**Participants**

This document has been created collaboratively by members of the System Infrastructure expert group. This document includes a combination of the contents of the GENIVI EA model and diagrams from the GENIVI Wiki.

**Change History**

The following table shows the change history for this specification.

|  |  |  |
| --- | --- | --- |
| Date | Version | Changes |
| 20.12.2012 | 0.1 draft | Initial Version |
| 21.12.2012 | 0.2 draft | Updates following review in the EG-SI telco 20.12.2012 |
| 03.01.2013 | 0.3 draft | Updates following review comments from Marko Hoyer, ADIT |
| 16.01.2013 | 0.4 draft | Updated following review from Guido Penella (pls see GT-1301) |
| 10.02.2013 | 1.0 released | Updated use cases following review comments from Marko Hoyer, ADIT |
| 13.12.2013 | 2.0 draft | Updated interface between the NSM and the NSMC |

Table of Contents

1 Lifecycle - Subdomain Overview 4

1.1 Subdomain Lifecycle Concept Refinement 5

1.2 Logical view of Lifecycle 6

1.3 Node State Manager Specification 6

1.3.1 Realization 8

1.3.2 Node State Manager Data 9

1.3.2.1 Node Boot Mode 9

1.3.2.2 Node Application Mode (NAM) 10

1.3.2.3 Node State 10

1.3.2.4 Node Session State 11

1.3.2.5 Running Reason 12

1.3.2.6 Restart Reason 13

1.3.2.7 Shutdown Reason 13

1.3.3 Node State Manager Model Tags 14

1.4 Split between Startup and Shutdown Management 14

1.4.1 Startup Management 15

1.4.2 Shutdown Management 15

1.4.2.1 Introduction 15

1.4.2.2 About the concept 16

1.5 Deployment Diagram 18

1.6 Node State Manager Requirements Traceability 18

2 Use Case Definition 20

2.1 WB - Application Control - SWL 21

2.2 WB - NSM - Add new session state 22

2.3 WB - NSM - Change Node Boot Mode 22

2.4 WB - NSM - List of registered consumers 23

2.5 WB - NSM - Updating the Node Application Mode 23

2.6 WB - NSM - Wakeup reason handling 24

2.7 WB - NSM/Health - Session reset on failed app 25

2.8 WB - Shutdown - Application blocking system shutdown 26

2.9 WB - Shutdown - Cancel shutdown NOK 29

2.10 WB - Shutdown - Cancel shutdown ok 29

2.11 WB - Shutdown - Consumer blocking system shutdown 31

2.12 WB - Shutdown - Fast shutdown 33

2.13 WB - Shutdown - Normal shutdown seq 33

2.14 WB - Shutdown - Phone call cancels shutdown 34

2.15 WB - Startup - Last user mode dependency 37

2.16 WB - State Mgmt - Session State Handling 37

3 Interface Definition 39

3.1 NsmAccess 39

3.2 NsmcAccess 40

3.3 org.genivi.NodeStateManager.Consumer 42

3.4 org.genivi.NodeStateManager.LifecycleConsumer 49

3.5 org.genivi.NodeStateManager.LifecycleControl 50

3.6 Defines 54

3.6.1 NSM\_\_pau8CoreSessionName\_t 55

3.6.2 NsmApplicationMode\_e 57

3.6.3 NsmDataType\_e 59

3.6.4 NsmErrorStatus\_e 60

3.6.5 NsmRestartReason\_e 62

3.6.6 NsmRunningReason\_e 64

3.6.7 NsmSeat\_e 67

3.6.8 NsmSessionState\_e 68

3.6.9 NsmShutdownReason\_e 69

3.6.10 NsmNodeState\_e 71

4 Requirements 73

4.1 Vehicle Requirements 73

4.2 SW Platform Requirements 76

# Lifecycle - Subdomain Overview

This document provides details the Component Specification for the Node State Manager (NSM).

The Node State Manager together with the Node State Machine (NSMC) make up the logical block Node State Management as shown in the subdomain diagram below.

## Subdomain Lifecycle Concept Refinement

## Logical view of Lifecycle

## Node State Manager Specification

The Node State Manager (NSM) is proposed as a P2 Abstract component.

