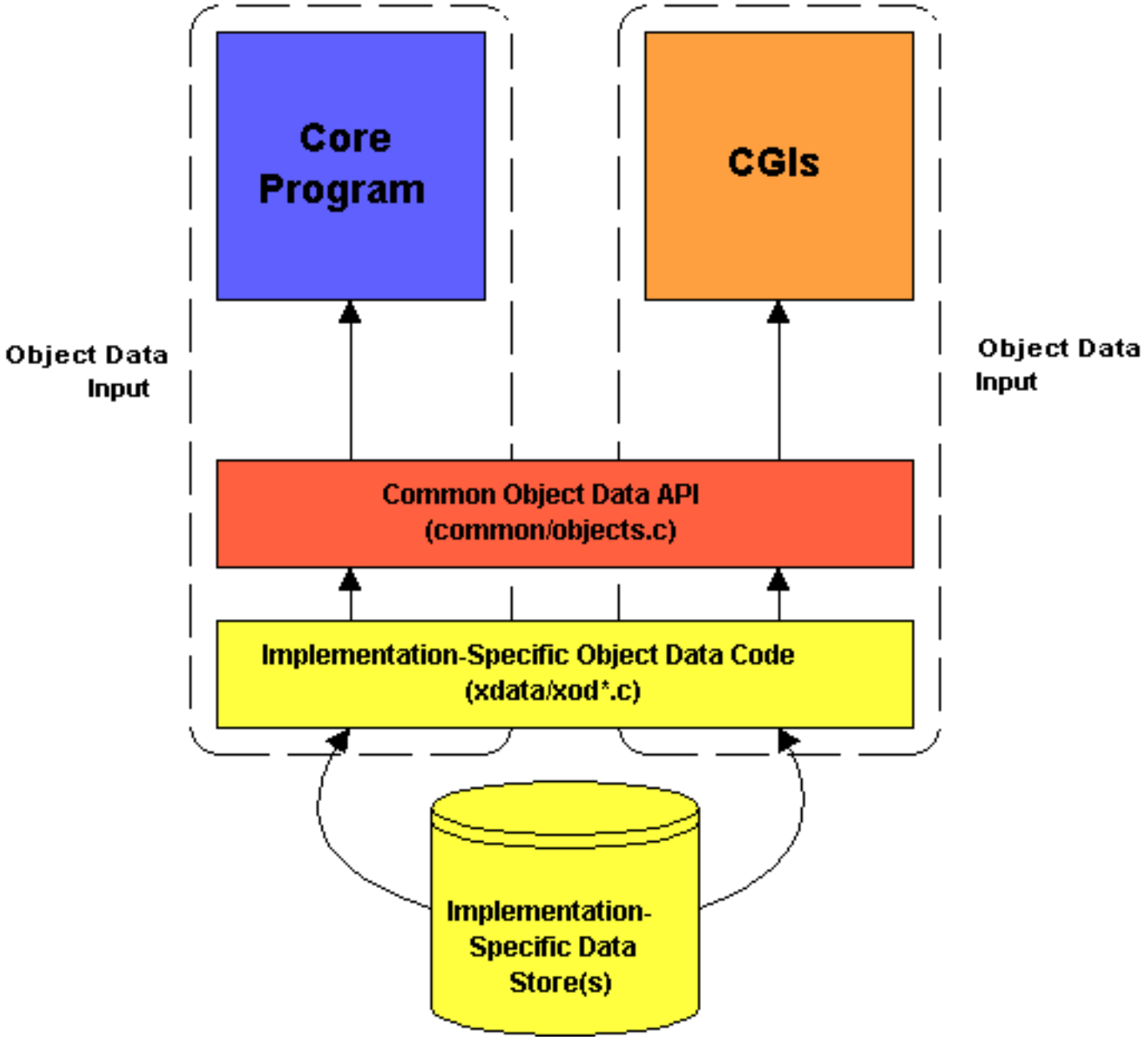
- External object data (XOD) API
- External comment data (XCD) API
- External extended data (XED) API
- External status data (XSD) API
- External retention data (XRD) API

