Best Practices Guide for NerveCenter Users and Developers

Windows and UNIX

Version 5.x – Version 6.x

July 2011

NCBPUD5200-04

Copyright

Disclaimers

LogMatrix, Inc. ("LogMatrix") makes no representations or warranties, either expressed or implied, by or with respect to anything in this manual, and shall not be liable for any implied warranties of merchantability or fitness for a particular purpose or for any indirect, special or consequential damages.

These applications are available through separate, individual licenses. Not every feature or application described herein is licensed to every customer. Please contact LogMatrix if you have licensing questions.

No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of LogMatrix. While every precaution has been taken in the preparation of this book, LogMatrix assumes no responsibility for errors or omissions. This publication and the features described herein are subject to change without notice.

The program and information contained herein are licensed only pursuant to a license agreement that contains use, reverse engineering, disclosure and other restrictions.

Trademarks

LogMatrix is registered in the U.S. Patent and Trademark Office. NerveCenter and the LogMatrix Logo are trademarks of LogMatrix, Inc.

All other products or services mentioned in this manual may be covered by the trademarks, service marks, or product names as designated by the companies who market those products.

Contacting LogMatrix

LogMatrix, Inc.
4 Mount Royal Ave, Suite 250
Marlborough, MA 01752

Phone 508-597-5300
Fax 774-348-4953

info@logmatrix.com

Website: www.logmatrix.com
Forum: http://community.logmatrix.com/LogMatrix
Blog: www.logmatrix.com/blog

Best Practices Guide for NerveCenter Developers and Users

The purpose of this document is to describe some of the processes and procedures that will help you get the most from your NerveCenter installation. Here we will discuss the best practices and guidelines for developers who design and deploy NerveCenter Models and for Users who employ NerveCenter to manage devices and applications.

This NerveCenter document discusses the uses of NerveCenter from a developer's point of view. There is a separate Best Practices for NerveCenter for a System Administrator/Security Administrator who needs to manage the system on which NerveCenter runs.

This document is written assuming you are running NerveCenter release 5.1 or later. If you are running an earlier release of NerveCenter please contact Customer Support for more information on steps you can take to either upgrade or better manage your current environment.

What is NerveCenter?

NerveCenter is a Framework that can be used to build and customize the network management and application management that meets your business needs. NerveCenter is built using a finite state machine architecture. This architecture allows you to build simple yet elegant models to manage any type of device or application.

NerveCenter was designed to work either standalone as your Network Management platform or in conjunction with other Network management products and devices to provide a custom solution that can be individualized to your site's needs.

At the heart of NerveCenter is its event correlation engine. For each device that it is monitoring, NerveCenter creates one or more finite state machines -or alarms- that define operational states it wants to detect. NerveCenter also defines rules that effect transitions between the operational states. These rules can be very simple; for example, a state transition can be caused by the receipt of a generic Simple Network Management Protocol (SNMP) trap. Or they can be quite complex and take advantage of NerveCenter's support for Perl expressions.

When deploying NerveCenter there are many areas to consider, just as with many large management products. We have tried to categorize the information into major activity groupings.

Model Writing

The most important activity you do to ensure your NerveCenter deployment will be a success is writing models. This document is not intended to teach you to write models, but instead to provide a set of guidelines on how to organize your group's approach to model creation and editing.

For more information on writing Models, refer to the Behavior Models Cookbook available in your online NerveCenter documentation package or consider accessing NerveCenter Training available through the NerveCenter Training Videos.

When writing models, consider these steps:

- Understand the issues and complexities of the device or application you intend to build the model for
- Obtain any additional MIBs necessary to support SNMP polling in your model
- For application models, obtain additional software that is required to communicate with the application
- Research available Perl Modules that can help you complete the model. CPAN is an excellent source for obtaining information about the available Perl modules.
- Backup all the models you have written. Machines lose drives and sometimes someone imports an older Model on top of your work, so make sure you save your work. You can either backup the nervecenter.ncdb file or use the Export Objects and Nodes Feature in the NerveCenter client to save the components of the Model you are working on.
- For complicated Perl code, consider writing the Perl outside of NerveCenter and then putting it in a subroutine or poll. It is much easier to debug Perl outside the server. Using NotePad++ or a Perl editor or IDE can help.

Why you should separate Development from Production systems

It is strongly recommended that all development of NerveCenter Models be done on a system separate from your production environment. This allows for a clean place to test new ideas, test for logic problems and ensure the Model works as desired. Ideally the development environment should be the same platform (Solaris, Linux or Windows) that you use for your production environment.

One of the advantages of the development environment is you can modify the NerveCenter server's configuration and try out different scenarios. You can also restart the server without impacting production.

