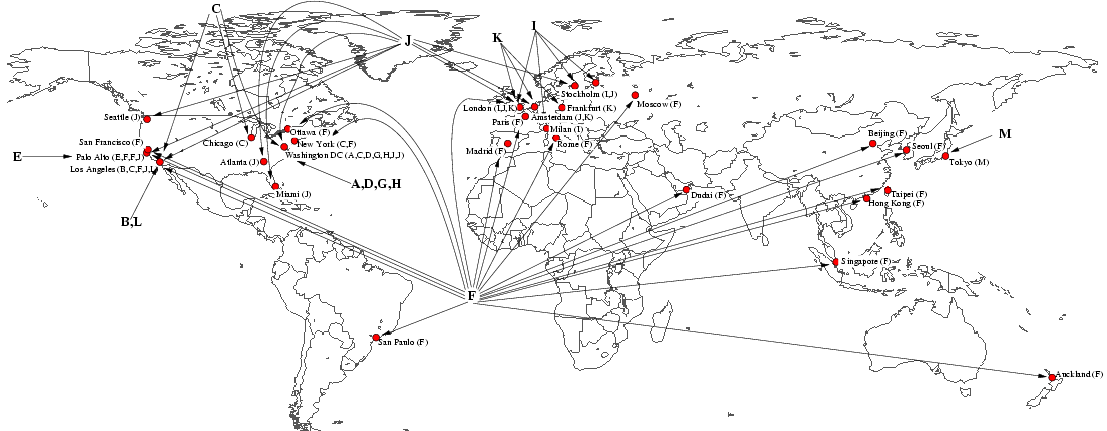
NMS-DNS Architecture



Daniel L. Needles - Daniel Needles, INC

John Hachey – Inovant (Visa)

Version 2.0

January 26, 2011

# Modification History

|  |  |  |
| --- | --- | --- |
| AUTHOR | DATE | COMMENTS |
| Daniel L. Needles and John Hachey | 4/19/2004 | Initial draft. |
| Daniel L. Needles | 1/26/2011 | Updated |



**COPYRIGHT © 2011,** **NMS Guru, INC**

Any copying, distribution, or use of any of the information contained within this document in any way or form is not permitted without the written consent of NMS Guru.

This is an unpublished work protected under the copyright laws. All rights reserved.

Table of Contents

[Introduction 1](#_Toc70325383)

[Total Cost of Ownership 2](#_Toc70325384)

[Introduction 2](#_Toc70325385)

[Services Provided 2](#_Toc70325386)

[Fiscal Benefits 3](#_Toc70325387)

[Overview 3](#_Toc70325388)

[Increased Effectiveness 3](#_Toc70325389)

[Reduced Administrative Cost 5](#_Toc70325390)

[Reduced Licensing Cost 5](#_Toc70325391)

[Fiscal Costs 5](#_Toc70325392)

[Personnel Triggers 6](#_Toc70325393)

[Product Triggers 7](#_Toc70325394)

[Policy Triggers 7](#_Toc70325395)

[Summary 8](#_Toc70325396)

[Architecture Overview 9](#_Toc70325397)

[Introduction 9](#_Toc70325398)

[NMS applications 10](#_Toc70325399)

[OpenView 10](#_Toc70325400)

[DNS 11](#_Toc70325401)

[DADA 13](#_Toc70325402)

[Introduction 13](#_Toc70325403)

[DADA Daemon 14](#_Toc70325404)

[DADA Primary Steps 15](#_Toc70325405)

[DADA Secondary Steps 16](#_Toc70325406)

[Conclusion 16](#_Toc70325407)

[As-Built 17](#_Toc70325408)

[Introduction 17](#_Toc70325409)

[Programs 18](#_Toc70325410)

[cron 18](#_Toc70325411)

[nmsdnsd.sh 18](#_Toc70325412)

[nmsdnsfeed.pl 19](#_Toc70325413)

[nmsdnsclean.pl 25](#_Toc70325414)

[nmsdnscreate.pl 25](#_Toc70325415)

[Configuration Files 25](#_Toc70325416)

[db.header 26](#_Toc70325417)

[db.cache 26](#_Toc70325418)

[nmsdns.props 26](#_Toc70325419)

[Log, Inventory, and DNS Files 30](#_Toc70325420)

[NMS-DNS Installation 31](#_Toc70325421)

[Package Inventory 31](#_Toc70325422)

[Obtain server information (Both Servers) 31](#_Toc70325423)

[Contact User (Both Servers) 33](#_Toc70325424)

[Verify OVW Starts. (Both Servers) 33](#_Toc70325425)

[Install DNS (bind) software. (Both Servers) 33](#_Toc70325426)

[Install NMS-DNS scripts and directories. (Both Servers) 33](#_Toc70325427)

[Save Files. (Both Servers) 34](#_Toc70325428)

[Configure NMS-DNS (Both Servers) 34](#_Toc70325429)

[Configure NMS-DNS (Primary Server Only) 34](#_Toc70325430)

[Configure NSLOOKUP (Both Servers) 35](#_Toc70325431)

[Configure Resolver (Both Servers) 35](#_Toc70325432)

[Configure OpenView (Both Servers) 36](#_Toc70325433)

[Configure Named Server (Both Servers) 37](#_Toc70325434)

[Configure UNIX Root User Startup Script (Both Servers) 37](#_Toc70325435)

[Configure Cron (Primary server only) 37](#_Toc70325436)

[Configure Startup Script (Both Servers) 38](#_Toc70325437)

[Backup Configuration Files (Both servers) 38](#_Toc70325438)

[Special Device Cases 38](#_Toc70325439)

[Test Run on Primary 38](#_Toc70325440)

[Verify OVW Starts. (Both Servers) 38](#_Toc70325441)

[Test System Startup Script (Both Servers) 39](#_Toc70325442)

[NMS-DNS Testing 40](#_Toc70325443)

[Introduction 40](#_Toc70325444)

[Basic System Checks 40](#_Toc70325445)

[Advanced Checks 41](#_Toc70325446)

[NMS-DNS 41](#_Toc70325447)

[OpenView 41](#_Toc70325448)

[Appendix A: Dependencies 43](#_Toc70325449)

[Appendix B: Initial Design Requirements 44](#_Toc70325450)

[Appendix C: Scripts 46](#_Toc70325451)

[Cron Entry 46](#_Toc70325452)

[NMSDNSd.sh 46](#_Toc70325453)

[NMSDNSFEED.pl 51](#_Toc70325454)

[NMSDNSCLEAN.pl 64](#_Toc70325455)

[NMSDNSCREATE.pl 73](#_Toc70325456)

[NMSDNS.pl 76](#_Toc70325457)

[NMSDNS.props 89](#_Toc70325458)

[IMPORTANT NOTICE 95](#_Toc70325459)

[Authors 97](#_Toc70325460)

# Introduction

John and I as consultants for over 50 Fortune 500 companies have architected many comprehensive NMS solutions. Most of these companies utilize a large number of NMS systems. These systems include: HP OpenView, IBM Tivoli, CA NetQoS, Concord, etc, Cisco CUOM, CANA, CiscoWorks, etc, JYRA, nGenius, and Visual Uptime. Usually each system administrates the NAME-to-IP and IP-to-NAME resolution for thousands of devices. Despite the advances in CMDB keeping the device inventories and in particular resolution consistent across all these systems is a constant headache. Often DNS is implemented synchronously within the vendors products and as a result any problem in DNS can cause cascading failures throughout the end-to-end solution.

The NMS-DNS solution streamlines the DNS administration by implementing a local DNS server with scripts that nightly populate DNS with OpenView’s monitored device inventory. Together they provide a common resolution source for all NMS applications and their servers. This:

* Increases the effectiveness of NMS applications.
* Simplifies the administration and reduces administration costs.
* Enhances diagnostic tools, providing interface information for router and switch ports.
* Reduces DNS licensing costs for products such as QIP.

This paper discusses:

* Total cost of ownership
* Overall architecture and design
* Initial as-built configuration
* Installation steps (for AIX)
* Post installation tests

# Total Cost of Ownership

## Introduction

Like any new technical tool, NMS-DNS is only as good as the processes and the people that support it. After the consultant leaves and the euphoria wears off, the NMS-DNS tool will continue to require diligent stewardship over its technological and budgetary needs. As long as the solution remains supported in both areas, the tool will persist and enable the goals of the company over its expected five-year lifespan. This section discusses the services provided and the fiscal consequences and requirements of an NMS-DNS solution.

## Services Provided

To understand the costs and how to best leverage the fiscal benefits of the NMS-DNS solution, it is important to understand its role and applications. The scripts and DNS installation provides five services:

* Highly available DNS services for all NMS applications.
* Self populating DNS database based on OpenView’s inventory.
* Notifies Network Administrator on detected error in OpenView and Network.
* Appends interface information to names of non primary interfaces for routers and switches.
* Eliminates errors in reporting down devices.

The NMS-DNS solution provides forward and reverse resolution between names and IP addresses of monitored routers, switches, and servers. The solution is automated, populated by scripts based on OpenView's inventory

It can be configured on two separate HPOV view systems, one which acts as the master of the other in order to keep the databases in sync.

The scripts also notify the NMS-DNS administrator about NMS and network errors it detects.

Further, the NMS-DNS solution appends interface information to non-primary interfaces on routers and switches. This enables tools such as trace route to show the interfaces being traversed though or interacted with.

Finally, by associating only the most relevant interface with the router or switch (such as a loopback address), NMSDNS can eliminate HPOV from indicating a device is down when only the interface is down.

## Fiscal Benefits

Together these five services provide the following three fiscal benefits:

* Increased effectiveness
* Reduced administration cost
* Reduced licensing cost

### Overview



The diagram above represents tool effectiveness as impacted by three vectors:

* Engineering and NOC administration workload
* The number NMS tools
* The number of monitored elements

The area in the diagram to the left of the first dotted line represents the current distribution of the available engineering hours. The second dotted line indicates the current visibility into the issues. It includes the current work being performed in the first box as well as the engineer’s todo and wish lists. The third dotted line indicates the visibility enabled by the NMS-DNS solution. As expected most of these services and facilities will be broken or not implemented due to the lack of visibility.

### Increased Effectiveness

Metaphorically, just as OpenView shines a light into the dark recesses of the network, the NMS-DNS project illuminates OpenView, the Network, and the processes that bind them. As an automated tool, NMS-DNS process runs daily, testing and verifying aspects of OpenView and the network. In the process it builds its database of shared devices and reports any errors to the administrator.

The scope and types of errors detected by NMS-DNS can indicate a lot about the effectiveness of HPOV currently implemented at a company. If the combination of people, technology, and processes are not running effectively, phone calls from irate customers become the primary mechanism for change. This is obviously less desirable than using OpenView as the primary notification mechanism.

The best way to improve the current situation is usually increased efficiency to stretch the current hours of the engineers. Through daily runs, the NMS-DNS solution will increase the company's visibility into potential issues and automate some tasks to increase efficiency.

For example, at one Fortune 500 company the initial run of the NMS-DNS solution identified the following problems in its environment:

|  |  |
| --- | --- |
| **Description** | **Issues** |
| Total Devices | 2,218 |
| Device name missing from OpenView | 311 |
| Device name missing/inaccessible from DNS | 315 |
| Device name missing/inaccessible from sysname | 344 |
| Domain missing from OpenView | 330 |
| Domain missing/inaccessible from DNS | 334 |
| Domain missing/inaccessible from sysname | 311 |
| SNMP-Invalid characters used | 3 |
| SNMP-string is unknown to OpenView | 774 |
| SNMP-sysname is not set | 15 |
| Duplicate-Object-In-OpenView | 33 |
| Illegal characters in network | 117 |
| Network-overlap | 17 |

Maintaining the NMS-DNS solution by correcting the reported errors will assist in creating a healthy monitoring solution to:

* Increase OpenView's response time and accuracy
* Reduced frequency and criticality of outages
* Shorten NOC response times
* Quicker and more accurate problem identification and resolution
* Problem anticipation
* Correct model of the network

Together they lead toward a NMS centric instead of a customer call centric monitoring solution. In this scenario, the ultimate goal is to reduce incoming call volume to 0 through proactive monitoring.

### Reduced Administrative Cost

The administration cost of DNS solutions such as QIP as well as other NMS applications such as OpenView are reduced though the automation of NMS-DNS and the single point of resolution. Secondly, augmenting interface descriptions in NMS-DNS with interface information, simplifies the trouble shooting of network issues. Given that the primary notification mechanism is "helpdesk calls," this greatly reduces the workload and potentially frees up hundreds of man-hours.

The actual and potential administrative cost reductions are mitigated by the increased visibility into existing issues described in the effectiveness section earlier. Although NMS system-based notification instead of help desk call-based notification is usually desired, sometimes the errors from NMS-DNS indicate that this is not the case. In these instances redirecting staff focus and revising procedures in the short term to address these dedected issues is required. If this is a political impossibility, accepting the current help desk centric instead of NMS centric monitoring solution becomes an important initiative to prevent further erosion of effectiveness and tactics that do not match the strategy of the organization.

### Reduced Licensing Cost

Many routers and switches are maintained in DNS applications such as QIP. Since the interface information is included within NMS-DNS, only the primary loopback or other host address is required to be retained in the corporate DNS. These saving can quickly add up. In one case the customer was paying $4.00 per interface per year. With a total of 3,800 interfaces that could be removed, NMS-DNS provided an annual saving of near $15,000. Additionally, by reducing the administration overhead of auditing and manually maintaining these devices in the DNS management product, the administration overhead was lowered by another 100 hours.

## Fiscal Costs

The NMS-DNS is a living set of processes, dependent on a variety of people, technology, and policies. This section describes the total cost of ownership over the lifespan of the NMS-DNS solution and documents the effect in these areas. Trigger events can necessitate changes in the NMS-DNS solution. In order provide proper stewardship both technologically and fiscally these trigger events need to be detected and appropriately acted upon in order to ensure the integrity of the NMS-DNS solution. Under budgeting can result in the failure of the NMS-DNS solution. Thus, understanding the trigger events is extremely important in maintaining and protecting the health of NMS-DNS. The following are descriptions and analysis of the impact of the more likely trigger events.

### Personnel Triggers

The most common personnel trigger events are staff changes. When the administrator of NMS-DNS changes, it will require 80 hours for the new employee with the proper skill set to fully come up to speed on the product. If only minor changes to the configuration are required, a minimum of 20 hours is required for the new employee to understand the impact of these changes. Based on past industry events, the lifecycle of NMS-DNS is expected to be around 5 years before the solution needs to be revisited. During this time period, an estimated yearly investment of 80 hours is required to maintain the solution using the following integrated skill sets:

* Perl ~2 yrs experience, including:
  + Procedures and standard procedural language.
  + Nested hashes.
  + GetOpt package.
  + 'Require' and 'use vars' constructs.
* Korn Shell ~ 1 yr experience, including:
  + Basic procedural language and variable declarations
  + Familiar with parameter passing and the $! variable.
* UNIX ~2 yrs experience, including:
  + UNIX scheduling program: cron.
  + Utilities such as tar, compress, etc.
  + Environment variables such as PATH, MANPATH, etc.
  + System startup scripts.
  + User startup scripts.
* OpenView ~1 yrs experience, including:
  + Inventory maintenance using seed files.
  + SNMP maintenance.
  + Map maintenance.
  + Usage of the ovtopodump and ovobjprint utilities
* DNS Admin ~ 1 yr experience.
  + Experience with bind 9.2
  + Familiar with SOA, NS, PTR, A, and CNAME records
  + Familiar with $INCLUDE and $ORIGIN directives

### Product Triggers

The NMS-DNS solution depends on OpenView, DNS, and UNIX to implement its solution. Changes in these environments create trigger events that must be acted upon

#### UNIX upgrade, patches, and replacement

The NMS-DNS solution was design and testing on Windows 2K, Solaris 2.8, AIX 5.1, and 4.3 servers. This makes the code extremely portable between operating systems. Although very few changes are probably required, it is recommended that a week for testing and development and another week of implementation time be allocated to ensure the proper corrections are performed.

#### OpenView upgrade, patches, and replacement

HP is notorious for changing very basic functionality in the OpenView product on even minor upgrades. Therefore it is recommended to allow two weeks to test and update the NMS-DNS solution in the lab and another week for implementation of the solution in production.

#### Addition of NMS tools and servers

The addition of NMS tools and servers into the monitoring solution at the company requires updating the resolvers of the operating systems on the respective devices. This can be accomplished within a period of two working days. Additionally three days should be allowed to enable the solution to be implemented in production.

### Policy Triggers

Policies embody the goals of processes within the organization. Like any solution these processes are an integral part of the NMS-DNS solution. The following section describes the more likely policy changes that will impact the NMS-DNS solution.

#### Primary interface policy (loopback3)

On multi-homed devices one primary interface is used to represent the device within NMS-DNS while the other interfaces have their names augmented with the interface name. Currently the policy that identifies which interface represents the primary interface uses loopback3 as the first choice. This logic is hard coded into the nmsdnsfeed.pl program. To alter this logic it will take roughly a week of coding and testing and another week of implementation in production.

#### Hostname resolution policy

On multi-homed devices one primary interface is used to represent the device within NMS-DNS while the other interfaces have their names augmented with the interface name. There is a general policy parameterized in the nmsdns.props file, which facilitates changing which types of interfaces take precedence. To implement these changes will require two days of coding and testing and another week of implementation in production

#### Diagnostic policies

Currently, NMS-DNS is designed to be available on all NMS-DNS systems. This means diagnostic tool in order to get the extra interface information and accurate IP to name resolution need to be issued from one of these servers. This is recommended since issues identified between the NMS system and the end device can sometimes not exist or differ between the user's workstation to the end device. However, it is possible to build a web interface to provide remote access to these tools. In the case of traceroute such a solution will require 3 weeks of development and testing and another week of implementation in production.

#### New device type addition (interface type)

On multi-homed devices one primary interface is used to represent the device within NMS-DNS while the other interfaces have their names augmented with the interface name. There is a general policy parameterized in the nmsdns.props file, which facilitates converting long interface names into abbreviations. To implement additions to this list will require two days of coding and testing and another week of implementation in production

## Summary

This section has described the NMS-DNS solution from a fiscal perspective. By exploring both the benefits as well as the costs associated with the solution, this section described the total cost of ownership.

# Architecture Overview

## Introduction

The NMS-DNS architecture consists of five components:

* NMS applications
* OpenView (inventory)
* DNS configuration
* The Distributed/Automated DNS Administration (DADA)
* NOC Operators.

The relationship between these components is shown in the diagram below.



In the NMS-DNS High Availability (HA) configuration, the external NMS systems such as Concord or NerveCenter first query the primary NMSDNS, then the backup NMSDNS, and finally the corporate DNS to resolve a host name or IP address. (This assumes a configuration including a primary and secondary OpenView with the databases synchronized.) The NOC operators switch between the primary and secondary OpenView servers in accordance with their failover policy.

The following sections of the document describe each of the layers in this architecture as it applies to NMS-DNS solution. The document touches on:

* NMS Applications
* OpenView
* DNS
* Distributed/Automated DNS Administration (DADA)

## NMS applications

Most companies with OpenView utilize a number of NMS applications which may include: CiscoWorks 2000, Netcool, NerveCenter, eHealth, JYRA, nGenius, and Visual Uptime. For each of these servers the DNS resolvers are configured to first look at the primary OpenView NMS-DNS, then the backup OpenView NMS-DNS, and finally the corporate DNS server.

## OpenView

OpenView provides the basis for the entire DNS inventory, which in turn is used by the NMS systems to resolve host names to IP addresses and company-versa. In conjunction with ICMP pings and SNMP gets, the DADA uses OpenView's database to completely describe the association between the machine-oriented IP addresses to the human-readable names for monitored devices. Specifically, the DADA nmsdnsfeed.pl PERL script uses the OpenView utilities ovtopodump and ovobjprint to extract information from OpenView's topology and object databases.

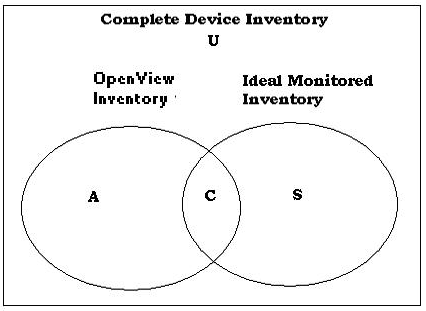
The domain of devices within the context of OpenView can be split into four sets:

A Represents OpenView's set of devices that are "actually" monitored (represented below by set A.)

S Represents the devices that "should" be monitored (represented below by set S)

C Represents the devices that are "correctly" monitored (represented below by set C)

U Represents the "universe" or complete device inventory that could be monitored.



From these sets two new sets can be created, representing inventory errors within OpenView.

E (Set A minus set C) Devices currently monitored that shouldn't be monitored ("Errored.")

M (Set A minus set C) Devices currently NOT monitored that should be monitored ("Missing.")

Inspection of OpenView's inventory can enable the company staff to identify the entirety of set E. Inspection of the inventory of other NMS systems can identify many of the elements of set M. However, if there are devices that should be monitored that are currently not part of the inventory in any of the NMS systems, some of these devices will be difficult to identify.

## DNS

The DNS tables associate machine oriented IP addresses with human readable Names. The DNS (BIND) server enables lookups in both directions. DNS uses the '.' within the domain name and IP address to direct the use of the data files. The DADA scripts insert three kinds of records into the local DNS server to assist the NMS applications and NOC staff.

* "A" record - Enables an application to look up an IP address by a Name.
* "CNAME" record - Enables an application to lookup a name by an alias.
* "PTR" record - Enables an application to lookup a name by an IP address.

The DNS tables are populated as follows:

* Multi-homed devices (Routers and Switches)
  + "A" Record:
    - The most appropriate interface resolves to the host name.
    - The remaining interfaces resolve to hostname-interface.
  + "PTR" Record
    - Every "A" record has a corresponding reverse lookup based "PTR" record.
  + "CNAME" Record
    - None.
* Normal devices
  + "A" Record:
    - Each interface resolves to the host name.
  + "PTR" Record
    - Every "A" record has a corresponding reverse lookup based "PTR" record.
  + "CNAME" Record
    - None.

## DADA

### Introduction



DADA runs on a daily basis to maintain the IP to NAME and NAME to IP resolution for all monitored devices based on OpenView's inventory. The diagram above superimposes the DADA process flow upon the NMS-DNS architecture described earlier.

Like the other NMS applications, the DADA architecture is highly available with a primary and a secondary server. The architectures are different for each. The primary server uses the object database of the local OpenView application to build the NMS-DNS configuration and device inventory. Further it pushes this NMS-DNS configuration and inventory to the secondary. In this manner the primary system is responsible via the nmsdnsd.pl program to audit and rebuild NMS-DNS based on the OpenView inventory for both the primary and secondary server.

### DADA Daemon



On the primary DADA server cron calls the nmsdnsd.sh program to update the NMS-DNS configuration and inventory for both the primary and secondary DADA servers. This Korn shell program performs the following six steps.

* nmsdnsd.sh determines if the server is local server the primary or secondary. If the server is neither, the program exits unless overridden by providing the primary or secondary server name as an argument on the command line.
* It backs up the existing DNS configuration files and stores these in the form nmsdns.[abbreviated day of the week].tar.Z.
* Performs on the primary or secondary DADA server.
  + In the case of the primary it builds the NMS-DNS configuration and data based on the local OpenView's inventory.
  + In the case of the secondary, it grabs the NMS-DNS configuration and data from the primary.
* Before the rebuilding of the configuration and inventory on NMS-DNS is complete, the program:
  + Wipes out the existing NMS-DNS configuration and inventory.
  + It stops the named (NMS-DNS) server.
  + Restarts the named (NMS-DNS) server.

At least one server is up at all times during the course of the script. Further this down time period is minimized. During each step, the nmsdnsd.sh program documents its progress to standard output. This information is directed via the cron entry to /opt/NMSDNS/logs/nmsdnsd.log

### DADA Primary Steps



The job of the primary DADA server is to update the DNS files and configuration based on the local OpenView's object database. The process starts when a cron job on the OpenView server initiates the DADA daemon described in the earlier section. This program determines that the local machine is the primary DADA server, stops the named server, and calls nmsdnsfeed.pl. This program queries OpenView and the network in order to build DNS tables for the local DNS server. Once this program completes, the DADA daemon calls nmsdnsclean.pl. This program examines the output of nmsdnsfeed.pl, corrects any problems in the data, and documents the changes. Finally, the DADA daemon calls the nmsdnscreate.pl program. This program is responsible for using the output from nmsdnsclean.pl to build out the local DNS tables and configuration. The DNS files include the named.conf configuration file for the named server, the database files for forward resolution of device name to IP address, and the database files for the reverse resolution of IP address to device name.

Sequentially nmsdnsfeed.pl, nmsdnsclean.pl, and nmsdnscreate.pl run and append their output to the same three logs:

* nmsdns.log: This contains all the debug and trace messages recorded by the programs.
* nmsdns.warn: Contains individually identified problems with individual devices.
* nmsdns.warn2: Summarizes the nmsdns.warn messages by error type, listing all the domains or devices associated with an error on a single line.

Additionally, all three programs use the same properties file for configuration variables: nmsdns.props. These three programs also use the same shared package: nmsdns.pl. This package contains procedures (both shared and unshared) used by the three programs.

### DADA Secondary Steps

The role and goals of the nmsdnsd.sh script on behalf of the secondary DADA server differs from the first. While primary server receives NMS-DNS tables and configuration based on its OpenView, the secondary retrieves the tables and configuration from the primary. Also unlike the section for the primary server, the section for the secondary's logic is contained entirely in the nmsdnsd.sh script.

Initially, nmsdnsd.sh program confirms that the server is the secondary, backups up the NMS-DNS configuration and inventory, and finally wipes out the current NMS-DNS configuration. The program then contacts the primary server (using rsh or ssh.) It tars and compresses the files on the primary. It then copies the compressed file to the secondary, using rcp or scp. Once the file is copied, the compressed file is expanded on the secondary. The final step is to move into place the personalized header file that distinguishes the secondary server from the primary server. This file is referenced by each of the forward and reverse lookup data files.

## Conclusion

The NMS-DNS architecture consists of five components: NMS applications, OpenView (inventory), DNS configuration, the Distributed/Automated DNS Administration (DADA), and NOC Operators. This section has described the design and purpose of these various components and their interoperability. The next section, As-Built, describes some of these components in greater detail to assist in the maintenance and enhancement of the DADA solution.