It is the central repository for information regarding the states/sessions inside the node. It collates information from multiple sources and uses this to determine the current states (there are different states existing for different purpose).

This information will be delivered to registered consumers or can be requested as needed using the provided interfaces. Sections of the raw data gathered by the NSM will also be made available on request to interested consumers.

The NSM also provides the shutdown management of the system, so one part of the information which is provided is the shutdown request notification event/message to the consumers.

The node state management is the last/highest level of escalation on the node, therefore it will be used by consumers that want to request a restart of the node.

Additionally if the system is a multi-node system the node state management has included a slave (done as a consumer) which knows the specifics of this configuration and knows about what event/message need to be transferred also to other node(s).

**Configuration:**

As mentioned previously the node state management consists of the Node State Manager (NSM) and the Node State Machine (NSMC). The NSM is a platform executable that provides 2 DBUS interfaces for interested clients. The NSMC is a library that will be linked into the NSM and will be replaceable as needed by any product based on a GENIVI platform.

The concept is that the NSMC will be an OEM defined state machine that will receive vehicle inputs (i.e. Clamp State) and Node States (shutdown requests, session states) and will, dependent on the state machine inject state changes into the NSM.

For example, a phone application has set the phone session active in the NSM. This information will be provided to the NSMC. This will have no effect on the state machine until it further receives a clamp change event indicating that the user has removed the key from the ignition. At this point the NSMC will want to initiate a shutdown but will delay this event because the phone call is still active. It is the OEM specific whether the phone session is timed out and a shutdown event is injected to the NSM or whether the shutdown is postponed.

### Realization





### Node State Manager Data

In this section we detail the data that the Node State Manager tracks and maintains, and where sensible provide examples of how this data is intended to be used (for more detailed information about the exact enumerations of these data please see section [Interface Definition](#_Interface_Definition))

#### Node Boot Mode

The Node Boot Mode (NBM) is a Node specific data item that is used to dictate the Kernel Image that should be started. Only one of the following modes can be active at any one time.

* Application (default mode)
* SW Loading (set only by an authorized application (Diagnosis, SWL or SHM)
* TestSoftware (generally only available in Factory and Development phase)

Updated via the Node State Manager when requested through a secured interface (only intended to be used by Diagnosis, SW Loading and Node Health Monitor)

It is intended that this persistent data will be used by the Lifecycle Support Library (running at some point before systemd) and therefore must be available very early in the system startup and with minimal library overhead.

#### Node Application Mode (NAM)

The Node Application Mode (NAM) is a Node specific data item that is used to define the functionality level that should be achieved in the current Lifecycle. Only one of the following modes can be active at any one time:

* Parking
* Factory
* Transport
* SWL

It can be updated via the Node State Manager when requested through a secured interface (intended to be used by Diagnosis, SW Loading and Node Health Monitor).

It will be used by the Lifecycle Support Library to determine the applications that should be started in the following lifecycle (i.e. the name of the target file that should be passed to systemd on the kernel command line).

So for instance when it is set to “Transport” the node will be started in a mode whereby PDC/RVC should be started but no other applications.

#### Node State

The Node State is a Node specific data item used to track the current state of the Head Unit.

It is intended that it is only to be updated by internal components within the Lifecycle subdomain.

Typically this will be used by the Node State Machine (NSMC) to trigger a node shutdown and by the Node State Machine to indicate to an external component that the shutdown has been completed.

Only one of the following states can be active at any one time:

* Start-up (default state on power on)
* Base running (set by Node Startup Controller (NSC) when it is initialized in the focussed.target)
* Last User Context (LUC) (set by NSC when all LUC components are running)
* Fully running (set by NSC when systemd signals unfocussed.target is complete)
* Fully operational (set by NSC when systemd signals lazy.target is complete)
* Degraded power (set by NSMC when we are not shutting down but supply manager indicates a “Poor” supply state
* Shutdown delay (set by NSMC when it receives a shutdown request and before it has started shutdown procedure)
* Shutting down (set by NSMC when it wants to initiate the shutdown procedure within the NSM)
* Fast shutdown (set by NSMC when it wants to initiate a fast shutdown within the NSM, based on a request from Diagnosis)
* Shutdown (set by NSM last consumer has completed shutdown – used by a component controlling the power supply to the node)

The Node State will be available to all interested clients in the system and can be used to trigger product specific handling, i.e. the “Shutdown delay” could be used by the Audio and HMI as a trigger to switch into a User Perceived off mode.