External Object Data (XOD) Overview

Nothing here yet... Wait for an alpha version of 0.0.7

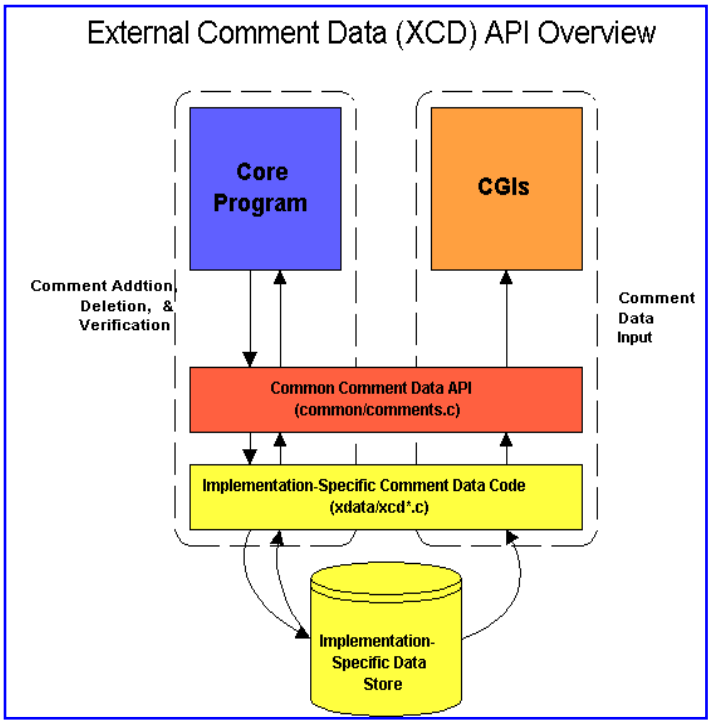


External Object Data (XOD) API Overview

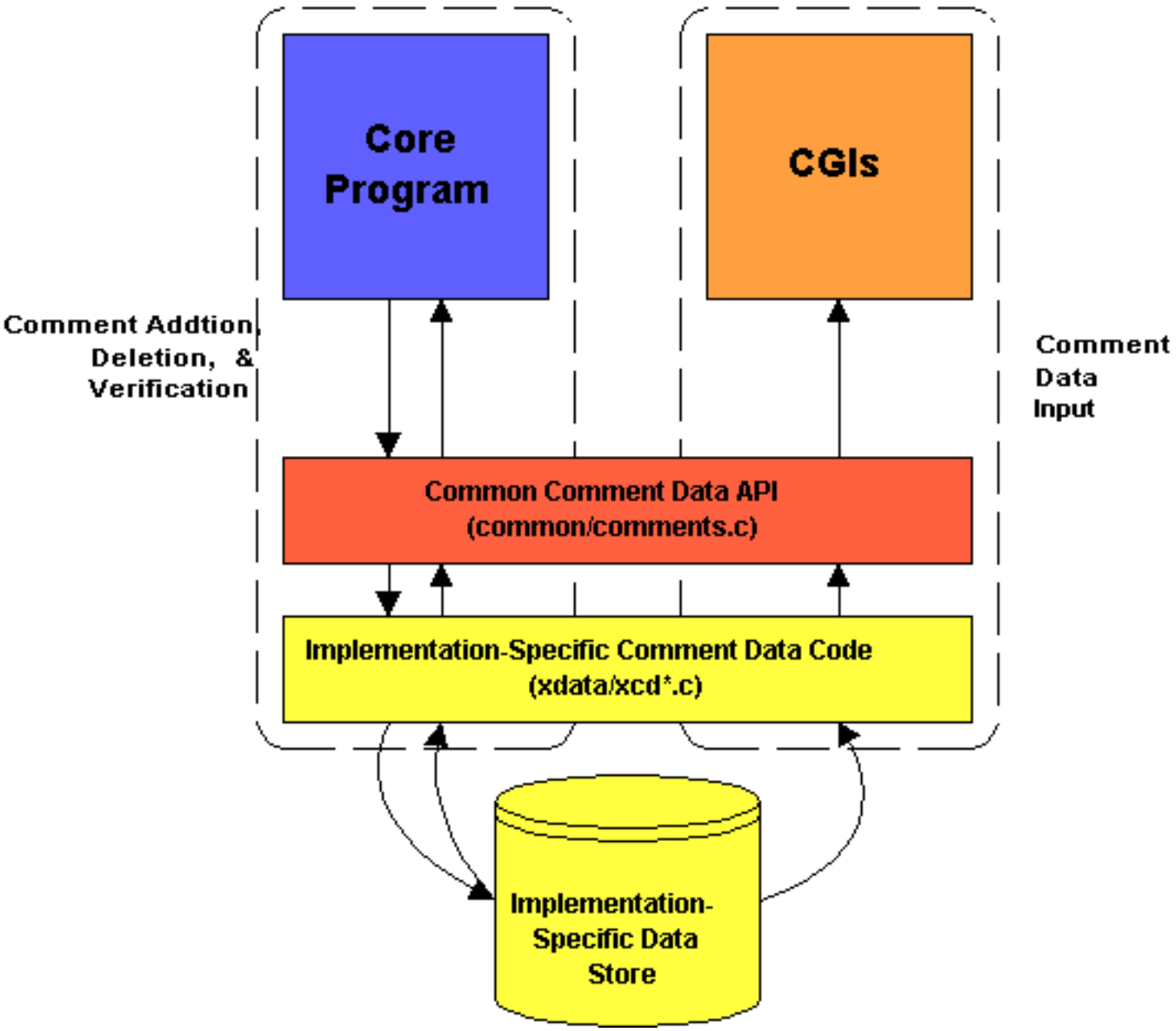


External Comment Data (XCD) Overview

Nothing here yet... Wait for an alpha version of 0.0.7

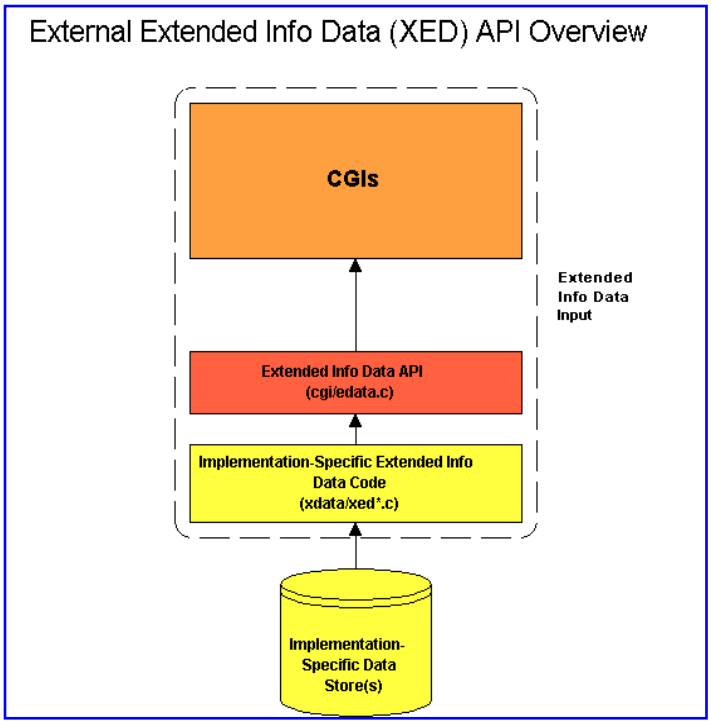


External Comment Data (XCD) API Overview

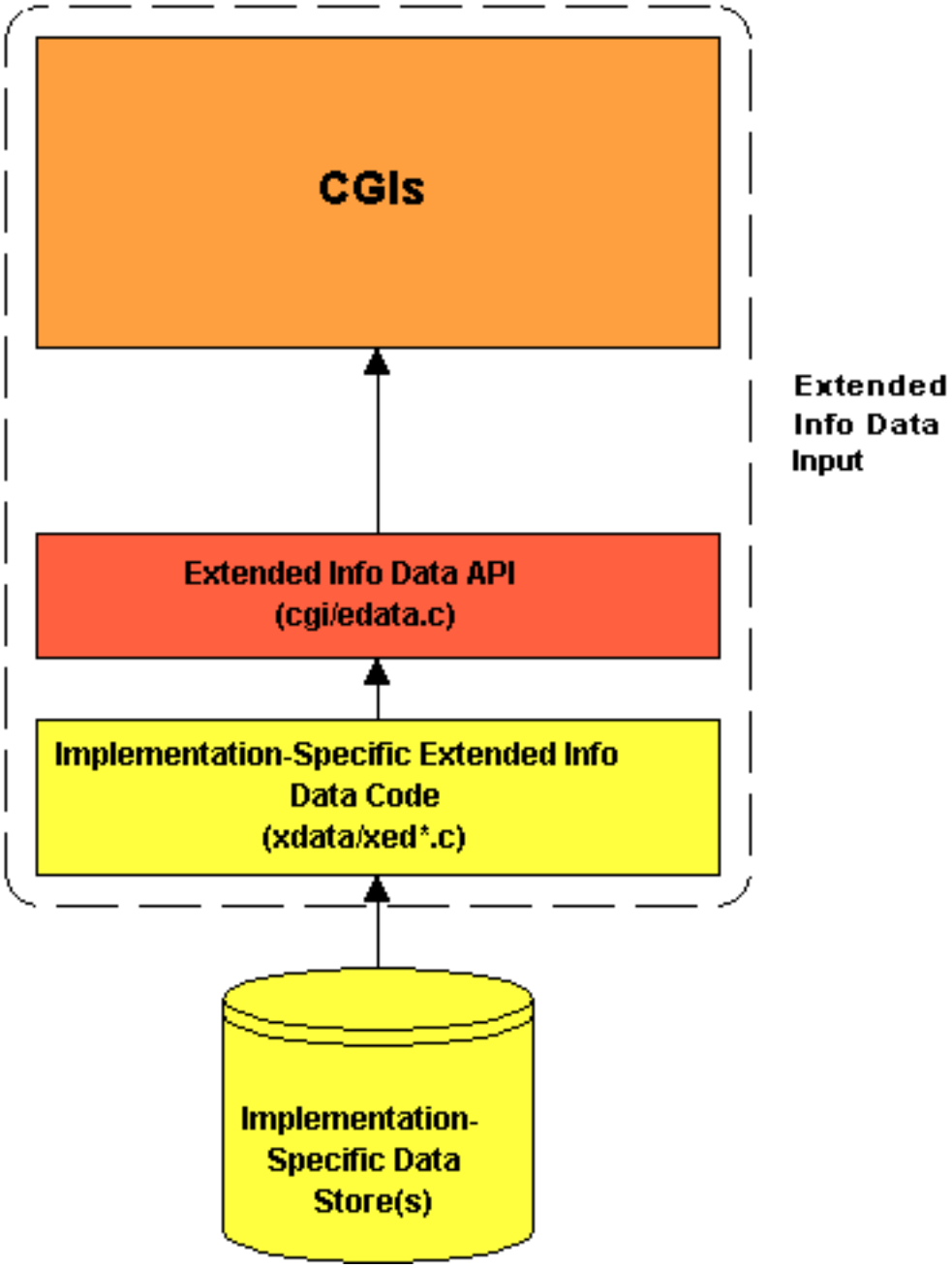


External Extended Data (XED) Overview

Nothing here yet... Wait for an alpha version of 0.0.7

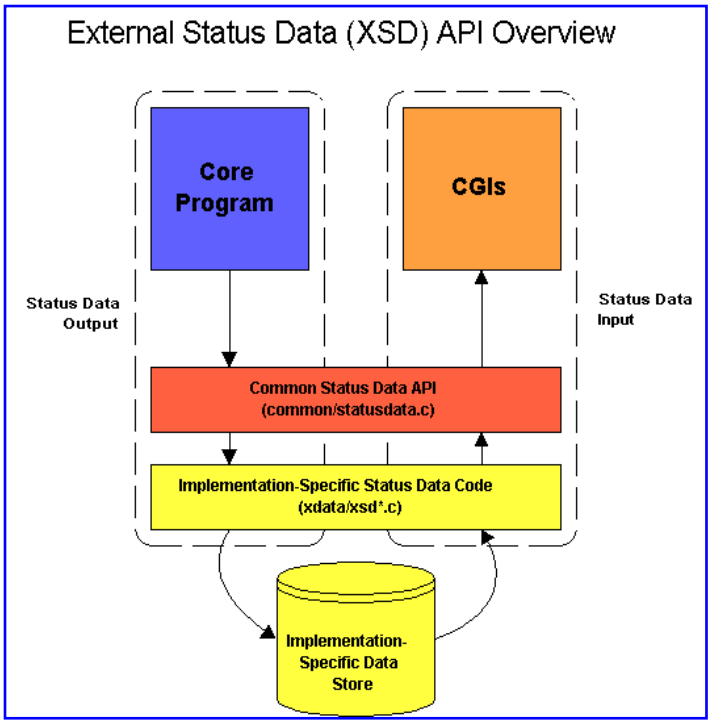


External Extended Info Data (XED) API Overview

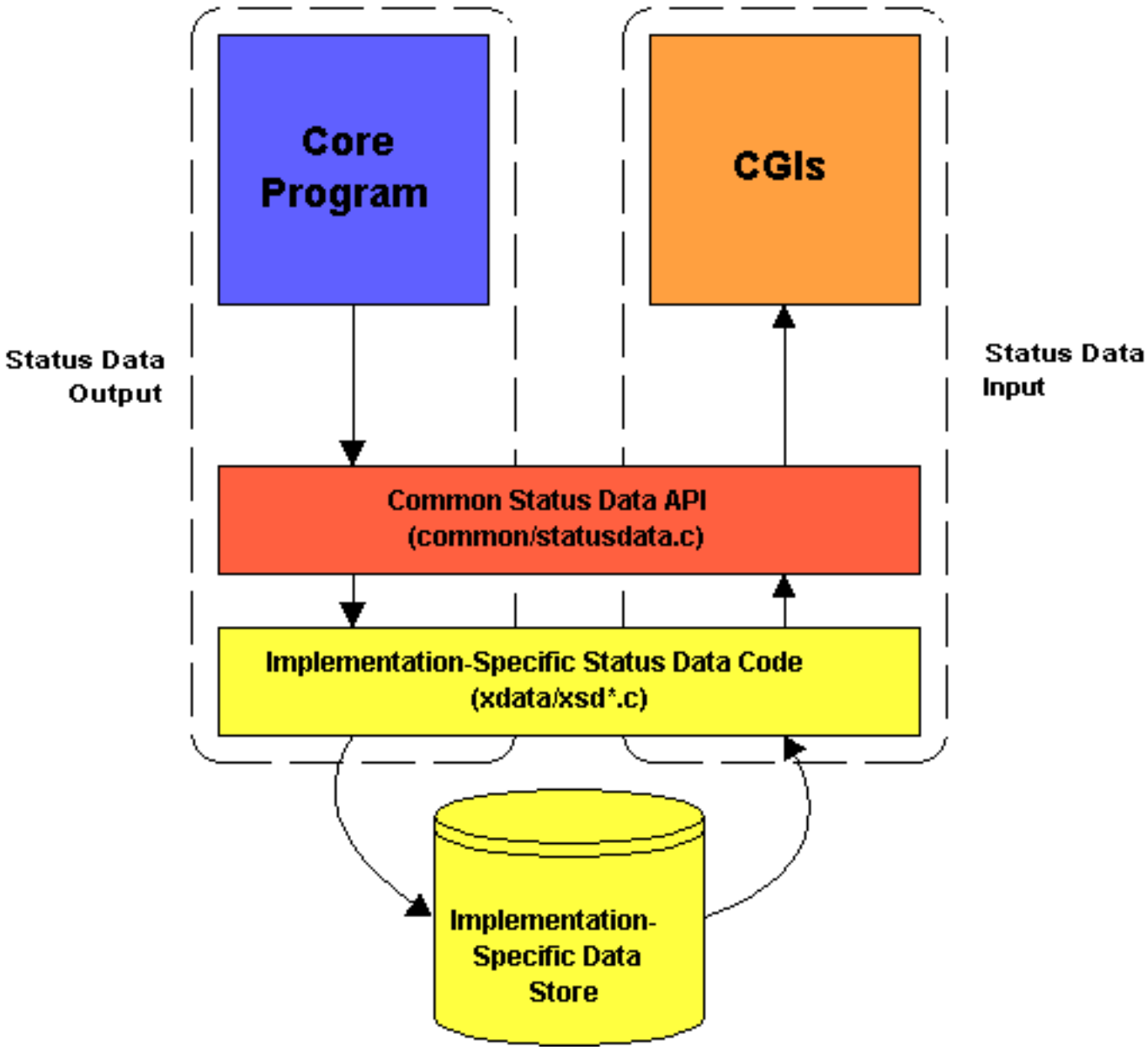


External Status Data (XSD) Overview

Nothing here yet... Wait for an alpha version of 0.0.7

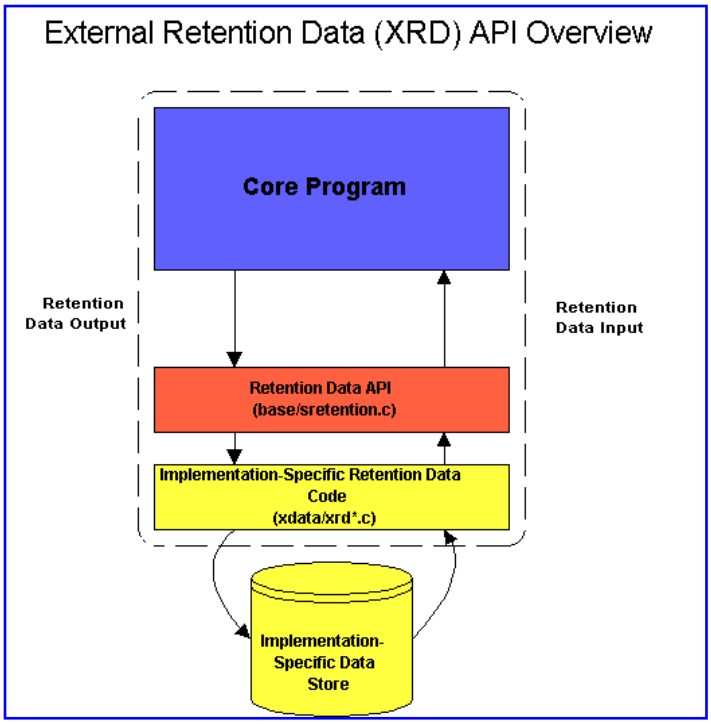


External Status Data (XSD) API Overview