# As-Built

## Introduction

|  |  |  |  |
| --- | --- | --- | --- |
| **Server** | **Type** | **File** | **Description** |
| Shared | Program | cron | Once a day cron starts the NMS-DNS rebuild process on the primary and secondary server. |
| Shared | Program | nmsdnsd.sh | The primary NMS-DNS rebuild program. |
| Primary | Program | nmsdnsfeed.pl | Extracts OpenView information to build an inventory list. |
| Primary | Program | nmsdnsclean.pl | Detects and corrects errors in the OpenView inventory list. |
| Primary | Program | nmsdnscreate.pl | Builds the NMS-DNS configuration and inventory based on the cleaned inventory list. |
| Primary | Program | nmsdns.pl | Procedures used by nmsdsnfeed.pl, nmsdnsclean.pl, and nmsdnscreate.pl. |
| Primary | Config | nmsdns.props | Variables used by nmsdnsfeed.pl, nmsdnsclean.pl, and nmsdnscreate.pl. |
| Primary | Log | nmsdns.log | Log and trace information used to debug the nmsdnsfeed.pl, nmsdnsclean.pl, and nmsdnscreate.pl programs |
| Primary | Log | nmsdns.warn | Full list of inventory errors detected by the nmsdnsfeed.pl, nmsdnsclean.pl, and nmsdnscreate.pl programs |
| Primary | Log | nmsdns.warn2 | Summary of inventory problems categorized by error type. |
| Primary | Device Inventory | nmsdns.h2n | Inventory list produced by nmsdnsfeed.pl and used by nmsdnsclean.pl. |
| Primary | Devices Inventory | nmsdns.host | Inventory list produced by nmsdnsclean.pl and used by nmsdnscreate.pl. |
| Primary | DNS | named.conf | The NMS-DNS configuration file for the named server. |
| Primary | DNS | db.IP-Address | The reverse lookup files, resolving IP address to host and domain name. |
| Primary | DNS | db.Domain | The forward lookup files, resolving host and domain name to an IP address. |
| Shared | Log | Standard-Out | Log and trace information produced by nmsdnsd.sh, used to debug the script. |
| Primary | Backup | nmsdns.[DAY].tar.Z | Backup of the data files and log files. |
| Primary | Backup | dns.tar.Z | Backup of the data files |

The complete DADA architecture consists of 16 files, including programs, configuration, logs, inventory, and DNS files. (The persistent files are documented in Appendix C.) The following is manifest of these files.

## Programs

Five programs nmsdnsd.sh, nmsdnsfeed.pl, nmsdnsclean.pl, nmsdnscreate.pl, and nmsdns.pl as well as cron embody the desired The company policies for the NMS-DNS solution. While the previous architecture section discussed the high level relationships of these programs, this sub-section describes the lower level tactical aspects that implement this policy.

### cron

The cron program operates on both the primary and secondary. A typical cron configuration will run the program on the primary everyday at 2:00 AM. The primary will build the NMS-DNS inventory and configuration files and pushes its data to the secondary.

### nmsdnsd.sh



On both the primary and secondary DADA servers, cron calls the same program to begin the DNS-NMS update - nmsdnsd.sh. This Korn shell program performs the following six steps.

* nmsdnsd.sh determines if the server is local server the primary or secondary. If the server is neither, the program exits unless overridden by providing the primary or secondary server name as an argument on the command line.
* It backs up the existing DNS configuration files and stores these in the form nmsdns.[abbreviated day of the week].tar.Z.
* Performs the logic for either the primary or secondary DADA server.
  + In the case of the primary it builds the NMS-DNS configuration and data based on the local OpenView's inventory.
  + In the case of the secondary, it grabs the NMS-DNS configuration and data from the primary.
* Right before the last step in the primary and secondary logic, the program wipes out the existing NMS-DNS configuration and inventory.
* Finally it stops and starts the named (NMS-DNS) server.

During each step, the nmsdnsd.sh program documents its progress to standard output. This information is directed via the cron entry to /opt/NMSDNS/logs/nmsdnsd.log

### nmsdnsfeed.pl

On the primary the first step in building the new NMS-DNS database is the program called nmsdnsfeed.pl. This program to extracts inventory information from OpenView in order to build the IP to device resolution database. In this role it provides four vital functions:

* Error Detection
* Host and Domain Name Resolution Logic
* Multi-Homed Determination Logic
* Primary Interface Logic (for multi-homed devices)

#### Error Detection

|  |  |
| --- | --- |
| **#** | **Error Message** |
| 1 | WARN Please update <PROPSFILE> with correct abbreviation for interface type <INTERFACE TYPE> (found in <INTERFACE NAME>) |
| 2 | WARN Unrecognized interface discovered with invalid characters (not a-z, A-Z, 0-9 or '-'): <INTERFACE NAME>. Interface name set to INVALID. Program nmsdnsfeed.pl in procedure IntNameClean() needs to be updated to detect this interface type. |
| 3 | WARN Unrecognized interface discovered: <INTERFACE NAME>. This will be left \"as-is. |
| 4 | WARN Program nmsdnsfeed.pl in procedure IntNameClean() needs to be updated to detect this interface type: <INTERFACE NAME>. |
| 5 | WARN The configuration file <PROPSFILE> does not exist. |
| 6 | WARN Unrecognized interface discovered: <INTERFACE NAME> Interface name set to INVALID. Program nmsdnsfeed.pl needs to be updated to detect this interface type. |
| 7 | WARN: <DEVICE NAME> has no SNMP address to poll inside OpenView. |
| 8 | WARN: <DEVICE NAME> <SNMP IP Address> encountered the following error with the SNMP string: <ERROR>.\*\* |
| 9 | WARN: $ipaddress has no identifiable node name that can be used for DNS. |
| 10 | WARN: DNS has no recorded domain for <DEVICE NAME> (<IP ADDRESS>). |

\*\* <ERROR> can be: SNMP-STRING-UNKNOWN-TO-OPENVIEW, SNMP-AGENT-DEAD, SNMP-SYSNAME-INVALID, SNMP-SYSNAME-NOT-SET, SNMP-STRING-UNKNOWN-ERROR, or SNMP-IP-ADDRESS-INVALID

#### Host and Domain Name Resolution Logic

For every node, the following logic occurs.



#### Multi-homed Logic

Before the name to IP address resolution can be determined the device needs to be identified as either multi-homed or not. For every node the following logic occurs.



The actual switches used in the multi-homed determination come from OpenView and are configurable via the /opt/NMSDNS/config/nmsdns.props file.

## IF A NODE IS MULTIHOMED. IF THE SWITCH IS 1 AND THE OPENVIEW FIELD VALUE

## IS TRUE, THEN THE NODE IS MULTIHOMED

$MULTIHOMED->{"isIPRouter"} = 1;

$MULTIHOMED->{"isSwitch"} = 1;

The list of switches that can be used are obtainable by setting

$DEBUGLOG = $D\_MH;

By running in /opt/NMSDNS/scripts/nmsdnsfeed.pl, the list of possible switches will be stored in the /opt/NMSDNS/logs/nmsdns.log file.

#### Primary Interface and Abbreviation Logic

If a device is determined to be multi-homed, then only one interface will represent the node. The other interfaces will be represented by the device name connected by hyphen to an abbreviation of the interface name and interface index. Any ':' or '.' in the index is converted to a '-', while any '/' is removed from the index entirely.



The hierarchy of the interface type is configurable. The /opt/NMSDNS/config/nmsdns.props file contains the order through which these interfaces are selected. The lower the number, the greater the priority. The text representing the interface type is an abbreviation which is determined before the table is referenced.

## INTERFACE HIERARCHY (BASED ON ABBREVIATIONS)

$IF->{"lp"} = 1;

$IF->{"sc"} = 2;

$IF->{"nt"} = 3;

$IF->{"vl"} = 4;

$IF->{"e"} = 5;

$IF->{"s"} = 6;

$IF->{"pix"} = 7;

As new interfaces are detected, the corresponding abbreviations can be added to the /opt/NMSDNS/config/nmsdns.props file. Initially this file contains:

## INTERFACE CLEANING

$INTABREV->{"agent's"} = "unk"; # 1

$INTABREV->{"amd"} = "hst"; # 2

$INTABREV->{"atm"} = "atm"; # 3

$INTABREV->{"bvi"} = "bvi"; # 4

$INTABREV->{"channel"} = "ch"; # 5

$INTABREV->{"cabletron"} = "ctron"; # 6

$INTABREV->{"cayman"} = "hst"; # 7

$INTABREV->{"cisco"} = "csco"; # 8

$INTABREV->{"compaq"} = "hst"; # 9

$INTABREV->{"ctron"} = "ctron"; # 10

$INTABREV->{"dc"} = "dc"; # 11

$INTABREV->{"dec"} = "hst"; # 12

$INTABREV->{"dialer"} = "dial"; # 13

$INTABREV->{"el574nd4"} = "e"; # 14

$INTABREV->{"el3c589"} = "e"; # 15

$INTABREV->{"enet"} = "enet"; # 16

$INTABREV->{"en"} = "en"; # 17

$INTABREV->{"eri"} = "eri"; # 18

$INTABREV->{"eth"} = "e"; # 19

$INTABREV->{"ethernet"} = "e"; # 20

$INTABREV->{"ether"} = "e"; # 21

$INTABREV->{"fastethernet"} = "fe"; # 22

$INTABREV->{"fddi"} = "fddi"; # 23

$INTABREV->{"fei"} = "fei"; # 24

$INTABREV->{"ge"} = "ge"; # 25

$INTABREV->{"gigabitethernet"} = "ge"; # 26

$INTABREV->{"hme"} = "hme"; # 27

$INTABREV->{"hssi"} = "hssi"; # 28

$INTABREV->{"interface"} = "hst"; # 28

$INTABREV->{"ibm"} = "hst"; # 29

$INTABREV->{"intel(r)"} = "hst"; # 30

$INTABREV->{"intel"} = "hst"; # 31

$INTABREV->{"ip"} = "hst"; # 32

$INTABREV->{"compaq"} = "hst"; # 33

$INTABREV->{"loopback"} = "lp"; # 34

$INTABREV->{"madge\_tr\_miniport\_driver"} = "tr"; # 35

$INTABREV->{"madge\_tr\_miniport\_driver-"} = "tr"; # 36

$INTABREV->{"network"} = "nt"; # 37

$INTABREV->{"qfe"} = "qfe"; # 38

$INTABREV->{"pix"} = "pix"; # 39

$INTABREV->{"sc"} = "sc"; # 40

$INTABREV->{"serial"} = "s"; # 41

$INTABREV->{"lan"} = "lan"; # 42

$INTABREV->{"smb"} = "hst"; # 43

$INTABREV->{"smc"} = "hst"; # 44

$INTABREV->{"tokenring"} = "tr"; # 45

$INTABREV->{"token\_ring"} = "tr"; # 46

$INTABREV->{"token-ring"} = "tr"; # 47

$INTABREV->{"token"} = "tr"; # 48

$INTABREV->{"tr"} = "tr"; # 49

$INTABREV->{"tunnel"} = "tu"; # 50

$INTABREV->{"tms380c26"} = "tms"; # 51

$INTABREV->{"virtual"} = "vt"; # 52

$INTABREV->{"visual"} = "hst"; # 53

$INTABREV->{"ip"} = "ip"; # 54

$INTABREV->{"vlan"} = "vl"; # 55

### nmsdnsclean.pl

This program surveys the output from NMSDNSFEED, cleans it, and reports any detected errors.

### nmsdnscreate.pl

The nmsdnscreate.pl program takes the scrubbed data from nmsdnsclean.pl and creates the DNS data files and configuration file. For every node the program uses the following logic to create A, CNAME, and PTR DNS records.



\* The multi-homed logic loop in NMSDNSFEED.pl is used to determine whether or not the node is multi-homed.

\*\* The primary interface logic loop in NMSDNSFEED.pl is used previously to determine which interface is the primary one for multi-homed devices.

## Configuration Files

There are three primary configuration files: nmsdns.conf, db.header, and db.cache.

### db.header

The db.header contains header information that is included in all the forward and reverse lookup files. It specifies the local NMS-DNS server, an email contact, and the pair of NMS-DNS servers.

$TTL 86400

@ SOA OpenViewPrimary.company.com. NMSADMIN.company.com. ( 1 10800 3600 604800 600 )

NS OpenViewPrimary.company.com.

NS OpenViewSecondary.company.com.

For each primary and secondary server, the db.header file needs to be updated with the three server references and the email contact.

### db.cache

The db.cache contains pointers to all the root servers. Since the NMS-DNS solution is isolated and does not use recursive inquires, the file is not actually used. However, bind requires it in order to run.

### nmsdns.props

All the configuration parameters for the PERL scripts in the NMS-DNS solution are defined in nmsdns.props. The multi-homed, if, and intabrev hashes are set here and have been previously described. In addition the files and directories used by the NMS-DNS solution are also defined here:

$LOGDIR = "/opt/NMSDNS/logs";

$PRGDIR = "/opt/NMSDNS/scripts";

$CFGDIR = "/opt/NMSDNS/config";

$OLDDIR = "/opt/NMSDNS/old";

$DNSDIR = "/opt/NMSDNS/db";

$HSTDIR = "/opt/NMSDNS/logs";

$DEFAULTNAME = "nmsdns";

## FILES: PROGRAM

#$DAEMONFILE = "$PRGDIR/$DEFAULTNAME" . "d.pl";

#$FEEDFILE = "$PRGDIR/$DEFAULTNAME" . "feed.pl";

#$CLEANFILE = "$PRGDIR/$DEFAULTNAME" . "clean.pl";

#$CREATEFILE = "$PRGDIR/$DEFAULTNAME" . "create.pl";

## FILES: CONFIGURATION

$PROPSFILE = "$CFGDIR/$DEFAULTNAME" . ".props";

$DBCACHEFILE = "$CFGDIR/db.cache";

$DBHEADERFILE = "$CFGDIR/db.header";

## FILES: LOG

$LOGFILE = "$LOGDIR/$DEFAULTNAME.log"; # LOG FILE

$WARNFILE = "$LOGDIR/$DEFAULTNAME.warn"; # WARN MESSAGES

$WARNFILE2 = "$LOGDIR/$DEFAULTNAME.warn2"; # WARN MESSAGES

## FILES: HOST DATABASES

$HOSTFILE = "$HSTDIR/$DEFAULTNAME.host"; # HOST FILE

$H2NFILE = "$HSTDIR/$DEFAULTNAME.h2n"; # H2N CMD FILE

$EXTRAFILE = "$HSTDIR/$DEFAULTNAME.extra";# MANUAL ENTRIES

## FILES: DNS FILES

$NAMEDFILE = "$DNSDIR/named.conf"; # HOST FILE

#$PTRFILE = "$DNSDIR/db.X.X.X.X"; # REVERSE LOOKUP FILES

#$ACNAMEFILE = "$DNSDIR/db.X.X.X.X"; # FORWARD LOOKUP FILES

Depending on the situation some of these files are opened read write and later read only. The following variables are used to track request to write to the files:

## FILE DESCRIPTOR SWITCHES

$LOGFD = 1;

$HOSTFD = 2;

$H2NFD = 4;

$WARNFD = 8; # USED FOR BOTH WARN FILES

$NAMEDFD = 16;

When the domain for a device cannot be determined, a default domain is used. This parameter allows the value used to be changed:

## IDENTIFYING INFO

$DEFAULTDOMAIN='company.com';

As part of the name and domain determination the company’s DNS server is used. The following parameter is used to point to the correct server:

## DNS SERVER INFO

$DNSSERVER = '10.10.10.15';

Due to a bug in OpenView, the sysname is not retained by the program. Thus, the NMS-DNS server must extract this information using SNMPWALK. The parameter to this program can be configured using the following variable:

## SPECIFIC FOR FEED

## SNMPWALK PARAMETERS

$SNMPTIMEOUT = 2;

$SNMPRETRIES = 2;

$SNMPCOMMUNITYSTRING = "public";

Several constants are also defined. The value used by NULL in the device tables as well as the values for true and false are defined:

## CONSTANT

$NullValue = "na";

use constant TRUE => 1;

use constant FALSE => 0;

The final configuration section specifies various types of output and to which files the output is directed. The files include: the general log, the host output from nmsdnsclean.pl, the h2n output from nmsdnsfeed.pl, the general warning file, the summary warning file (warn2), and named configuration file. There are fourteen different types of messages that can be directed to these files. They include:

|  |  |  |
| --- | --- | --- |
| **#** | **BIT** | **Description** |
| 1 | $D\_GOOD | Output good interfaces and hosts. |
| 2 | $D\_HOST | Host file output |
| 3 | $D\_NAMED | Named configuration file output |
| 4 | $D\_ADMDOWN | Output admin down interfaces and hosts |
| 5 | $D\_INVALID | Output errored interfaces and hosts |
| 6 | $D\_INT | Basic process tracing |
| 7 | $D\_MAIN | More process tracing commands |
| 8 | $D\_LOGFILE | In-process interface listing |
| 9 | $D\_HOSTNAME | Detailed in-process interface listing |
| 10 | $D\_MH | Multihome detection. |
| 11 | $D\_1STPASS | Trace 1st stage: Parse data for host info |
| 12 | $D\_PROCESS | Trace 2nd stage: Process host info |
| 13 | $D\_2NDPASS | Trace 3rd stage: Parse data for interface info |
| 14 | $D\_WARN | WARNING messages - raw. |
| 15 | $D\_WARN2 | WARNING messages - summarized. |
| 16 | $D\_H2N | Node level FIRST PASS tracing. |

These bits are assigned to the debug buffers representing the files:

|  |  |
| --- | --- |
| **Buffer** | **File** |
| $DEBUGLOG | Log file |
| $DEBUGHOST | Cleaned inventory |
| $DEBUGH2N | Raw inventory |
| $DEBUGWARN | Individual detailed warning messages |
| $DEBUGWARN2 | Errored devices and domains grouped by warning message type |
| $DEBUGNAMED | Named configuration file |

There are two general parameters as well. DEBUGMAX is used to redirect all output to a file. DEBUG is used to shut of a particular set of bits for all output. If the bit is not set in DEBUG, the output represented by that bit will not be directed to any file.

Together these parameters for the debug section of the configuration file:

######################## DEBUG PARAMETERS ########################

## DEBUG RULES:

## DEBUG manages all the output from this program.

## 1. DEBUGMAX represents the number if all bits were turned on and provides

## an easy way to turn up all bits for [OUTPUT STATEMENT], DEBUG, and

## [MASTER OUTPUT FILE SWITCHES].

## 2. [OUTPUT STATEMENT] switch statements categorizes the type of output.

## Of these the leading switches are used to specify output associated

## with the classification of device.

## 3. DEBUG is a master switch which can globally turn off output of

## some of the OUTPUT STATEMENTS by setting their bit to 0.

## 4. [MASTER OUTPUT FILE SWITCHES] switch statements are used to funnel

## [OUTPUT STATEMENT] types to the correct output.

###################################################################

## OUTPUT STATEMENT SWITCHES

## MAXIMUM DEBUG LEVEL

$DEBUGMAX=(2\*\*14)-1;

## TYPE OF DEVICES THAT CAN BE SAVED IN HOST FILE AND H2N FILE

$D\_GOOD = 1; ## 1 Output good interfaces and hosts.

$D\_HOST = 1; ## 1 Host file output

$D\_NAMED = 1; ## 1 Host file output

$D\_ADMDOWN = 2; ## 2 Output admin down interfaces and hosts

$D\_INVALID = 4; ## 3 Output errored interfaces and hosts

## BASIC PROCESS FLOW DEBUGING

$D\_INT = 8; ## 4 Basic process tracing

$D\_MAIN = 16; ## 5 More process tracing commands

## SPECIAL DEBUGGING

$D\_LOGFILE = 32; ## 6 In-process interface listing

$D\_HOSTNAME = 64; ## 7 Detailed in-process interface listing

$D\_MH = 128; ## 8 Multihome detection.

## DETAILED PROCESS FLOW DEBUGING

$D\_1STPASS = 256; ## 9 Trace 1st stage: Parse data for host info

$D\_PROCESS = 512; ## 10 Trace 2nd stage: Process host info

$D\_2NDPASS = 1024; ## 11 Trace 3rd stage: Parse data for interface info

## ERROR DETECTION OUTPUT

$D\_WARN = 2048; ## 12 WARNING messages - raw.

$D\_WARN2 = 4096; ## 13 WARNING messages - summarized.

## DATA OUTPUT

$D\_H2N = 8192; ## 14 Node level FIRST PASS tracing.

## MASTER OUTPUT FILE SWITCHES

$DEBUG = $DEBUGMAX; # MASTER SWITCH. OUTPUT ONLY OCCURS IF

# THESE BITS ARE ON. DEFAULT: ALL ON

$DEBUGLOG = $D\_INT | $D\_MAIN | $D\_LOGFILE | $D\_WARN | $D\_H2N;

#$DEBUGLOG = $DEBUGMAX; # LOG FILTER. LOG STATEMENTS TO $LOGFILE

$DEBUGHOST = $D\_HOST; # CLEANED INVENTORY

$DEBUGH2N = $D\_H2N; # RAW INVENTORY FROM OPENVIEW

$DEBUGWARN = $D\_WARN; # WARN MESSAGES TO $WARNFILE

$DEBUGWARN2 = $D\_WARN2; # WARN MESSAGES TO $WARNFILE

$DEBUGNAMED = $D\_NAMED; # HOST FILE

####################################################################

## Log, Inventory, and DNS Files

The debug parameters enable configuration of which statements go to which files. There are three files or file sets for each area:

The /opt/NMSDNS/config/nmsdns.props specifies three log files:

## FILES: LOG

$LOGFILE = "$LOGDIR/$DEFAULTNAME.log"; # LOG FILE

$WARNFILE = "$LOGDIR/$DEFAULTNAME.warn"; # WARN MESSAGES

$WARNFILE2 = "$LOGDIR/$DEFAULTNAME.warn2"; # WARN MESSAGES

The log file contains trace and debug information. The warn file contains individual problems discovered in the inventory. The warn file 2 contains a list of devices or domains by error type, summarizing the content of the warn file.

The /opt/NMSDNS/config/nmsdns.props specifies three database files:

## FILES: HOST DATABASES

$HOSTFILE = "$HSTDIR/$DEFAULTNAME.host"; # HOST FILE

$H2NFILE = "$HSTDIR/$DEFAULTNAME.h2n"; # H2N CMD FILE

$EXTRAFILE = "$HSTDIR/$DEFAULTNAME.extra";# MANUAL ENTRIES

The h2nfile represent the inventory outputted from nmsdnsfeed.pl and is used by nmsdnsclean.pl. The hosts file represents the inventory outputted by nmsdnsclean.pl and is used by nmsdnscreate.pl to create the DNS database files and configuration file.

The /opt/NMSDNS/config/nmsdns.props specifies three sets of DNS files:

## FILES: DNS FILES

$NAMEDFILE = "$DNSDIR/named.conf"; # HOST FILE

#$PTRFILE = "$DNSDIR/db.X.X.X.X"; # REVERSE LOOKUP FILES

#$ACNAMEFILE = "$DNSDIR/db.X.X.X.X"; # FORWARD LOOKUP FILES

The named file is the DNS configuration file, providing a schema for the forward and reverse lookup files. The PTRFILE (PTR files) represent the reverse lookup files while the ACNAMEFILE (A and CNAME file) represents the reverse lookup files.

# NMS-DNS Installation

## Package Inventory

The following components are required for the NMS-DNS architecture:

|  |  |  |
| --- | --- | --- |
| **Software** | **Version** | **Size** |
| UNIX AIX (with scp and ssh) | 5.1 | 0\*\* |
| OpenView (with snmpwalk) | 7.1 | 0\*\* |
| Perl | 5.6.0 (built in to OS) | 0\*\* |
| Bind (DNS Server) | 9.2.2 | 64 meg |
| NMS-DNS (Custom software) | 1.0 | 46 K (200K uncompressed) |

OpenView and PERL are already installed on UNIX AIX. The bind binary file can be downloaded from:

[http://aixpdslib.seas.ucla.edu/packages/bind.html](http://www.sunfreeware.com)

In the case of AIX, if the file system needs to be expanded, use SMIT and perform the following. Select system storage -> file systems -> add/change/show/delete file systems -> Journaled file systems (JFS) -> change/show characteristics. Next, pick the right file system from the list and then increase its size and select OK. Once the prompt returns, select cancel. (DON'T PRESS OK OR THE ALLOCATION WILL OCCUR AGAIN.)

## Obtain server information (Both Servers)

Before beginning the following bits of information are required:

|  |  |  |
| --- | --- | --- |
| **Description** | **Command** | **Example Data** |
| Determine CMX (Change Order) | - | - |
| Contact Operations and inform them the OpenView will be bounced and the server rebooted during this CMX. | - | - |
| Get domain | cd /etc; cat resolv.conf | company.com |
| Verify PERL location | which perl | This should match the PERL scripts' first line:  /usr/bin/perl |
| Verify KORN shell location | which sh | This should match the KORN shell script first line. (NOTE: AIX uses /usr/bin/sh to represent KORN shell instead of /usr/bin/ksh) |
| OpenView IP Address | ifconfig -a | 10.10.10.15 |
| OpenView server name | uname -n | OpenViewPrimary |
| Determine if the OpenView server is the primary NMS-DNS or secondary | - | primary |
| Desired starting debug level for the NMS-DNS solution. (The default is probably correct.) | vi /opt/NMSDNS/config/props.conf | $D\_INT|$D\_MAIN|$D\_LOGFILE|$D\_WARN|$D\_H2N |
| Determine if either ssh and scp or rsh and rcp are available issued on the primary against the secondary without authorization confirmation. (ssh/scp is preferred against rsh/rcp.) | cd tmp; touch a; scp a SECONDARY:/tmp; ssh SECONDARY "ls -l /tmp/a" | -rw-r--r-- 1 root system 0 May 13 23:51 /tmp/a |
| Determine who the administrator of NMS-DNS is going to be and obtain their email address. | - | OpenView-Admin@company.com |
| Determine the primary and secondary DNS servers for this environment | cat /etc/resolv.conf | 10.10.10.15 |
| Verify that root privileges are accessible on both the primary and secondary NMS-DNS servers. | sudo su | - |
| Available space on /opt/NMSDNS (26 meg) | df -k | /dev/tvlv03 196608 74204 63% 335 1% /opt/NMSDNS |
| Available space on /usr/local (40 meg) | df -k | /dev/hd2 1835008 289852 85% 34212 8% /usr |
| Available space on /tmp or other temporary location (40 meg) | df -k | /dev/hd3 507904 458332 10% 120 1% /tmp |
| Location of BIND compressed and tared binary files | - | OpenViewSecondary:/tmp/bind.9.2.2.tar.Z |
| Location of NMS-DNS compressed and tared files | - | OpenViewSecondary:/tmp/nmsdns.tar.Z |

## Contact User (Both Servers)

Contact support.