#### Node Session State

An important part of the concept is the handling of Sessions within the node. It is intended that sessions can be used by applications to notify the Node State Management about their current state (typically active or not). The information provided by these sessions will be used by the Node State Machine when it comes to determining the next Node State.

For instance the Phone application can set a “Phone Session” to active to indicate that a phone call is currently in progress. The NSMC would use this information to determine that if it receives a clamp state signal that normally triggers a shutdown, that the shutdown should be delayed until the phone call is no longer active (or until a product specific timeout occurs).

The Node Session State (NSS) contains information about the current sessions that are active in that Head Unit. All defined “sessions” can be active in parallel and there is no direct interaction between sessions. The following sessions have been identified within the platform and are created by default by the Node State Manager:

* Heating/Ventilation/Air Con active
* Rear View Camera active
* Park Distance Control active
* HMI active
* Network Passive State
* Network active
* Phone Session active
* Entertainment Mode active
* SW Loading Session active
* Diagnostic Session active

Additionally clients can create new sessions that they can then use for whatever internal usage they want. For instance an OEM could define for a product that they want a “Navigation” session that is used to determine the mode that an installed navigation application is running in. The NSMC that is also product specific would then be able to use this “Navigation” session as part of his internal decision matrix.

As it is possible that a GENIVI platform will operate in a multi-seat environment it is also possible to set the “Vehicle Seat” for whom the session is currently active.

The sessions will be updated via the Node State Manager when triggered by a session controlling application through a public API.

#### Running Reason

The Running Reason contains information about why the Head Unit was awoken in the current Lifecycle and will be used for Wakeup events as well as Restart events. Only one of the following reasons can be active in any Lifecycle:

* WakeupCan - Wake up because of CAN activity
* WakeupMediaEject - Wake up because of 'Eject' button
* WakeupMediaInsertion - Wake up because of media insertion
* WakeupHevac - Wake up because of user activating the HEVAC unit in the car
* WakeupPhone - Wake up because of a phone call being received
* WakeupPowerOnButton – Wake up because user presses the "Power ON" button in the car
* StartupFstp - System was started due to a first switch to power
* StartupSwitchToPower - System was switched to power
* RestartSwRequest - System was restarted due to an internal SW Request (i.e. SWL)
* RestartInternalHealth - System was restarted due to an internal health problem
* RestartExternalHealth - System was restarted due to an external health problem
* RestartUnexpected - System was restarted due to an unexpected kernel restart.
* RestartUser - Target was reset due to user action (i.e user 3 finger press)

This data item will be updated via the NSM based around the value that it expects to find in the kernel command line. It is expected that a product component will exist in the Bootloader/UBOOT that will receive the real trigger (i.e. Front Panel, Vehicle Bus, …) and update the kernel command line as needed.

It will be used by a the NSM state machine to determine correct actions to events (i.e. WakeupPowerOnButton would result in the target staying alive for a product defined period before performing an automatic shutdown).

It would also be used by the Lifecycle Support Library (running at some point before systemd) to assist in the decision about the feature set required for the current Lifecycle (i.e. Disc eject requires only enough functionality to eject the media from the drive).

#### Restart Reason

The Restart reason contains information about why the Head Unit was shut down in the previous Lifecycle. Only one of the following reasons can be active at any one time:

* Unknown (default value set newly on each lifecycle)
* Shutdown (set by the NSM when it performs an intentional shutdown)
* SW Loading
* Diagnosis
* Application failure

Updated via the NSM when requested to perform a system restart by an authorized application

Used in the subsequent lifecycle by Node Health Monitor as part of its health tracking over multiple lifecycles (i.e. an intentional system restart due to SW Loading is not a problem but multiple restarts with “Unknown” is)

Used in the subsequent lifecycle by Error Mgmt to log an error code if an unexpected system restart occurred

#### Shutdown Reason

The Shutdown reason contains information about why the Head Unit was shut down in the previous Lifecycle. Only one of the following reasons can be active at any one time:

* Clamp State (default value)
* Thermal Bad
* Thermal Poor
* Supply Bad
* Supply Poor

It is updated via the NSM when an event in the system triggers the shutting down of the system (i.e Thermal Mgmt reports a dangerously high temperature)

Used in the subsequent lifecycle by Error Mgmt to log an error code if a non standard shutdown occurred

Used in the subsequent lifecycle by Node Health Monitor as part of its health tracking over multiple lifecycles

### Node State Manager Model Tags

| Custom Properties | |
| --- | --- |
|  | isIndirectlyInstantiated = True |

|  |  |
| --- | --- |
| Tagged Values | |
|  | Implementation-URL = http://git.projects.genivi.org/?p=lifecycle/node-state-manager.git;a=summary. |
|  | Implementation-Version = >=2.0.0. |
|  | Priority = P1 |
|  | Specification-URL = https://collab.genivi.org/wiki/display/genivi/SysInfraEGLifecycleConcept. |
|  | Specification-Version =2.0.0. |

## Split between Startup and Shutdown Management

Systemd does stop and unload the components in its shutdown concept. That means it will take a lot of time to make them functional again in case of a cancel shutdown situation. The IVI system have to come back fast without losing any context, if a reboot can be avoided for that cancel shutdown. Node State Management will only call registered consumers in the shutdown phase. Those consumers will drive the components into a stable state and ensure that everything have been stored which will be needed for the next startup.

Note: The components won’t be shutdown (Exceptions are existing like the flash filesystem). Therefore in addition the shutdown management concept will include/use the systemd shutdown concept as well where it makes sense and supports legacy/adapted components.

### Startup Management

In the following diagram you can see how the Node State Manager is used during the startup sequence to register for notifications during the shutdown sequence

### Shutdown Management

#### Introduction

In automotive the aim in a shut-down phase is to trigger real needed activities only. That means a lot of standard/traditional activities for shut-down are not in the focus in an IVI-system. If you can archive the requested start-up KPI with a cold boot the IVI system will be switched off at the end of the shut-down phase.  
The main activities which are need in shut-down are the following:

* log off the user
* write / specify the last user context
* unregister at the vehicle network
* bring hardware in a safe state (drives,...)
* mute audio
* switch off display(s)
* store application and system persistence data

#### About the concept

The basic idea is the following:

During start-up phase the applications,... will register at the Node State Management which are interested to be involved during a shut-down phase.

The order of registration in start-up is the inverted order of the execution in shut-down phase. Currently we think that we do not need a specific specification of the shut-down behavior. There will be a protocol defined between the registered consumers and the Node State Management.

There will be specific consumers which are responsible to set a shutdown target or a list of conflicts to systemd, which enables the de-initialization of (legacy) components via systemd.

## Deployment Diagram



Figure: 1

The Node State Machine (NSMC) is a library that is linked into the Node State Manager executable. As can be seen in the diagram above it provides the NsmcAccess interface (defined later in this document) which will be used by the Node State Manager (NSM) to inform the NSMC above state changes in the system.

## Node State Manager Requirements Traceability



Figure: 2

# Use Case Definition



Figure: 3

## WB - Application Control - SWL

In this use case we show Lifecycle components are used during the application update process.

Specifically we show the following interactions

SWL -> NSM -> activation of the SWL Session

Applications -> SHM -> de-registration and re-registration of watchdogs either side of the update process

**Sequence diagram: WB - Application SW Update**



Figure: 6

## WB - NSM - Add new session state

In this use case it is shown how a new session state can be added to the NSM and how clients can register for notifications on the session state



**Sequencediagram: WB - NSM - Add new session state**

## WB - NSM - Change Node Boot Mode

In this use case it is shown how certain applications will be able to change the Node Boot Mode via an interface provided by the NSM.