It is too easy to accidently introduce a code error in Perl. Testing all your Models on a development platform gives you the best chance to discover any of these issues prior to deployment.

Perl Style guide

A major component of writing models is writing your Perl code. To effectively write and manage the Perl used in NerveCenter, a Perl style guide is a must. The guide you choose to write can be simple or quite detailed, but it should contain a few common features.

- Naming conventions – all variables, filenames (logs or scripts), and FireTriggers need to follow naming conventions. These conventions allow another developer to easily understand the purpose of the code. The naming conventions you define may be the most important piece of your Style Guide.
- Use Comments – Brief comments should be included in the Perl code to ensure everyone understands what the code does. Longer descriptions can be placed in the Notes feature provided in NerveCenter.
- Make good use of Notes – The notes feature is an excellent place to put lengthy descriptions of the Design and functionality of Perl code. It is also a good place to store revision history for the model component.
- Get buy-in – A style guide is only useful if everyone uses it. Try to get input and buy-in from as many people in your development group as possible. It is important that the style guide match your organization's culture. Style guides that are too cumbersome or inflexible will be ignored. The purpose of the style guide is to make you more successful not to generate unnecessary work.
- Don't use **die** function – Do not use the **die** function in Perl. This function has the effect of terminating your process which in this case is the actual NerveCenter Server. Refer to the Perl Documentation at http://perldoc.perl.org/functions/die.html for more information on the **die** function call.
- Don't use other potentially fatal functions such as **exit()** and **croak()** – any of these functions will/may cause the NerveCenter server to exit. The **croak()** feature can be found in the Carp module. Please be sure to understand the Perl Modules you include in your internal NerveCenter Perl code. All Perl modules you choose to use should be documented, and any retrieved from CPAN will have full documentation and often comment threads on usage in the Perl forums.
- Trigger Names - Don't re-use trigger names between polls, trap masks or Perl subroutines. It will make it virtually impossible to determine who triggered the event. Have trigger names suggest their source - namely who fired them. You could put a 'T'

prefix for Trap masks, a 'P' for Poll functions or a 'PL' for Perl subroutines. It doesn't matter what you choose as long as you are consistent.

- Use Perl Modules – When writing code that is used frequently, take advantage of Perl Modules if possible. This does require the ability to compile the Perl code if the Module is not present, but that can be done onsite or you can request assistance from NerveCenter support engineering. Writing your own Perl Modules is also an excellent idea. This will lead to more consistent and efficient models as you proceed with your NerveCenter deployment.

All Perl written in models is executed within the NerveCenter Server. This means that attention must be paid to the type of actions you perform in your Perl code. In addition to not using the **die** function, **do not make Perl function calls to routines that may delay the processing of the Server.** For example, do not make a call to write to a database file on a remote machine that could delay for minutes at a time. This will cause the server to wait for a response and could backup polls in the process.

Other examples of processes that could delay server activity:

- Waiting for response from remote applications
- DNS IP Address or hostname resolution
- SQL processing, especially large amounts of data

To perform tasks that require waiting on other processes, consider using the Command Action in the Alarm.

Resources for Learning Perl

There are a number of good resources for learning about Perl. There is a wealth of information on the Internet for learning Perl. You can access perlmonks.org, learn.perl.org or CPAN to learn about Perl.

A number of books are also available. Some of the more popular ones are:

- The Camel Book (**PROGRAMMING PERL**) by Wall, Christiansen, & Orwant
- **LEARNING PERL** by Randal Schwartz
- **THE PERL COOKBOOK** by Tom Christiansen
- Perl Best Practices by Damian Conway

Perl Debugging inside the Server

Currently there is no debugger for Perl inside the Server. If you are writing complicated Perl and use Perl often in your environment, it may be worth using a Perl editor or purchasing an IDE for Perl. You can use these tools to verify the Perl before adding it to your NerveCenter Models.

To troubleshoot Perl inside NerveCenter, there are a few steps you can take to help with the Process.

- Take advantage of the Log to File capability with Alarms. This feature can be used to verify that FireTriggers are happening as expected. Setup a dummy alarm that receives the extra Triggers (with the correct property group)
- Write temporary results to a log file within the Perl. This is very helpful when designing a new model that is not behaving as expected. For Performance reasons, extra logging should be removed for Production environments, but while in development, it really helps to see what is going on. Using the following simple structure works well:

```
my $debug_on = 1;
if ( $debug_on != 0 ) {
  # logging will be done now
   my $PLOAD;
   open PLOAD, ">>/opt/OSInc/userfiles/logs/pload\.csv";
   print PLOAD “$var1, $var2,  $var3 “;
   print PLOAD "\nLogged all my data “;
   close PLOAD;
}
```

FireTriggers

Care should be taken with FireTrigger calls also. Having multiple poll routines call the same FireTrigger function will lead to confusion. Having extra FireTriggers that do not have corresponding Alarms is not an issue. They will simply be ignored until there is an action defined which is associated with them. So don’t be afraid to use extra FireTrigger calls.