For example, sometimes issuing a wall command to notifying users is sufficient:

echo "Please logoff immediately. Change order CMX# is in effect. NMSADMIN Ext XXXX" | wall

Verify that OVW is down:

ps -ef | grep ovw

## Verify OVW Starts. (Both Servers)

Verify that OVW starts and then exit.

## Install DNS (bind) software. (Both Servers)

The bind package untars at the root directory under the /usr/local file system per the installation instructions. The bind package requires around 50 megs of space under /usr with an additional 50 megs available on this or another file system to enable the unbundling and configuring of the files.

cd /tmp

<Download bind package to /tmp>

cd /

zcat /tmp/bind.9.2.2.tar.Z | tar xvf -

rm /tmp/bind.9.2.2.tar.Z

## Install NMS-DNS scripts and directories. (Both Servers)

As root a new directory should be created with the command:

mkdir /opt/NMSDNS

Verify that the directory is made with the following command:

ls -l /opt/NMSDNS

Next the NMS-DNS directories and files can be unbundled by untaring the 81 K compressed package through the following series of commands:

cd /tmp

<Download NMS-DNS package to /tmp>

cd /opt/NMSDNS

zcat /tmp/nmsdns.tar.Z | tar xvf -

This will create the following directory structure:

/opt/NMSDNS/logs

/opt/NMSDNS/scripts

/opt/NMSDNS/config

/opt/NMSDNS/db

/opt/NMSDNS/old

The log directory will hold the log files; the scripts directory holds all the NMS-DNS scripts, the config directory holds the NMS-DNS configuration files and the DNS files: db.cache and db.header.

Due to the possible expansion of the log files, the /opt file system should have at least 25 megs of additional space AFTER the unbundling of this package.

## Save Files. (Both Servers)

Save the old configuration files with the following commands:

cd /opt/NMSDNS/config

cp -p /.profile profile.old

cp -p /.nslookuprc nslookuprc.old

cp -p /etc/rc.tcpip rc.tcpip.old

## Configure NMS-DNS (Both Servers)

Once the files are in place, they need to be configured:

/opt/NMSDNS/config/db.header file need to be updated with:

* The local NMS-DNS server name and domain for the SOA record
* The correct email address for the NMS-DNS admin
* The correct NMS-DNS servers and domains for the NS records

## Configure NMS-DNS (Primary Server Only)

/opt/NMSDNS/config/nmsdns.cron documents the root cron table including the NMS-DNS component. Create a new file with the following commands:

cd /opt/NMSDNS/config

crontab -l > tmp

tail -2 nmsdns.cron >> tmp

mv tmp nmsdns.cron

chmod 755 nmsdns.cron

The NMS-DNS cron process runs only on the primary NMS-DNS server.

/opt/NMSDNS/config/nmsdns.props provides the global properties for all PERL based NMS-DNS programs. If desired the $DEBUGLOG variable can be changed to increase or decrease the amount of trace and debug information. The following are some examples:

$DEBUGLOG = 0; # No tracing.

$DEBUGLOG = $D\_INT|$D\_MAIN|$D\_LOGFILE|$D\_WARN|$D\_H2N; # Moderate tracing

$DEBUGLOG = $DEBUGMAX; # Maximum tracing

In addition the DNS server for the company needs to be added to this configuration. This server is documented in the /etc/resolv.conf file. For example the line could be configured to the following:

$DNSSERVER = 10.10.10.13; # Primary Lucent DNS server.

NOTE: Make sure the DNSSERVER is for the company’s DNS server and NOT the local NMS-DNS box.

Also in the same file you can configure the default domain used if the correct domain for the device is unknown. For example,

$DEFAULTDOMAIN = 'company.com'; # Default domain if domain unknown.

/opt/NMSDNS/scripts/nmsdnsd.sh needs to be updated with the proper primary and secondary NMS-DNS server names as well as the email contact information at the top of the file.

## SERVER IDENTIFICATION

PRIMARY="OpenViewPrimary"

SECONDARY="OpenViewSecondary"

#PRIMARY="OpenViewPrimary"

#SECONDARY="OpenViewSecondary"

## REMOTE COMMAND TYPES

RSH="rsh"

RCP="rcp"

#RSH="ssh"

#RCP="scp"

## ADMINISTRATION CONTACT

CONTACT='OpenView-Admin@company.com'

## Configure NSLOOKUP (Both Servers)

In the user's root directory you can create the file /.nslookuprc. To change the timeout from 5 seconds to 1 seconds enter the following line in this file:

set timeout=1

set retry=1

## Configure Resolver (Both Servers)

Next on each of the UNIX servers the /etc/resolv.conf file was created specifying the two OpenView servers as well as the corporate DNS server. To determine the new entries to the resolv.conf, run the following command on each of the OpenView servers:

ifconfig -a

Identify the IP address and add this to the resolv.conf file. For example the updated file could read:

nameserver 10.10.10.13

nameserver 10.10.10.14

nameserver CORPORATE-DNS-RESOLVER-ADDRESS

domain company.com

While testing takes place make sure to change the resolv.conf file to be writeable to everyone to enable changes in case root access is lost.

chmod 777 /etc/resolv.conf

Next, verify the DNS servers are valid with the following commands:

nslookup 10.10.10.13

nslookup 10.10.10.14

ping 10.10.10.13

ping 10.10.10.14

## Configure OpenView (Both Servers)

In order to get the correct timeout two environment variables need to be configured for OpenView. (These can be configured generally for the root user -- see UNIX Root user startup scripts below.) Since these are merely environmental variables, they can also be placed in the OpenView and OpenView's NNM startup file. Specifically, using vi place the following in /usr/OV/bin/OpenView.pre:

RES\_RETRY=1

RES\_TIMEOUT=1

export RES\_RETRY RES\_TIMEOUT

To propagate these changes into OpenView's node manager configuration file. First verify the file doesn't exist:

ls /usr/OV/bin/netnmrc.pre

Then use the following commands:

cp -p /usr/OV/bin/OpenView.pre /usr/OV/bin/netnmrc.pre

chmod 755 /usr/OV/bin/OpenView.pre /usr/OV/bin/netnmrc.pre

Alternatively if the files does exist, simply use vi to add the same lines to this file as you added to the file /usr/OV/bin/OpenView.pre:

RES\_RETRY=1

RES\_TIMEOUT=1

export RES\_RETRY RES\_TIMEOUT

In addition the Universal Password Manager (UPM) servers need to be identified and added to OpenView. These servers provide distributed sudo services. Since the NMS-DNS is authoritative for company.com, if the UPM servers are not in OpenView, the upm scripts will fail to find the appropriate servers. This is a very important step since no access to these servers mean that no one can obtain the root password via UPM. Since these servers are so critical they should also be added to the /etc/hosts file so that in the case of an OpenView corruption, the root password is still obtainable.

Finally, the Cisco works servers and other desired devices should be added to OpenView and possibly the host file as well.

## Configure Named Server (Both Servers)

Then the /var/run directory is required by the named (NMS-DNS) server and can be created with the commands:

mkdir /var/run

chmod 755 /var/run

## Configure UNIX Root User Startup Script (Both Servers)

In the /.profile configuration file for the root user directories were appended to the PATH and MANPATH variables:

# ADDED /usr/local/bin and /usr/local/sbin for NAMED (DNS) executable

PATH=/usr/local/bin:/usr/local/sbin:/usr/OV/bin:$PATH

# ADDED /usr/local/man for NAMED (DNS) executable man page

MANPATH=/usr/local/man:$MANPATH

export PATH MANPATH

The DNS resolver can take an extra ordinary time before timing out on a device name or IP address lookup. However, two environment variables can be configured to reduce the delay.

RES\_TIMEOUT Overrides the default value of the retrans field of the \_res structure, which is the value of the RES\_TIMEOUT constant defined in the /usr/include/resolv.h file. This value is the base time-out period in seconds between queries to the name servers. After each failed attempt, the time-out period is doubled. The time-out period is divided by the number of name servers defined. The minimum time-out period is 1 second.

RES\_RETRY Overrides the default value for the retry field of the \_res structure, which is 4. This value is the number of times the resolver tries to query the name servers before giving up. Setting RES\_RETRY to 0 prevents the resolver from querying the name servers.

These variables should be add to the /.profile:

## ADD TO PREVENT LONG TIMEOUTS FOR DNS QUERIES

RES\_RETRY=1

RES\_TIMEOUT=1

export RES\_RETRY RES\_TIMEOUT

## Configure Cron (Primary server only)

Earlier the nmsdns.cron file was created. This can be used to create the right cron entry:

cd /opt/NMSDNS/config

crontab -l > cron.old

crontab nmsdns.cron

## Configure Startup Script (Both Servers)

Edit the /etc/rc.tcpip script. Search for:

#start /usr/sbin/named $src\_running

Add beneath it the line:

/usr/local/sbin/named -c /opt/NMSDNS/db/named.conf

NOTE: DO NOT PREPEND "START" TO THIS COMMAND

## Backup Configuration Files (Both servers)

Copy the changed files into the config directory with the following commands

cd /opt/NMSDNS/config

cp -p /etc/rc.tcpip .

cp -p /.nslookuprc .

cp -p /.profile .

cp -p /tmp/nmsdns.tar.Z .

## Special Device Cases

Special properties can be exhibited for certain devices. For example, in the case of some firewalls, OpenView does not use ICMP (ping) but instead use SNMP for availability tests. In these cases these entries need to be fully qualified within the /usr/OV/conf/seedfile. Then:

ovw

ovstop netmon

ovstart netmon

nmdemaindpoll <DEVICE>

## Test Run on Primary

To populate and start the primary NMS-DNS server and push these results to the secondary.

Contact support

Bounce OpenView:

ovstop

ovstart

Update the environment variables

. /.profile

Verify the local resolv.conf entries are commented out for the first run.

Run NMS-DNS daemon

crontab -l | tail -1 | awk '{ print $6,$7,$8,$9 }' | xargs sh

Uncomment the resolv.conf entries.

## Verify OVW Starts. (Both Servers)

Verify that OVW starts

## Test System Startup Script (Both Servers)

Reboot the server and confirm that named started on its own. When the system comes up, enter the command to confirm the named server has come up:

ps -ef | grep named | egrep -v

If the named server is not listed, you can use the following command to turn on debugging and to redirect the output to the screen:

named -c /opt/NMSDNS/db/named.conf -d 11 -f -g

# NMS-DNS Testing

## Introduction

As part of the post installation and troubleshooting of the NMS-DNS solution, there are a number of tests that can be performed. These tests can be broken into basic system checks, and more advanced tests, which verify aspects of the NMS-DNS and OpenView applications.

## Basic System Checks

There are a number of basic checks that should be performed to verify the resources and processes are in place.

Verify the cron job is setup on the primary server for the root user:

crontab -l | tail -2

Verify the named server is running. Repeat this command and verify the process ID and timestamp to verify that the process is not bouncing:

ps -ef | grep named

Verify there is an additional 25 megs of storage available for the file system responsible for the /opt/db directory:

df -k /opt/db

Verify the DNS timeout and resolver switching:

ping NO-SUCH-NODE

ping SERVER-NOT-IN-OPENVIEW-BUT-AVAILABLE-THROUGH-CORPORATE

Verify the files and directories are in place for NMS-DNS:

ls -lR /opt/NMSDNS

The following directory structure should exist:

/opt/NMSDNS/logs

/opt/NMSDNS/scripts

/opt/NMSDNS/config

/opt/NMSDNS/db

/opt/NMSDNS/old

Additionally the following files should be present:

/opt/NMSDNS/config/nmsdns.props

/opt/NMSDNS/config/db.header

/opt/NMSDNS/config/db.cache

/opt/NMSDNS/config/db.cron

/opt/NMSDNS/scripts/nmsdns.pl

/opt/NMSDNS/scripts/nmsdnsclean.pl

/opt/NMSDNS/scripts/nmsdnscreate.pl

/opt/NMSDNS/scripts/nmsdnsd.sh

/opt/NMSDNS/scripts/nmsdnsfeed.pl

If the database has been created, in addition there should be:

/opt/NMSDNS/db/named.conf

/opt/NMSDNS/db/db.header

/opt/NMSDNS/db/db.cache

/opt/NMSDNS/db/db.1\*

/opt/NMSDNS/db/db.\*company\_domains\*

/opt/NMSDNS/logs/nmsdns.h2n

/opt/NMSDNS/logs/nmsdns.host

/opt/NMSDNS/logs/nmsdns.log

/opt/NMSDNS/logs/nmsdns.warn

/opt/NMSDNS/logs/nmsdns.warn2

/opt/NMSDNS/logs/nmsdnsd.log

## Advanced Checks

There are a number of more advanced checks that could be performed to verify the resources and processes are in place. These checks revolve around the prominent aspects of the solution: NMS-DNS and OpenView.

### NMS-DNS

Using grep, awk, and vi grab PTR records and create a series of host commands. Execute these commands and verify the line count of the results matches the line count of the host commands. Any discrepancies should be investigated.

Next, take the results from the host commands and create another command file using host to perform the forward lookup. Compare the line count of the results to the line count of the host commands. Any discrepancies should be investigated.

Finally compare the results from the second test with the input of the first test and verify that they match.

Finally, survey the following content and scan for issues:

/opt/NMSDNS/logs/nmsdns.h2n

/opt/NMSDNS/logs/nmsdns.host

/opt/NMSDNS/logs/nmsdns.log

/opt/NMSDNS/logs/nmsdns.warn

/opt/NMSDNS/logs/nmsdns.warn2

/opt/NMSDNS/logs/nmsdnsd.log

### OpenView

Open ovw and check 10 nodes randomly.

Verify the hostnames in OpenView, using ovtopodump -rl. Make sure the named devices follow the correct conventions. For any devices that still are identified by their IP address, create a series of host commands to identified devices resolved by NMS-DNS, but not by OpenView. Next, manually check these devices and verify that technical or environmental issues prevented correct resolution.

# Appendix A: Dependencies

The following components were used with the NMS-DNS architecture:

|  |  |
| --- | --- |
| **Software** | **Version** |
| UNIX AIX (with scp and ssh) | 5.1 (built in) |
| OpenView (with snmpwalk) | 7.1 (built in) |
| Perl | 5.6.0 (built in) |
| Bind (DNS Server) | 9.2.1 |

OpenView and PERL were already installed on UNIX AIX. The bind binary files were installed under the /usr file system.

# Appendix B: Initial Design Requirements

* DADA
  + The scripts use "scp" to copy the updated OpenView DNS configuration from primary to backup.
  + The scripts use "ssh" to activate the new DNS configuration on the backup
  + Scripts are activated via root users "cron."
  + Performs the creation of the local DNS database daily.
  + Scripts track exceptions and log these to local log files as well as notify the appropriate party.
  + NAME determined by the logic on the following page
  + The determination of whether a monitored device is a switch/router or not is through the SNMP sys OID.
  + Each of the local DNS servers is populated based on the monitored device inventory of the primary OpenView.
  + The domain for the local DNS will be determined by a reverse lookup up (IP address to fully qualified system name) against the company’s DNS.
  + The initial timeout for caching in the local DNS is 48 hours.
  + Five-second timeouts with 2 retries for SNMP response
  + Exception notification should be sent via email to the admin and OpenView event log.
* NMS Client Servers
  + NMS Server resolver implements a three-tier escalation: primary, backup, and the company’s DNS.
  + Two-second timeouts with 2 retries for DNS resolution
* DNS Servers
  + Each of the local DNS servers is populated based on the monitored device inventory of the primary OpenView.
  + The domain for the local DNS will be determined by a reverse lookup up (IP address to fully qualified system name) against the company’s DNS.
  + Each DNS server is authoritative for all the company’s domains.
  + Zone transfers will NOT occur between the local DNS and the company’s DNS.
  + The company DNS servers do not have visibility to one another.
  + Local DNS systems should mirror the production DNS domains for the monitored devices.

# Appendix C: Scripts

## Cron Entry

Cron initiates the rebuilding and restart of the primary and secondary NMS-DNS server.

## CREATE NMS-DNS EVERY HOUR AT 5 AFTER

5 \* \* \* \* /opt/NMSDNS/scripts/nmsdnsd.sh > /opt/NMSDNS/logs/nmsdnsd.log 2>&1

## NMSDNSd.sh

This is the primary program called by cron on the primary NMS-DNS server that rebuilds the database and configuration files before restarting the named server.

#!/usr/bin/sh

##########################################################################

# 03/30/2003 Daniel L. Needles Version 1.0 #

# PROGRAM: nmsdnsd.sh #

# USAGE: nmsdnsd.sh [SERVER NAME] #

# #

# PURPOSE: Repopulates and restarts named server #

# Code currently supports only the primary server. #

# 1. Backup of DNS files taken on the primary #

# 2. DNS files replaced on the primary #

# 3. Primary Named server is restarted. #

# 4. DNS files on the primary are pushed to the secondary #

# 5. Backup of DNS files taken on secondary. #

# 6. DNS files replaced on the secondary #

# 7. Secondary Named server is restarted. #

# NOTE: rsh does not like to run in the background. This script must #

# run in the foreground. #

##########################################################################

# DLN20030507: Instead of having the primary build files and have the #

# secondary pull the files independently, now have the #

# primary build the files and immediately push them to #

# the secondary. #

# DLN20030507: Corp (sa7Xnetv1) uses rsh/rcp instead of ssh/scp. However #

# rsh only allows individual commands and not interactive #

# input. Accordingly the code was changed to be not interact#

##########################################################################

# DLN20030520: Production release. #

##########################################################################

## SERVER IDENTIFICATION

PRIMARY="PRIMARY"

SECONDARY="SECONDARY"

## REMOTE COMMAND TYPES

RSH="rsh"

RCP="rcp"

#RSH="ssh"

#RCP="scp"

## ADMINISTRATION CONTACT

CONTACT='NMSADMIN@company.com'

## DOUBLE CHECKING FOR DIRECTORIES REMOVED WITH rm -r

DNSCHECK="/opt/NMSDNS/db"

## DIRECTORY STRUCTURE

DEFAULTNAME="nmsdns"

ALLDIR="/opt/NMSDNS/DNS"

PRGDIR="/opt/NMSDNS/scripts"

CFGDIR="/opt/NMSDNS/config"

DNSDIR="/opt/NMSDNS/db"

OLDDIR="/opt/NMSDNS/old"

LOGDIR="/opt/NMSDNS/logs"

#HSTDIR="/opt/NMSDNS/logs"

## NMS-DNS CREATION PROGRAMS

## XXX: SWITCH COMMENT TO SKIP PRIMARY STEPS

FEED="$PRGDIR/nmsdnsfeed.pl"

CLEAN="$PRGDIR/nmsdnsclean.pl"

CREATE="$PRGDIR/nmsdnscreate.pl"

#FEED="echo echo"

#CLEAN="echo echo"

#CREATE="echo echo"

##########################################################################

## GET TO THE RIGHT PLACE

cd $PRGDIR

## GET LOCAL NAME

A=$(uname -n)

B=$(whoami)

## CHECK IF ROOT

if [ $B != "root" ]

then

echo "ERROR: $0: Must be root to run $0"

exit -1

fi

## OVERRIDE LOCAL NAME

if [ $# -eq 1 ]

then

A=$1

fi

## DETERMINE WHICH NAME WE HAVE

if [ $A = $PRIMARY ]

then

A="PRIMARY"

elif [ $A = $SECONDARY ]

then

A="SECONDARY"

echo "ERROR: $0: Program must be run on the primary OpenView server ($PRIMARY) NOT the secondary ($SECONDARY)."

exit -2

else

echo -n "ERROR: $0: Program must be run on a primary OpenView server ($PRIMARY) NOT "

`uname -n`

exit -3

fi

if [ $DNSDIR = $DNSCHECK ]

then

echo "PASSED: $DNSDIR"

else

echo "ERROR: $0: Fatal warning. $DNSDIR will be removed with rm -r. Correct script to accept this."

exit -4

fi

## ANNOUNCE SERVER

echo ""

echo "\*\*\*\*\*\*\*\*\* BEGIN WORK ON $A \*\*\*\*\*\*\*\*\*"

date

## BACK STUFF UP

echo ""

echo "\*\*\*\*\*\*\* SAVE CURRENT DNS ENVIRONMENT \*\*\*\*\*\*\*"

cd $ALLDIR

F="nmsdns.$(date | awk '{ print $1 }').tar"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#tar cvf $F $PRGDIR/named.run $DNSDIR/db.\* $DNSDIR/named.conf $LOGDIR/nmsdns\*

echo "tar cvf $F $PRGDIR $DNSDIR $LOGDIR"

tar cvf $F $PRGDIR $DNSDIR $LOGDIR

wait $!

compress -f $F

mv $F.Z $OLDDIR

## BENCHMARK CURRENT STATE

echo ""

echo "\*\*\*\*\*\*\* DOCUMENT BENCHMARK \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

ps -ef | grep named | egrep -v grep

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

ls -lR $ALLDIR

## CLEAR PAST LOGS

## XXX: COMMENT OUT TO SKIP PRIMARY STEPS

rm $PRGDIR/named.run $LOGDIR/nmsdns.h2n $LOGDIR/nmsdns.host $LOGDIR/nmsdns.log 2>/dev/null

rm $LOGDIR/nmsdns.warn $LOGDIR/nmsdns.warn2 2>/dev/null

cd $PRGDIR

echo "\*\*\*\*\*\*\* GRAB DATA FROM LOCAL PRIMARY OPENVIEW \*\*\*\*\*\*\*"

date

$($FEED)

wait $!

sleep 1

echo "\*\*\*\*\*\*\* SCRUB OPENVIEW DATA AND NOTE PROBLEMS \*\*\*\*\*\*\*"

date

$($CLEAN)

wait $!

sleep 1

## CLEAN CURRENT ENVIRONMENT

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#rm $DNSDIR/db.\* $DNSDIR/named.conf

## XXX: COMMENT OUT TO SKIP PRIMARY STEPS

echo 'y' | rm -r $DNSDIR

wait $!

mkdir $DNSDIR

## PROVE IT IS CLEAN

echo ""

echo "\*\*\*\*\*\*\* DOCUMENT THAT IT IS NOW CLEAN \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

ps -ef | grep named | egrep -v grep

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

ls -lR $ALLDIR

echo "\*\*\*\*\*\*\* CREATE NEW DNS ENVIRONMENT \*\*\*\*\*\*\*"

date

$($CREATE)

wait $!

sleep 1

## BOUNCE DNS's NAMED DAEMON (BIND)

echo ""

echo "\*\*\*\*\*\*\* BOUNCE THE DNS SERVER \*\*\*\*\*\*\*"

## 1. KILL NAMED DAEMON

## XXX: COMMENT OUT TO SKIP PRIMARY STEPS

ps -ef | grep named | egrep -v grep | awk '{ print $2 }' | xargs kill -9

sleep 1

## 2. RESTART NAMED DAEMON

date

## XXX: COMMENT OUT TO SKIP PRIMARY STEPS

nohup /usr/local/sbin/named -c $DNSDIR/named.conf -d 2

sleep 5

## VERIFY IT WORKED IF NOT BACKUP

## HOOK HERE TO ADD THIS LOGIC

## PROVE DNS SERVER IS BACK UP

echo ""

echo "\*\*\*\*\*\*\* RESTORED \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

ps -ef | grep named | egrep -v grep

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

ls -lR $ALLDIR

## TO PREVENT CONFIGURING SECONDARY TURN THIS BIT ON

#exit

echo "\*\*\*\*\*\*\* PUSH FILES FROM PRIMARY SERVER TO SECONDARY \*\*\*\*\*\*\*"

echo "\*\*\*\*\*\*\* BUNDLE FILES \*\*\*\*\*\*\*"

date

cd $ALLDIR

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

## STILL DO NOT COPY ALL OF CFGDIR DUE TO db.header FILE.

#tar cvf dns.tar db.\* named.conf db.cache

tar cvf dns.tar $PRGDIR $DNSDIR $CFGDIR/db.cache $CFGDIR/nmsdns.props

wait $!

compress -f dns.tar

wait $!

## CREATE ROOT NMSDNS DIRECTORY IF DOESNT EXIST

$RSH $SECONDARY "mkdir -p $ALLDIR"

wait $!