**Sequence diagram: WB - NSM - Change Node Boot Mode**



## WB - NSM - List of registered consumers

To fulfill the requirement that all registered components are listed for debugging a trace output will be included whenever a consumer registers or de-registers

## WB - NSM - Updating the Node Application Mode

In this use case we show the interaction required to change the Node Application Mode

**Sequence diagram: WB - NSM - Updating the Node Application Mode**



## WB - NSM - Wakeup reason handling

In this use case we show how the Wakeup reason is passed as a common line parameter to the kernel and on to the Node State Management and hence available to read by a consumer

**Sequence diagram: WB - NSM - Wakeup reason handling**



## WB - NSM/Health - Session reset on failed app

In this use case we see that if an application fails and it was registered as controlling a session that was active that the NSM would reset the session state

**Sequence diagram: WB - NSM/Health - Session reset on failed app**



## WB - Shutdown - Application blocking system shutdown

The system should shutdown but this is delayed due to the state of one application in the system (i.e. the user is making a phonecall). The system will stay on until the bocking event is resolved or until a configurable timeout for that application has expired.

**Trigger:** Shutdown is requested due to Clamp State

**Pre-conditions:**

* System is running and phone call is in progress

**Main Flow:**

1. Clamp State change is detected and reported to the Node State Manager
2. Node State Manager evaluates operational mode of the system
3. NSM delays system shutdown until Phone call is ended
4. Phone call ends
5. NSM is notified of operational mode change to Normal
6. NSM initiates shutdown by informing all consumers of shutdown request
7. All consumers accept shutdown request
8. NSM requests power off

**Alternative Flow [A1]:**

1. Clamp State change is detected and reported to the Node State Manager
2. Node State Manager evaluates operational mode of the system
3. NSM delays system shutdown until Phone call is ended
4. NSM times out the Phone Application
5. NSM initiates shutdown by informing all consumers of shutdown request
6. All consumers accept shutdown request
7. NSM requests power off

**Exception Flow [Ex1]:**

**Post-conditions:**

* System is shutdown

**Notes:**

Assumptions, issues and all other notes go here.

**Sequence diagram: Application Blocking Shutdown**



## WB - Shutdown - Cancel shutdown NOK

Here an ongoing shutdown is attempted to be cancelled but it is too late in the process and the shutdown completes and the system starts up immediatly

**Sequence diagram: WB - Shutdown - Cancel shutdown NOK**



## WB - Shutdown - Cancel shutdown ok

An ongoing system/node shutdown is cancelled and the target returns to full operational mode

**Trigger:** User requests restart of the system, interrupting an ongoing shutdown

**Pre-conditions:**

* System is up and running normally

**Main Flow:**

1. User turns off engine and removes key
2. Node receives change of clamp state
3. NSM evaluates if shutdown is allowed
4. NSM initiates shutdown sequence, informing consumers of event
5. Consumers start to acknowledge shutdown
6. User re-inserts key and starts engine
7. Node receives change of clamp state
8. NSM initiates system restart informing consumers of event
9. Consumers accept state change
10. System is back running in fully configured operational state

**Alternative Flow [A1]:**

1. first step
2. second step

**Exception Flow [Ex1]:**

1. User turns off engine and removes key
2. Node receives change of clamp state
3. NSM evaluates if shutdown is allowed
4. NSM initiates shutdown sequence, informing consumers of event
5. Consumers start to acknowledge shutdown
6. User re-inserts key and starts engine
7. Node receives change of clamp state
8. NSM initiates system restart informing consumers of event
9. A consumer fails to acknowedgle running state
10. NSM initiates system restart following normal recovery procedures
11. System is restarted

**Post-conditions:**

* System is back running in fully configured operational state

**Notes:**

Assumptions, issues and all other notes go here.

**Sequence diagram: WB - Cancel shutdown ok**



## WB - Shutdown - Consumer blocking system shutdown

The user initiates a system shutdown but one consumer is not responding to shutdown requests. The system will timeout the blocking consumer and shut down the system after a configuable time.

**Trigger:** System shutdown initiated

**Pre-conditions:**

* System is running in a normal state

**Main Flow:**

1. Platform State Manager informs all consumers of shutdown request
2. All consumers confirm shutdown request
3. Platform State Manager sends power off request

**Alternative Flow [A1]:**

1. Platform State Manager informs all consumers of shutdown request
2. One consumer does not confirm shutdown request immediatly
3. Platform State Manager delays system shutdown
4. Blocking consumer confirms shutdown
5. Platform State Manager sends power off request

**Exception Flow [Ex1]:**

1. Platform State Manager informs all consumers of shutdown request
2. One consumer does not confirm shutdown request
3. Platform State Manager delays system shutdown
4. Platform State Manager times blocking consumer
5. Platform State Manager sends power off request

**Post-conditions:**

* System has shutdown

**Notes:**

Assumptions, issues and all other notes go here.

