Node Health Monitor

Component specification of the Node Health Monitor

GENIVI Document

|  |  |
| --- | --- |
| **Author:** | David Yates, Continental Automotive Gmbh |
|  | |
| **Revision:** | 1.3.4 |

**Sponsored by:**

**Abstract:**

**Keywords:**

**License:**

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Copyright © , Company ABC, Company XYZ.  
All rights reserved.

The information within this document is the property of the copyright holders and its use and disclosure are restricted. Elements of GENIVI Alliance specifications may be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights (and such third parties may or may not be members of GENIVI Alliance). GENIVI Alliance and the copyright holders are not responsible and shall not be held responsible in any manner for identifying, failing to identify, or for securing proper access to or use of, any or all such third party intellectual property rights.

GENIVI and the GENIVI Logo are trademarks of GENIVI Alliance in the U.S. and/or other countries. Other company, brand and product names referred to in this document may be trademarks that are claimed as the property of their respective owners.

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

The full license text is available at http://creativecommons.org/licenses/by-sa/4.0

The above notice and this paragraph must be included on all copies of this document that are made.

GENIVI Alliance

2400 Camino Ramon, Suite 375

San Ramon, CA 94583, USA

# Table of Contents

1 Table of Contents 3

2 History 4

3 Terms and Abbreviations 4

4 Introduction 5

4.1 Component overview 5

4.2 Responsibilities of the Node Health Monitor 7

5 External component view 8

5.1 System context 8

5.2 Interfaces 9

5.3 Dbus connection 10

6 Internal design 13

6.1 Component diagram 13

6.2 Project layout 14

6.3 Internal functions 15

6.4 Configuration values 15

6.5 Data storage format 16

6.6 Sequence diagrams 17

6.7 Project layout and compilation 18

6.8 Unit test and quality 19

7 Open Issues 19

8 References 19

9 Checklists 20

9.1 Node State Manager 20

9.2 Health Monitoring 21

9.3 Profile Manager 23

9.4 Persistency 24

# History

The following table lists the document’s history. Please add an entry every time the document is changed to be able to trace the modifications and extensions.

| **Revision** | **Date** | **Author, Editor** | **Reason** |
| --- | --- | --- | --- |
| **1.0** | 10/30/2013 | Jean-Pierre Bogler | Initial revision describing NodeHealthMonitor version 1.3.3 |
| 1.3.3 | 13/12/2013 | David Yates | Updated for release of NHM 1.3.3 within GENIVI |
| 1.3.4 | 22/01/2015 | David Yates | Updated with GENIVI Creative Commons Licence |

# Terms and Abbreviations

The following terms and abbreviations are used within this document.

| **Term** | **Definition** |
| --- | --- |
| SSW | System software (Continental department) |
| OIP | Open infotainment platform |
| NHM | Node Health Monitor |
| NSM | Node State Manager |
| DLT | Diagnostic Log and Trace |
| PCL | Persistence Client Library |
| IPC | Inter process communication |

# Introduction

The document gives an overview about the Node Health Monitor (NHM). The subchapters will explain the responsibilities of the NHM and outline its interfaces, architecture and design.

## Component overview

The NHM is a system software component that observes the system health and initiates configurable actions, if issues are identified. The component has been specified by the GENIVI consortium and was developed as part of the Continental “Open Infotainment Platform” (OIP).

The NHM is one component within the Domain Lifecycle and sits within the Sub-Domain of Resource Management.

Figure 1 - Lifecycle Domain View

Figure 2 – Component View

The component is licensed with the “Mozilla Public License” and relies heavily on the features provided within systemd.

## Responsibilities of the Node Health Monitor

The Node Health Monitor is started by systemd and will interact with application plug-ins and systemd to get notifications when a component has failed. He will be responsible for:

* Providing interfaces, which are used to:
  + register failures of applications
  + retrieve error statistics
  + request system restarts
* Observe applications started by systemd and detect if they fail
* Inform the system about failing applications
* Maintain a count of the number of failures in the current life cycle as well as statistics on number of failures in last X life cycles (i.e. 3 failures in last 32 life cycles)
* Observe the life cycle accordingly to catch unexpected system restarts

Additionally the NHM will test a number of product defined criteria with the aim to ensure that user land is stable and functional. It will be able to validate that:

* Defined files are present
* Defined processes are still running
* User defined processes can be executed with an expected result
* Inter Process Communication on defined dbus (bus address) is possible

If the NHM believes that there is an issue with user land then it will initiate a system restart.

# External component view

The NHM offers interfaces and communicates to other components of the system using the dbus for Inter Process Communication (IPC). This chapter describes the interfaces of the NHM and how it interacts with the rest of the system.