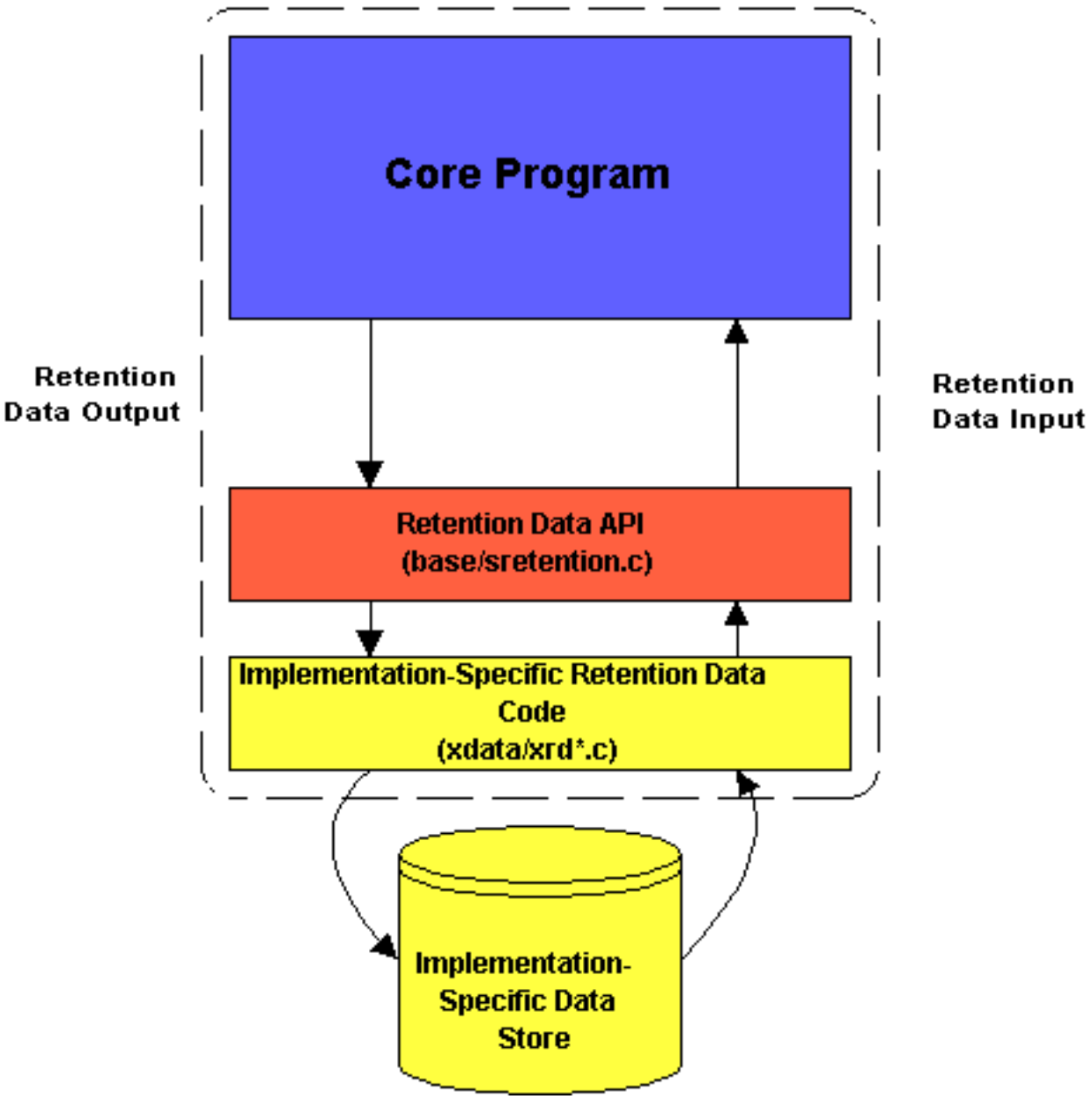


External Retention Data (XRD) Overview

Nothing here yet... Wait for an alpha version of 0.0.7



External Retention Data (XRD) API Overview



Indirect Host and Service Checks

Introduction

Chances are, many of the services that you're going to be monitoring on your network can be checked directly by using a plugin on the host that runs NetSaint. Examples of services that can be checked directly include availability of web, email, and FTP servers. These services can be checked directly by a plugin from the NetSaint host because they are publicly accessible resources. However, there are a number of things you may be interested in monitoring that are not as publicly accessible as other services. These "private" resources/services include things like disk usage, processor load, etc. on