NDOUTILS Documentation

Last Updated: December 28, 2005

CONTENTS

1. INTRODUCTION

a) Purpose

The NDOUTILS addon is designed to store all configuration and event data from Nagios in a database. Storing information from Nagios in a database will allow for quicker retrieval and processing of that data and will help serve as a foundation for the development of a new PHP-based web interface in Nagios 3.0.

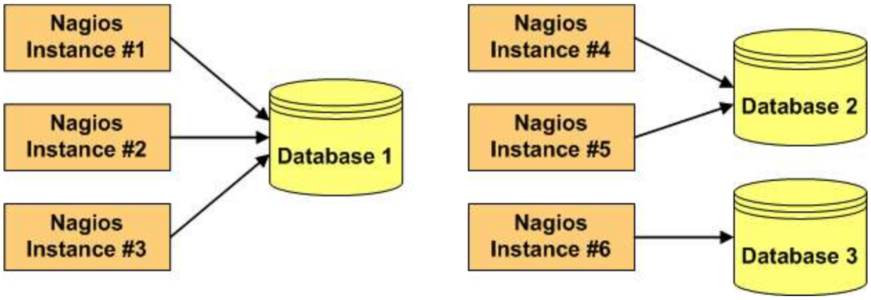
MySQL databases are currently supported by the addon and PostgreSQL support is in development. Support for other database servers may be added if there is sufficient user interest.

b) Design Overview

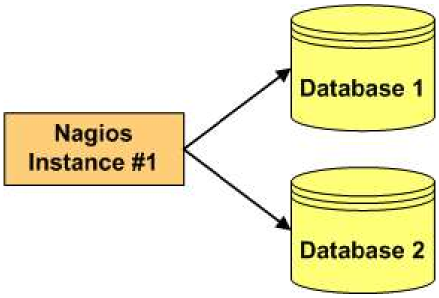
The NDOUTILS addon was designed to work for users who have:

- Single Nagios installations
- Multiple standalone or "vanilla" Nagios installations
- Multiple Nagios installations in distributed, redundant, and/or failover environments

Data from each Nagios process (hereafter referred to as "instances") can be stored either in the same database or different databases than data from other Nagios instances.



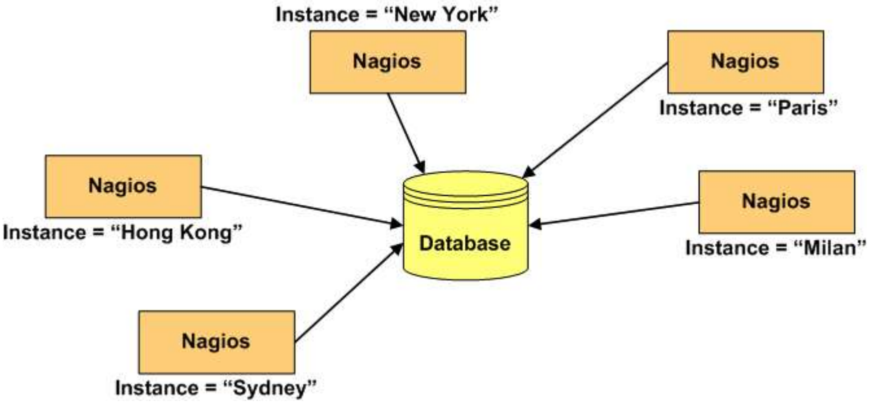
Although not yet supported, future development should allow for data from any given Nagios instance to be stored in multiple databases if desired.



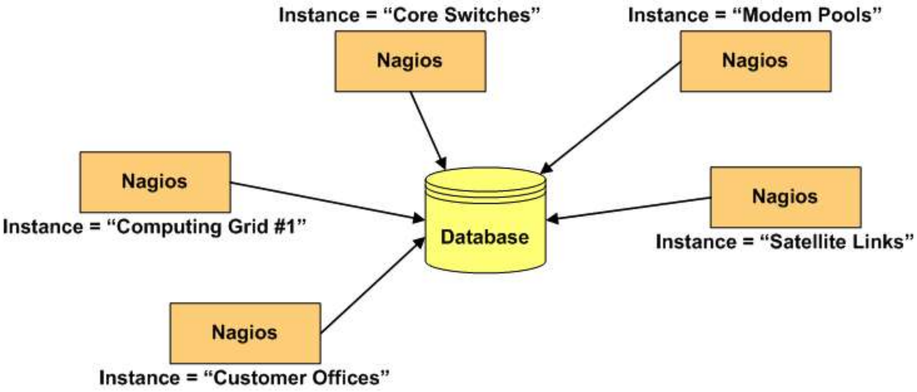
c) Instances

Each Nagios process, whether it is a standalong monitoring server, or part of a distributed, redundant, or failover monitoring setup, is referred to as an "instance". In order to maintain the integrity of stored data, each Nagios instance must be labeled with a unique identifier or name.