## System context

In Figure 3 the NHM and its actions in the system context are shown. In the system context, the following NHM relevant actions are done:

* The NHM is started by systemd
* Beside of the NHM, systemd starts all other user applications and observes them
* systemd is able to detect when an application it started fails
* The NHM is notified when systemd detects a failed application and will store information about the app.
* So-called “restart applications” may request a restart at the NHM
* The NHM informs the NodeStateManager (NSM) about failing apps and forwards restart requests (if the requesting application is configured within the NHM).

The further actions performed by the NHM, like storing the failed applications and performing user land checks (see chapter 4.1) do not affect the remaining system. A description of these tasks can be found in the internal design documentation in chapter .

|  |
| --- |
| Figure 4 – NHM in system context |
|  |

## Interfaces

The NHM provides type definitions, defines, constants, variables and external interface definitions (functions, not dbus methods) via the header file “NodeHealthMonitor.h”. Currently there are only two enumerations declared, which are used to identify numeric values that are sent over the dbus.

|  |  |  |
| --- | --- | --- |
| Enumeration | Value | |
| Name | Description |
| NhmAppStatus\_e | NhmAppStatus\_Failed | Used when an application has failed. |
| NhmAppStatus\_Restarting | Used when an application has failed but is in process of being restarted. |
| NhmAppStatus\_Ok | Used when an application failed but has correctly been restarted. |
| Description | This enum will be used to report the status of an application before/after or during a failure. | |

|  |  |  |
| --- | --- | --- |
| Enumeration | Value | |
| Name | Description |
| NhmErrorStatus\_e | NhmErrorStatus\_Ok | Used to state that the method worked as expected. |
| NhmErrorStatus\_Error | Used to state that an error occurred handling the request. |
| NhmErrorStatus\_UnknownApp | Set when the passed string does not correspond to a failed application. |
| NhmErrorStatus\_RestartNotPossible | Used when an application requests a node restart but it is not currently possible. |
| Description | This enum will be used for indicating the status of dbus method calls. | |

## Dbus connection

|  |
| --- |
|  |
| *Fig. 3 - NHM dbus connection* |

In this diagram, the dbus connection of the NHM with its relevant information is outlined.

The main methods of the NHM are offered on the “session” dbus. The NHM connects to the bus using  
the well-known bus name “org.genivi.NodeHealthMonitor”.

The dbus methods are exported under the dbus object path “/org/genivi/NodeHealthMonitor” in “org.genivi.NodeHealthMonitor.Info”.

A description of the dbus interface is provided in the “org.genivi.NodeHealthMonitor.Info.xml” XML model (that can be processed with gdbus-codegen).

In addition to the own interface, the NHM implements the “LifecycleConsumer” interface of the NSM and registers as a shutdown client. In this way, the NHM is called during every shut down and can maintain a flag to indicate the successful completion of the life cycle.

The NHM offers the following dbus methods:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | | | |
| Type | Dir | Name | Description |
| RegisterAppStatus | s (char\*) | in | AppName | The unit name of the application that has failed. |
| i (int) | in | AppStatus | This can be used to specify the status of the application that has failed. It will be based upon the enumeration NHM\_ApplicationStatus\_e. |
| Description | This method will be used by an NHM client to register that an application has failed or recovered from a previous failure. The Node Health Monitor will maintain an internal list of the applications that are currently in a failed state. Additionally it will maintain a count of the currently failed applications that can be used to trigger a system restart if the value gets too high. | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | | | |
| Type | Dir | Name | Description |
| ReadStatistics | s (char\*) | in | AppName | This will be the name of the application for which the calling application wants to know the failure count for. If this value is an empty string the NHM will return the failure statistics for the whole node. |
| u (uint) | out | CurrentFailCount | This value will be the number of failures that have occurred in this lifecycle. |
| u (uint) | out | TotalFailures | This will be the total number of failures that have occurred in the past X amount of lifecycles. The value of X will be configurable. |
| u (uint) | out | TotalLifecycles | This value will be the number of lifecycles that are being used for the statistics collection (i.e. 5 failures in 8 LCs). |
| i (int) | out | ErrorStatus | This parameter will be used as a return value |
| Description | This method can be used to read the failure count of either a particular application or of the Node itself. | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Parameters | | | |
| Type | Dir | Name | Description |
| RequestNodeRestart | s (char\*) | in | AppName | This is the unit name of the application that has failed. |
| i (int) | out | ErrorStatus | This parameter will be used as a return value. |
| Description | This method can be used by an NHM client to request a node restart if a critical application cannot be recovered. The Node Health Monitor will have the possibility to internally evaluate whether the failed application is important enough to warrant the restarting of the node. The NHM will then forward the request to the NSM who will evaluate whether a restart is allowed at the current time. | | | |

The NHM sends the following dbus signals:

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Parameters | | |
| Type | Name | Description |
| AppHealthStatus | s (char\*) | AppName | This is the unit name of the application whose status has changed. |
| i (int) | AppStatus | The new status of the application. |
| Description | This signal is used by clients to be notified about AppHealth status changes. | | |

# Internal design

This chapter gives an overview about the internal design of the NHM.