remote machines. Private resources like these cannot be checked without the use of an intermediary agent. Service checks which require an intermediary agent of some kind to actually perform the check are called *indirect* checks.

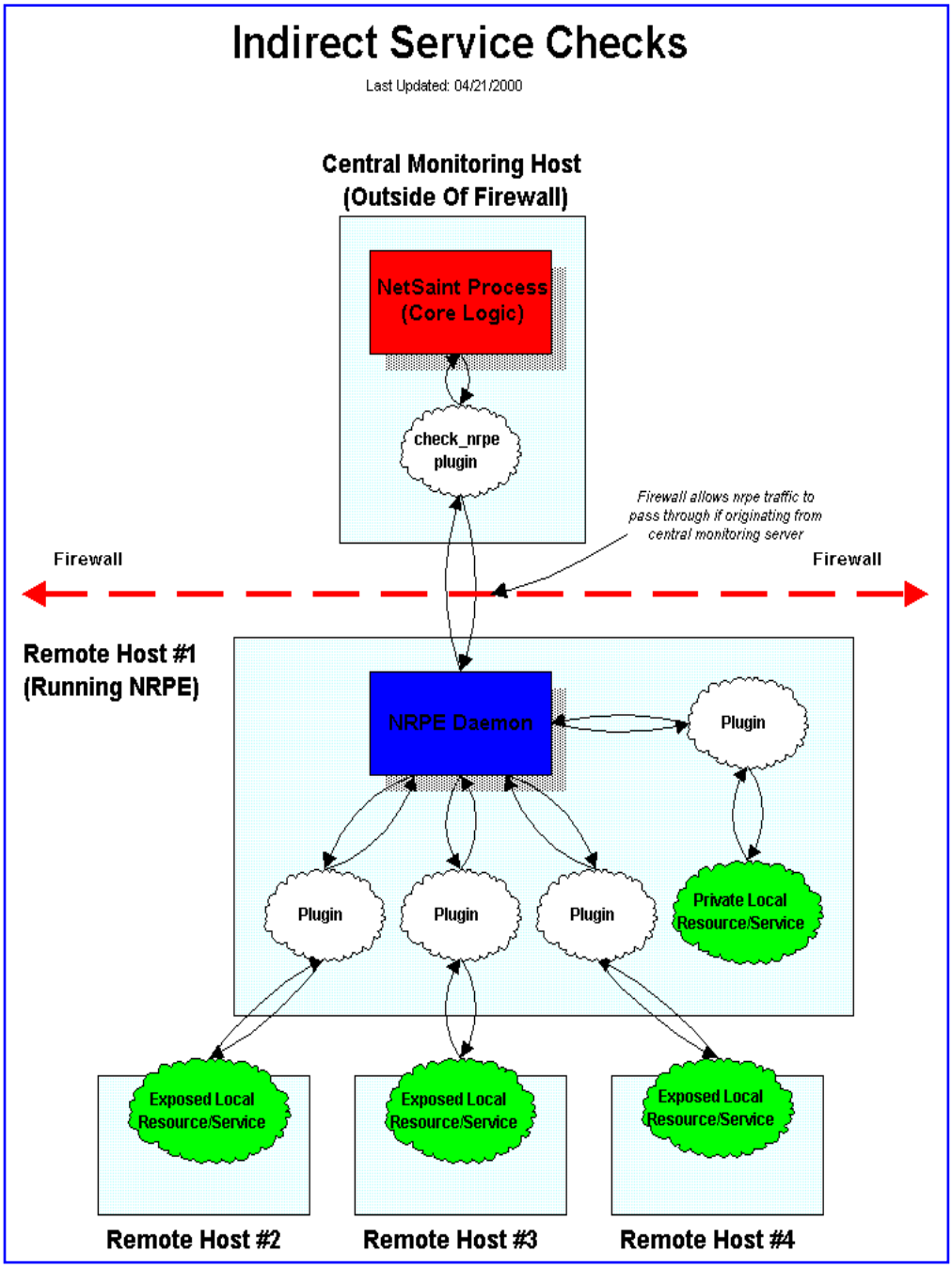
Indirect checks are useful for:

- Monitoring "local" resources (such as disk usage, processer load, etc.) on remote hosts
- Monitoring services and hosts behind firewalls
- Obtaining more realistic results from checks of time-sensitive services between remote hosts (i.e. ping response times between two remote hosts)

There are several methods for performing indirect active checks (passive checks are not discussed here), but I will only talk about how they can be done by using the nrpe addon. The nrpep and netsaint_statd can also be used to perform indirect checks.