## ISSUE THIS COMMAND:

## rcp dns.tar.Z sa73netv1:/opt/NMSDNS/dns.tar.Z

echo "$RCP $ALLDIR/dns.tar.Z $SECONDARY:$ALLDIR/dns.tar.Z"

$RCP $ALLDIR/dns.tar.Z $SECONDARY:$ALLDIR/dns.tar.Z

wait $!

echo "\*\*\*\*\*\*\* SECONDARY SERVER RESTORE \*\*\*\*\*\*\*"

echo ""

echo "\*\*\*\*\*\*\* SECONDARY: SAVE CURRENT DNS ENVIRONMENT \*\*\*\*\*\*\*"

## NAME THE DAILY TAR FILE (SAVE 7 FILES: Mon-Sun)

F="$ALLDIR/nmsdns.$(date | awk '{ print $1 }').tar"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#tar cvf $F $PRGDIR/named.run $DNSDIR/db.\* $DNSDIR/named.conf $LOGDIR/nmsdns\*

echo "SECONDARY: $RSH $SECONDARY tar cvf $F $PRGDIR $DNSDIR $LOGDIR"

$RSH $SECONDARY "tar cvf $F $PRGDIR $DNSDIR $LOGDIR"

## COMPRESS THE FILE

$RSH $SECONDARY "compress -f $F"

## MOVE IT INTO POSITION

echo "SECONDARY: $RSH $SECONDARY mv $F.Z $OLDDIR"

$RSH $SECONDARY "mv $F.Z $OLDDIR"

## BENCHMARK CURRENT STATE

echo ""

echo "\*\*\*\*\*\*\* SECONDARY: DOCUMENT BENCHMARK \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

$RSH $SECONDARY "ps -ef | grep named | egrep -v grep"

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

$RSH $SECONDARY "ls -lR $ALLDIR"

## CLEAR PAST LOGS

$RSH $SECONDARY "rm $PRGDIR/named.run $LOGDIR/nmsdns.h2n $LOGDIR/nmsdns.host $LOGDIR/nmsdns.log 2>/dev/null"

$RSH $SECONDARY "rm $LOGDIR/nmsdns.warn $LOGDIR/nmsdns.warn2 2>/dev/null"

## CLEAN CURRENT ENVIRONMENT

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#rm $DNSDIR/db.\* $DNSDIR/named.conf

## XXX: COMMENT OUT TO SKIP PRIMARY STEPS

$RSH $SECONDARY "echo 'y' | rm -r $DNSDIR"

$RSH $SECONDARY "mkdir $DNSDIR"

## PROVE IT IS CLEAN

echo ""

echo "\*\*\*\*\*\*\* SECONDARY: DOCUMENT THAT IT IS NOW CLEAN \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

$RSH $SECONDARY "ps -ef | grep named | egrep -v grep"

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

$RSH $SECONDARY "ls -lR $ALLDIR"

## UNPACKAGE FILES ON SECONDARY

echo ""

echo "\*\*\*\*\*\*\* SECONDARY: UNPACKAGE FILES ONTO SECONDARY SERVER \*\*\*\*\*\*\*"

date

$RSH $SECONDARY "cd $ALLDIR; zcat dns.tar.Z | tar xvf -"

## PERSONALIZE TO THIS SERVER

echo "\*\*\*\*\*\*\* SECONDARY: PERSONALIZE THIS DNS SERVER \*\*\*\*\*\*\*"

date

$RSH $SECONDARY "cp $CFGDIR/db.header $DNSDIR"

## BOUNCE DNS's NAMED DAEMON (BIND)

echo ""

echo "\*\*\*\*\*\*\* SECONDARY: BOUNCE THE DNS SERVER \*\*\*\*\*\*\*"

date

## 1. KILL NAMED DAEMON

$RSH $SECONDARY "ps -ef | grep named | egrep -v grep | awk '{ print \$2 }' | xargs kill -9"

## 2. RESTART NAMED DAEMON

$RSH $SECONDARY "nohup /usr/local/sbin/named -c $DNSDIR/named.conf -d 2"

## VERIFY IT WORKED IF NOT BACKUP

## HOOK HERE TO ADD THIS LOGIC

## PROVE DNS SERVER IS BACK UP

echo ""

echo "\*\*\*\*\*\*\* RESTORED \*\*\*\*\*\*\*"

date

echo "PROCESSES:"

$RSH $SECONDARY "ps -ef | grep named | egrep -v grep"

echo "FILES:"

## FIX: KSH CANNOT HANDLE BIG CMD LINES DUE TO '\*'

#ls -l $DNSDIR/db.\* $DNSDIR/named.conf

$RSH $SECONDARY "ls -lR $ALLDIR"

## FOOTER

echo ""

echo "\*\*\*\*\*\*\* END \*\*\*\*\*\*\*"

date

## SEND NOTIFICATION TO ADMIN

cat $LOGDIR/nmsdnsd.log $LOGDIR/$DEFAULTNAME.warn2 | mailx -s "NMS-DNS run" $CONTACT

cat $LOGDIR/$DEFAULTNAME.warn2 | mailx -s "CORP EAST NMS-DNS OpenView and Device Error Detection" 'NMSADMIN@company.com';

## NMSDNSFEED.pl

This is the program that takes information from OpenView and SNMP polls in order to construct tables that represent the DNS named space.

#!/usr/bin/perl

##########################################################################

# 03/30/2003 Daniel L. Needles Version 1.0 #

# PROGRAM: NMSDNSFEED.pl #

# USAGE: NMSDNSFEED.pl [-DEBUG #] [-propsfile FILE] [-logfile FILE] #

# #

# PURPOSE: Extract node information from OpenView into tabular format #

##########################################################################

# DLN20030331: Used $desc instead of $id since index numbers #

# change from OpenView version to version. #

##########################################################################

# DLN20030520: Production release. #

# DLN20030522: Added check for $intname in GOOD, BAD, and other node adds#

# to prevent adding NODE-INTNAME to the database when INTNAM#

# is NULL. Otherwise entries would include NODE- #

##########################################################################

use strict;

my $VERSION = 25;

## CONFIGURABLE GLOBAL VARIABLES (ALLOWS FOR SIMPLE PARAMETER FILE)

use vars qw /$NullValue $MULTIHOMED $PROPSFILE

$SNMPTIMEOUT $SNMPRETRIES $SNMPCOMMUNITYSTRING

$IF $INTABREV $TBLHEADER $DEFAULTMASK $DEFAULTDOMAIN $LOCATION

$NAMEDFILE $LOGFILE $H2NFILE $HOSTFILE $WARNFILE $WARNFILE2

$EXTRAFILE $DEFAULTNAME

$DEBUG $DEBUGMAX

$DEBUGLOG $DEBUGNAMED $DEBUGHOST $DEBUGH2N $DEBUGWARN $DEBUGWARN2

$D\_GOOD $D\_ADMDOWN $D\_INVALID

$D\_INT $D\_MAIN

$D\_LOGFILE $D\_HOSTNAME $D\_NAMED $D\_MH

$D\_1STPASS $D\_PROCESS $D\_2NDPASS

$D\_WARN $D\_WARN2 $D\_H2N

$LOGDIR $PRGDIR $CFGDIR $DNSDIR $HSTDIR $OLDDIR

$DNSSERVER/;

use vars qw / $LOGFD $HOSTFD $H2NFD $WARNFD $NAMEDFD

$summary $networks $domains

$domains2 $domains3 $domains4 $ips $nodeints

$fields $named $VALIDMASKS

$iphostname $ipaddress $node $nodedns $sysname $sysnamedns

$DNSnode $DNSdns $domain $interfacecount $intname $extra

$intstatus $intip $intstatus $intip $intmask $getsnmp

$bad $dispose $intnetworks $DBCACHEFILE $DBHEADERFILE/;

## CONSTANT

my $TRUE = 1;

my $FALSE = 0;

my @OVLIST;

my @DUMP;

my @MARKERS;

my $MARKERS;

my ($id,$I,$J,$desc,$val,$item,$node,$line,$mh);

my ($tmp,$tmp1,$tmp2,$tmp3,$tmp4,$j,$multi,$nodeint);

############################# PROCEDURES #################################

require "./nmsdns.pl";

###########################################################################

############################## MAIN PROGRAM ###############################

###########################################################################

## Initialize Program

InitPrg($LOGFD | $WARNFD | $H2NFD );

## LIST OF NODES TO PULL FROM THE DATABASE WITH COMMANDS

@OVLIST=`/usr/OV/bin/ovtopodump -rl | grep "NODE ID:" | awk -F: '{ print \$2 }' | sed -e 's: ::' | sort -n | sed -e 's:^:\/usr\/OV\/bin\/ovobjprint -o :'`;

## LIST COMMANDS TO BE RUN

$I=1;

LogMsg($D\_MAIN,"CMD: PROGRAMS TO RUN (1 PER NODE)\n");

foreach $node (@OVLIST) {

LogMsg($D\_MAIN,"CMD: $I: $node\n");

$I++;

}

## CHECK FOR MAJOR FAILURE

if ( $I < 5 ) {

LogMsg($D\_WARN,"OpenView not responding.\n");

LogMsg($D\_WARN2,"OpenView not responding.\n");

exit -1;

}

## LIST OF POTENTIAL MARKERS TO ID MULTIHOMED DEVICES

## -- ENABLES AUDIT OF WHAT IS BEING USED VERSES SHOULD BE USED

@MARKERS=`grep Field /usr/OV/fields/C/\\* | awk '{ print \$2 }' | grep "\\\"is" | sed -e 's:"::g'`;

LogMsg($D\_MH,"MH: TOTAL MULTIHOMED SWITCHES AVAILABLE: $#MARKERS\nMH: ");

foreach $item (@MARKERS) {

chomp($item);

$MARKERS->{$item} = 1;

}

## LIST OF VARIABLES ALLOWED TO IDENTIFY MULTIHOMED

foreach $item (sort keys %{$MARKERS}) {

LogMsg($D\_MH,",$item");

}

LogMsg($D\_MH,"\n");

## HEADER FOR TABLE INFO

LogMsg($D\_LOGFILE,"TBL,node,DNSnode,hostname,ipaddress,sysname,sysnamedns,interfacecount,intname,intstatus,intip,intmask,IsMultiHomed,NodePriInterface\n");

## MAIN PROGRAM LOOP

$J=1;

foreach $node (@OVLIST) {

## COMMAND TO EXTRACT INFO ON A NODE FROM OPENVIEW

LogMsg($D\_MAIN,"CMD: ################################################################\nCMD: $J $node\n");

$J++;

## PULL DATA ABOUT NODE

@DUMP=`$node`;

## NULLIFY SHORT TERM MEMORY

$iphostname="$NullValue";

$ipaddress="$NullValue";

$sysname="$NullValue";

$sysnamedns="$NullValue";

$DNSnode="$NullValue";

$DNSdns="$NullValue";

$interfacecount="$NullValue";

$intname="$NullValue";

$extra="$NullValue";

$intstatus="$NullValue";

$intip="$NullValue";

$intmask="$NullValue";

$node="$NullValue";

$nodedns="$NullValue";

$domain="$NullValue";

$getsnmp="$NullValue";

$bad="";

$dispose=$FALSE;

undef($nodeint);

my $nodeint;

undef($multi);

my $multi;

$mh=$FALSE;

## PASS 1 - DETERMINE NODE LEVEL ATTRIBUTES.

for ($j=0; $j<=$#DUMP; $j++) {

# GRAB A LINE

$line=$DUMP[$j];

LogMsg($D\_MAIN,"CMD: $line\n");

LogMsg($D\_1STPASS,"FIRST PASS:$line\n");

# PARSE A VALID LINE

if ( $line =~ /\s+([0-9]+)\s+(.\*?)\t\t(.\*)/) {

# BREAK LINE INTO THREE VALUES

$id=$1; $desc=$2; $val=$3;

# REMOVE WHITE SPACE FROM VALUES

chomp($id); chomp($desc); chomp($val);

$id=~s: ::g; $desc=~s: ::g; $val=~s: ::g;

# CHECK FILTER FOR MULTIHOMED

LogMsg($D\_1STPASS," ID=$id;DESC=$desc;VAL=$val\n");

if ($MULTIHOMED->{"$desc"} == 1) {

if ( $val eq "TRUE" ) {

LogMsg($D\_MH," MULTIHOME: Use AND TRUE\n");

$mh=$TRUE;

$multi->{"$desc"} = 1;

} else {

LogMsg($D\_MH," MULTIHOME: Use BUT FALSE\n");

$multi->{"$desc"} = -1;

}

} else {

if ( $val eq "TRUE" ) {

LogMsg($D\_MH," MULTIHOME: Ign BUT TRUE\n");

$multi->{"$desc"} = -2;

} else {

LogMsg($D\_MH," MULTIHOME: Ign AND FALSE\n");

$multi->{"$desc"} = -3;

}

}

# CHECK FIELD ID TO IDENTIFY VALUES

#id == 11

if ($desc eq "IPHostname" ) {

## GRAB DEVICE NAME

$iphostname=$val;

$iphostname=~s:"::g;

LogMsg($D\_1STPASS," GRAB: IP HOSTNAME: $iphostname\n");

## CHECK FOR BUG THIS IS A STANDBY NODE

#id==172

} elsif ($desc eq "SNMPsysDescr") {

if ($val =~ /sysName:/ ) {

$dispose=$TRUE;

}

## GRAB ADDRESS USED FOR SNMP POLLS

#id==177

} elsif ($desc eq "SNMPipAddress") {

$ipaddress=$val;

$ipaddress=~s:"::g;

LogMsg($D\_1STPASS," GRAB: SNMP ipAddress: $ipaddress\n");

## GET SNMP SYSNAME SINCE OPENVIEW DOESNT STORE IT FOR US

if ( ! ($ipaddress =~ /0\.0\.0\.0/ )) {

## OPENVIEW KNOWS THE CORRECT SNMP READ STRING FOR ITS INVENTORY

LogMsg($D\_MAIN,"sysname=/usr/OV/bin/snmpwalk -t $SNMPTIMEOUT -r $SNMPRETRIES $ipaddress system.sysName ");

$sysname=`/usr/OV/bin/snmpwalk -t $SNMPTIMEOUT -r $SNMPRETRIES $ipaddress system.sysName`;

LogMsg($D\_MAIN," = $sysname\n");

## NO RESPONSE MEANS NEW DEVICE OR UNKNOWN SNMP READ STRING FOR DEVICE

## TRY A FEW VALUES BEFORE GIVING UP: OpenView's best guess and

## global value

if ($sysname eq "") {

LogMsg($D\_MAIN,"TRY 2 getsnmp=/usr/OV/bin/xnmsnmpconf -read -verbose $ipaddress | grep community | awk -F= print 2\n");

$getsnmp=`/usr/OV/bin/xnmsnmpconf -read -verbose $ipaddress | grep "^community" | awk -F= '{ print \$2 }'`;

LogMsg($D\_MAIN,"TRY 3 = $getsnmp\n");

## OPENVIEW: GIVES NULL FOR THE DEFAULT SNMP STRING AS WELL AS

## SNMP STRINGS IT DOESNT KNOW. SO GUESS FIRST

if ($getsnmp eq "" ) {

$getsnmp=$SNMPCOMMUNITYSTRING;

}

## REMOVE LEADING SPACES FROM SNMPSTRING

if ($getsnmp =~ /\s+(.\*)/ ) {

$getsnmp=$1;

}

## USE WHAT WE GOT TO PULL NODE

LogMsg($D\_MAIN,"TRY 4 sysname=/usr/OV/bin/snmpwalk -t $SNMPTIMEOUT -r $SNMPRETRIES -c $getsnmp $ipaddress system.sysName ");

$sysname=`/usr/OV/bin/snmpwalk -t $SNMPTIMEOUT -r $SNMPRETRIES -c $getsnmp $ipaddress system.sysName`;

LogMsg($D\_MAIN," = $sysname\n");

}

## NO RESPONSE MEANS STRING REJECTED - BAD STRING

if ($sysname eq "") {

$sysname="SNMP-string is unknown to OpenView";

## WE GOT A RESPONSE. TRY TO PROCESS

} else {

## DISCARD OID NAME AND OID VARIABLE TYPE AND KEEP ONLY THE OUTPUT

$sysname=~/.\*:(.\*)/;

$sysname=$1;

## REMOVE WHITE SPACE AT THE BEGINING OF SYSNAME DATA

if ($sysname =~ /\s+(.\*)/ ) {

$sysname=$1;

}

## REMOVE WHITE SPACE AT THE END OF SYSNAME

if ($sysname =~ /(.\*)\s+/ ) {

$sysname=$1;

}

## IF NEEDED, EXTRACT DNS FROM SYSNAME

if ($sysname=~ /(.\*?)\.(.\*)/) {

$sysname=$1;

$sysnamedns=$2;

}

## CHECK FOR WRONG CHARACTER USED IN SYSNAME

if ( $sysname && !($sysname =~ /^([a-zA-Z0-9\_-]+)$/ )) {

$sysname="SNMP-Invalid characters used ($sysname)";

## CHECK FOR NULL SYSNAME RETURNED

} elsif ( ! $sysname ) {

$sysname="SNMP-sysname is not set";

}

}

# $sysname="SNMP-string is unknown";

## INVALID SNMP IP ADDRESS

} else {

$sysname="SNMP-IP address is unknown";

}

LogMsg($D\_1STPASS," SNMP sysname: $sysname obtained with SNMPREAD: $getsnmp\n");

## DETERMINE THE BEST INTERFACE FOR THE NODE

#id==216

} elsif ($desc eq "TopMInterfaceCount") {

LogMsg($D\_1STPASS," GRAB Interface Count: $val\n");

$interfacecount=$val;

my $first=1;

my $primeip="";

$val=$j+$val+1;

for ( $j=$j+1; $j<$val; $j++) {

$line=$DUMP[$j];

LogMsg($D\_1STPASS,"FIRST PASS (INTERFACE):$line\n");

# STRIP LEADING HEADER ON FIRST LINE

if ($first) {

if ( $line =~ /\s+([0-9]+)\s+(.\*?)\t\t(.\*)/) {

$line=$3;

}

}

$first=0;

## OPENVIEW INVENTORY ISSUE. SOME IBM DEVICES HAVE EMBEDDED SPACES MESSING UP

## THE DELIMITER USE FIRST TO GET INTO THE NEXT STANSA.

if ($line =~ /"IBM (.\*?)\s+(.\*?)\s+(.\*?)\s+(.\*?)\s+.\*"/) {

$intname="IBM-$1";

$intstatus=$2;

$intip=$3;

$intmask=$4;

$first=2;

}

# FROM INTERFACE INFO PARSE OUT NAME, STATUS, AND IP

if (($first) || ($line =~ /"(.\*?)\s+(.\*?)\s+(.\*?)\s+(.\*?)\s+.\*"/)) {

if (! $first) {

$intname=$1;

$intstatus=$2;

$intip=$3;

$intmask=$4;

} else {

$first=0;

}

LogMsg($D\_1STPASS," INT: Name:$intname Status:$intstatus IP:$intip MASK: $intmask\n");

## FORCE LOWER CASE

$intname=~ tr/A-Z/a-z/;

## BEST GUESS AT MAST

if (! ( $intmask =~ /\d+\.\d+\.\d+\.\d+/ )) {

$intmask=$DEFAULTMASK;

}

# FOR MULTIHOMED PICKUP BEST INTERFACE TO REPRESENT THE NODE IN DNS

LogMsg($D\_1STPASS," DETERMINE BEST INTERFACE FOR NODE IN DNS\n");

if ($intname =~ /([a-z\_-]+)([0-9\.-:\/]+)/) {

## BREAK UP INTEFACES INTO TYPE AND INDEX

$tmp1=$1; $tmp2=$2;

## ONE OFF SPECIAL REQUIREMENT. LOOPBACK3 IS KING

if ( $intname eq "loopback3" ) {

$nodeint=$intname;

$primeip=$intip;

LogMsg($D\_1STPASS," Special Case. Grab loopback3 as it. ($tmp1)($tmp2)\n");

## SKIP OTHER CHECKS. WE FOUND WHAT WE WANTED

} elsif ($nodeint eq "loopback3") {

## DO NOTHING.

## GRAB FIRST

} elsif ($nodeint eq "") {

$nodeint=$intname;

$primeip=$intip;

LogMsg($D\_1STPASS," First interface. Grab it $nodeint ($tmp1)($tmp2)\n");

## ALL OTHER CASES

} else {

$nodeint =~ /([a-z\_-]+)([0-9\.-:\/]+)/;

$tmp3=$1; $tmp4=$2;

LogMsg($D\_1STPASS," Checking NEW ($tmp1)($tmp2) against OLD ($tmp3)($tmp4)\n");

## CONVERT TO TYPE INTO A NUMERIC PRIORITY

$tmp1=($IF->{"$INTABREV->{$tmp1}"})?

$IF->{"$INTABREV->{$tmp1}"}:500;

$tmp3=($IF->{"$INTABREV->{$tmp3}"})?

$IF->{"$INTABREV->{$tmp3}"}:500;

## IF SUB OR TEMP INTERFACES MAKE THEN VERY BAD CHOICES

$tmp=$tmp2;

if ( $tmp=~ /\// ) {

$tmp2=1000;

}

if ( $tmp=~ /\./ ) {

$tmp2=2000;

}

if ( $tmp=~ /\:/ ) {

$tmp2=3000;

}

$tmp=$tmp4;

if ( $tmp=~ /\// ) {

$tmp4=1000;

}

if ( $tmp=~ /\./ ) {

$tmp4=2000;

}

if ( $tmp=~ /\:/ ) {

$tmp4=3000;

}

## THE TYPE OF INTERFACE IS BETTER THAN THE SAVED COPY

if ( $tmp1 < $tmp3 ) {

LogMsg($D\_1STPASS," Superior type of interface $intname ($tmp1) replaces $nodeint ($tmp3)\n");

$nodeint=$intname;

$primeip=$intip;

## SAME TYPE OF INTERFACE BUT NEW INTERFACE HAS A LOWER NUMBER (i.e. LOOPBACK0 is better than LOOPBACK2)

} elsif (( $tmp1 == $tmp3 ) && ( $tmp2 < $tmp4 )) {

if ( ( $tmp1 != $IF->{"lp"}) || ($intstatus=~ /[Uu]p/ ) ) {

LogMsg($D\_1STPASS," Superior numerator $intname ($tmp2) replaces $nodeint ($tmp4)\n");

$nodeint=$intname;

$primeip=$intip;

} else {

LogMsg($D\_1STPASS," Loop back superior numerator $intname ($tmp2) over $nodeint ($tmp4), but is down\n");

}

}

}

# IP ADDRESS WITHOUT ANY NAME

} elsif ( $intname =~ /\d+\.\d+\.\d+\.\d+/ ) {

if (!$nodeint) {

$nodeint="$NullValue";

$primeip="";

}

# NO INTERFACE NUMBER ... COULD BE A PRINTER

} elsif ( $intname =~ /[a-z\_-]+/ ) {

if (!$nodeint) {

$nodeint="$NullValue";

$primeip="";

}

} else {

LogMsg($D\_WARN,"WARN Unrecognized interface discovered: \"$intname\". Interface name set to INVALID. Program nmsdnsfeed.pl needs to be updated to detect this interface type. \n");

if (!$nodeint) {

$nodeint="INVALID";

$primeip="";

}

}

# INTERFACEW DATA CORRUPT OR UNEXPECTED

} else {

LogMsg($D\_1STPASS, " SKIP: Bad interface,\n");

}

}

## CLEAN PRIMARY NODE INTERFACE SO COMPARE WORKS LATER

$nodeint=IntNameClean($nodeint,$node);

## GRAB SNMP IP IF NO PRIMARY IP FOUND

if (! $primeip) {

$primeip=$ipaddress;

}

## GET DNS NAME

LogMsg($D\_MAIN,"/usr/local/bin/host $primeip $DNSSERVER | grep domain name pointer | awk print 5");

$DNSnode=`/usr/local/bin/host $primeip $DNSSERVER | grep "domain name pointer" | awk '{ print \$5 }'`;

LogMsg($D\_MAIN," = $DNSnode\n");

## REMOVE TRAILING NEWLINE

chomp($DNSnode);

## BREAK HOST AND DNS OUT OF DNSNAME

if ($DNSnode && ($DNSnode ne "$NullValue") ) {

if ($DNSnode=~ /(.\*?)\.(.\*)\./) {

$DNSnode=$1;

$DNSdns=$2;

}

} else {

$DNSnode=$NullValue;

}

# IGNORE NON INTEFACE LINES

} else {

LogMsg($D\_1STPASS," GRAB: SKIP: Not used\n");

}

# NON DATA LINE PROBABLY BLANK LINE OR OTHER HEADER INFO

} else {

LogMsg($D\_1STPASS," SKIP: Bad syntax\n");

}

}

### OPENVIEW NUANCE. HOSTNAME SET TO IPADDRESS IF HOSTNAME NOT KNOWN.

### IF THIS IS THE CASE CLEAR IT. SPECIFICALLY THE LOGIC IS:

## If the node supports IP:

## 1a) If a non-migratable software loopback IP address

## (other than 127.0.0.1) exists on the node, and the address

## resolves to an IP hostname, that hostname is used.

## 1b) Otherwise, NNM chooses the name associated with the lowest

## numbered non-migratable IP address that resolves to an

## IP hostname.

## 1c) If no IP addresses resolve to an IP hostname, the

## lowest numbered IP address is formatted

## as a string and used as the hostname.

## [ "lowest numbered" means when compared as integers. ]

$iphostname=("$iphostname"=~/\d+\.\d+\.\d+\.\d+/)?$NullValue:$iphostname;

### DETERMINE HOST NAME BASED ON NODE LEVEL DATA

# ## CLEAN HOST NAME UP (LAB PROBLEMS)

# $sysname=~s:\_:-:g;

## TRACK ERRORS

$bad="";

# IF MULTIHOMED

if ($mh) {

# IF INVALID SYSNAME

if (!$sysname || ("$sysname" eq "$NullValue") || ("$sysname" =~ "^SNMP-")) {

$bad= "SysName";

# IF INVALID DNS ENTRY

if (!$DNSnode || ("$DNSnode" eq "$NullValue")) {

$bad.= " DNS";

# IF INVALID OPENVIEW ENTRY CANNOT CREATE DNS ENTRY

if (!$iphostname || ("$iphostname" eq "$NullValue")) {

$bad.= " OpenView";

$node=$NullValue;

# ELSE CREATE DNS ENTRY WITH OPENVIEW NAME

} else {

$node=$iphostname;

}

# ELSE CREATE DNS ENTRY WITH DNS NAME

} else {

$node=$DNSnode;

}

# ELSE CREATE DNS ENTRY WITH SYSNAME NAME

} else {

$node=$sysname;

}

# NOT MULTIHOMED

} else {

# IF INVALID DNS ENTRY

if (!$DNSnode || ("$DNSnode" eq "$NullValue")) {

$bad= " DNS";

# IF INVALID SYSNAME

if (!$sysname || ("$sysname" eq "$NullValue") || ("$sysname" =~ /^SNMP-/) ) {

$bad.= " SysName";

# IF INVALID OPENVIEW ENTRY CANNOT CREATE DNS ENTRY

if (!$iphostname || ("$iphostname" eq "$NullValue")) {

$bad.= " OpenView";

$node=$NullValue;

# ELSE CREATE DNS ENTRY WITH OPENVIEW NAME

} else {

$node=$iphostname;

}

# ELSE CREATE DNS ENTRY WITH SYSNAME NAME

} else {

$node=$sysname;

}

# ELSE CREATE DNS ENTRY WITH DNS NAME

} else {

$node=$DNSnode;

}

}

## GENERAL CLEAN UP

chomp($domain);

## FORCE LOWER CASE

$node=~ tr/A-Z/a-z/;