You can choose the name of each Nagios instance to suit your needs. For instance, you could choose to name Nagios instances based on their geographical location...



Or you could name Nagios instances based on their purpose...


Instance = "Core Switches"
Instance = "Modem Pools"
Nagios
Nagios
Nagios
Instance = "Computing Grid #1"
Database
Nagios
Instance = "Satellite Links"
Nagios
Instance = "Customer Offices"

How you name Nagios instances is up to you. The key point to remember is that each and every Nagios process must have its own unique instance name.

More information on how instance names come into play will be discussed in sections 3 and 4.

2. COMPONENTS

a) Overview

There are four main components that make up the NDO utilities:

1. NDOMOD Event Broker Module
2. LOG2NDO Utility
3. FILE2SOCK Utility
4. NDO2DB Daemon

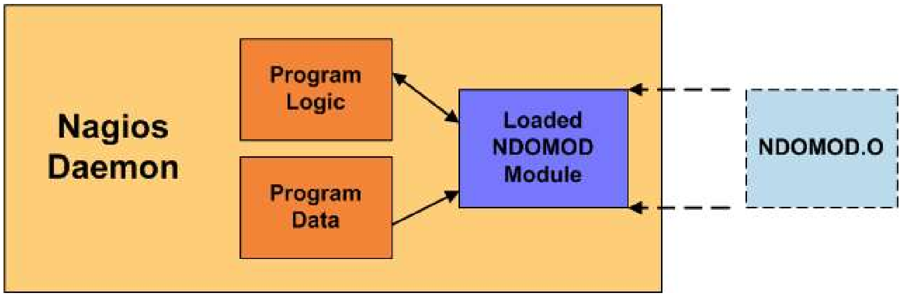
Each component is described in more detail on the following pages.

The rest of this page has been intentionally left blank.

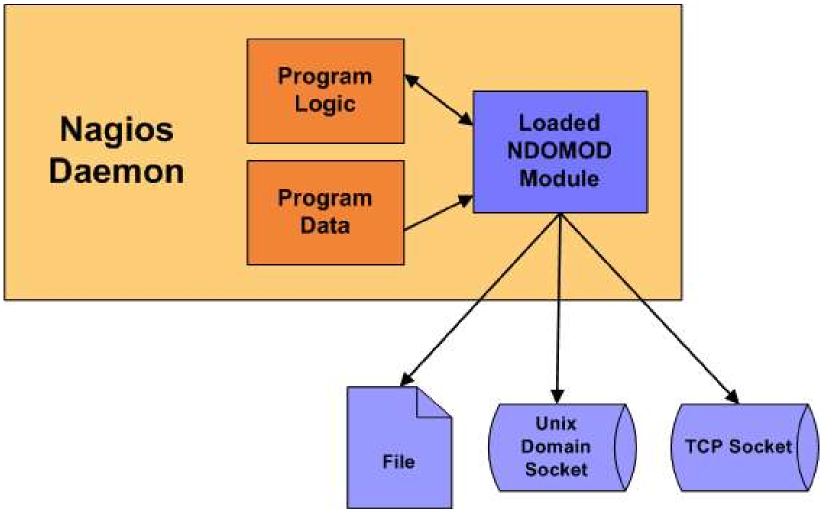
b) The NDOMOD Event Broker Module

The NDO utilities includes a Nagios event broker module (NDOMOD.O) that exports data from the Nagios daemon.

Assuming that Nagios has been compiled with the event broker enabled (this is the default), you can configure Nagios to load the NDOMOD module during runtime. Once the module has been loaded by the Nagios daemon, it can access all of the data and logic present in the running Nagios process.



The NDOMOD module has been designed to export configuration data, as well as information about various runtime events that occur in the monitoring process, from the Nagios daemon. The module can send this data to a standard file, a Unix domain socket, or a TCP socket.