Indirect Service Checks

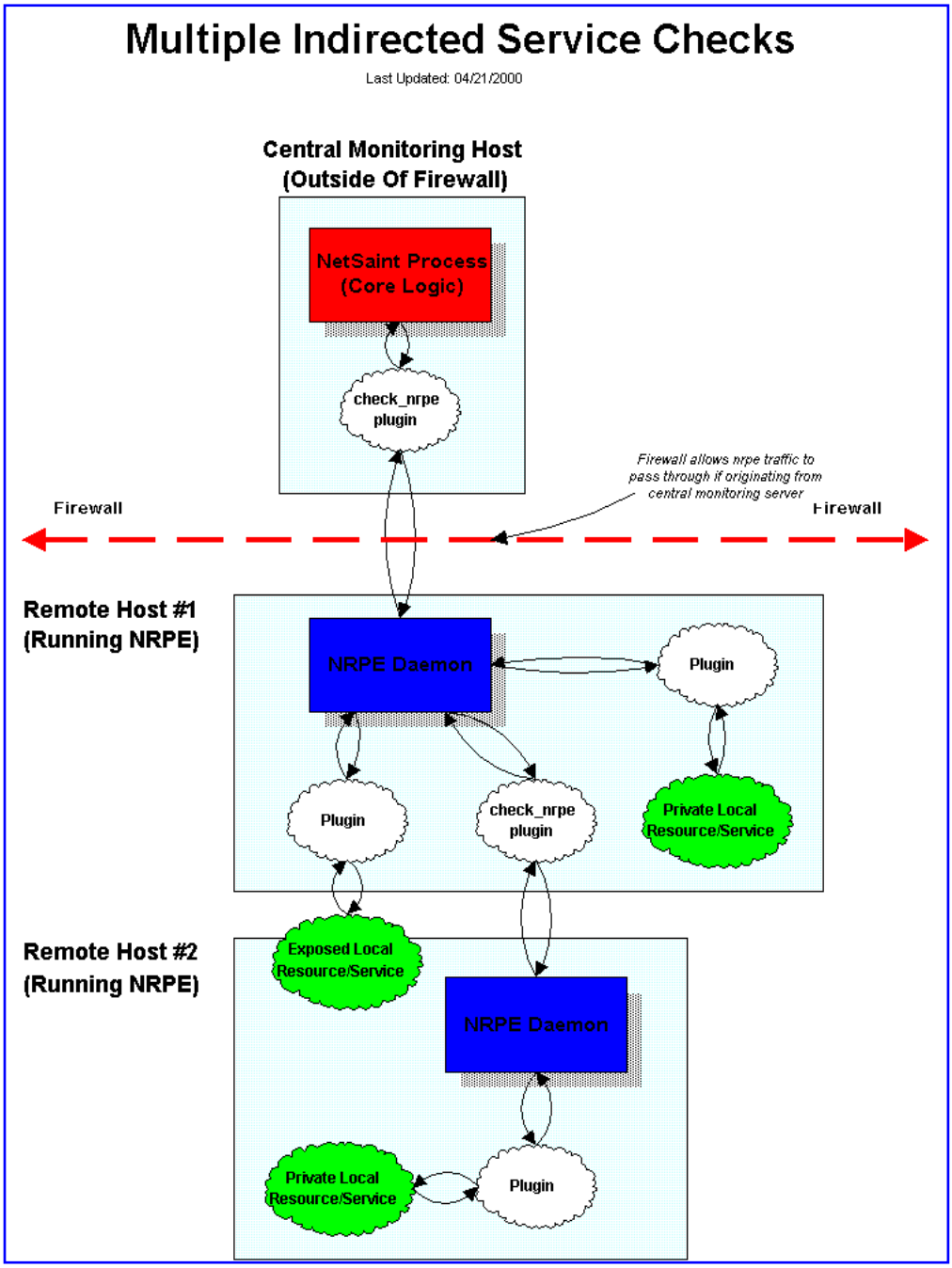
The diagram below shows how indirect service checks work. Click the image for a larger version...



Multiple Indirected Service Checks

If you are monitoring servers that lie behind a firewall (and the host running NetSaint is outside that firewall), checking services on those machines can prove to be a bit of a pain. Chances are that you are blocking most incoming traffic that would normally be required to perform the monitoring. One solution for performing active checks (passive checks could also be used) on the hosts behind the firewall would be to poke a tiny hold in the firewall filters that allow the NetSaint host to make calls to the *nrpe* daemon on one host inside the firewall. The host inside the firewall could then be used as an intermediary in performing checks on the other servers inside the firewall.

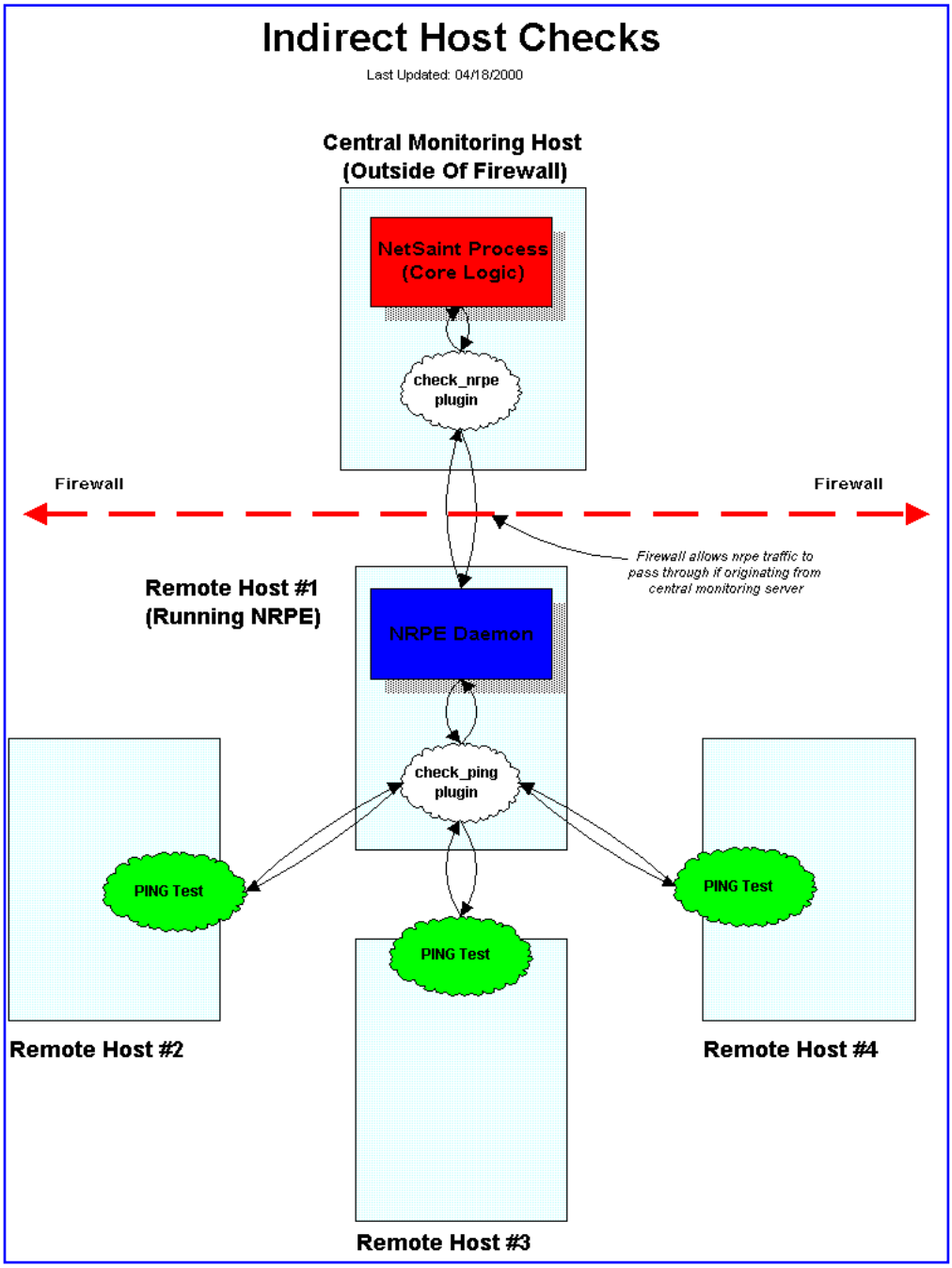
The diagram below show how multiple indirect service checks work. Notice how the *nrpe* daemon is running on hosts #1 and #2. The copy that runs on host #2 is used to allow the *nrpe* agent on host #1 to perform a check of a "private" service on host #2. "Private" services are things like process load, disk usage, etc. that are not directly exposed like SMTP, FTP, and web services. Click on the diagram for a larger image...