## UNLESS IP ADDRESS BREAK HOST AND DNS OUT OF NODE

if ($node=~ /\d+\.\d+\.\d+\.\d+/) {

$node="na";

$nodedns="na";

} elsif ($node=~ /(.\*?)\.(.\*)/) {

$node=$1;

$nodedns=$2;

}

## INVALID IP ADDRESS AND SNMP DIFFICULTIES

if ( $ipaddress =~ /0\.0\.0\.0/ ) {

LogMsg($D\_WARN, "WARN: SNMP address missing for $node.\n");

#REMEMBER

$summary->{"SNMP-Missing IP Address"}->{"DATA"}.= ",$node";

$summary->{"SNMP-Missing IP Address"}->{"CNT"}++;

} elsif ($sysname =~ "^SNMP-") {

LogMsg($D\_WARN,"WARN: SNMP poll error for $node ($ipaddress): $sysname.\n");

# REMEMBER NODES

$summary->{"$sysname"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"$sysname"}->{"CNT"}++;

$sysname = "$NullValue";

}

## INVALID NODE

if ($node eq "$NullValue") {

LogMsg($D\_WARN,"WARN: Name missing for $ipaddress.\n");

LogMsg($D\_WARN,"IP Address: \"$ipaddress\"\n");

LogMsg($D\_WARN," : Node \"$node\"\n");

LogMsg($D\_WARN," : Node Domain \"$nodedns\"\n");

LogMsg($D\_WARN," : DNS node \"$DNSnode\"\n");

LogMsg($D\_WARN," : DNS domain \"$DNSdns\"\n");

LogMsg($D\_WARN," : SYSNAME node \"$sysname\"\n");

LogMsg($D\_WARN," : SYSNAME domain \"$sysnamedns\"\n");

LogMsg($D\_WARN," : IP Hostname \"$iphostname\"\n");

LogMsg($D\_WARN," : IsMultihomed \"$mh\"\n");

LogMsg($D\_WARN," : NULL VALUE \"$NullValue\"\n");

} elsif ( $bad ) {

$bad=~s: :,:g;

$bad=~s:^,::;

LogMsg($D\_WARN, "WARN: Name missing in ($bad) for $ipaddress.\n");

}

## SUMMARIZE

if ( $bad =~ "DNS" ) {

$summary->{"Device name missing/unaccessible from DNS"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Device name missing/unaccessible from DNS"}->{"CNT"}++;

}

if ( $bad =~ "OpenView" ) {

$summary->{"Device name missing in OpenView"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Device name missing in OpenView"}->{"CNT"}++;

}

if ( $bad =~ "SysName" ) {

$summary->{"Device name missing/unaccessible from sysname"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Device name missing/unaccessible from sysname"}->{"CNT"}++;

}

## HOSTNAME

LogMsg($D\_HOSTNAME,"\nNode: \"$node\"\n");

LogMsg($D\_HOSTNAME," : SYSNAME node \"$sysname\"\n");

LogMsg($D\_HOSTNAME," : IP Hostname \"$iphostname\"\n");

LogMsg($D\_HOSTNAME," : IP Address \"$ipaddress\"\n");

LogMsg($D\_HOSTNAME," : IsMultihomed \"$mh\"\n");

LogMsg($D\_HOSTNAME," : NULL VALUE \"$NullValue\"\n");

## TRACK ERRORS

$bad="";

### DETERMINE DOMAIN

if (!$DNSdns || ("$DNSdns" eq "$NullValue")) {

$bad="DNS";

if (!$nodedns || ("$nodedns" eq "$NullValue")) {

$bad.=" OpenView";

if (!$sysnamedns || ("$sysnamedns" eq "$NullValue")) {

$bad.=" SysName";

$domain="$NullValue";

} else {

$domain=$sysnamedns;

}

} else {

$domain=$nodedns;

}

} else {

$domain=$DNSdns;

}

## GENERAL CLEAN UP

chomp($domain);

## FORCE LOWER CASE

$domain=~ tr/A-Z/a-z/;

## NO DOMAIN FOUND

if ( $domain eq "$NullValue" ) {

LogMsg($D\_WARN,"\nDOMAIN \"$domain\"\n");

LogMsg($D\_WARN," : NODE DNS \"$nodedns\"\n");

LogMsg($D\_WARN," : DNS DNS \"$DNSdns\"\n");

LogMsg($D\_WARN," : SYSNAME \"$sysnamedns\"\n");

LogMsg($D\_WARN," : NULL VALUE \"$NullValue\"\n");

LogMsg($D\_WARN,"WARN: Domain not available for $node ($ipaddress.) Guessing $DEFAULTDOMAIN\n");

## SOME DOMAINS NOT FOUND

} elsif ( $bad ) {

$bad=~s: :,:g;

$bad=~s:^,::g;

LogMsg($D\_WARN, "WARN: Domain missing in ($bad) for $node ($ipaddress).\n");

}

## LOG IT

LogMsg($D\_HOSTNAME,"\nDOMAIN \"$domain\"\n");

LogMsg($D\_HOSTNAME," : NODE DNS \"$nodedns\"\n");

LogMsg($D\_HOSTNAME," : DNS DNS \"$DNSdns\"\n");

LogMsg($D\_HOSTNAME," : SYSNAME \"$sysnamedns\"\n");

LogMsg($D\_HOSTNAME," : NULL VALUE \"$NullValue\"\n");

LogMsg($D\_HOSTNAME," : DEFAULT DOMAIN\"$DEFAULTDOMAIN\"\n");

## SUMMARIZE

if ( $bad =~ "DNS" ) {

$summary->{"Domain missing/unaccessible from DNS"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Domain missing/unaccessible from DNS"}->{"CNT"}++;

}

if ( $bad =~ "OpenView" ) {

$summary->{"Domain missing from OpenView"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Domain missing from OpenView"}->{"CNT"}++;

}

if ( $bad =~ "SysName" ) {

$summary->{"Domain missing/unaccessible from sysname"}->{"DATA"}.= ",$node ($ipaddress)";

$summary->{"Domain missing/unaccessible from sysname"}->{"CNT"}++;

}

## DETAIL DOMAIN IF DEBUG ON

LogMsg($D\_HOSTNAME,"\nDOMAIN \"$domain\"\n");

LogMsg($D\_HOSTNAME," : NODE DNS \"$nodedns\"\n");

LogMsg($D\_HOSTNAME," : DNS DNS \"$DNSdns\"\n");

LogMsg($D\_HOSTNAME," : SYSNAME \"$sysnamedns\"\n");

LogMsg($D\_HOSTNAME," : NULL VALUE \"$NullValue\"\n");

### PASS 2: PROCESS INTERFACE LEVEL DATA

for ($j=0; $j<=$#DUMP; $j++) {

# GRAB A LINE

$line=$DUMP[$j];

LogMsg($D\_2NDPASS,"LAST PASS:$line\n");

# PARSE A VALID LINE

if ( $line =~ /\s+([0-9]+)\s+(.\*?)\t\t(.\*)/) {

# BREAK LINE INTO THREE VALUES

$id=$1; $desc=$2; $val=$3;

chomp($id); chomp($desc); chomp($val);

$id=~s: ::g; $desc=~s: ::g; $val=~s: ::g;

# CHECK FILTER FOR MULTIHOMED

LogMsg($D\_2NDPASS," ID=$id;DESC=$desc;VAL=$val;\n");

# CHECK FIELD ID TO IDENTIFY VALUES

## PARSE ALL INTERFACES ON MULTIHOMED AND NON MULTIHOMED DEVICES

#id==216

if ($desc eq "TopMInterfaceCount") {

LogMsg($D\_2NDPASS," GRAB Interface Count: $val\n");

$interfacecount=$val;

my $first=1;

$val=$j+$val+1;

for ( $j=$j+1; $j<$val; $j++) {

$line=$DUMP[$j];

# STRIP LEADING HEADER ON FIRST LINE

if ($first) {

if ( $line =~ /\s+([0-9]+)\s+(.\*?)\t\t(.\*)/) {

$line=$3;

}

}

$first=0;

## OPENVIEW INVENTORY ISSUE. SOME IBM DEVICES HAVE EMBEDDED SPACES MESSING UP

## THE DELIMITER USE FIRST TO GET INTO THE NEXT STANSA.

if ($line =~ /"IBM (.\*?)\s+(.\*?)\s+(.\*?)\s+(.\*?)\s+.\*"/) {

$intname="IBM-$1";

$intstatus=$2;

$intip=$3;

$intmask=$4;

$first=2;

}

# FROM INTERFACE INFO PARSE OUT NAME, STATUS, AND IP

if (($first) || ($line =~ /"(.\*?)\s+(.\*?)\s+(.\*?)\s+(.\*?)\s+.\*"/)) {

if (! $first ) {

$intname=$1;

$intstatus=$2;

$intip=$3;

$intmask=$4;

} else {

$first=0;

}

LogMsg($D\_2NDPASS," INT: Name:$intname Status:$intstatus IP:$intip MASK: $intmask\n");

## FORCE LOWER CASE

$intname=~ tr/A-Z/a-z/;

## SAVE INTNAME

$extra="$intname~$intstatus~$intip~$intmask";

## CLEAN UP INTERFACE NAMES FOR DNS STUFF

# if ($mh) {

$intname=IntNameClean($intname,$node);

# }

## LOG DEVICE TYPE INFO IF DEBUG INCLUDE D\_LOGFILE

if ( $D\_LOGFILE & $DEBUG ) {

LogMsg($D\_LOGFILE,"TBL,$node,$DNSnode,$iphostname,$ipaddress,$sysname,$sysnamedns,$interfacecount,$intname,$intstatus,$intip,$intmask,$mh");

## MULTIHOMED DEVICE PRINT EXTRA OUTPUT

# if ( $mh ) {

LogMsg($D\_LOGFILE,",$nodeint");

foreach $item (sort keys %{$MARKERS}) {

if ($multi->{"$item"} == 1) {

$extra.="~TU";

LogMsg($D\_LOGFILE, ",TU");

} elsif ($multi->{"$item"} == -1) {

LogMsg($D\_LOGFILE, ",FU");

$extra.="~FU";

} elsif ($multi->{"$item"} == -2) {

LogMsg($D\_LOGFILE, ",TI");

$extra.="~TI";

} elsif ($multi->{"$item"} == -3) {

$extra.="~FI";

LogMsg($D\_LOGFILE, ",FI");

} elsif ( $MULTIHOMED->{"$item"} ) {

$extra.="~OU";

LogMsg($D\_LOGFILE, ",OU");

} else {

$extra.="~OI";

LogMsg($D\_LOGFILE, ",OI");

}

}

LogMsg($D\_LOGFILE,"\n");

# }

}

## PRINT OUTPUT

# IS NO NODE NAME AVAILABLE?

LogMsg($D\_PROCESS, "COMPARE PRIMARY \"$nodeint\" AGAINST \"$intname\"\n");

$extra="$intname~$intstatus~$intip~$intmask";

## NODE IS NULL, IBM MAINFRAME INTERFACE, OR MARKED DISPOSE

if ( ("$node" eq "$NullValue" ) || ($intname eq "IP" ) || ($dispose)) {

if ( $mh && $intname && ( $nodeint ne $intname )) {

LogMsg($D\_INVALID,"$intip $node-$intname.$domain $node.$domain\n");

H2NPopulate($D\_INVALID,$intip,$intmask,$domain,$node,"$node-$intname",$extra);

} else {

LogMsg($D\_INVALID,"$intip.$domain $node.$domain\n");

H2NPopulate($D\_INVALID,$intip,$intmask,$domain,$node,$NullValue,$extra);

}

# IS INTERFACE MARKED ADMIN DOWN BY OPENVIEW?

} elsif ( $intip =~ /(\d+\.\d+\.\d+\.\d+)[A-Za-z]+/ ) {

$intip=$1;

if ( $mh && $intname && ( $nodeint ne $intname )) {

LogMsg($D\_ADMDOWN,"$intip $node-$intname.$domain $node.$domain\n");

H2NPopulate($D\_ADMDOWN,$intip,$intmask,$domain,$node,"$node-$intname",$extra);

} else {

LogMsg($D\_ADMDOWN,"$intip $node.$domain\n");

H2NPopulate($D\_ADMDOWN,$intip,$intmask,$domain,$node,$NullValue,$extra);

}

# OTHERWISE IT IS A GOOD NODE

} else {

if ( $mh && $intname && ( $nodeint ne $intname )) {

LogMsg($D\_GOOD,"$intip $node-$intname.$domain $node.$domain\n");

H2NPopulate($D\_GOOD,$intip,$intmask,$domain,$node,"$node-$intname",$extra);

} else {

LogMsg($D\_GOOD,"$intip $node.$domain\n");

H2NPopulate($D\_GOOD,$intip,$intmask,$domain,$node,$NullValue,$extra);

}

}

# CLEAR

LogMsg($D\_MAIN,"CMD: Interface $intip,$intmask,$domain,$node,$intname\n");

$intname="$NullValue";

$extra="$NullValue";

$intstatus="$NullValue";

$intip="$NullValue";

$intmask="$NullValue";

# INTERFACEW DATA CORRUPT OR UNEXPECTED

} else {

LogMsg($D\_2NDPASS, " SKIP: Bad interface,\n");

}

}

# PARSABLE LINE BUT WE DIDN'T EXPECT IT.

} else {

LogMsg($D\_2NDPASS," SKIP: Parsed but not recognized\n");

}

# NON DATA LINE PROBABLY BLANK LINE OR OTHER HEADER INFO

} else {

LogMsg($D\_2NDPASS," SKIP: Bad syntax\n");

}

}

}

##############################################################################

## OUTPUT FILE: GENERATE H2N RAW DATA FILE

##############################################################################

LogMsg($D\_MAIN,"CMD: Post Processing\n");

my ($tblcnt,$tblextra,$tdomain,$tnetwork,$thost,$tintip,$thostint);

close(HOST);

LogMsg($D\_H2N,"$TBLHEADER");

foreach $tdomain (sort keys %{$intnetworks}) {

foreach $tnetwork (sort keys %{ %{ $intnetworks->{"$tdomain"} } } ) {

foreach $thost (sort keys %{ %{ %{ $intnetworks->{"$tdomain"}->{"$tnetwork"} } } } ) {

foreach $tintip (sort keys %{ %{ %{ %{ $intnetworks->{"$tdomain"}->{"$tnetwork"}->{"$thost"} } } } } ) {

foreach $thostint (sort keys %{ %{ %{ %{ %{ $intnetworks->{"$tdomain"}->{"$tnetwork"}->{"$thost"}->{"$tintip"} } } } } } ) {

LogMsg($D\_H2N,"$tdomain,$tnetwork,$thost,$tintip,$thostint,");

$tblcnt=$intnetworks->{"$tdomain"}->{"$tnetwork"}->{"$thost"}->{"$tintip"}->{"$thostint"}->{"Cnt"};

$tblextra=$intnetworks->{"$tdomain"}->{"$tnetwork"}->{"$thost"}->{"$tintip"}->{"$thostint"}->{"Extra"};

LogMsg($D\_H2N,"$tblcnt,$tblextra\n");

#print "OUTPUT: $tblcnt,$tblextra\n";

}

}

}

}

}

##############################################################################

## ERROR SUMMARY FILE

##############################################################################

my $item;

my $cnt;

my $list;

my $i=0;

## FILLOUT A HEADER

LogMsg($D\_WARN2,"### NMS-DNS SUMMARY OF OpenView AND DEVICE CONFIG ERRORS\n#\n")

;

foreach $item (sort keys %{ $summary} ) {

$cnt=0;

$i++;

$cnt=$summary->{$item}->{"CNT"};

LogMsg($D\_WARN2,"# $i: $cnt issues of: $item\n");

}

LogMsg($D\_WARN2,"\n\n");

## NOW ENTER THE DETAILS

foreach $item (sort keys %{ $summary} ) {

$cnt=0;

$cnt=$summary->{$item}->{"CNT"};

$list=$summary->{$item}->{"DATA"};

$list=~s:^,::;

LogMsg($D\_WARN2,"$item ($cnt): $list\n");

}

LogMsg($D\_MAIN,"CMD: End.\n");

## NMSDNSCLEAN.pl

This program cleans the output from nmsdnsfeed.pl, which is stored in the nmsdns.h2n file.

#!/usr/bin/perl

##########################################################################

# 03/30/2003 Daniel L. Needles Version 1.0 #

# PROGRAM: NMSDNSclean.pl #

# USAGE: NMSDNSclean.pl [-DEBUG #] [-propsfile FILE] [-logfile FILE] #

# #

# PURPOSE: Take output from nmsdnsfeed, detect problems, and corrects #

# and reports on any issues. #

##########################################################################

# 5/15/3 Daniel L. Needles: Turn off Network:Mask checks since class C #

# used exclusively for DNS files. #

# 1. Turn off Missing Masks. #

# 2. Turn off Overlapping domains for networks. #

# 3. Turn off Overlapping networks for domains. #

# 4. Turn off Output host file. #

##########################################################################

# DLN20030520: Production release. #

##########################################################################

use strict;

my $VERSION = 25;

## CONFIGURABLE GLOBAL VARIABLES (ALLOWS FOR SIMPLE PARAMETER FILE)

use vars qw /$NullValue $MULTIHOMED $PROPSFILE

$SNMPTIMEOUT $SNMPRETRIES $SNMPCOMMUNITYSTRING

$IF $INTABREV $TBLHEADER $DEFAULTMASK $DEFAULTDOMAIN

$NAMEDFILE $LOGFILE $H2NFILE $HOSTFILE $WARNFILE $WARNFILE2

$EXTRAFILE $DEFAULTNAME $LOCATION

$DEBUG $DEBUGMAX

$DEBUGLOG $DEBUGNAMED $DEBUGHOST $DEBUGH2N $DEBUGWARN $DEBUGWARN2

$D\_GOOD $D\_ADMDOWN $D\_INVALID

$D\_INT $D\_MAIN

$D\_LOGFILE $D\_HOSTNAME $D\_MH $D\_HOST

$D\_1STPASS $D\_PROCESS $D\_2NDPASS

$D\_WARN $D\_WARN2 $D\_H2N

$LOGDIR $PRGDIR $CFGDIR $DNSDIR $HSTDIR $OLDDIR

$DEFAULTDOMAIN/;

use vars qw / $LOGFD $HOSTFD $H2NFD $WARNFD $NAMEDFD

$summary $networks $domains

$domains2 $domains3 $domains4 $ips $nodeints

$fields $named $VALIDMASKS $DBCACHEFILE $DBHEADERFILE/;

## CONSTANTS

use constant TRUE => 1;

use constant FALSE => 0;

require "./nmsdns.pl";

###########################################################################

############################## MAIN PROGRAM ###############################

###########################################################################

## INITIALIZE PROGRAMS.

InitPrg($LOGFD | $HOSTFD | $WARNFD );

## LOAD THE OUTPUT FROM $DEFAULTNAME.pl

H2NLoadClean();

###########################################################################

##############################################################################

## REPLACE MISSING NETWORK MASKS ##################

##############################################################################

my $field;

my $node;

my ($domain,$intnetwork,$host,$intip,$hostint,$cnt,$extra);

my $newintnetwork;

my ($bits,$nk2);

my ($nk1,$mk1,$nk2,$mk2,$nk1T,$mk1T,$nk2T,$mk2T);

my @nk1;

my @mk1;

my @nk2;

my @mk2;

#LogMsg($D\_INT,"\nENTERING ADDING MISSING MASKS ##########\n");

### LOOP THROUGH ALL ENTRIES

#foreach $field (sort keys %{ $fields } ) {

#

# ## BREAK THE FIELDS OUT

# $node = $fields->{"$field"};

# ($domain,$intnetwork,$host,$intip,$hostint)= split /,/, $field, 6;

#

# ## BREAK OUT THE NETWORK AND MASK INFO

# ($nk1,$mk1)= split /:/, $intnetwork;

# $newintnetwork= $intnetwork;

#

# ## IF MASK OR NETWORK IS NULL LETS DO SOME WORK TO FIND A NETWORK:MASK SET

# if ( $mk1 eq "$NullValue" || $nk1 eq "$NullValue" ) {

#

# # USE IP ADDRESS AS NETWORK AND CONVERT TO A NUMBER

# $nk1 = $intip;

# @nk1 = split /\./, $intip, 4;

# $nk1T= $nk1[3]+$nk1[2]\*(2\*\*8)+$nk1[1]\*(2\*\*16)+$nk1[0]\*(2\*\*24);

#

# # START WITH THE LARGEST NETWORK MASK AND WORK TO MORE

# # GENERAL TO FIND THE MOST SPECIFIC MATCH

# for ( $bits=0; $bits < 25; $bits++ ) {

#

# ## HYPOTHESIS MASK AND NETWORK

# $mk1T=((2\*\*32)-1) - ((2\*\*($bits))-1);

# $nk1T= $nk1T & $mk1T;

#

# ## LOG

# ## CHECK AGAINST EACH KNOWN MASK

# foreach $nk2 (sort keys %{ $networks } ) {

#

#

# $mk2=$networks->{"$nk2"};

#

# ## CONVERT REAL MASK TO NUMERIC

# @mk2 = split /\./, $mk2, 4;

# $mk2T= $mk2[3]+$mk2[2]\*(2\*\*8)+$mk2[1]\*(2\*\*16)+$mk2[0]\*(2\*\*24);

#

# ## CONVERT REAL NETWORK TO NUMERIC

# @nk2 = split /\./, $nk2, 4;

# $nk2T= $nk2[3]+$nk2[2]\*(2\*\*8)+$nk2[1]\*(2\*\*16)+$nk2[0]\*(2\*\*24);

#

# ## IF BOTH MASK AND NETWORK MATCH, USE IT

# ## AND FORCE BREAKOUT OF GREATER LOOP

# if ( ( $mk2T == $mk1T ) && ( $nk1T == $nk2T ) ) {

# $newintnetwork= "$nk2:$mk2";

# $bits=40;

# }

# }

# }

#

# # REPORT ERROR ONLY. NO ALTERNATIVE MASK WAS FOUND

# if ($newintnetwork =~ /$NullValue/ ) {

# LogMsg($D\_WARN,"WARN OpenView Reported an unknown mask for $host. No replacement available. Assuming Class C.\n");

#

# if (!($summary->{"Mask-missing. Assuming Class C."}->{"DATA"} =~ /,$host,|,$host$/)) {

# $summary->{"Mask-missing. Assuming Class C."}->{"DATA"}.= ",$host";

# }

# $summary->{"Mask-missing. Assuming Class C."}->{"CNT"}++;

#

# ## MARK INVALID ENTRY AS DEAD

# LogMsg($D\_WARN,"REPLACED $field ");

# delete($fields->{"$field"});

#

# ## ADD REPLACEMENT TO HASH WITH CORRECT NETWORK:MASK

# $intnetwork="$nk1[0].$nk1[1].$nk1[2]:255.255.255.0";

# $field="$domain,$intnetwork,$host,$intip,$hostint";

# $fields->{"$field"}=$host;

# LogMsg($D\_WARN," WITH $field\n");

#

# # ENABLE DOMAIN AND NETWORK CHECKS SINCE ITS BEEN PATCHED

# $domains->{"$intnetwork"}->{"DATA"}.= "$host.$domain $intip ($hostint) ";

## $domains->{"$intnetwork"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

# $domains2->{"$intnetwork"}->{"$domain"}->{"DATA"} .= "$host $intip ($hostint): ";

# $domains2->{"$intnetwork"}->{"$domain"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

#

# # REPORT ERROR AND USE ALTERNATIVE MASK. REGISTER NODE IN OTHER DOMAIN/NETWORK CHECKS

# } else {

# LogMsg($D\_WARN,"WARN OpenView Reported an unknown mask for $host. Replacement found.\n");

#

# ## ROLLUP STATS

# if (!($summary->{"Mask-replaced"}->{"DATA"} =~ /,$host,|,$host$/)) {

# $summary->{"Mask-replaced"}->{"DATA"}.= ",$host";

## }

# $summary->{"Mask-replaced"}->{"CNT"}++;

#

# ## MARK INVALID ENTRY AS DEAD

# LogMsg($D\_WARN,"REPLACED $field");

# delete($fields->{"$field"});

#

# ## ADD REPLACEMENT TO HASH WITH CORRECT NETWORK:MASK

# $intnetwork=$newintnetwork;

# $field="$domain,$intnetwork,$host,$intip,$hostint";

# $fields->{"$field"}=$host;

# LogMsg($D\_WARN," WITH $field\n");

#

# # ENABLE DOMAIN AND NETWORK CHECKS SINCE ITS BEEN PATCHED

# $domains->{"$intnetwork"}->{"DATA"}.= "$host.$domain $intip ($hostint) ";

# $domains->{"$intnetwork"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

# $domains2->{"$intnetwork"}->{"$domain"}->{"DATA"} .= "$host $intip ($hostint): ";

# $domains2->{"$intnetwork"}->{"$domain"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

# }

# }

#}

#LogMsg($D\_INT,"\nEXITING ADDING MISSING MASKS ##########\n");

##############################################################################

## CHECK FOR OVERLAPPING DOMAINS ##################

##############################################################################

### Network design issues. NO DELETE

#

### Domains made up of IP and a MASK.

#my ($dm1,$dm2,$nk1,$nk2,$mk1,$mk2);

#my $K;

#

## DOT NOTATIONS REPRESENTED BY ARRAYS

#my @nk1;

#my @mk1;

#my @nk2;

#my @mk2;

#my @ip1;

#my $i;

#my $j;

#my $restart=TRUE;

#LogMsg($D\_INT,"\nENTERING CHECKING NETWORK:MASK OVERLAPS ##########\n");

#

#while ( $restart && ($j < 15)) {

# $restart=FALSE;

## $j++;

## print "RUN $j\n";

### GO THROUGH ALL REGISTERED NETWORK/MASK COMBOS

#foreach $dm1 (sort keys %{ $domains }) {

# ## COUNT

# $i++;

#

# ## BREAK OUT NETWORK AND MASK

# ($nk1,$mk1)= split /:/, $dm1, 2;

#

# ## IF MASK NOT NULL, PROCESS

# if ( $mk1 ne "$NullValue" ) {

#

# ## BREAK OUT OCTETS FROM FIRST NETWORK/MASK COMBO

# @nk1= split /\./, $nk1, 4;

# @mk1= split /\./, $mk1, 4;

# $mk1T= $mk1[3]+$mk1[2]\*(2\*\*8)+$mk1[1]\*(2\*\*16)+$mk1[0]\*(2\*\*24);

#

# ## CROSS REFERENCE THIS NETWORK/MASK AGAINST

# ## ALL REGISTERED NETWORK/MASK COMBOS

# foreach $dm2 (sort keys %{ $domains }) {

#

##LogMsg($D\_WARN,"$dm1 $dm2: ");