## Component diagram

The following diagram shows the components that are used by the NHM to provide its services. The NHM uses the following components:

* systemd: The function “sd\_notify” from the library “libsystemd-daemon” is used to notify systemd that NHM is started and to trigger the watchdog provided by systemd.
* DLT: In accordance with the GENIVI compliance, the NHM uses “Diagnostic Log and Trace” (DLT) to report status, error and debug messages.
* glib: glib is a package containing multiple libraries, which are split into different functionalities.
  + “libglib” provides support for list, array, timer and main loop functions
  + “libgio” is used to realize inter process communication via the dbus.
  + “libgio-unix” is used to catch the Linux specific SIGTERM signal, which is needed to terminate processes.
  + “libgobject” is a generic part used by the other glib libraries.
* PCL: The GENIVI persistent client library is used to write and read persistent values.

|  |
| --- |
|  |

|  |
| --- |
|  |

## Project layout

|  |
| --- |
|  |
|  |

This tree diagram shows the structure of the NHM code project. The sources of the NHM are located in subfolders, based on their content. The following subfolders are used:

* cfg: systemd service files, initial configuration, “package config” file and other administrative files.
* gen: Destination for generated code.
* inc: Public header files (see chapter ).
* mod: XML models of the dbus interfaces.
* src: Source code of the NHM.
* tst: Unit test for the NHM and stubs to replace system functions for the test.

Since the NHM is based on autotools the following standard files are used: AUTHORS, autogen.sh, ChangeLog, configure.ac, COPYING, Makefile.am, NEWS and README.

## Internal functions

It is not intended to provide an in depth overview of the internal functions of the NHM but the component is broken down into the following modules:

* nhm-main.c: The main function of the NHM providing dbus callbacks, configuration and data handling
* nhm-systemd: Functions to register for systemd signals and evaluate them to detect failed applications.
* nhm-helper: Common helper functions used in multiple places of the NHM.

## Configuration values

The NHM can be configured via the file “node-health-monitor.conf” that is located in the “confdir” (usually /etc/). The table below shows the available configuration keys and their default values (used if the key is not present or not readable).

| *Tab. 1 – Configuration keys* | |
| --- | --- |
| **Key** | **Detailed description** |
| historic\_lc\_count | Amount of life cycles that are stored by the NHM. Has to be a positive value.  Set to 0 (NHM default), to only monitor current life cycle. |
| max\_failed\_apps | Max. amount of apps. in fail state.  If the number of failed apps. exceeds this amount, a node restart will be requested. Has to be a positive value.  Set to 0 (NHM default) to disable restart because of failed apps. |
| no\_restart\_apps | Semicolon separated list of apps. for which a restart will be rejected, when they call the dbus method 'RequestNodeRestart'.  Leave empty (NHM default) to allow all apps to initiate restarts. |
| ul\_chk\_interval | Interval in s, in which NHM performs 'userland' checks. Always has to be a positive value.  Set to 0 (NHM default) to disable userland checks. |
| monitored\_files | Semicolon separated list of files that are observed by the NHM. If a configured file cannot be found, the NHM resets the system.  Leave empty (NHM default) to disable restarts because of files. |
| monitored\_procs | Semicolon separated list of processes that are observed by the NHM. If the process does not return with 0, the NHM resets the system.  Leave empty (NHM default) to disable restarts because of processes. |
| monitored\_dbus | Semicolon separated list of dbus addresses that are observed by the NHM (check if dbus is alive by pinging org.freedesktop.DBus).  Leave empty to disable (default) restarts because of failed busses. |

## Data storage format

The NHM maintains the PCL key value “ERG\_OIP\_SHUTDOWN\_FLAG”.

It evaluates and writes the flag at startup and at shut down. This way, this NHM is able to detect an unexpected termination of the previous Lifecycle.

Furthermore, information about failed applications is written in the file “lcdata” in the “datadir” (usually /var/run/lib”), every time an application changes its state.

The structure of the contents in the LC data is shown below.

|  |
| --- |
|  |
|  |

At the start of the file, the version of the NHM is stored. The version is followed by the number of stored life cycles. For each life cycle, the first information is the shutdown state. After that, the number of failed applications in the life cycle is stored. For each failed app., the app. name is stored (str. length and string) and the fail count.

## Sequence diagrams

In the use case shown below, whenever systemd detects a failing service (application) it changes the property “ActiveState” of the service to “failed”.