**Sequence diagram: Consumer Blocking Shutdown**



## WB - Shutdown - Fast shutdown

This use case relates to the request to shutdown the system by Diagnosis as part of a Coding process.

The requirement is that the system is shutdown as fast as possible. It is accepted that this fast shutdown can result in the loss of cached persistent data from that Lifecycle.

**Sequence diagram: WB - Shutdown - Fast shutdown**



## WB - Shutdown - Normal shutdown seq

In this use case the NSM has received a Clamp state change notification that results in the NSM initiating a shutdown of the system.

Nothing blocks the shutdown and the sequence runs to the point that the power is removed

**Sequence diagram: WB - Shutdown - Normal sequence**



## WB - Shutdown - Phone call cancels shutdown

During the shutdown process, and before the Phone stack has been shutdown, a phone call is received. This event forces the cancellation of the ongoing shutdown process.

The shutdown will either be restarted after a product configurable timeout (if the call is not ended) or when the call had finished (if there are no other reasons for the Lifecycle to remain).

**Sequence diagram: WB - Shutdown - Phone call cancels shutdown**



## WB - Startup - Last user mode dependency

During the System Startup the "Last User Mode" must be evaluated when determining the dependency graph.

**Sequence diagram: Last User Mode Flow**



## WB - State Mgmt - Session State Handling

The system will run in one of a number of Application states (i.e. Phone, Diagnostic, SWL, ...).

These states can be triggered by external components (i.e. Tester) and internal components (i.e. phone application)

**Sequence diagram: Session State Handling**



# Interface Definition

## NsmAccess

This interface will be used by the NSMC to read and write data available in the NSM. It provides a pair of generic set/get interfaces that can be used to read/write all data available in the NSM.

Operations

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **NsmGetData()** int  Public | This is a generic interface that can be used by the NSMc to read a specific data item that from the NSM.  If the function has read the requested data correctly then it will return a positive value equal to the amount of data read. If the data item does not exist then it will return 0 and if another error occurs then it will return a negative value | NsmDataType\_e [in] dataType  This will specify the data to be read  puint8\_t [inout] dataPtr  This will be a memory pointer allocated by the caller that the NSM will use to store the requested data item. The data will be then be cast to the appropriate type based on the dataType value  int [in] dataSize  This will be used by the NSMC to specify the max size of the data allowed to be written and will be used by the NSM to specify the size of the data actually written |
| **NsmSetData()** NsmErrorStatus\_e  Public | This is a generic interface that can be used by the NSMc to write a specific data item to the NSM.  If an error occurs then it will be returned in the return value which is of type NsmErrorStatus\_e | NsmDataType\_e [in] dataType  This will specify the data to be written  puint8\_t [inout] dataPtr  This will be a memory pointer allocated by the NSMC and filled with the data to be written. It will be read and cast to the appropriate type based on the dataType value  int [in] dataSize  This will be used by the NSMC to specify the size of the data to be written |
| **NsmGetInterfaceVersion()** int  Public | Request the interface version from the Node State Manager |  |

## NsmcAccess

This interface will be a private protected interface to be used by the Node State Manager to communicate with the Node State Machine

Operations

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **NsmcReqNodeRestart()** bool\_t  Public | This will be used by the NSM to request a node restart when requested by one of its clients.  It passes through the reason for the request (i.e. SWL, App Failure) and the type of restart required (i.e Fast or Normal)  If the request can’t be performed at the current time then it will return false otherwise true. | **NsmRestartReason\_e** enRestartReason  **unsigned int** u32RestartType |
| **NsmcLucRequired()** bool\_t  Public | This will be used by the NSM to check whether in the current Lifecycle the LUC should be started.  This allows the product to define its own handling for specific Application modes |  |
| **NsmcSetData()** void  Public | This is a generic interface that can be used by the NSM to inform the NSMC about changes to data items (i.e. events that have occurred in the system) | NsmDataType\_e [in] dataType  This will specify the data to be written  puint8\_t [in] dataPtr  This will be a memory pointer allocated by the NSM and filled with the appropriate data. It will be read and cast by the NSMC to the appropriate type based on the dataType value  int [in] dataSize  This will be used by the NSM to specify the size of the data to be written |
| **NsmcInit()** bool\_t  Public | This function will be used to initialise the Node State Machine, it will be called by the Node State Manager. At the point where this function returns the machine is available to accept events via its interfaces from the NSM.  It is envisaged that in this call the NSMC will create and transfer control of the NSMC to its own thread and will return in the original thread.  If the request can not be performed at the current time then it will return false otherwise true. |  |
| **NsmcGetInterfaceVersion()** int  Public | Request the interface version from the Node State Machine |  |