#

# ## ONLY NEED TO CHECK NON NULL AND THINGS

# ## WE HAVENT CHECKED YET. (A=B <=> B=A)

# ($nk2,$mk2)= split /:/, $dm2, 2;

# if ( $domains->{$dm2}->{DATA} eq "") {

##LogMsg($D\_WARN,"KILL 2\n");

# delete($domains->{$dm2});

# } elsif ( $domains->{$dm1}->{DATA} eq "") {

##LogMsg($D\_WARN,"KILL 1\n");

# delete($domains->{$dm1});

# } elsif ( ("$dm2" gt "$dm1") && ($mk2 ne "$NullValue") ) {

## } elsif ( ("$dm2" ne "$dm1") && ($mk2 ne "$NullValue") ) {

##print "Compare $dm1 $dm2\n";

#

# ## BREAK OUT OCTETS FROM SECOND NETWORK/MASK COMBO

# my @x1;

# @nk2= split /\./, $nk2, 4;

# @mk2= split /\./, $mk2, 4;

# $mk2T= $mk2[3]+$mk2[2]\*(2\*\*8)+$mk2[1]\*(2\*\*16)+$mk2[0]\*(2\*\*24);

#

# $nk1T= $nk1[3]+$nk1[2]\*(2\*\*8)+$nk1[1]\*(2\*\*16)+$nk1[0]\*(2\*\*24);

# $nk2T= $nk2[3]+$nk2[2]\*(2\*\*8)+$nk2[1]\*(2\*\*16)+$nk2[0]\*(2\*\*24);

#

# ## IS MASK 1 MORE SPECIFIC?

# if ( $mk2T < $mk1T ) {

##LogMsg($D\_WARN,"MASK 1 ");

#

# ## IF NETWORK 1 LONGER THAN NETWORK 2 and

# ## NETWORK 1 AND MASK 2 = NETWORK 2,

# ## THEN

# ## NETWORK 2 IS A VAGUE DUPLICATE AND SHOULD

# ## BE REPLACED ON ALL DEVICES

# if ( ( $nk1T & $mk2T ) == $nk2T ) {

##LogMsg($D\_WARN," HIT ");

#

# # GRAB MORE SPECIFIC MASK

# LogMsg($D\_WARN,"WARN 1 Domain clash between devices: $domains->{$dm1}->{DATA} and devices:$domains->{$dm2}->{DATA}. Domains $dm1 and $dm2 overlap. Replace more general network:mask.\n");

#

# my @list= split /\~/, $domains->{$dm2}->{IDX};

# my $item;

# $summary->{"Network-overlap"}->{"DATA"}.= ",$dm1 and $dm2 ($#list)";

# $summary->{"Network-overlap"}->{"CNT"}++;

##LogMsg($D\_WARN,"WARN 1 Replacing $#list\n");

##LogMsg($D\_WARN,"STUFF 1 $dm2 $domains->{$dm2}->{IDX}\n");

#

# ## REMOVE ALL DEVICES ASSOCIATED WITH A NETWORK

# foreach $item (@list) {

# if ( $item ne "" ) {

#

# ## SAVE AND BREAK OUT VALUES

# $host=$fields->{"$item"};

# ($domain,$intnetwork,$host,$intip,$hostint)= split /,/, $item, 5;

# ## MARK INVALID ENTRY AS DEAD

# LogMsg($D\_WARN,"REPLACED $item");

# delete($fields->{"$item"});

# $restart=TRUE;

#

# ## BUILD NETWORK FROM IP

# @ip1= split /\./, $intip, 4;

# my $k; $nk1="";

# for ($k=0; $k<4; $k++) {

# $nk1[$k] = $ip1[$k] & $mk1[$k];

# if ( $mk1[$k] ) {

# $nk1.= ".$nk1[$k]";

# }

# }

# $nk1=~ s:^\.::;

#

# ## ADD REPLACEMENT TO HASH WITH CORRECT NETWORK:MASK

# $intnetwork="$nk1:$mk1";

# $domains->{"$intnetwork"}->{"DATA"}.= "$host.$domain $intip ($hostint) ";

# $item="$domain,$intnetwork,$host,$intip,$hostint";

# $domains->{"$intnetwork"}->{"IDX"}.= "~$item";

#

# ## ADD REPLACEMENT TO HASH WITH CORRECT NETWORK:MASK

# $fields->{"$item"}=$host;

# LogMsg($D\_WARN," WITH $item\n");

##print "BAD $item";

# }

# }

#

# ## REMOVE

# delete($domains->{$dm2});

# } else {

##LogMsg($D\_WARN," MISS\n");

# }

# } else {

##LogMsg($D\_WARN,"MASK 2 ");

# ## IF NETWORK 2 LONGER THAN NETWORK 1 and

# ## NETWORK 2 AND MASK 1 = NETWORK 1,

# ## THEN

# ## NETWORK 1 IS A VAGUE DUPLICATE AND SHOULD

# ## BE REPLACED ON ALL DEVICES

# if ( ( $nk2T & $mk1T ) == $nk1T ) {

##LogMsg($D\_WARN," HIT ");

# LogMsg($D\_WARN,"WARN 2 Domain clash between devices: $domains->{$dm2}->{DATA} and devices:$domains->{$dm1}->{DATA}. Domains $dm2 and $dm1 overlap.\n");

# my @list= split /\~/, $domains->{$dm1}->{IDX};

# my $item;

#

# $summary->{"Network-overlap"}->{"DATA"}.= ",$dm2 and dm1 ($#list)";

# $summary->{"Network-overlap"}->{"CNT"}++;

#

##LogMsg($D\_WARN,"WARN 2 Replacing $#list\n");

##LogMsg($D\_WARN,"STUFF 2 $dm1 $domains->{$dm2}->{IDX}\n");

# ## REMOVE ALL DEVICES ASSOCIATED WITH A NETWORK

# foreach $item (@list) {

# if ( $item ne "" ) {

#

# ## SAVE AND BREAK OUT VALUES

# $host=$fields->{"$item"};

# ($domain,$intnetwork,$host,$intip,$hostint)= split /,/, $item, 5;

# ## MARK INVALID ENTRY AS DEAD

# LogMsg($D\_WARN,"REPLACED $item");

# delete($fields->{"$item"});

# $restart=TRUE;

#

# ## BUILD NETWORK FROM IP

# @ip1= split /\./, $intip, 4;

# my $k; $nk1="";

# for ($k=0; $k<4; $k++) {

# $nk1[$k] = $ip1[$k] & $mk2[$k];

# if ( $mk1[$k] ) {

# $nk1.= ".$nk1[$k]";

# }

# }

# $nk1=~ s:^\.::;

#

# ## ADD REPLACEMENT TO HASH WITH CORRECT NETWORK:MASK

# $intnetwork="$nk1:$mk2";

# $domains->{"$intnetwork"}->{"DATA"}.= "$host.$domain $intip ($hostint) ";

# $item="$domain,$intnetwork,$host,$intip,$hostint";

# $domains->{"$intnetwork"}->{"IDX"}.= "~$item";

#

# $fields->{"$item"}=$host;

# LogMsg($D\_WARN," WITH $item\n");

##print "BAD $item";

# }

# }

#

# ## REMOVE

# delete($domains->{$dm1});

# } else {

##LogMsg($D\_WARN," MISS\n");

# }

# }

##print "\n";

# }

# }

# }

#}

##foreach $field (sort keys %{ $fields } ) {

## LogMsg($D\_WARN,"NOW $j: $field\n");

##}

#LogMsg($D\_INT,"Overlapping Networks: In $j loops checked $i cross references\n");

#}

#

### FAIL STOP MEASURE

#if ($j > 15 ) {

# LogMsg($D\_WARN2,"Too many nested overlapping Networks: In $j loops checked $i cross references. Aborting checks.\n");

#}

#

#LogMsg($D\_INT," Processing $j loops with $i cross checks.\n");

#LogMsg($D\_INT,"\nEXITING CHECKING NETWORK:MASK OVERLAPS ##########\n");

###########################################################################

##############################################################################

## CHECK FOR OVERLAPPING NODES ##################

##############################################################################

my $item;

my $item2;

my $i;

my $j;

LogMsg($D\_INT,"\nENTERING CHECKING DUPLICATE NAMES ##########\n");

foreach $item (sort keys %{ $named}) {

$i++;

my @named = split / XzY /,$named->{"$item"}->{"DATA"};

## COUNT OF NODES WITH THIS IP ADDRESS GREATER THAN 1

if ( $#named > 1 ) {

$j++;

LogMsg($D\_WARN,"WARN $item is repeated $#named times. Includes:");

foreach $item2 (@named) {

LogMsg($D\_WARN,"$item2,");

}

LogMsg($D\_WARN,"\n");

$summary->{"Duplicate-Name"}->{"DATA"}.= ",$item ($#named)";

$summary->{"Duplicate-Name"}->{"CNT"}++;

}

}

LogMsg($D\_INT," Processing $j of $i entries.\n");

LogMsg($D\_INT,"\nEXITING CHECKING DUPLICATE NAMES ##########\n");

##############################################################################

## CHECK FOR OVERLAPPING IPS ##################

##############################################################################

my $ip1;

my $ipst;

my $i;

my $j;

LogMsg($D\_INT,"\nENTERING CHECKING IP ADDRESS OVERLAP ##########\n");

foreach $intip (sort keys %{ $ips}) {

my @ipst = split / XzY /,$ips->{"$intip"};

## COUNT OF NODES WITH THIS IP ADDRESS GREATER THAN 1

$i++;

if ( $#ipst > 1 ) {

$j++;

LogMsg($D\_WARN,"WARN Duplicate entries for IP address: $intip. Includes:");

foreach $ip1 (@ipst) {

LogMsg($D\_WARN,"$ip1,");

}

LogMsg($D\_WARN,"\n");

$summary->{"Duplicate-IP-address"}->{"DATA"}.= ",$#ipst entries for $intip";

$summary->{"Duplicate-IP-address"}->{"CNT"}++;

}

}

LogMsg($D\_INT," Processing $j of $i entries.\n");

LogMsg($D\_INT,"\nEXITING CHECKING IP ADDRESS OVERLAP ##########\n");

##############################################################################

## CHECK FOR OVERLAPPING MASKS ##################

##############################################################################

# Network design issue. Same network different masks. NO DELETE

my $i;

my $j;

my $dm1;

my $dm2;

LogMsg($D\_INT,"\nENTERING CHECKING MASK MISMATCH FOR NETWORK ##########\n");

## LOOP THROUGH NETWORKS/MASK

foreach $dm1 (sort keys %{ $domains }) {

$i++;

## IF MASK NOT NULL...

($nk1,$mk1)= split /:/, $dm1, 2;

if ( !( $mk1 =~ /$NullValue/ )) {

## CROSS REFERENCE NETWORK/MASKS

foreach $dm2 (sort keys %{$domains }) {

## IF MASK NOT NULL AND NOT ALREADY PROCESSED (NET/MASK 1 verses NET/MASK 2 is the same

## as NET/MASK 2 verses NET/MASK 1. Ergo NET/MASK 2 > NET/MASK 1)...

($nk2,$mk2)= split /:/, $dm2, 2;

if ( ("$dm2" gt "$dm1") && (!( $mk2 =~ /$NullValue/ )) ) {

my @x1;

## IF NETWORKS MATCH BUT MASKS DONT, WE HAVE A PROBLEM.

if ( "$nk1" eq "$nk2" && "$mk1" ne "$mk2" && "$mk1" ne "$NullValue" && $mk2 ne "$NullValue" ) {

LogMsg($D\_WARN,"WARN Devices using network $nk1 have two different masks: $mk2 and $mk1. (devices: $domains->{$dm2}->{DATA} and devices:$domains->{$dm1}->{DATA}.)\n");

$summary->{"Mask mismatch"}->{"DATA"}.= ",$nk1 uses both $mk1 and $mk2";

$summary->{"Mask mismatch"}->{"CNT"}++;

$j++;

}

}

}

}

}

LogMsg($D\_INT," Processing $j of $i entries.\n");

LogMsg($D\_INT,"\nEXITING CHECKING MASK MISMATCH FOR NETWORK ##########\n");

###########################################################################

##############################################################################

## CHECK FOR OVERLAPPING NETWORKS FOR DOMAINS #############

##############################################################################

## THIS IS NORMAL FOR ROUTERS AND OTHER DEVICES

#my $nk;

#foreach $nk (sort keys %{ $domains2} ) {

# my $cnt=0;

# if ( $nk ne "$NullValue" ) {

# foreach $dm2 (sort keys %{ %{ $domains2->{"$nk"} } } ) {

# if ( $nk ne "$NullValue" ) {

# $cnt++;

# }

# }

# if ( $cnt > 1 ) {

# LogMsg($D\_WARN,"WARN $cnt domains share network $nk. H2N will create overlapping reverse lookup tables for multiple domains: ");

# foreach $dm2 (sort keys %{ %{ $domains2->{"$nk"} } } ) {

# LogMsg($D\_WARN,"Domain $dm2 (devices $domains2->{$nk}->{$dm2}->{DATA}): ");

# }

# LogMsg($D\_WARN,"\n");

# $summary->{"Multiple domains sharing a network"}->{"DATA"}.= ",$nk ($cnt)";

# $summary->{"Multiple domains sharing a network"}->{"CNT"}++;

# }

# }

#}

############################################################################

##############################################################################

## CHECK FOR INTERFACE NAME DUPLICATES ##################

##############################################################################

## DO NOT DELETE

my $hostint;

my $hostint2;

my @hostints;

my $extra;

my $cnt;

my $i;

my $j;

LogMsg($D\_INT,"\nENTERING CHECKING INTERFACE NAME DUPLICATES ##########\n");

## LOOP THROUGH ALL HOST INTERFACE NAMES

foreach $hostint (sort keys %{$nodeints}) {

## COUNT

$i++;

## COUNT UP NUMBER OF DEVICES THAT USE IT.

$cnt=$nodeints->{"$hostint"}->{"Cnt"};

## IF GREATER THAN 1, DO SOMETHING

if ($cnt > 1 ) {

$j++;

## PRINT DETAILS ON EACH ELEMENT

$extra=$nodeints->{"$hostint"}->{"DATA"};

@hostints = split /XzG /, $extra;

LogMsg($D\_WARN,"WARN Interface name clash between $cnt interfaces. ");

foreach $hostint2 (@hostints) {

LogMsg($D\_WARN,"$hostint2: ");

}

LogMsg($D\_WARN,"\n");

$summary->{"Reuse of node-interface name"}->{"DATA"}.= ",$hostint ($cnt)";

$summary->{"Reuse of node-interface name"}->{"CNT"}++;

}

}

LogMsg($D\_INT," Processing $j of $i entries.\n");

LogMsg($D\_INT,"\nEXITING CHECKING INTERFACE NAME DUPLICATES ##########\n");

###########################################################################

##############################################################################

## NORMALIZE NETWORKS TO CLASS C.

## DNS IS NOT CLASS OR CIDR AWARE. HOWEVER IT USES THE '.' AS A DELIMITER

## SO IT EFFECTIVELY IS CLASS BIASED SINCE THIS DELIMITER BREAKS UP THE OCTETS.

## TO REDUCE OVERLAP NMSDNS WILL ASSUME A CLASS C. (GOING TO BEYOND 24 BIT

## SPECIFICATION GETS REALLY UGLY.

##############################################################################

LogMsg($D\_INT,"\nENTERING NORMALIZE NETWORKS ##########\n");

my ($newintnetwork,$domain,$intnetwork,$host,$intip,$hostint,$extra,$field);

my ($mk,$nk);

my @ip;

foreach $field (sort keys %{ $fields } ) {

($domain,$intnetwork,$host,$intip,$hostint,$extra)= split /,/, $field, 6;

@ip= split /\./, $intip, 4;

## ASSUME CLASS C

$newintnetwork="$ip[0].$ip[1].$ip[2]:255.255.255.0";

## DO WE NEED TO UPDATE?

if ( "$newintnetwork" ne "$intnetwork" ) {

## REMOVE OLD ENTRY

delete($fields->{"$field"});

## ADD NEW ENTRY

$fields->{"$domain,$newintnetwork,$host,$intip,$hostint,$extra"} = 1;

LogMsg($D\_WARN,"REPLACED $host.$domain ($intip): $intnetwork with $newintnetwork\n");

($nk,$mk)= split /:/, $intnetwork, 2;

$summary->{"Informational: Mask $mk replaced by class C mask"}->{"DATA"}.= ",$host.$domain $hostint $intip";

$summary->{"Informational: Mask $mk replaced by class C mask"}->{"CNT"}++;

}

}

LogMsg($D\_INT,"\nEXITING NORMALIZE NETWORKS ##########\n");

##############################################################################

## OUTPUT COMMAND FILE

##############################################################################

LogMsg($D\_INT,"\nENTERING OUTPUT COMMAND FILE ##########\n");

foreach $field (sort keys %{ $fields } ) {

chomp($field);

($domain,$intnetwork,$host,$intip,$hostint,$extra)= split /,/, $field, 6;

LogMsg($D\_HOST,"$domain,$intnetwork,$host,$intip,$hostint\n");

}

##############################################################################

## ERROR SUMMARY FILE

##############################################################################

LogMsg($D\_INT,"\nENTERING ERROR SUMMARY FILE ##########\n");

my $item;

my $cnt;

my $list;

my $i;

foreach $item (sort keys %{ $summary} ) {

$i++;

$cnt=$summary->{$item}->{"CNT"};

$list=$summary->{$item}->{"DATA"};

$list=~s:^,::;

LogMsg($D\_WARN2,"$item ($cnt): $list\n");

}

LogMsg($D\_INT," Processing $i\n");

LogMsg($D\_INT,"\nEXITING ERROR SUMMARY FILE ##########\n");

LogMsg($D\_MAIN,"CMD: End of Clean.\n");

## NMSDNSCREATE.pl

This program creates the DNS forward and reverse lookup files as well as the named configuration file based on the output from nmsdnscleen.pl which is stored in the nmsdns.host file.

#!/usr/bin/perl

##########################################################################

# 03/30/2003 Daniel L. Needles Version 1.0 #

# PROGRAM: NMSDNSCREATE.pl #

# USAGE: NMSDNSCREATE.pl [-DEBUG #] [-propsfile FILE] [-logfile FILE] #

# #

# PURPOSE: Create DNS files from a clean table of data. #

##########################################################################

# DLN20030520: Production release. #

##########################################################################

use strict;

my $VERSION = 25;

## CONFIGURABLE GLOBAL VARIABLES (ALLOWS FOR SIMPLE PARAMETER FILE)

use vars qw /$NullValue $MULTIHOMED $PROPSFILE

$SNMPTIMEOUT $SNMPRETRIES $SNMPCOMMUNITYSTRING

$IF $INTABREV $TBLHEADER $DEFAULTMASK $DEFAULTDOMAIN

$NAMEDFILE $LOGFILE $H2NFILE $HOSTFILE $WARNFILE $WARNFILE2

$EXTRAFILE $DEFAULTNAME

$DEBUG $DEBUGMAX

$DEBUGLOG $DEBUGNAMED $DEBUGHOST $DEBUGH2N $DEBUGWARN $DEBUGWARN2

$D\_GOOD $D\_ADMDOWN $D\_INVALID

$D\_INT $D\_MAIN

$D\_LOGFILE $D\_NAMED $D\_NAMED $D\_MH

$D\_1STPASS $D\_PROCESS $D\_2NDPASS

$D\_WARN $D\_WARN2 $D\_H2N

$LOGDIR $PRGDIR $CFGDIR $DNSDIR $HSTDIR $OLDDIR $LOCATION

$DEFAULTDOMAIN/;

use vars qw / $LOGFD $HOSTFD $H2NFD $WARNFD $NAMEDFD

$summary $networks $domains

$domains2 $domains3 $domains4 $ips $nodeints

$fields $named $VALIDMASKS $DBCACHEFILE $DBHEADERFILE/;

## CONSTANTS

use constant TRUE => 1;

use constant FALSE => 0;

require "nmsdns.pl";

###########################################################################

############################## MAIN PROGRAM ###############################

###########################################################################

my ($domain,$intnetwork,$host,$intip,$hostint,$cnt,$extra);

my ($network,$mask);

my $field;

my ($fnamed,$rnamed,$forward,$reverse,$high,$low,$i,$idx,$idx2,$fidx,$ridx);

my @mk;

my @nk;

my @ip;

## INITIALIZE PROGRAMS.

InitPrg($LOGFD | $WARNFD | $NAMEDFD );

## LOAD THE OUTPUT FROM $DEFAULTNAME.pl

H2NLoadCreate();

###########################################################################

open(DB, ">$DNSDIR/db.127.0.0") || die("Can't open 'Loopback db file'");

print DB "\$TTL 86400\n";

print DB "\$INCLUDE db.header\n\n";

print DB "1 PTR localhost.\n";

close(DB);

###########################################################################

## FIRST PASS. DEDUPLICATE DATA

###########################################################################

foreach $field (sort keys %{ $fields } ) {

($domain,$intnetwork,$host,$intip,$hostint,$extra)= split /,/, $field, 6;

($network,$mask) = split /:/, $intnetwork;

@nk= split /\./, $network;

@mk= split /\./, $mask;

## H2N FEATURE DOESNT WORK WITH 9.2.1

## REVERSE LOOKUPS FAIL DUE TO A LACK OF $GENERATE CLAUSES AND NS RECORDS

# if ( $mk[3] > 0 ) {

# $low= $mk[3] & $nk[3];

# $high=$low+(255 - $mk[3]);

# if ( "$low" eq "$high" ) {

# $nk[3]=$low;

# } else {

# $nk[3]="$low-$high";

# }

# }

## CREATE IP ADDRESS FILE NAME

$idx="db";

for ( $i=0; $i<=3; $i++) {

if ( $mk[$i] ) {

$idx.=".$nk[$i]";

}

}

## CREATE IP ADDRESS IDENTIFIER

$idx2="";

for ( $i=3; $i>=0; $i--) {

if ( $mk[$i] ) {

$idx2.=".$nk[$i]";

}

}

$idx2.=".in-addr.arpa";

## REMEMBER KEY ITEMS for NAMED.CONF FILE (rnamed,fnamed)

## REVERSE LOOKUP DATAFILES (reverse)

## FORWARD LOOKUP DATAFILES (forward)

($domain,$intnetwork,$host,$intip,$hostint,$extra)= split /,/, $field, 6;

$idx2 =~ s:^\.::;

#print "$mask:$idx2 $intip $host $hostint\n";

$rnamed->{"$idx2"}=$idx;

$reverse->{"$idx:$idx2"}->{"$host,$intip,$hostint,$domain"}=1;

$fnamed->{"$domain"}="db.$domain";

$forward->{"$domain"}->{"$host,$intip,$hostint"}=1;

}

###########################################################################

## NAMED FILE

###########################################################################

#print "NAMED: $D\_NAMED\n";

LogMsg($D\_NAMED,"\noptions {\n");

LogMsg($D\_NAMED,"\tdirectory \"$DNSDIR\";\n");

LogMsg($D\_NAMED,"\trecursion no;\n");

LogMsg($D\_NAMED,"};\n\n");

LogMsg($D\_NAMED,"zone \".\"\t\t{ type hint;\tfile \"db.cache\"; };\n");

LogMsg($D\_NAMED,"zone \"0.0.127.in-addr.arpa\"\t{ type master;\tfile \"db.127.0.0\"; };\n");

foreach $idx2 (sort keys %{ $rnamed } ) {

#print "NAMED: REV\n";

LogMsg($D\_NAMED,"zone \"$idx2\"\t{ type master;\tfile \"$rnamed->{$idx2}\"; check-names ignore; };\n");

}

foreach $idx2 (sort keys %{ $fnamed } ) {

#print "NAMED: FWD\n";

LogMsg($D\_NAMED,"zone \"$idx2\"\t{ type master;\tfile \"$fnamed->{$idx2}\"; check-names ignore; };\n");

}

###########################################################################

## FORWARD LOOK UP FILES

###########################################################################

foreach $idx (sort keys %{ $forward } ) {

open(FWD, ">$DNSDIR/db.$idx") || die("Can't open '$idx'");

print FWD "\$TTL 86400\n";

print FWD "\$INCLUDE db.header\n\n";

print FWD "localhost A 127.0.0.1\n";

foreach $idx2 (sort keys %{ %{ $forward->{$idx} } } ) {

($host,$intip,$hostint) = split /,/, $idx2, 3;

if ( $hostint eq "$NullValue" ) {

print FWD "$host A $intip\n";

} else {

print FWD "$hostint A $intip\n";

# print FWD "$host CNAME $hostint\n";

}

}

close(FWD);

}

###########################################################################

## REVERSE LOOK UP FILES

###########################################################################

foreach $idx (sort keys %{ $reverse } ) {

($fidx,$ridx) = split /:/, $idx, 2;

open(FWD, ">$DNSDIR/$fidx") || die("Can't open '$idx'");

print FWD "\$TTL 86400\n";

print FWD "\$INCLUDE db.header\n\n";

foreach $idx2 (sort keys %{ %{ $reverse->{$idx} } } ) {

($host,$intip,$hostint,$domain) = split /,/, $idx2, 4;

@ip= split /\./, $intip,4;

#print "I: $idx:$idx2 $host,$intip,$hostint $ip[3]\n";

if ( $hostint eq "$NullValue" ) {

print FWD "$ip[3]\t\tPTR\t$host.$domain.\n";

} else {

print FWD "$ip[3]\t\tPTR\t$hostint.$domain.\n";

}

}

close(FWD);

}

###########################################################################

## COPY db.cache

###########################################################################

`cp $DBCACHEFILE $DNSDIR`;

`cp $DBHEADERFILE $DNSDIR`;

## NMSDNS.pl

The procedures used by the PERL programs in the NMS-DNS solution are contained in this file.