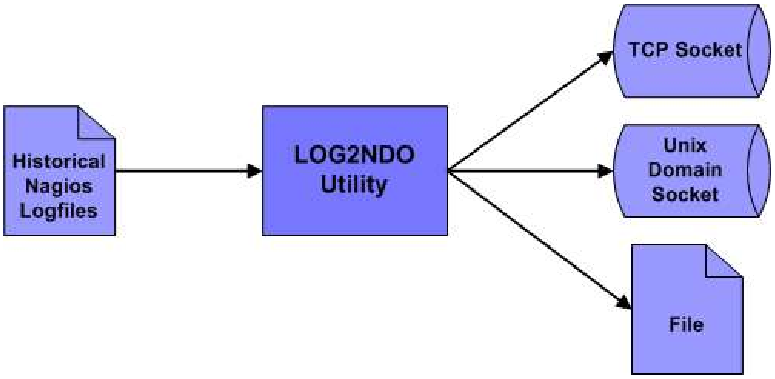
The NDOMOD module writes data in a format that the NDO2DB daemon (described later) can understand.

If the NDOMOD module is writing its output to a file, you can configure it to periodically rotate and/or process the output file using a predefined Nagios command. This can be useful if you want to transfer the output file to another physical machine (using SSH, etc.) and send its contents to the NDO2DB daemon using the FILE2SOCK utility (described later).

If the NDOMOD module is writing it output to a TCP or Unix domain socket, it has some resistance to connection dropouts. The module will attempt to cache its output until it can (re)connect to the socket for writing. This is helpful if the process that creates and listens on the socket needs to be restarted, etc.

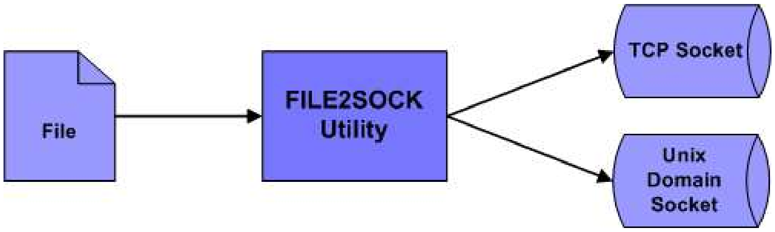
c) The LOG2NDO Utility

The LOG2NDO utility has been designed to allow you to import historical Nagios and NetSaint log files into a database via the NDO2DB daemon (described later). The utility works by sending historical log file data to a standard file, a Unix domain socket, or a TCP socket in a format the NDO2DB daemon understands. The NDO2DB daemon can then be used to process that output and store the historical logfile information in a database.



d) The FILE2SOCK Utility

The FILE2SOCK utility is quite simple. Its reads input from a standard file (or STDIN) and writes all of that data to either a Unix domain socket or TCP socket. The data that is read is not processed in any way before it is sent to the socket.

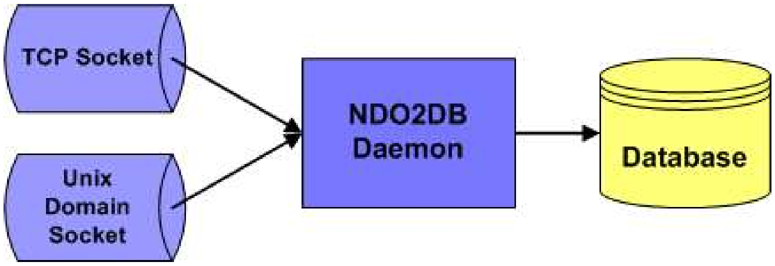


This utility is useful if you are directing the output of the NDOMOD event broker module and/or LOG2NDO utility to a standard file. Once these components finish writing their output to a file, you can use the FILE2SOCK utility to send the contents of the file to the NDO2DB daemon's TCP or Unix domain socket.

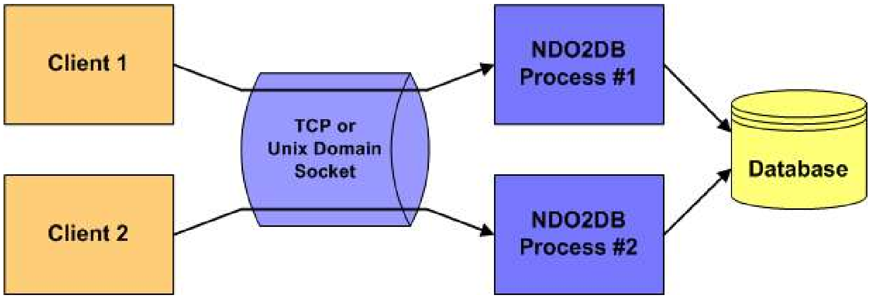
e) The NDO2DB Daemon

The NDO2DB utility is designed to take the data output from the NDOMOD and LOG2NDO components and store it in a MySQL or PostgreSQL database.