## org.genivi.NodeStateManager.Consumer

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  org.genivi.NodeStateManager.Consumer |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **BootMode** int32  Public  Static  «DBUSProperty» | This property will be based upon an external BootMode definition that will be product specific and hence not defined here. This is simply a framework that can be used as needed | *Default:* |
| **RestartReason** int32  Public  Static  «DBUSProperty» | This property will be based upon the NsmRestartReason\_e | *Default:* |
| **ShutdownReason** int32  Public  Static  «DBUSProperty» | This property will be based upon the NsmShutdownReason\_e | *Default:* |
| **WakeUpReason** int32  Public  Static  «DBUSProperty» | This property will be based upon the NsmWakeupReason\_e | *Default:* |

Operations

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **GetAppHealthCount()**  Public | This method can be used to read the number of failed applications in the system | int32 [out] Count  Number of currently failing apps |
| **GetApplicationMode()**  Public | The method is used by other applications to get the application mode. | int32 [out] ApplicationModeId  Will be based on NsmApplicationMode\_e  int32 [out] ErrorCode |
| **GetNodeState()**  Public | The method is used by other applications to get the NodeState without the need of registration to the signal. | int32 [out] NodeStateId  Will be based on the NsmNodeState\_e  int32 [out] ErrorCode  NsmErrorStatus\_e value |
| **GetSessionState()**  Public | The method is used by applications to get the state of a session. | string [in] SessionName  int32 [in] SeatID  This parameter will be based upon the enum NsmSeat\_e  int32 [out] SessionState  Will be based on the NsmNodeState\_e but it will not be bounded by the values in that enum as extended session states are possible for new product sessions  int32 [out] ErrorCode  NsmErrorStatus\_e value |
| **LifecycleRequestComplete()**  Public | The method is used by a client to notify the NSM that a LifecycleRequest has been completed. | uint32 [in] RequestId  ID of the Lifecycle request  int32 [in] Status  Status of the request to be performed. This will be based on the enum NsmErrorStatus\_e  int32 [out] ErrorCode  NsmErrorStatus\_e value |
| **NodeApplicationMode()**  Public | Clients can register for notifications when the Node Application Mode is updated inside the Node State Manager. This signal is sent to registered clients and will include the current Node Application Mode as a parameter | int32 [out] ApplicationModeId  This parameter will be based upon the NsmApplicationMode\_e |
| **NodeState()**  Public | Clients can register for notifications when the Node State is updated inside the Node State Manager. This signal is sent to registered clients and will include the current Node State as a parameter | int32 [out] NodeStateId  This parameter will be based upon the NsmNodeState\_e |
| **RegisterSession()**  Public | The method is used by other applications to register a new session whose state should be observed and distributed by the NSM. | string [in] SessionName  Session name to be registered  string [in] SessionOwner  This is the name of the application that is registering the new session (this must be the applications systemd unit filename)  int32 [in] SeatID  This parameter will be based upon the enum NsmSeat\_e  int32 [in] SessionState  Will be based on the NsmNodeState\_e but it will not be bounded by the values in that enum as extended session states are possible for new product sessions  int32 [out] ErrorCode  Returned error code of the method.  NsmErrorStatus\_e value |
| **RegisterShutdownClient()**  Public | The method is used by other applications to register themselves as shutdown client. Any client that registers must provide a method in their DBUS object called NSMLifecycleRequest. This method will take one parameter which is the Lifecycle Request (i.e. normal shutdown, fast shutdown, runup). For an example of the required client interface please see the Boot Manager component who will be a client of the NSM | string [in] BusName  Bus name of remote ap-plication.  string [in] ObjName  Object name of remote object that provides the shutdown interface  int32 [in] ShutdownMode  Shutdown mode for which client wants to be informed (i.e normal, fast etc)  uint32 [in] TimeoutMs  Max. Timeout to wait for response from shutdown client.  int32 [out] ErrorCode  Returned error code of the method.  NsmErrorStatus\_e value |
| **SessionStateChanged()**  Public | This signal is sent to registered clients when a particular session is state is changed. The client can register for notification about a specific session through the use of the Session Name.  The signal will include the session name and the new session state | string [out] SessionName  The Session name will be based upon either the pre-defined platform session names or using a newly added product defined session name  int32 [out] SeatID  This parameter will be based upon the enum NsmSeat\_e  int32 [out] SessionState  This parameter will be based upon the NsmSessionState\_e but it will not be bounded by the values in that enumeration. The listed values are the default values that are mandatory for platform sessions, but product sessions may have additional session states |
| **SetSessionState()**  Public | The method is used by applications to set the state of a session. | string [in] SessionName  This is a null terminated string that identifies the name of the session that should be updated  string [in] SessionOwner  This parameter defines the name of the application that is setting the state of the session. This must be the applications systemd unit filename.  int32 [in] SeatID  This parameter will be based upon the enum NsmSeat\_e  int32 [in] SessionState  Will be based on the NsmNodeState\_e but it will not be bounded by the values in that enum as extended session states are possible for new product sessions  int32 [out] ErrorCode  NsmErrorStatus\_e value |
| **UnRegisterSession()**  Public | The method is used by other applications to remove a new session from the session list hosted by NSM. | string [in] SessionName  Session name to be unregistered  string [in] SessionOwner  This is the name of the application that originally registered the session. It will be validated that this value matches the stored value from the registration  int32 [in] SeatID  This parameter will be based upon the enum NsmSeat\_e  int32 [out] ErrorCode  Returned error code of the method.  NsmErrorStatus\_e value |
| **UnRegisterShutdownClient()**  Public | The method is used by other applications to unregister themselves as shutdown client. | string [in] BusName  Bus name of remote ap-plication.  string [in] ObjName  Object name of remote object that provides the shutdown interface  int32 [in] ShutdownMode  Shutdown mode from which client wants to be unregistered  int32 [out] ErrorCode  Returned error code of the method.  NsmErrorStatus\_e value |