#!/usr/bin/perl

##########################################################################

# 03/30/2003 Daniel L. Needles Version 1.0 #

# PROGRAM: NMSDNS.pl #

# USAGE: require nmsdns.pl #

# #

# PURPOSE: Subrootines for NMS-DNS project. #

##########################################################################

# DLN20030520: Production release. #

# DLN20030522: Do not append interface to node if node ends with #

# interface name. This logic existed but was higher in the #

# code and some itterations slipped through resulting in #

# names such as HOST-na-na-na. That is a "-na" was added per#

# run #

# This is caused by the fact that if sysname and DNS name #

# are not available, the OpenView name is used which may #

# already have an interface postfix. It will have this #

# postfix if the node was originally added with a non prime #

# interface (i.e. not LOOPBACK or SC or the most prime int #

# on the device.) #

# DLN20030523: Check for trailing '-' on interfaces before returning. #

##########################################################################

# LOAD MODULES

use Getopt::Long; # Load module to handle command line opts

my $VERSION=25;

## CONSTANTS

use constant TRUE => 1;

use constant FALSE => 0;

##########################################################################

## BOOTSTRAP PARAMETER -- GETS ALL OTHER VARS FROM THIS

$PROPSFILE="/opt/NMSDNS/config/nmsdns.props";

## BOOTSTRAP PARAMETER -- FOR EXTERNAL PROGRAMS WHEN CALLING INITPRG WITH THESE FLAGS

## BEFORE THE PROPSFILE IS LOADED

$LOGFD=1;

$HOSTFD=2;

$H2NFD=4;

$WARNFD=8;

$NAMEDFD=16;

#LOCAL VARIABLES ####################

#$intnetworks;

$iphostname="$NullValue";

$ipaddress="$NullValue";

$node="$NullValue";

$nodedns="$NullValue";

$sysname="$NullValue";

$sysnamedns="$NullValue";

$DNSnode="$NullValue";

$DNSdns="$NullValue";

$domain="$NullValue";

$interfacecount="$NullValue";

$intname="$NullValue";

$extra="$NullValue";

$intstatus="$NullValue";

$intip="$NullValue";

$intmask="$NullValue";

$getsnmp="$NullValue";

$bad="";

$dispose=$FALSE;

## SPECIFIC FOR THIS PROGRAM

#$networks;

#$domains;

#$domains2;

#$domains3;

#$domains4;

#$ips;

#$nodeints;

#$fields;

#$named;

#$VALIDMASKS;

############################# PROCEDURES #################################

##########################################################################

# PROCEDURE: H2NPopulate

# PURPOSE: DETERMINE NETWORK INFO FOR H2N

##########################################################################

sub H2NPopulate {

my $MASK = shift;

my $intip = shift;

my $intmask = shift;

my $domain = shift;

my $host = shift;

my $hostint = shift;

my $extra = shift;

my $tmp;

## CLEAN UP EXTRA CONTENT INTO A SINGLE FIELD BY REMOVING ',' DELIMITER

## AS WELL AS ANY RECORD DELIMITERS (NEWLINES)

$extra=~s:,:~:g;

$extra=~s:\n::g;

## LOCAL VARIABLES

my $intnetwork="";

my $K;

my @intmask = split /\./, $intmask, 4;

my @intip = split /\./, $intip, 4;

## ENTERING

LogMsg($D\_INT,"\nENTERING H2NPopulate() ##########\n");

## ADD TO NETWORKS IF MASTER DEBUG ON AND TYPE COMPATIBLE

## OUTPUT, DEAD NODE, OR NULL NODE (NO HOSTNAME KNOWN)

## EVENTUALLY THESE ARE USED TO CREATE THE H2N COMMAND FILE

if ( $MASK & $DEBUG & $DEBUGHOST ) {

## BUILD MASK TABLE FOR SEARCHING UNKNOWN MASKS

if ( ! ( $intmask =~ /^[0-9\.]+$/ ) ) {

$intmask="$NullValue";

$intnetwork="$NullValue";

## VALID CHARACTERS. BEGIN WORK

} else {

## PARSE THROUGH BYTES IN MASK AND IP TO FIND NETWORK ID

for ($K=0; $K<4; $K++) {

## GRAB ONLY NESSISARY BITS

if ( $intmask[$K] == 0 ) {

$K=5;

} elsif ( $intmask[$K] < 255 ) {

$intip[$K] = ($intip[$K]) & ($intmask[$K]);

$intnetwork = "$intnetwork.$intip[$K]";

} else {

$intnetwork = "$intnetwork.$intip[$K]";

}

}

## REMOVE LEADING PERIOD

$intnetwork=~ s:^\.::;

}

## BUILD OPTION FOR H2N

$intnetwork= "$intnetwork:$intmask";

# LogMsg($D\_2NDPASS," Add $intnetwork\n");

## FILL LOGICAL FLATFILE

if (defined($intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"})) {

$intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"}->{"Cnt"}++;

$intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"}->{"Extra"}.="~~$extra";

# LogMsg($D\_2NDPASS," $domain,$intnetwork,$host,$intip,$hostint");

} else {

$intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"}->{"Cnt"}=1;

$intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"}->{"Extra"}="$extra";

# LogMsg($D\_2NDPASS," $domain,$intnetwork,$host,$intip,$hostint");

$tmp=$intnetworks->{"$domain"}->{"$intnetwork"}->{"$host"}->{"$intip"}->{"$hostint"}->{"Cnt"};

# LogMsg($D\_2NDPASS,",$tmp\n");

}

} else {

# LogMsg($D\_2NDPASS," Skip Network add for $intip:$intmask since $MASK not in $DEBUG and $DEBUGHOST\n");

}

LogMsg($D\_INT,"LEAVING H2NPopulate() ##########\n\n");

}

##########################################################################

# PROCEDURE: H2NLoad

# PURPOSE: DETERMINE NETWORK INFO FOR H2N

##########################################################################

sub H2NLoadClean {

my ($domain,$intnetwork,$host,$intip,$hostint,$cnt,$extra);

my ($line,$lineno,$lineact);

my ($dm1,$ip1,$nk1,$mk1,$mk1T);

my @ip1;

my @mk1;

my @nk1;

my @domain;

my $err;

my $i;

my $node;

my $STAGE;

LogMsg($D\_INT,"\nENTERING H2NLoad() ##########\n");

$lineno=0;

## CALCULATE VALID MASKS (ALL HIGH ORDER BITS ON; ALL LOW ORDER BITS OFF)

$cnt=0;

## BITS ABOVE CLASS C (DNS EASY IMPLEMENTATION)

for ($i=31; $i>=8; $i--) {

$cnt+=2\*\*$i;

$VALIDMASKS->{$cnt}=1;

}

## BITS BELOW CLASS C (DNS HARD IMPLEMENTATION)

for ($i=7; $i>=0; $i--) {

$cnt+=2\*\*$i;

$VALIDMASKS->{$cnt}=2;

}

## PREPAIR HEADER

chomp($TBLHEADER);

## PROCESS DATA

$STAGE=0;

while ($STAGE < 2 ) {

## DETERMINE WHICH FILE IS BEING PROCESSED: OPENVIEW OR MANUAL

## PROCESSING OPENVIEW OUTPUT

if ($STAGE == 0) {

## IF END OF OPENVIEW OUTPUT, ...

if (!( $line = <H2N> )) {

## PROCESS MANUAL FILE

if ( $EXTRAFILE && -f $EXTRAFILE ) {

$STAGE=1;

## NO MANUAL FILE; QUIT

} else {

$STAGE=2;

loop;

}

## STILL OPENVIEW DATA TO PROCESS (STAGE 0)

}

}

## PROCESSING MANUAL OUTPUT

if ($STAGE == 1) {

## IF END OF MANUAL FILE, SIGNAL END

if (!($line = <EXTRA>)) {

$STAGE=2;

loop;

}

}

## PREPROCESS LINE

$lineno++;

chomp($line);

## ENTIRE LINE CHECKS

if ( $line =~ /^#/ ) {

LogMsg($D\_MAIN,"Comment line: $line. Record discarded.\n");

} elsif ( $line eq "$TBLHEADER" ) {

LogMsg($D\_MAIN,"Header: $line. Record discarded.\n");

} elsif ( $line =~ /^\s+$/ ) {

LogMsg($D\_MAIN,"Blank line: $line. Record discarded.\n");

} elsif ( $line =~ /^$/ ) {

LogMsg($D\_MAIN,"Empty line: $line. Record discarded.\n");

## LINE AS A WHOLE VALID

} else {

## CUT LINE INTO DATA VIA ',' DELIMITER

( $domain, $intnetwork, $host, $intip, $hostint, $cnt, $extra) = split /,/, $line, 7;

$node="$host.$domain $intip ($hostint) ($intnetwork)";

## DOCUMENT PROGRESS

LogMsg($D\_HOSTNAME,"Domain: $domain\n");

LogMsg($D\_HOSTNAME,"NetworkNMask: $intnetwork\n");

LogMsg($D\_HOSTNAME,"Hostname: $host\n");

LogMsg($D\_HOSTNAME,"IP Address: $intip\n");

LogMsg($D\_HOSTNAME,"Interface Name: $hostint\n");

LogMsg($D\_HOSTNAME,"Count: $cnt\n");

LogMsg($D\_HOSTNAME,"Extra: $extra\n");

## REMOVE LINES WITH NULL VALUES

if ( $domain eq "" || $intnetwork eq "" || $host eq "" || $intip eq "" || $hostint eq "" ) {

LogMsg($D\_WARN,"WARN Nulls detected in the entry $line.\n");

LogMsg($D\_WARN,"REMOVED $line.\n");

$summary->{"Missing-Data"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Missing-Data"}->{"CNT"}++;

## REMOVE LINES WITH IP SET TO PROGRAMMED NULL

} elsif ( $intip eq "$NullValue" ) {

LogMsg($D\_WARN,"WARN IP Address set to NULL $host.$domain $intip ($hostint) ($intnetwork). Record discarded\n");

LogMsg($D\_MAIN,"IP Address set to NULL $host.$domain $intip ($hostint) ($intnetwork) ($extra). Record discarded\n");

LogMsg($D\_WARN,"REMOVED $line.\n");

$summary->{"Null-IP-In-OpenView"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Null-IP-In-OpenView"}->{"CNT"}++;

## REMOVE LINES WITH INVALID CHARACTERS IN IP

} elsif ( ! ($intip && $intip =~ /^[0-9\.]+$/ ) ) {

LogMsg($D\_WARN,"$host ($node) has invalid characters in its IP Address\n");

$summary->{"Invalid characters in IP Address"}->{"DATA"}.= ",'$intip': $host.$domain $hostint ($intip)";

$summary->{"Invalid characters in IP Address"}->{"CNT"}++;

## REMOVE LINES WITH IPs WITH LEADING 0 OR .0.0

} elsif ( $intip =~ /^0/ || $intip =~ /\.0\.0/ ) {

LogMsg($D\_WARN,"WARN IP Address set to invalid value $host.$domain $intip ($hostint) ($intnetwork). Record discarded\n");

LogMsg($D\_WARN,"WARN IP Address set to invalid value $host.$domain $intip ($hostint) ($intnetwork) ($extra). Record discarded\n");

LogMsg($D\_WARN,"REMOVED $line.\n");

$summary->{"Null-IP-In-OpenView"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Null-IP-In-OpenView"}->{"CNT"}++;

## REMOVE LINES WITH INVALID CHARACTERS IN IP

} elsif ( ! ($intip && $intip =~ /^[0-9\.]+$/ ) ) {

LogMsg($D\_WARN,"$host ($node) has invalid characters in its IP Address\n");

$summary->{"Invalid characters in IP Address"}->{"DATA"}.= ",'$intip': $host.$domain $hostint ($intip)";

$summary->{"Invalid characters in IP Address"}->{"CNT"}++;

## REMOVE LINES WITH IPs WITH LEADING 0 OR .0.0

} elsif ( $intip =~ /^0/ || $intip =~ /\.0\.0/ ) {

LogMsg($D\_WARN,"WARN IP Address set to invalid value $host.$domain $intip ($hostint) ($intnetwork). Record discarded\n");

LogMsg($D\_WARN,"WARN IP Address set to invalid value $host.$domain $intip ($hostint) ($intnetwork) ($extra). Record discarded\n");

LogMsg($D\_WARN,"REMOVED $line.\n");

$summary->{"Leading or multiple 0 octets in IP Address"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Leading or multiple 0 octets in IP Address"}->{"CNT"}++;

## REMOVE LINES WITH PROGRAMMED NULL HOST

} elsif ( $host eq "$NullValue" ) {

LogMsg($D\_WARN,"WARN Device name set to NULL $host.$domain $intip ($hostint) ($intnetwork). Record discarded\n");

LogMsg($D\_MAIN,"Device name set to NULL $host.$domain $intip ($hostint) ($intnetwork) ($extra). Record discarded\n");

LogMsg($D\_WARN,"REMOVED $line.\n");

$summary->{"Null-Host-In-OpenView"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Null-Host-In-OpenView"}->{"CNT"}++;

## REMOVE LINES WITH INVALID HOSTNAME

} elsif ( !($host=~/^[A-Za-z0-9-\_]+$/) ) {

LogMsg($D\_WARN,"$host ($node) has invalid characters (ONLY characters:A-Z,a-z,0-9,'-', or '\_' allowed.)\n");

$summary->{"Invalid characters in host name"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Invalid characters in host name"}->{"CNT"}++;

## REMOVE LINES WITH INVALID INTERFACE HOSTNAME

} elsif ( !($hostint=~/[A-Za-z0-9-\_]+/) ) {

LogMsg($D\_WARN,"$host ($node) interface has invalid characters (ONLY characters: A-Z,a-z,0-9,'-', or '\_' allowed.)\n");

$summary->{"Invalid characters in host-interface name"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Invalid characters in host-interface name"}->{"CNT"}++;

## NON CACULATED ERROR DETECTIONS COMPLETE

} else {

## PROCESS IP ADDRESS (DETECT IP OCTETS OUT OF RANGE)

@ip1= split /\./, $intip;

$err=0;

foreach $i (@ip1) {

if ( ($i < 0) || ($i > 255) ) {

$err++;

}

}

if ( $err ) {

LogMsg($D\_WARN,"$host ($node) has $err errored octets in IP address.\n");

$summary->{"IP has invalid octets."}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"IP has invalid octets."}->{"CNT"}++;

## MORE CALCULATED CHECKS

} else {

## VALIDATE NETWORK AND MASK

($nk1,$mk1) = split /:/,$intnetwork,2;

########### MASK CHECKS #################

## DIGITAL MASK?

if ( $mk1 =~ /^[0-9\.]+$/ ) {

@mk1 = split /\./, $mk1, 4;

$mk1T= $mk1[3]+$mk1[2]\*(2\*\*8)+$mk1[1]\*(2\*\*16)+$mk1[0]\*(2\*\*24);

## NON VALID MASK. USE DEFAULT

if ( $VALIDMASKS->{$mk1T} <= 0 ) {

$summary->{"Illegal mask value. Default used."}->{"DATA"}.= ",$intnetwork: $host.$domain $hostint ($intip)";

$summary->{"Illegal mask value. Default used."}->{"CNT"}++;

$mk1="$NullValue";

}

## NON DIGITAL MASK. USE DEFAULT

} elsif ( $mk1 ne "$NullValue") {

$summary->{"Illegal characters in mask. Default used."}->{"DATA"}.= ",'$mk1': $host.$domain $hostint ($intip)";

$summary->{"Illegal characters in mask. Default used."}->{"CNT"}++;

$mk1="$NullValue";

}

########### NETWORK CHECKS #################

## INVALID CHARACTERS?

if ( ! ($nk1 && $nk1 =~ /^[0-9\.]+$/ ) && ($nk1 ne "NullValue" ) && ($nk1 ne "")) {

LogMsg($D\_WARN,"$host ($node) has invalid characters in its Network Address\n");

$summary->{"Ilegal characters in network. Calculate true network."}->{"DATA"}.= ",'$nk1': $host.$domain $hostint ($intip)";

$summary->{"Ilegal characters in network. Calculate true network."}->{"CNT"}++;

$nk1="$NullValue";

## NULL. BUT NOT A BIG DEAL. SET TO DEFAULT.

} elsif ( $nk1 eq "") {

$nk1="$NullValue";

## VALID CHARACERS IN NETWORK

} else {

## PROCESS NETWORK (DETECT IP OCTETS OUT OF RANGE)

@nk1= split /\./, $nk1;

$err=0;

foreach $i (@nk1) {

if ( ($i < 0) || ($i > 255) ) {

$err++;

}

}

if ( $err ) {

LogMsg($D\_WARN,"$host ($node) has $err errored octets in Network address.\n");

$summary->{"Network has octets out of range. Calculate true network."}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Network has octets out of range. Calculate true network."}->{"CNT"}++;

$nk1= "$NullValue";

}

}

## RECALCULATE NETWORK:MASK

$intnetwork="$nk1:$mk1";

## COLLECT VALID MASKS BY NETWORK (TRACK OVERLAPS)

if ( !( $intnetwork =~ /$NullValue/ ) ) {

if ($networks->{"$nk1"} && ($mk1 ne $networks->{"$nk1"})) {

LogMsg($D\_WARN,"WARN conflicting masks on network $nk1: $mk1 and $networks->{$nk1}\n");

$summary->{"Multiple-Masks-Used-On-Network"}->{"DATA"}.= ",$nk1 uses $mk1 and $networks->{$nk1}";

$summary->{"Multiple-Masks-Used-On-Network"}->{"CNT"}++;

$networks->{"$nk1"}= $mk1;

}

}

## CHECK COUNTS (RECOVERABLE ERROR)

if ( $cnt > 1 ) {

LogMsg($D\_WARN,"WARN duplicate entries in OpenView: $host.$domain $intip ($hostint)\n");

$summary->{"Duplicate-Object-In-OpenView"}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Duplicate-Object-In-OpenView"}->{"CNT"}++;

}

## DETECT COMMON INVALID DOMAIN (RECOVERABLE)

if ( $domain eq "company" ) {

$summary->{"Invalid domain COMPANY. Using default."}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Invalid domain COMPANY. Using default."}->{"CNT"}++;

$domain = $DEFAULTDOMAIN;

## CHECK DOMAIN (RECOVERABLE ERRORS)

} elsif ( $domain eq "$NullValue" ) {

LogMsg($D\_WARN,"WARN $host ($intip $intnetwork) does not have domain set. Using default $DEFAULTDOMAIN\n");

$summary->{"Missing domain. Default used."}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Missing domain. Default used."}->{"CNT"}++;

$domain=$DEFAULTDOMAIN;

## VALID DOMAIN IS LETTERS ONLY PUNCTUATED BY PERIODS

} else {

@domain= split /\./, $domain;

$err=0;

foreach $dm1 (@domain) {

if ( $dm1 =~ /A-Za-z/ ) {

$err++;

}

}

if ( $err ) {

LogMsg($D\_WARN,"$host ($node) has $err errored sub domains in domain of $host. Default used.\n");

$summary->{"Invalid Domain. Default used."}->{"DATA"}.= ",$host.$domain $hostint ($intip)";

$summary->{"Invalid Domain. Default used."}->{"CNT"}++;

$domain=$DEFAULTDOMAIN;

}

}

################ MAIN DATA ELEMENT PROCESSING ##################

# MEMORIZE RECORD

$fields->{"$domain,$intnetwork,$host,$intip,$hostint"} = "$host.$domain ($intip) $extra";

$lineact++;

## OVERLAPPING NETWORKS AND MASKS OVER IP SPACE

## (IGNORE PARTIALLY NULL RECORDS)

if ( ! ($intnetwork =~ /$NullValue/) ) {

## COLLECT HOSTS ON NETWORK:MASK

$domains->{"$intnetwork"}->{"DATA"}.= "$host.$domain $intip ($hostint) ";

$domains->{"$intnetwork"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

## COLLECT HOSTS ON NETWORK:MASK:DOMAIN

if ( $domain ne "$NullValue" ) {

$domains2->{"$intnetwork"}->{"$domain"}->{"DATA"} .= "$host $intip ($hostint): ";

$domains2->{"$intnetwork"}->{"$domain"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

}

## COLLECTS FIRST ORDER OCTETS (FOR -a H2N OPTION)

$nk1=~ /(.\*?)\./;

$domains4->{"$1"}++;

}

## COLLECT DOMAINS (NAMES)

if ( $domain ne "$NullValue") {

$domains3->{"$domain"}++;

}

## COLLECT DUPLICATE INTERFACE-HOST NAMES

if ( $hostint ne "$NullValue" ) {

if ( $nodeints->{"$hostint"} ) {

$nodeints->{"$hostint"}->{"DATA"} .= "XzG $host.$domain $hostint ($intip) $extra";

$nodeints->{"$hostint"}->{"IDX"}.= "~$domain,$intnetwork,$host,$intip,$hostint";

$nodeints->{"$hostint"}->{"Cnt"}++;

} else {

$nodeints->{"$hostint"}->{"DATA"} = "$host.$domain $hostint ($intip) $extra";

$nodeints->{"$hostint"}->{"IDX"} = "$domain,$intnetwork,$host,$intip,$hostint";

$nodeints->{"$hostint"}->{"Cnt"}=1;

}

## COLLECT DUPLICATE NODES IN DATABASE

## (INDICATES DIFFERENT OBJECT IDs)

} else {

if (($named->{"$host"}->{"LINENO"}) &&

($lineno ne $named->{"$host"}->{"LINENO"}+1)) {

$named->{"$host"}->{"DATA"}.= " XzY $host.$domain ($intip $intnetwork)";

} else {

$named->{"$host"}->{"DATA"}= "$host.$domain ($intip $intnetwork)";

}

$named->{"$host"}->{"LINENO"}=$lineno;

}

## COLLECT OVERLAPPING IPs

if ( $ips->{"$intip"} ) {

$ips->{"$intip"}.= " XzY $host.$domain ($hostint)";

} else {

$ips->{"$intip"}= "$host.$domain ($hostint) ";

}

}

}

}

}

LogMsg($D\_INT," Imported $lineact of $lineno lines\n\n");

## CHECK FOR LARGE ENOUGH LOAD

if ( $lineact < 5 ) {

LogMsg($D\_WARN,"E: OpenView did not produce 5 host interfaces. Skipping.\n");

LogMsg($D\_WARN2,"E: OpenView did not produce 5 host interfaces. Skipping.\n");

exit -1;

}

LogMsg($D\_INT,"LEAVING H2NLoad() ##########\n\n");

}

##########################################################################

# PROCEDURE: H2NLoadCreate

# PURPOSE: DETERMINE NETWORK INFO FOR H2N

##########################################################################

sub H2NLoadCreate {

my ($domain,$intnetwork,$host,$intip,$hostint,$cnt,$extra);

my $field;

my $lineact=0;

LogMsg($D\_INT,"\nENTERING LOAD HOST H2N FILE ##########\n");

#print "OPENING\n";

while ( $field = <HOST> ) {

chomp($field);

($domain,$intnetwork,$host,$intip,$hostint,$extra)= split /,/, $field, 6;

$fields->{"$domain,$intnetwork,$host,$intip,$hostint"}=1;

$lineact++;

#print "LOADED: $field\n";

}

LogMsg($D\_INT,"\nEXITING LOAD HOST H2N FILE ##########\n");

## CHECK FOR LARGE ENOUGH LOAD

if ( $lineact < 5 ) {

LogMsg($D\_WARN,"E: OpenView did not produce 5 host interfaces. Skipping.\n");

LogMsg($D\_WARN2,"E: OpenView did not produce 5 host interfaces. Skipping.\n");

exit -1;

}

}

##########################################################################

# PROCEDURE: IntNameClean

# PURPOSE: Cleans up inteface name

##########################################################################

sub IntNameClean {

my $intname = shift;

my $nodet = shift;

my ($name, $idx, $junk);

LogMsg($D\_INT,"\nENTERING IntNameClean() ##########\n");

# CANT DUE TO '.' in subinterfaces

# ## EXTRACT DNS FROM HOSTNAME IF THERE

# if ($intname=~ /(.\*?)\.(.\*)/) {

# $intname=$1;

# }

# IP ADDRESS

if ( $intname =~ /\d+\.\d+\.\d+\.\d+/ ) {

$intname="";

# ## SPECIAL CASE NODE CONTAINS INTERFACE KILL INTERFACE

# } elsif ( $nodet =~ /$intname$/ ) {

# $name="";

# $idx="";

# $intname="$name$idx";

## SPECIAL CASE

} elsif ( $intname =~ /^eth-(.\*)/ ) {

$name="eth";

$idx=$1;

chomp($idx);

$intname="$name$idx";

## SPECIAL CASE

} elsif ( $intname =~ /^tms(.\*)/ ) {

$name="tms";

$idx=$1;

chomp($idx);

$intname="$name$idx";

## SOMETHING INTERESTING

} else {

## CLEAN OUT ILLEGAL DNS CHARACTERS

$intname=~ s/:/-/g;

$intname=~ s/\./-/g;

$intname=~ s/\///g;

## FORCE LOWER CASE

$intname=~ tr/A-Z/a-z/;

## TRUNCATE PROPER NAME

if ( $intname =~ /([a-z-\_]+)([0-9][0-9-]\*)(.\*)/ ) {

$name=$1;

$idx=$2;

$junk=$3;

## FIND NAME

if ( $INTABREV->{$name} ) {

$name=$INTABREV->{$name};

## REMOVE TRAILING '-' ON INDEX

$idx=~ s/-$//;

## REBUILD INTERFACE NAME

$intname="$name$idx";

## NO ABBREV FOUND; FLAG IT AND DETERMINE THE DAMAGE

} else {

## INVALID CHARACTERS USED IN INDEX?

if ( !( $intname =~ /^[a-z0-9-\_]+$/ )) {

LogMsg($D\_WARN,"WARN: (2) Unrecognized interface discovered with invalid characters (not a-z, A-Z, 0-9 or '-' or '\_'): \"$intname\". Interface name set to INVALID. Program nmsdnsfeed.pl in procedure IntNameClean() needs to be updated to detect this interface type. \n");

$summary->{"Interface parsable, but unrecognized and contains invalid characters"}->{"DATA"}.= ",$intname";

$summary->{"Interface parsable, but unrecognized and contains invalid characters"}->{"CNT"}++;

$intname="INVALID";

## ELSE KEEP THE INTNAME

} else {

## FIND NAME

if ( $INTABREV->{$intname} ) {

$name=$INTABREV->{$intname};