Indirect Host Checks

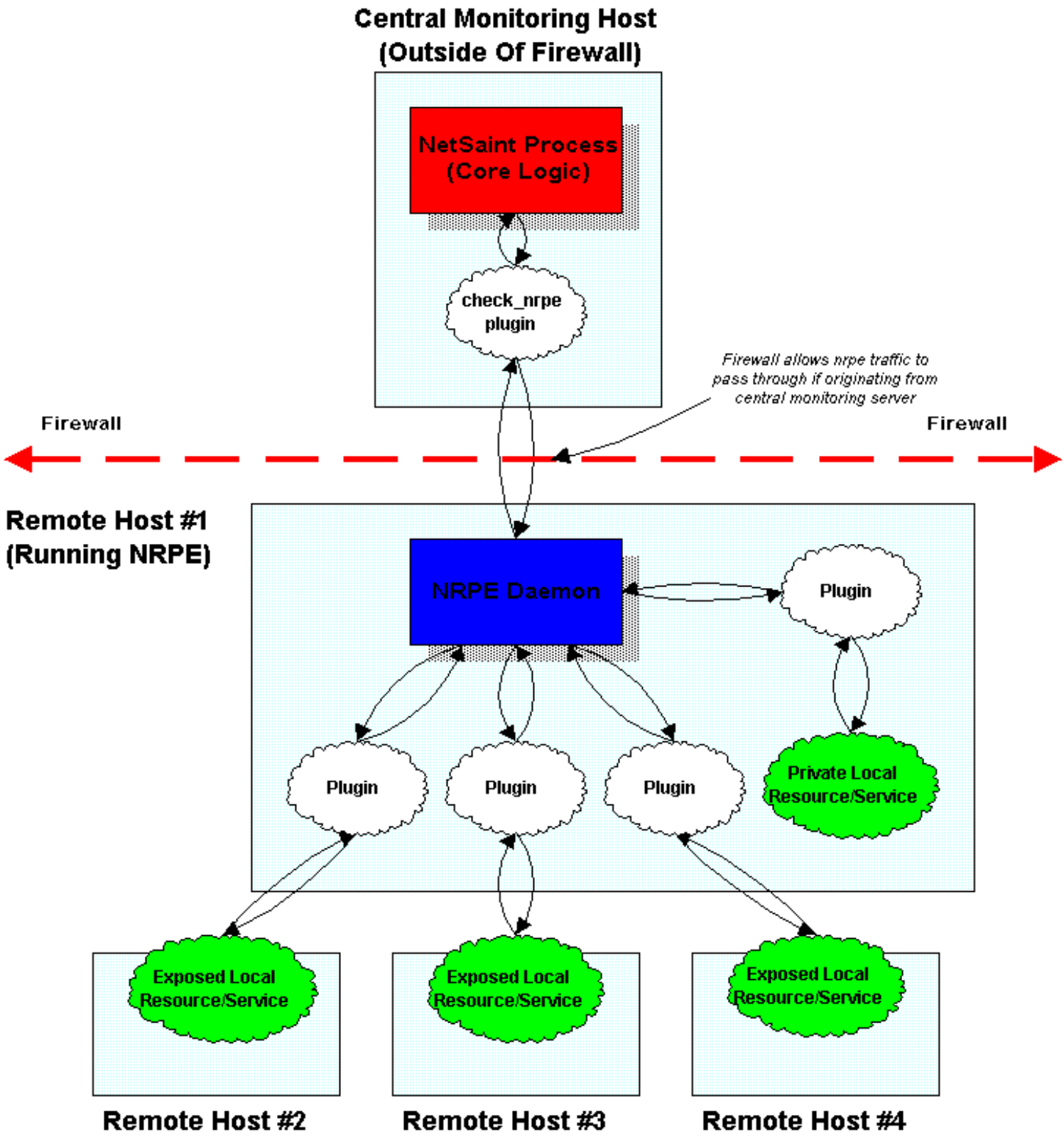
Indirect host checks work on the same principle as indirect service checks. Basically, the plugin used in the host check command asks an intermediary agent (i.e. a daemon running on a remote host) to perform the host check for it. Indirect host checks are useful when the remote hosts being monitored are located behind a firewall and you want to restrict inbound monitoring traffic to a particular machine. That machine (remote host #1 in the diagram below) performs will perform the host check and return the results back to the top level *check_nrpe* plugin (on the central server). It should be noted that with this setup comes potential problems. If remote host #1 goes down, the *check_nrpe* plugin will not be able to contact the *nrpe* daemon and NetSaint will believe that remote hosts #2, #3, and #4 are down, even though this may not be the case. If host #1 is your firewall machine, then the problem isn't really an issue because NetSaint will detect that it is down and mark hosts #2, #3, and #4 as being unreachable.

The diagram below shows how an indirect host check can be performed by using the *nrpe* daemon and *check_nrpe* plugin. Click the image for a larger version.