When designing your model in a test area, FireTriggers are also a useful tool for analyzing behavior. An example of this technique is:

```
If ( delta( snmp.snmpInBadCommunityNames ) >= 1 or
     delta( snmp.snmpInBadCommunityUses ) >= 1 )
{
      FireTrigger("authFail");
} else {
```

```
        # Fire a trigger for now While debugging.  Once happy with poll, take this out.
        FireTrigger("authGOOD");
}
```

MIB Compilation

MIB compilation is the process of building all your vendor and private MIB files along with the standard MIB data provided in the NerveCenter kit, to produce one file which the NerveCenter server loads and uses for all SNMP processing. If you wish to poll devices using NerveCenter, it is critical to build your MIB file correctly.

When compiling your MIB for NerveCenter, the make sure you **always** backup all the components you used to generate the MIB. This is a very important step. Once you are happy with your MIB file, you do not want to waste time rebuilding and reloading it on your system.

For those users running newer installations of NerveCenter (NC 5.1.01) and later, use **mibTool** to compile you MIB files and not the older **mibcomp**. **mibTool** is the recommended application for compiling MIBs.

When compiling your MIBs, consider the following:

- Use the newer 5.1.xx version of the mibcomp.txt file (this can be found in /opt/OSInc/mib on your UNIX platform)
- mibTool is a Java application and requires a minimum Java version of 1.5.0. Make sure Java is in your **$path** setting. Use the command **java –version** to see the version of Java installed on your system.
- You can use the old 'mibcomp.txt' files for input, but you may see errors or warnings.
- Redirect output of MIB compiler to a log file for review. On UNIX you may also use **script** to generate a script log of the actions and output.
- Once you have run the MIB compiler, review the logs for errors and completion information.

- If you are running a version of NerveCenter prior to NC5.1, then mibTool needs to create the NerveCenter MIB such that it can be read by the older release. To accomplish this, run mibTool with the "-namemap" switch. For example:
  # mibTool -namemap -mibcomp mibcomp.txt
  This will create an index file, mibnamemap.txt, in addition to the NerveCenter MIB file, nervectr.mib. The older MIB format requires both files and this index file must be kept with the output NerveCenter MIB file.

Using mibTool

Usage: **mibTool** <options>

-o <filename> Send the compiled MIB information to the specified NerveCenter MIB filename, default is "nervectr.mib"
-loc <path> Declare a repository location. All MIBs under this location will be checked if entity lookups fail.
-inc <mib name> Includes the specified MIB in the output file nervectr.mib
-mibcomp <filename> Compiler will Read an old style mibcomp file for processing.
-namemap Generates output compatible with older versions of NerveCenter, also generates a mibnamemap.txt
-strict Tells compiler to compile the MIB files in strict mode. When in strict mode, all imports must all be completely accurate or compilation will fail.
-e <filename> Redirects error output to the specified file.

Example: Showing the use of mibTool:
mibTool –mibcomp mibcomp.txt

Example: Showing the use of error output files:
mibTool –mibcomp mibcomp.txt –strict >mibcomp_out.log 2> mibcomp_error.log

Reloading MIBs on your NerveCenter server

- Stop your NerveCenter server using ncstop
- Save a copy of the original MIB file
- Copy new MIB into the MIB directory
- Copy new mibnamemap to the MIB directory
- Restart NerveCenter
- Verify your new MIB file is loaded and working correctly. Check your property groups, try a few polls, and make sure Nodes are in the correct property group. You should verify a Poll or Alarm for every property group you use to make sure all of the MIB variables you need are present.

Using Statistics to Analyze Performance

Prior to NerveCenter 5.2 release, there are limited statistics available for monitoring the NerveCenter server. Those statistics that are present can give you a great deal of insight into the running of the server, especially after upgrades with new Models or newer managed nodes.

Take advantage of the statistics and TraceCounters.log and TraceQueues.log to verify the performance of your server. Customer Support can supply you with Perl scripts to assist in analyzing your NerveCenter 5.1 Trace log files.

For 5.2 and later releases, the statistics data can be obtained from the NerveCenter client application and NerveCenter **ncadmin** application. This feature is a first place to look when analyzing server behavior.

It is very useful to monitor these files on your development platform and understand how your NerveCenter server performs in your environment. When upgrading to a new NerveCenter version or when deploying new Models, run the statistics again and make sure you have not caused degradation in performance.

The NerveCenter 5.2 Users Guide provides a detailed description of logging and statistics files and how to manage and use them.