This changed property triggers a “PropertiesChanged” signal to be sent on the dbus. The reception of this signal leads to a callback in the NHM and the function “nhm\_systemd\_unit\_properties\_changed” is called. Only if the property “ActiveState” changed, the call is processed further and the internal function “nhm\_systemd\_unit\_active\_state\_changed” is called. The function verifies, if the “ActiveState” transition leads to a failed or recovered application and forwards the call to “nhm-main.c”, if the application state really changed.

|  |
| --- |
|  |
| *Fig. 7 – ActiveState change sequence* |

In the function “nhm\_main\_register\_app\_status“, the new application status finally is reported to the NSM and the NHM “AppStatus” signal is sent out on the dbus. Furthermore, the updated information about the application is persisted for the current life cycle and a check is initiated if the amount of failed applications should lead to a restart of the system.

## Project layout and compilation

As explained in chapter , and shown in the tree diagram, the NHM is an autotools project. Therefore, building and installing the project typically involves the following steps:

* autogen.sh
* make
* (optional) make check
* make install

The available configuration values and the required package dependencies are explained in the README file.

## Unit test and quality

The NHM is delivered with a unit test that can be started via the “make check” command. A coverage analysis with the tool “BullsEye” resulted in a function-coverage of 100 % and a branch-coverage of 84 % for NHM in version 1.3.3. The unit test should be considered in every coming source code adaption. A static code analysis with the tool “Klocwork” and a memory leak check with “valgrind” did not show any findings for the NHM source.

# Open Issues

| **Open Issue** | **Detailed description** |
| --- | --- |
|  |  |

# References

|  |  |
| --- | --- |
| **Name of the Document** | **Version or Release Date** |
|  |  |

# Checklists

## Node State Manager

|  |  |
| --- | --- |
| **Application or Service Name** | **<Name>** |
| **Shutdown notification required** | yes  *The Node Health Monitor sets a flag during the shutdown process that it uses in the runup to validate that a valid shutdown occurred in the previous lifecycle. Additionally, it stores all its statistics in the persistency during the shutdown* |
| **Type of Shutdown required** | normal |
| **Ability to perform system restarts** | yes  *The NHM can use the NSM Restart functionality in case of critical application failures* |
| **Using existing sessions** | no |
| **Planning on adding own sessions** | no |

## Health Monitoring

|  |  |
| --- | --- |
| **Application or Service Name** | **Node Health Monitor** |
| **Monitoring Required** | yes |
| **Which functionality** | There is only one process thread in the Node Health monitor and it will will send heartbeats to systemd |
| **Monitoring only in cooperation w/ product** | No |
| **How to detect** | <…>   * *The Node Health Monitor will itself be monitored by systemd and will have an appropriate OnFailure component with a product specific escalation strategy* * *The Node Health Monitor will send heartbeats to systemd from its one and only thread* |
| **How to recover** | <…>   * *The Node Health Monitor will provide an escalating recovery strategy:* * *it will use a shared persistency counter with the NHM to check how many times the NHM has failed without a successful rundown* * *If this counter is less than 3 then it will increment the counter and request a node restart via the Node State Manager* * *If this counter is more than 3 then it will delete the NHM persistency data and reset the counter* |
| **Monitoring of child processes required?** | <…>  *No, there are no child processes* |
| **Unit file configuration** | ***[Unit]***  OnFailure=  ***[Service]***  RestartSec=  TimeoutStartSec=  TimeoutStopSec=  TimeoutSec=  WatchdogSec=  Restart=  StartLimitInterval=  StartLimitBurst=  StartLimitAction= |

## Profile Manager

|  |  |
| --- | --- |
| **Application or Service Name** | **<Name>** |
| **Does the component handle user commands or user data?** | no  *The health monitoring is a system function and is not user specific in any way* |
| **Synchronous user switching** | N/A |
| **How are multiple users supported** | N/A |
| **How are multiple seats supported** | N/A |
| **How are dependencies to other user specific components handled** | N/A |
| **Recommend timeout and timeout handling.** | N/A |

## Persistency

|  |  |
| --- | --- |
| **Application or Service Name** | **NodeHealthMonitor** |
| **Persistency Storage Required** | yes |
| **Which functionality** | Store statistics of failures in the system and also shutdown lifecycle flag |
| **Type of data** | system |
| **Properties of the data** | ReadWrite |
| **Coding Data** | no |
| **When is data access needed** | no |
| **Data Size** | Max size is dependent on the number of applications in the system that can fail. We have set a max limit of 255 Apps with each app requiring 32 bytes |
| **Data Priority** | no |
| **Data caching** | yes |

|  |  |
| --- | --- |
| **Application or Service Name** | **<Name>** |
| **Persistency Storage Required** | No  All in Key Value store |
| **Which functionality** | N/A |
| **Type of file** | N/A |
| **Data Size** | N/A |
| **Data caching** | N/A |