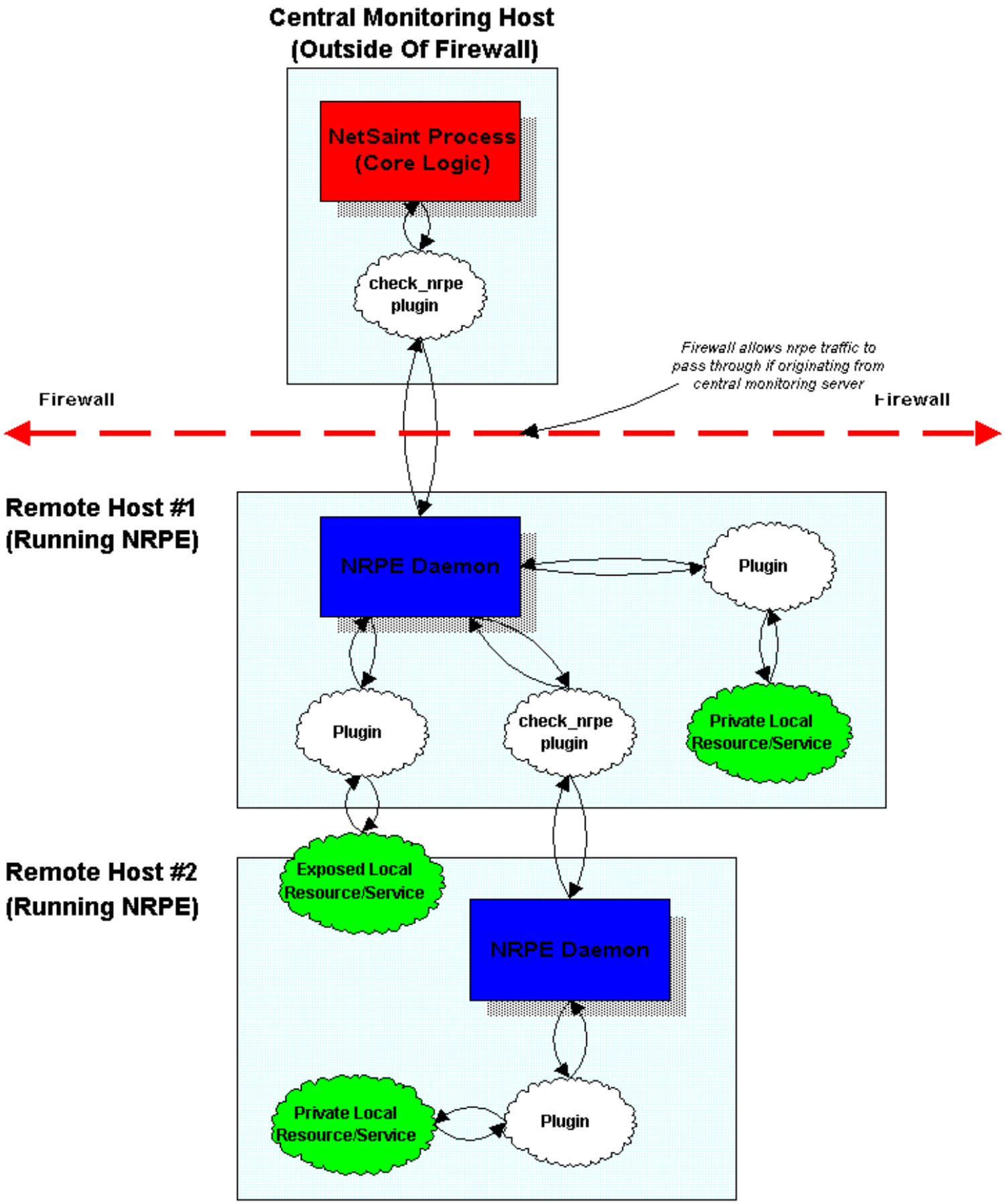
Indirect Service Checks

Last Updated: 04/21/2000



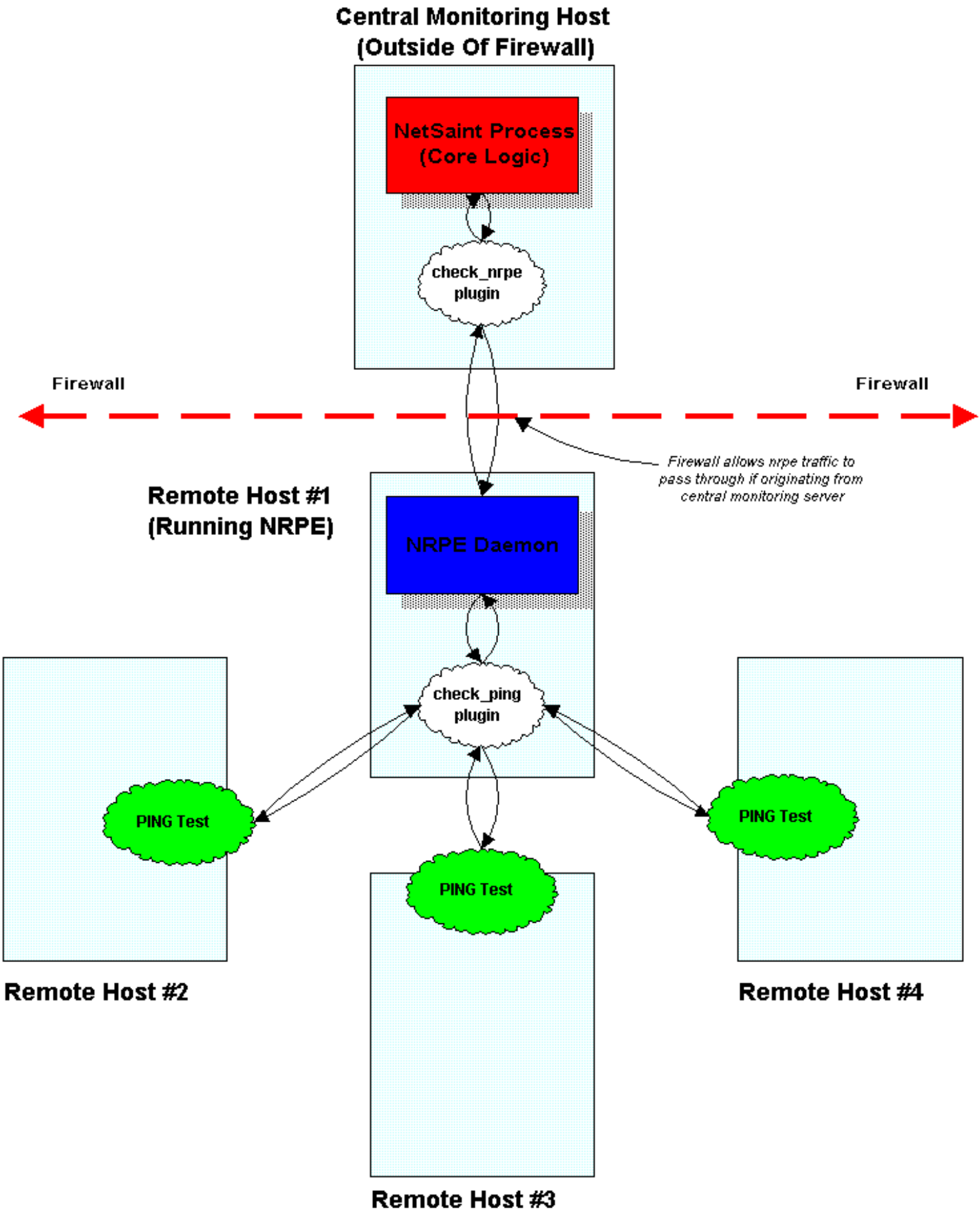
Multiple Indirected Service Checks

Last Updated: 04/21/2000



Indirect Host Checks

Last Updated: 04/18/2000



Monitoring Service and Host Clusters

Introduction

Several people have asked how to go about monitoring clusters of hosts or services, so I decided to write up a little documentation on how to do this. Its fairly straightforward, so hopefully you find things easy to understand...

First off, we need to define what we mean by a "cluster". The simplest way to understand this is with an example. Let's say that your organization has five hosts which provide redundant DNS services to your organization. If one of them fails, its not a major catastrophe because the remaining servers will continue to provide name resolution services. If you're concerned with monitoring the availability of DNS service to your organization, you will want to monitor five DNS servers. This is what I consider to be a *service* cluster. The service cluster consists of five separate DNS services that you are monitoring. Although you do want to monitor each individual service, your main concern is with the overall status of the DNS service cluster, rather than the availability of any one particular service.

If your organization has a group of hosts that provide a high-availability (clustering) solution, I would consider those to be a *host* cluster. If one particular host fails, another will step in to take over all the duties of the failed server. As a side note, check out the High-Availability Linux Project for information on providing host redundancy with Linux.

Plan of Attack

There are several ways you could potentially monitor service or host clusters. I'll describe the method that I believe to be the easiest. Monitoring service or host clusters involves two things:

- Monitoring individual cluster elements
- Monitoring the cluster as a collective entity

Monitoring individual host or service cluster elements is easier than you think. In fact, you're probably already doing it. For service clusters, just make sure that you are monitoring each service element of the cluster. If you've got a cluster of five DNS servers, make sure you have five separate service definitions (probably using the *check_dns* plugin). For host clusters, make sure you have configured appropriate host definitions for each member of the cluster (you'll also have to define at least one service to be monitored for each of the hosts). **Important:** You're going to want to disable notifications for the individual cluster elements (host or service definitions). Even though no notifications will be sent about the individual elements, you'll still get a visual display of the individual host or service status in the status CGI. This will be useful for pinpointing the source of problems within the cluster in the future.

Monitoring the overall cluster can be done by using the previously cached results of cluster elements.

Although you could re-check all elements of the cluster to determine the cluster's status, why waste bandwidth and resources when you already have the results cached? Where are the results cached? Cached results for cluster elements can be found in the status log (assuming you are monitoring each element). The *check_cluster* plugin is designed specifically for checking cached host and service states in the status log. **Important:** Although you didn't enable notifications for individual elements of the cluster, you will want them enabled for the overall cluster status check.

Using the *check_cluster* Plugin

The check_cluster plugin is designed to check the overall status of a host or service cluster. It works by checking the cached status information of individual host or service cluster elements in the status log.

More to come... The check_cluster plugin can temporarily be obtained from http://www.netsaint.org/download/alpha.

Monitoring Service Clusters

First off, you're going to have to define a service for monitoring the cluster. This service will perform the check of the overall status of the cluster. You are probably going to want to have notifications enabled for this service so you know when there are problems that need to be looked at. You probably don't care so much about the status of any one of the services that are members of the cluster, so you can disable notifications in those those service definitions.

Okay, let's assume that you have a *check_service_cluster* command defined as follows:

command[check_service_cluster]=/usr/local/netsaint/libexec/check_cluster --service $ARG1$ $ARG2$ /usr/local/netsaint/var/status.log < $ARG3$

Let's say you have five services that are members of the service cluster. If you want NetSaint to generate a warning alert if two or more services in the cluster and in a non-ok state or a critical alert if three or more are in a non-ok state, the <*check_command*> argument of the service you define to monitor the cluster looks something like this:

check_service_cluster!2!3!/usr/local/netsaint/etc/servicecluster.cfg

The $ARG3$ macro will be replaced with */usr/local/netsaint/etc/servicecluster.cfg* when the check is made. Since this is the file from which the *check_cluster* plugin will read the names of cluster members, you'll need to create that file and add the services that are members (one per line). The format of a service entry is the short name of the host the service is associated with, followed by a semi-colon, and then the service description. An example of the file contents would be as follows:

host1;DNS Service
host2;DNS Service

host3;DNS Service
host4;DNS Service
host5;DNS Service
host6;DNS Service

Monitoring Host Clusters

Monitoring host clusters is very similiar to monitoring service clusters. Obviously, the main difference is that the cluster members are hosts and not services. In order to monitor the status of a host cluster, you must define a service that uses the *check_cluster* plugin. The service should *not* be associated with any of the hosts in the cluster, as this will cause problems with notifications for the cluster if that host goes down. A good idea might be to associate the service with the host that NetSaint is running on. After all, if the host that NetSaint is running on goes down, then NetSaint isn't running anymore, so there isn't anything you can do as far as monitoring (unless you've setup redundant monitoring hosts)...

Anyway, let's assume that you have a *check_host_cluster* command defined as follows:

command[check_host_cluster]=/usr/local/netsaint/libexec/check_cluster --host $ARG1$ $ARG2$ /usr/local/netsaint/var/status.log < $ARG3$

Let's say you have six hosts in the host cluster. If you want NetSaint to generate a warning alert if two or more hosts in the cluster are not up or a critical alert if four or more hosts are not up, the *<check_command>* argument of the service you define to monitor the cluster looks something like this:

check_host_cluster!2!4!/usr/local/netsaint/etc/hostcluster.cfg

The $ARG3$ macro will be replaced with */usr/local/netsaint/etc/hostcluster.cfg* when the check is made. Since this is the file from which the *check_cluster* plugin will read the names of cluster members, you'll need to create that file and add the short names of all hosts (as they were defined in your host definitions) that are members (one per line). An example of the file contents would be as follows:

host1
host2
host3
host4
host5
host6

That's it! NetSaint will periodically check the status of the host cluster and send notifications to you when its status is degraded (assuming you've enabled notification for the service). Note that for the host definitions of each cluster member, you will most likely want to disable notifications when the

host goes down (using the *<notify_down>* option). Remeber that you don't care as much about the status of any individual host as you do the overall status of the cluster. Depending on your network layout and what you're trying to accomplish, you may wish to leave notifications for unreachable states enabled (using the *<notify_unreachable>* option) for the host definitions.