} else {

LogMsg($D\_WARN,"WARN Please update $PROPSFILE with correct abbreviation for interface type $name (found in $intname)\n");

$summary->{"Interface parsable, but unrecognized"}->{"DATA"}.= ",$intname";

$summary->{"Interface parsable, but unrecognized"}->{"CNT"}++;

}

}

}

## NULL CONVERT THESE TO NORMALIZED NULL

} elsif ( $intname eq "" ) {

$intname="";

## NULL IGNORE THESE

} elsif ( $intname eq "$NullValue" ) {

$intname="";

## INVALID CHARACTERS

} elsif ( !( $intname =~ /^[a-zA-Z0-9-\_]+$/ )) {

## TRY TO FIND NAME

if ( $INTABREV->{$intname} ) {

$intname=$INTABREV->{$intname};

## GIVE UP

} else {

LogMsg($D\_WARN,"WARN: Unrecognized interface discovered with invalid characters (not a-z, A-Z, 0-9, '-', or '\_'): \"$intname\". Interface name set to INVALID. Program nmsdnsfeed.pl in procedure IntNameClean() needs to be updated to detect this interface type. \n");

$summary->{"Unparsable interface with invalid characters"}->{"DATA"}.= ",$intname";

$summary->{"Unparsable interface with invalid characters"}->{"CNT"}++;

$intname="INVALID";

}

## OTHERWISE UNKNOWN FORMAT BUT VALID CHARACTERS SO DONT TOUCH

} else {

## TRY TO FIND NAME

if ( $INTABREV->{$intname} ) {

$intname=$INTABREV->{$intname};

## GIVE UP

} else {

LogMsg($D\_WARN,"WARN Unrecognized interface discovered: \"$intname\". This will be left \"as-is.\" Program nmsdnsfeed.pl in procedure IntNameClean() needs to be updated to detect this interface type.\n");

$summary->{"Unparsable interface"}->{"DATA"}.= ",$intname";

$summary->{"Unparsable interface"}->{"CNT"}++;

}

}

}

LogMsg($D\_INT,"\nEXITING IntNameClean() ##########\n");

## SPECIAL CASE NODE CONTAINS INTERFACE KILL INTERFACE

## CAUSED BY USING OPENVIEW NAME FOR ROUTER OR SWITCH WHICH WAS ADDED

## TO OPENVIEW WITHOUT THE RIGHT (MOST PRIME) INTERFACE

if ( $nodet =~ /$intname$/ ) {

$intname="";

## REMOVE TRAILING '-'

} elsif ( $intname=~ /(.\*)-$/ ) {

$intname=$1;

}

return($intname);

}

##########################################################################

# PROCEDURE: commandline

# PURPOSE: Parses commandline parameters and sets their values

##########################################################################

sub commandline {

LogMsg($D\_INT,"\nENTERING commandline() ##########\n");

GetOptions("propsfile=s" => \$PROPSFILE,

"DEBUG=i" => \$DEBUGLOG,

"logfile=s" => \$LOGFILE,

"hostfile=s" => \$HOSTFILE,

"extrafile=s" => \$EXTRAFILE,

"h2nfile=s" => \$H2NFILE,

"warnfile=s" => \$WARNFILE,

"namedfile=s" => \$NAMEDFILE);

LogMsg($D\_INT," propsfile => $PROPSFILE\n");

LogMsg($D\_INT," DEBUG => $DEBUG\n");

LogMsg($D\_INT," logfile => $LOGFILE\n");

LogMsg($D\_INT," h2nfile => $H2NFILE\n");

LogMsg($D\_INT," warnfile => $WARNFILE\n");

LogMsg($D\_INT," hostfile => $HOSTFILE\n");

LogMsg($D\_INT," extrafile => $EXTRAFILE\n");

LogMsg($D\_INT," namedfile => $NAMEDFILE\n");

LogMsg($D\_INT,"LEAVING commandline () ##########\n\n");

}

##########################################################################

# PROCEDURE: loadprops

# PURPOSE: Parses properties files and extracts values.

##########################################################################

sub loadprops {

LogMsg($D\_INT,"\nENTERING loadprops() ##########\n");

#print "PROPSFILE $PROPSFILE\n";

# LOAD THE PROPERTIES FILE

if ( -f "$PROPSFILE" ) {

LogMsg($D\_INT," The configuration file \"$PROPSFILE\" is being loaded.\n");

require "$PROPSFILE";

} else {

LogMsg($D\_WARN,"WARN: The configuration file \"$PROPSFILE\" does not exist.\n");

}

LogMsg($D\_INT,"\nEXITING loadprops() ##########\n");

}

##########################################################################

# PROCEDURE: LogMsg

# PURPOSE: Write messages to logfiles, standard out, depending on

# three things:

# 1. Type of log message (specified in the call to the process via a

# statement debug switch)

# 2. Master debug switch

# 3. Output debug switches.

##########################################################################

sub LogMsg {

my $lvl = shift;

my $msg = shift;

# NOTE: Must not call LogMsg() to

# print this or we enter an

# infinite loop!

#($DEBUG & $D\_INT) && print "\nENTERING LogMsg() ##########\n";

## DEBUG LOG FILE

if ( $LOGFILE && ($lvl & $DEBUG & $DEBUGLOG)) {

# print "$msg";

print LOG "$msg";

}

## H2N FILE LIST OF COMMANDS

if ( $H2NFILE && ($lvl & $DEBUG & $DEBUGH2N )) {

print H2N "$msg";

}

## ROLLED UP DIAGNOSTIC MESSAGES

if ( $lvl & $DEBUG & $DEBUGHOST ) {

print HOST "$msg";

}

## HOST FILE LIST OF HOSTS (USED LATER TO CREATE H2N COMMANDS)

if ( $lvl & $DEBUG & $DEBUGNAMED ) {

print NAMED "$msg";

}

## INDIVIDUAL DIAGNOSTIC MESSAGES

if ( $lvl & $DEBUG & $DEBUGWARN ) {

print WARN "$msg";

}

## ROLLED UP DIAGNOSTIC MESSAGES

if ( $lvl & $DEBUG & $DEBUGWARN2 ) {

print WARN2 "FAILURE: $msg\n";

}

# NOTE: Must not call LogMsg() to

# print this or we enter an

# infinite loop!

#($DEBUG & $D\_INT) && print "LEAVING LogMsg() ##########\n\n";

}

##########################################################################

# PROCEDURE: InitPrg

# PURPOSE: Parse command line, load parameter file, open other files, etc.

##########################################################################

sub InitPrg {

my $write = shift;

## LOG FILE NOT READY SKIP

LogMsg($D\_INT,"\nENTERING InitPrg() ##########\n");

## PARSE COMMAND LINE OPTIONS commandline();

commandline();

## LOAD PROPERTIES FILE

loadprops();

## OPEN LOG FILE

if ( $LOGFILE) {

if ( $write & $LOGFD ) {

LogMsg($D\_MAIN,"opening to append to log file '$LOGFILE'\n");

open(LOG, ">>$LOGFILE") || die("Can't open '$LOGFILE'");

} else {

LogMsg($D\_MAIN,"opening to read from log file '$LOGFILE'\n");

open(LOG, "<$LOGFILE") || die("Can't open '$LOGFILE'");

}

select(LOG); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

# OPEN EXTRA HOST FILE (MANUAL ENTRIES)

if ( $EXTRAFILE && -f $EXTRAFILE ) {

LogMsg($D\_MAIN,"opening to read from MANUAL file '$EXTRAFILE'\n");

open(EXTRA, "<$EXTRAFILE") || die("Can't open '$EXTRAFILE'");

}

select(EXTRA); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

# OPEN H2N COMMAND FILE

if ( $H2NFILE) {

if ( $write & $H2NFD) {

LogMsg($D\_MAIN,"opening to write a new H2N file '$H2NFILE'\n");

open(H2N, ">$H2NFILE") || die("Can't open '$H2NFILE'");

} else {

if ( -f $H2NFILE ) {

LogMsg($D\_MAIN,"opening to read from H2N file '$H2NFILE'\n");

open(H2N, "<$H2NFILE") || die("Can't open '$H2NFILE'");

}

}

select(H2N); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

# OPEN NAMED FILE

if ( $NAMEDFILE) {

if ( $write & $NAMEDFD ) {

LogMsg($D\_MAIN,"opening to write to host file '$NAMEDFILE'\n");

open(NAMED, ">$NAMEDFILE") || die("Can't open '$NAMEDFILE'");

} else {

if ( -f $NAMEDFILE ) {

LogMsg($D\_MAIN,"opening to read from host file '$NAMEDFILE'\n");

open(NAMED, "<$NAMEDFILE") || die("Can't open '$NAMEDFILE'");

}

}

select(NAMED); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

# OPEN HOST FILE

if ( $HOSTFILE) {

if ( $write & $HOSTFD ) {

LogMsg($D\_MAIN,"opening to write to host file '$HOSTFILE'\n");

open(HOST, ">$HOSTFILE") || die("Can't open '$HOSTFILE'");

} else {

if ( -f $HOSTFILE ) {

LogMsg($D\_MAIN,"opening to read from host file '$HOSTFILE'\n");

open(HOST, "<$HOSTFILE") || die("Can't open '$HOSTFILE'");

}

}

select(HOST); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

# OPEN WARN FILE

if ( $WARNFILE) {

if ( $write & $WARNFD ) {

LogMsg($D\_MAIN,"opening to append to verbose warn file '$WARNFILE'\n");

open(WARN, ">>$WARNFILE") || die("Can't open '$WARNFILE'");

} else {

LogMsg($D\_MAIN,"opening to read from verbose warn file '$WARNFILE'\n");

open(WARN, "<$WARNFILE") || die("Can't open '$WARNFILE'");

}

select(WARN); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

# OPEN WARN2 FILE

if ( $WARNFILE2) {

if ( $write & $WARNFD ) {

LogMsg($D\_MAIN,"opening to append to concise warn file '$WARNFILE2'\n");

open(WARN2, ">>$WARNFILE2") || die("Can't open '$WARNFILE2'");

} else {

LogMsg($D\_MAIN,"opening to read from concise warn file '$WARNFILE2'\n");

open(WARN2, "<$WARNFILE2") || die("Can't open '$WARNFILE2'");

}

select(WARN2); $|=1; select(STDOUT);

# LogMsg($D\_MAIN," done\n");

}

LogMsg($D\_INT," DEBUG = $DEBUG\n");

LogMsg($D\_INT," DEBUGLOG = $DEBUGLOG\n");

LogMsg($D\_INT," DEBUGWARN = $DEBUGWARN\n");

LogMsg($D\_INT," DEBUGWARN2 = $DEBUGWARN2\n");

LogMsg($D\_INT," DEBUGHOST = $DEBUGHOST\n");

LogMsg($D\_INT," DEBUGH2N = $DEBUGH2N\n");

LogMsg($D\_INT," DEBUGNAMED = $DEBUGNAMED\n");

LogMsg($D\_INT,"\nEXITING InitPrg() ##########\n");

}

# END # PROCEDURES ########################################################

## REQUIRED BY PACKAGES

1;

## NMSDNS.props

This configuration file enables the user to change the parameters to the PERL scripts in a single location. (The NMSDNS.sh script has to be configured separately.)

#!/usr/bin/perl

#######################################################################

#

# Copyright (C) 2003 - Daniel L. Needles, INC.

#

#######################################################################

#######################################################################

#

# Properties supported by this program:

#

# Property Name Description and Default

#

### GLOBAL OPTIONS

# DEBUG : Debug level. Values 1-255 supported.

# : Default = 0.

#

# DEBUGLOG : Debug level for logs. Values 1-255 supported.

# : NOTE: Bits must also be set in DEBUG

# : Default = 0.

#

# DEBUGPRINT : Debug level for standard out. Values 1-255 supported.

# : NOTE: Bits must also be set in DEBUG

# : Default = 0.

#

# NullValue : Value assigned when a '' is detected.

# : Default = NA

#

# PROPSFILE : Represents the location of this file.

# : Default = "/opt/NMSDNS/NMSADMIN/nmsdnsfeed.props"

#

# LOGFILE : Represents the location of log messages.

# : Default = "/tmp/NMSDNSFEED.log";

#

# MULTIHOMED : Pointer to a hash of tags that identify if

# : a device is multihomed or not.

# : Default

# $MULTIHOMED->{"isIPRouter"} = 1;

# $MULTIHOMED->{"isSwitch"} = 1;

# $MULTIHOMED->{"isConnector"} = 1;

#

# SNMPTIMEOUT : Timeout for snmpwalk to get sysname.

# : Default = 2

#

# SNMPRETRIES : Retry count for snmpwalk to get sysname.

# : Default = 2

#

# SNMPCOMMUNITYSTRING : Retry count for snmpwalk to get sysname.

# : Default = "zeepnet22";

#

#######################################################################

#######################################################################

#

# Add your settings here

#

#######################################################################

# INITIAL SETTINGS OF CONFIGURABLES

my $VERSION = 25;

## DIRECTORIES

#$LOGDIR = "/tmp";

#$PRGDIR = "/tmp";

#$CFGDIR = "/tmp";

#$OLDDIR = "/tmp";

#$DNSDIR = "/tmp";

#$HSTDIR = "/tmp";

$LOGDIR = "/opt/NMSDNS/logs";

$PRGDIR = "/opt/NMSDNS/scripts";

$CFGDIR = "/opt/NMSDNS/config";

$OLDDIR = "/opt/NMSDNS/old";

$DNSDIR = "/opt/NMSDNS/db";

$HSTDIR = "/opt/NMSDNS/logs";

$DEFAULTNAME = "nmsdns";

## FILES: PROGRAM

#$DAEMONFILE = "$PRGDIR/$DEFAULTNAME" . "d.pl";

#$FEEDFILE = "$PRGDIR/$DEFAULTNAME" . "feed.pl";

#$CLEANFILE = "$PRGDIR/$DEFAULTNAME" . "clean.pl";

#$CREATEFILE = "$PRGDIR/$DEFAULTNAME" . "create.pl";

## FILES: CONFIGURATION

$PROPSFILE = "$CFGDIR/$DEFAULTNAME" . ".props";

$DBCACHEFILE = "$CFGDIR/db.cache";

$DBHEADERFILE = "$CFGDIR/db.header";

## FILES: LOG

$LOGFILE = "$LOGDIR/$DEFAULTNAME.log"; # LOG FILE

$WARNFILE = "$LOGDIR/$DEFAULTNAME.warn"; # WARN MESSAGES

$WARNFILE2 = "$LOGDIR/$DEFAULTNAME.warn2"; # WARN MESSAGES

## FILES: HOST DATABASES

$HOSTFILE = "$HSTDIR/$DEFAULTNAME.host"; # HOST FILE

$H2NFILE = "$HSTDIR/$DEFAULTNAME.h2n"; # H2N CMD FILE

$EXTRAFILE = "$HSTDIR/$DEFAULTNAME.extra";# MANUAL ENTRIES

## FILES: DNS FILES

$NAMEDFILE = "$DNSDIR/named.conf"; # HOST FILE

#$PTRFILE = "$DNSDIR/db.X.X.X.X"; # REVERSE LOOKUP FILES

#$ACNAMEFILE = "$DNSDIR/db.X.X.X.X"; # FORWARD LOOKUP FILES

## CONSTANT

$NullValue = "na";

$UnkValue = "unk";

use constant TRUE => 1;

use constant FALSE => 0;

## FILE DESCRIPTOR SWITCHES

$LOGFD = 1;

$HOSTFD = 2;

$H2NFD = 4;

$WARNFD = 8; # USED FOR BOTH WARN FILES

$NAMEDFD = 16;

## IDENTIFYING INFO

$DEFAULTDOMAIN='company.com';

## DNS SERVER INFO

$DNSSERVER = 'X.X.X.X';

## SPECIFIC FOR FEED

## SNMPWALK PARAMETERS

$SNMPTIMEOUT = 3;

$SNMPRETRIES = 1;

$SNMPCOMMUNITYSTRING = "public";

## IF A NODE IS MULTIHOMED. IF THE SWITCH IS 1 AND THE OPENVIEW FIELD VALUE

## IS TRUE, THEN THE NODE IS MULTIHOMED

$MULTIHOMED->{"isIPRouter"} = 1;

$MULTIHOMED->{"isSwitch"} = 1;

## LOCATIONS

$LOCATION->{"rt"} = 1;

$LOCATION->{"fw"} = 1;

$LOCATION->{"st"} = 1;

$LOCATION->{"ms"} = 1;

## INTERFACE HIERARCHY (BASED ON ABBREVIATIONS)

$IF->{"lp"} = 1;

$IF->{"sc"} = 2;

$IF->{"nt"} = 3;

$IF->{"vl"} = 4;

$IF->{"e"} = 5;

$IF->{"s"} = 6;

$IF->{"pix"} = 7;

$IF->{"unk"} = 8;

## INTERFACE CLEANING

$INTABREV->{"agent's"} = "unk"; # 1

$INTABREV->{"amd"} = "hst"; # 2

$INTABREV->{"atm"} = "atm"; # 3

$INTABREV->{"bvi"} = "bvi"; # 4

$INTABREV->{"channel"} = "ch"; # 5

$INTABREV->{"cabletron"} = "ctron"; # 6

$INTABREV->{"cayman"} = "hst"; # 7

$INTABREV->{"cisco"} = "csco"; # 8

$INTABREV->{"compaq"} = "hst"; # 9

$INTABREV->{"cpu"} = "hst"; # 9

$INTABREV->{"ctron"} = "ctron"; # 10

$INTABREV->{"dc"} = "dc"; # 11

$INTABREV->{"dec"} = "hst"; # 12

$INTABREV->{"dialer"} = "dial"; # 13

$INTABREV->{"e"} = "e"; # 14

$INTABREV->{"el574nd4"} = "e"; # 14

$INTABREV->{"el3c589"} = "e"; # 15

$INTABREV->{"enet"} = "enet"; # 16

$INTABREV->{"en"} = "en"; # 17

$INTABREV->{"eri"} = "eri"; # 18

$INTABREV->{"eth"} = "e"; # 19

$INTABREV->{"ethernet"} = "e"; # 20

$INTABREV->{"ether"} = "e"; # 21

$INTABREV->{"fastethernet"} = "fe"; # 22

$INTABREV->{"fddi"} = "fddi"; # 23

$INTABREV->{"fei"} = "fei"; # 24

$INTABREV->{"ge"} = "ge"; # 25

$INTABREV->{"gigabitethernet"} = "ge"; # 26

$INTABREV->{"hme"} = "hme"; # 27

$INTABREV->{"hssi"} = "hssi"; # 28

$INTABREV->{"interface"} = "hst"; # 28

$INTABREV->{"ibm"} = "hst"; # 29

$INTABREV->{"intel(r)"} = "hst"; # 30

$INTABREV->{"intel"} = "hst"; # 31

$INTABREV->{"ip"} = "hst"; # 32

$INTABREV->{"compaq"} = "hst"; # 33

$INTABREV->{"loopback"} = "lp"; # 34

$INTABREV->{"madge\_tr\_miniport\_driver"} = "tr"; # 35

$INTABREV->{"madge\_tr\_miniport\_driver-"} = "tr"; # 36

$INTABREV->{"network"} = "nt"; # 37

$INTABREV->{"qfe"} = "qfe"; # 38

$INTABREV->{"pix"} = "pix"; # 39

$INTABREV->{"sc"} = "sc"; # 40

$INTABREV->{"serial"} = "s"; # 41

$INTABREV->{"lan"} = "lan"; # 42

$INTABREV->{"smb"} = "hst"; # 43

$INTABREV->{"smc"} = "hst"; # 44

$INTABREV->{"tokenring"} = "tr"; # 45

$INTABREV->{"token\_ring"} = "tr"; # 46

$INTABREV->{"token-ring"} = "tr"; # 47

$INTABREV->{"token"} = "tr"; # 48

$INTABREV->{"tr"} = "tr"; # 49

$INTABREV->{"tunnel"} = "tu"; # 50

$INTABREV->{"tms380c26"} = "tms"; # 51

$INTABREV->{"unk"} = "unk"; # 55

$INTABREV->{"virtual"} = "vt"; # 52

$INTABREV->{"visual"} = "hst"; # 53

$INTABREV->{"ip"} = "ip"; # 54

$INTABREV->{"vlan"} = "vl"; # 55

$INTABREV->{"vln"} = "vl"; # 55

##########################################################################

############################## DEBUG PARAMETERS ########################

## DEBUG RULES:

## DEBUG manages all the output from this program.

## 1. DEBUGMAX represents the number if all bits were turned on and provides

## an easy way to turn up all bits for [OUTPUT STATEMENT], DEBUG, and

## [MASTER OUTPUT FILE SWITCHES].

## 2. [OUTPUT STATEMENT] switch statements categorizes the type of output.

## Of these the leading switches are used to specify output assoicated

## with the classification of device.

## 3. DEBUG is a master switch which can globally turn off output of

## some of the OUTPUT STATEMENTS by setting their bit to 0.

## 4. [MASTER OUTPUT FILE SWITCHES] switch statements are used to funnel

## [OUTPUT STATEMENT] types to the correct output.

##########################################################################

## OUTPUT STATEMENT SWITCHES

## MAXIMUM DEBUG LEVEL

$DEBUGMAX=(2\*\*14)-1;

## TYPE OF DEVICES THAT CAN BE SAVED IN HOST FILE AND H2N FILE

$D\_GOOD = 1; ## 1 Output good interfaces and hosts.

$D\_HOST = 1; ## 1 Host file output

$D\_NAMED = 1; ## 1 Host file output

$D\_ADMDOWN = 2; ## 2 Output admin down interfaces and hosts

$D\_INVALID = 4; ## 3 Output errored interfaces and hosts

## BASIC PROCESS FLOW DEBUGING

$D\_INT = 8; ## 4 Basic process tracing

$D\_MAIN = 16; ## 5 More process tracingmands

## SPECIAL DEBUGGING

$D\_LOGFILE = 32; ## 6 In-process interface listing

$D\_HOSTNAME = 64; ## 7 Detailed in-process interface listing

$D\_MH = 128; ## 8 Multihome detection.

## DETAILED PROCESS FLOW DEBUGING

$D\_1STPASS = 256; ## 9 Trace 1st stage: Parse data for host info

$D\_PROCESS = 512; ## 10 Trace 2nd stage: Process host info

$D\_2NDPASS = 1024; ## 11 Trace 3rd stage: Parse data for int info

## ERROR DETECTION OUTPUT

$D\_WARN = 2048; ## 12 WARNING messages - raw.

$D\_WARN2 = 4096; ## 13 WARNING messages - summarized.

## DATA OUTPUT

$D\_H2N = 8192; ## 14 Node level FIRST PASS tracing.

## MASTER OUTPUT FILE SWITCHES

$DEBUG = $DEBUGMAX; # MASTER SWITCH. OUTPUT ONLY OCCURS IF

# THESE BIT ARE ON. DEFAULT: ALL ON

$DEBUGLOG = $D\_INT | $D\_MAIN | $D\_LOGFILE | $D\_WARN | $D\_H2N | $D\_1STPASS | $D\_HOSTNAME;

#$DEBUGLOG = $DEBUGMAX; # LOG FILTER. LOG STATEMENTS TO $LOGFILE

#$DEBUGLOG = 0; # LOG FILTER. SAVE NOTHING

$DEBUGHOST = $D\_HOST; # CLEANED INVENTORY

$DEBUGH2N = $D\_H2N; # RAW INVENTORY FROM OPENVIEW

$DEBUGWARN = $D\_WARN; # WARN MESSAGES TO $WARNFILE

$DEBUGWARN2 = $D\_WARN2; # WARN MESSAGES TO $WARNFILE

$DEBUGNAMED = $D\_NAMED; # HOST FILE

##########################################################################

## REQUIRED FOR PACKAGES

1;

# IMPORTANT NOTICE

ALL INFORMATION PROVIDED IN THIS PAPER IS PROVIDED "AS IS" WITH ALL FAULTS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPIED. DANIEL NEEDLES, INC. DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

DANIEL NEEDLES, INC SHALL NOT BE LIABLE FOR ANY INDIRECT, SPEICAL, CONSEQUENTIAL, EXEMLARY, PUNITIVE OR INCIDENTAL DAMAGES INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR REVENUES, COSTS OF REPLACEMENT GOODS, LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE ANY PRODUCT MENTIONED IN THIS PAPER, DAMAGES RESULTING FROM USE OF OR RELIANCE ON THE INFORMATION PRESENT, EVEN IF DANIEL NEEDLES, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DANIEL NEEDLES, INC IS NOT LIABLE FOR THE ACCURACY OR UTILITY OF THE INFORMATION CONTAINED IN THIS WHITE PAPER. DANIEL NEEDLES INC'S DISCUSSION OF ANOTHER COMPANY'S PRODUCTS AND/OR SERVICES DOES NOT CONSTITUTE EITHER AN ENORSEMENT OR A RECOMMENDATION. THE CONTENTS OF THIS PAPER ARE FOR INFORMATION PURPOSES ONLY.

# About NMS Guru

NMS Guru architects and manages comprehensive enterprise management solutions through principal consultants with decades of experience and deep roots into the industry. Specialties include: monitoring, performance, configuration, provisioning, change, and security solutions for networks, systems, applications, and business processes. These practices are integrated holistically by weaving together the strategic initiatives from above (OSS/BSS, BPM, ITIL, FCAPS, TMN, etc) with the tactical realities from below (tools, people, knowledge and processes.) The result is increased operational awareness and extended useful lifespan of the enterprise management solution.

NMS Guru is headquartered in Austin, TX. (Along with NMS tools: IBM Tivoli, CA NetQOS, SolarWinds, and many others.) For more information, visit the website at <http://nmsguru.com> or call 1.512.617.6694.

**Author**

Dan Needles is the founder of NMS Guru. Over the past 20 years as an Enterprise IT Operations Architect he has designed and implemented fault, performance and configuration management solutions and products for over 50 fortune 500 companies and government entities. He can be contacted via: guru@nmsguru.com or call 1.512.627.6694.

John Hachey is a Senior Engineer with VISA Corp. Mr. Hachey has over 18 years of networking and network management experience, and maintains product certifications with most NMS product vendors, including Cisco, HP’s OpenView, and Open’s NerveCenter. He can be reached at: [JHachey@VISA.com](mailto:John.Hachey@VISA.com). or [Hachey@citlink.net](mailto:Hachey@citlink.net).