When it starts, the NDO2DB daemon creates either a TCP or Unix domain socket and waits for clients to connect. NDO2DB can run either as a standalone, multi-process daemon or under INETD (if using a TCP socket).



Multiple clients can connect to the NDO2DB daemon's socket and transmit data simultaneously. A seperate NDO2DB process is spawned to handle each new client that connects. Data is read from each client and stored in a user-specified database for later retrieval and processing.



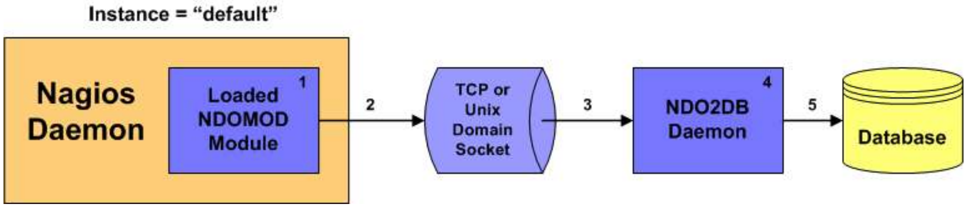
The NDO2DB daemon currently supports only MySQL databases, but future support for PostgreSQL databases is planned.

3. EXAMPLE CONFIGURATIONS

a) Single Server, Single Instance Setup

The easiest configuration occurs when you have a single Nagios instance running on your network. In this case, installing and configuring the various components of the NODUTILS addon is fairly straighforward.

The following diagram illustrates how the various components can fit together in a single server, single Nagios instance setup....



Here's a description of what's happening at each point in the diagram:

1. The NDOMOD module is configured with an instance name of "default" since there is only one instance of Nagios that is running on the network.
2. While the Nagios daemon is running and performing its usual business of monitoring the network, the NDOMOD module is sending configuration data and event information to the TCP or Unix domain socket that was created by the NDO2DB daemon.
3. The NDO2DB daemon reads data that is coming into the socket from the NDOMOD module.
4. The NDO2DB daemon processes and transforms data that has been received from the NDOMOD module.
5. The processed data is stored in a database for later retrieval and processing.

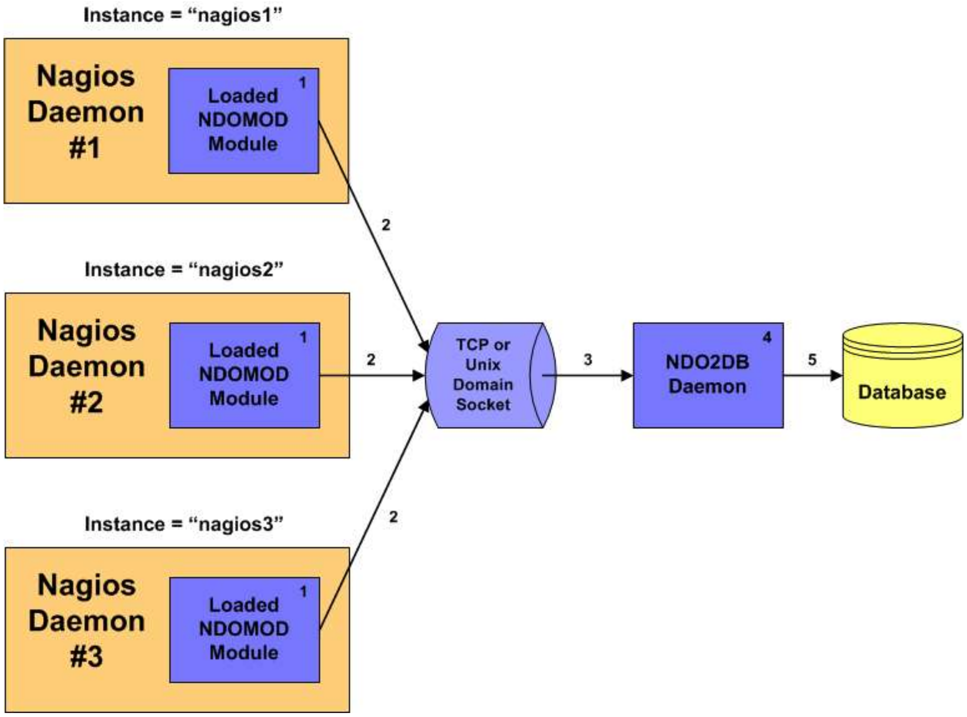
This example assumes that:

1. Nagios is configured to load the NDOMOD module at startup.
2. The NDO2DB daemon (which is a seperate process from the Nagios daemon) is running.

b) Single Server, Multiple Instance Setup

Another simple configuration can be used when you have a multiple Nagios instances running on a single server. Installing and configuring the various components of the NODUTILS addon is similiar as to what was shown in the previous example.