Large-Scale Host and Service Monitoring

Introduction

Many people have asked how well NetSaint scales when monitoring hundreds and even thousands of hosts and services. If configured properly it can be done. In fact, it *is* being done by a few large ISPs that I've spoken with. Right now I don't have any documentation put together on tips for setting things up for large-scale monitoring, but I will as 0.0.6 progresses further...

Fun Stuff

Have a little too much free time on your hands? Well, instead of playing Quake , you could try out some of the following things that you can do with NetSaint...

Create a virtual network assistant that speaks!

The Lowdown

By utilizing event handlers and some speech software, you can have NetSaint talk to you and tell you whats wrong with your network.

Completely Scientific Ratings

Funness Rating:	100%
Ability To Impress Co-Workers Rating:	100%
Usefulness Rating:	30%
Wise Use Of System Resources Rating:	5%

The Upsides

- It gives immediate audio feedback on the status of your network, which is quite useful if you're in the server room working on other things
- It will impress your boss and co-workers...

The Downsides

- Its a bit of a waste of system resources, so its not really fit for production machines
- If you're in the server room alone on a weekend or at night, having a machine start talking can scare the living daylights out of you. It has happened to me before...

Give Me Details!

First off, you need speech software installed on your system. I would recommend using the Festival Speech Synthesis System developed by The Centre for Speech Technology Research at the University of Edinburgh. This package provides the basic framework needed for converting text into spoken word. In order to use Featival in a practical manner you'll also need to install speechd. The speechd software implements /dev/speech and will queue all text written to it for processing by the Festival sound system. Once you've installed the speech software and *tested* it to make sure it works properly, you can move on to the next step...

In order to make NetSaint report system status via the speech software you'll have to write some event handlers. If you want audio alerts for only certain hosts or services, you'll have to define event handlers in the appropriate host and service definitions. If you want audio alerts for everything, you can just use the global_host_event_handler and global_service_event_handler definitions in the main configuration file. I've chosen to use global event handlers to make things easier to implement.

The global event handler definitions in my main configuration file look like this...

global_host_event_handler=global-hst-event-handler

global_service_event_handler=global-svc-event-handler

The command definitions for my global event handlers look like this...

command[global-hst-event-handler]=/usr/local/netsaint/libexec/eventhandlers/hst_event_handler $HOST NAME$ "$HOSTALIAS$" $HOSTSTATE$ $STATETYPE$ $HOSTATTEMPT$

command[global-svc-event-handler]=/usr/local/netsaint/libexec/eventhandlers/svc_event_handler $HOSTNAME$ "$HOSTALIAS$" "$SERVICEDESC$" $SERVICESTATE$ $STATETYPE$ $SERVICEATTEMPT$

So what do the event handler scripts look like? The event handlers I use are listed below. A copy of these scripts is available in the *eventhandlers/* subdirectory of the distribution. You may have to modify things to work on your system...

Global Host Event Handler (hst_event_handler)

```
#!/bin/sh

########################################
# NetSaint Global Host Event Handler
#
# Arguments:
#
# $1 = host short name
# $2 = host alias (long name)
# $3 = state
# $4 = state type (HARD or SOFT)
# $5 = current attempt number
#
########################################

echocmd="/bin/echo"
festivalcmd="/dev/speech"

case $4 in
    HARD)
        case $3 in
            UP)
                $echocmd "Good news!  Host $2 has RECOVERED!" > $festivalcmd
                ;;
            DOWN)
                $echocmd "Attention...  Host $2 is $3.  This is a critical state. Please check host
status!" > $festivalcmd
                ;;
            UNREACHABLE)
                $echocmd "Attention...  Host $2 is $3.  This is a critical state.  Please check
network connectivity!" > $festivalcmd
                ;;
        esac
        ;;
    SOFT)
        case $3 in
            UP)
                ;;
            DOWN)
                $echocmd "Attention... Host $2 is in a $4 $3 state.  Attempt number $5" >
$festivalcmd
                ;;
            UNREACHABLE)
                $echocmd "Attention... Host $2 is in a $4 $3 state.  Attempt number $5" >
$festivalcmd
                ;;
        esac
        ;;
esac

exit 0
```

Global Service Event Handler (svc_event_handler)

```
#!/bin/sh

##########################################
# NetSaint Global Service Event Handler
#
# Arguments:
#
# $1 = host short name
# $2 = host alias (long name)
# $3 = service description
# $4 = state
# $5 = state type (HARD or SOFT)
# $6 = current attempt number
#
##########################################

echocmd="/bin/echo"
festivalcmd="/dev/speech"

case $5 in
    HARD)
        case $4 in
            OK)
                $echocmd "Good news!  Service $3 on $2 has RECOVERED!" > $festivalcmd
                ;;
            CRITICAL)
                $echocmd "Attention...  Service $3 on $2 is in a $4 state.  Please check service!"
> $festivalcmd
                ;;
            WARNING)
                $echocmd "Attention...  Service $3 on $2 is in a $4 state.  Please check service!"
> $festivalcmd
                ;;
            UNKNOWN)
                $echocmd "Attention...  Service $3 on $2 is in a $4 state.  Please check service!"
> $festivalcmd
                ;;
        esac
        ;;
    SOFT)
        case $4 in
            OK)
                ;;
            CRITICAL)
                $echocmd "Attention.  Service $3 on $2 is in a $5 $4 state." > $festivalcmd
                ;;
            WARNING)
                $echocmd "Attention.  Service $3 on $2 is in a $5 $4 state." > $festivalcmd
                ;;
            UNKNOWN)
                $echocmd "Attention.  Service $3 on $2 is in a $5 $4 state." > $festivalcmd
                ;;
        esac
        ;;
esac

exit 0
```

That's it! Fire up NetSaint and listen to it tell you about your problems...

Securing NetSaint

Introduction

This is intended to be a brief overview of some things you should keep in mind when installing NetSaint, so as to not set it up in an insecure manner. This document is new, so if anyone has additional notes or comments on securing NetSaint, please drop me a note at netsaint@netsaint.org

Do Not Run NetSaint as Root!

NetSaint doesn't need to run as root, so don't do it. Even if you start NetSaint at boot time with an init script, you can force it to drop privileges after startup and run as another user/group by using the netsaint_user and netsaint_group directives in the main config file.

Enable External Commands Only If Necessary

By default, external commands are disabled. This is done to prevent an admin from setting up NetSaint and unknowingly leaving its command interface open for use by "others".. If you are planning on using event handlers or issuing commands from the web interface, you will have to enable external commands. If you aren't planning on using event handlers or the web interface to issue commands, I would recommend leaving external commands disabled.

Set Proper Permissions On The External Command File

If you enable external commands, make sure you set proper permissions on the */usr/local/netsaint/var/rw* directory. You only want a few users (probably only NetSaint and the web server) to have permissions to write to the command file. Instructions on setting up permissions for the external command file can be found here.

Require Authentication In The CGIs

I would strongly suggest requiring authentication for accessing the CGIs. Once you do that, read the documentation on the default rights that authenticated contacts have, and only authorize specific contacts for additional rights as necessary. Instructions on setting up authentication and configuring authorization rights can be found here. If you disable the CGI authentication features using the use_authentication directive in the CGI config file, the command CGI will refuse to write any commands to the external command file. After all, you don't want the world to be able to control NetSaint do you?

Use Full Paths In Command Definitions

When you define service checks, event handlers, notification commands, etc in command definitions, make sure you specify the *full path* to any scripts or binaries you're executing.

Hide Sensitive Information With $USERn$ Macros

The CGIs read the main config file and host config file(s), so you don't want to keep any sensitive information (usernames, passwords, etc) in there. If you need to specify a username and/or password in a command definition, use a $USERn$ macro to hide it. $USERn$ macros are defined in one or more resource files. The CGIs will not attempt to read the contents of resource files, so you can set more restrictive permissions (600 or 660) on them. See the sample *resource.cfg* file in the base of the NetSaint distribution for an example of how to define $USERn$ macros.