The following diagram illustrates how the various components can fit together in a single server, multiple Nagios instance setup....



You'll notice that the diagram above is similiar to the one for the single-server, single instance configuration. The main difference is that there are now three (3) different Nagios daemons instead of just one.

1. Each Nagios daemon loads the NODMOD module at startup with a unique instance name. In this example the instances are simply named "nagios1", "nagios2" and "nagios3".

2. Each NDOMOD module sends configuration data and event information for its specific instance of the Nagios daemon to the TCP or Unix domain socket that was created by the NDO2DB daemon.

6. The NDO2DB daemon reads data that is coming into the socket from the three NDOMOD modules.

7. The NDO2DB daemon processes and transforms data that has been received from the NDOMOD modules.

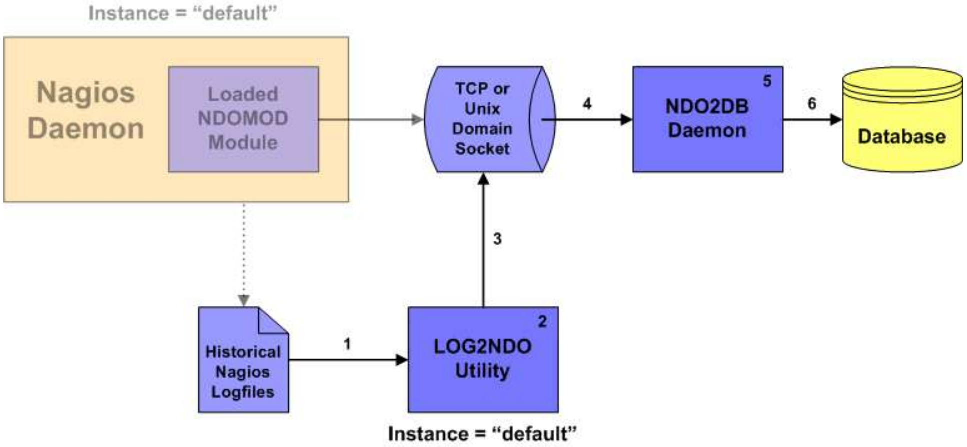
3. The processed data is stored in a database for later retrieval and processing. Data from each instance of Nagios is kept seperate (using the instance names as unique identifiers) in the database.

b) Single Server, Single Instance Log File Import

There are two reasons you'll probably want to import your Nagios log files into the same database that Nagios configuration and event data is stored in:

1. Historical log file data isn't imported into the database by default and having a record of events that occurred before you implemented the NDOUTILS addon is probably desireable.

2. The NDOMOD module is not able to process realtime log entries from the time right after the Nagios daemon starts to the time that the NDOMOD module is loaded by the Nagios daemon. This "blackout period" is unavoidable and causing log entries such as "Nagios 2.0 starting..." to be missed by the NDOMOD module. Thus, importing day-old Nagios log files on a daily basis (via a cron job) is recommended.



Here's a description of what's happening at each point in the diagram:

1. Historical Nagios log files are read by the LOG2NDO utility.

2. The LOG2NDO utility processes the contents of the log files and tags them with an instance name of "default". This instance name must match the same instance name used by the NDOMOD module in the Nagios daemon.

3. Historical log file data is sent to the TCP or Unix domain socket in a format that the NDO2DB daemon can understand.

4. The NDO2DB daemon reads the log file data from the Unix domain socket.

5. The NDO2DB daemon processes the log file data.

6. Historical log file data is stored in a database for later retrieval and processing. The NDO2DB daemon will perform some checks to make sure it doesn't re-import duplicate historical log entries, so running the LOG2NDO utility on the same historical log file multiple times shouldn't have any negative side effects.

That's it! Pretty simple.

4. INSTALLATION

**** TODO ****

5. NDO2DB DATABASE DATA DICTIONARY

** TODO **