## org.genivi.NodeStateManager.LifecycleConsumer

This is the interface that must be provided by clients that are interested in being notified about shutdown events

Operations

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **LifecycleRequest()** void  Public | This method must be provided by a client registered for shutdown events. | uint32 [in] Request  The type of the life cycle request. Can be  NSM\_SHUTDOWNTYPE\_RUNUP  NSM\_SHUTDOWNTYPE\_NORMAL or  NSM\_SHUTDOWNTYPE\_FAST  uint32 [in] RequestId  The RequestId will be a unique identifier that the client must use in subsequent response completion messages (i.e. in LifecycleRequestComplete)  int [out] ErrorCode  This is the status of the call the valid response are  NsmErrorStatus\_Ok -> the client has completed the request within the context of this call  NsmErrorStatus\_Error -> the client can not perform the request  NsmErrorStatus\_ResponsePending -> the client will call the NSM LifecycleRequestComplete method when it has completed the request |

## org.genivi.NodeStateManager.LifecycleControl

This is a set of methods that can be used to control the Lifecycle. It is assumed that some authentication will be carried out within the methods to be ensured that the process calling the interface is allowed to run that method

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  org.genivi.NodeStateManager.LifecycleControl |  |

Operations

| **Method** | **Notes** | **Parameters** |
| --- | --- | --- |
| **CheckLucRequired()**  Public | This method will be called exclusivley by the Boot Manager to find out whether the LUC Applications should be started in the current Lifecycle.  This is required whilst in certain Node Application Modes (i.e. Transport, Factory) we do not want the LUC Applications started. Internally the Node State Manager will need to call a product interface to get the mapping back for the current NAM. | bool [out] LucWanted  This will be a simple TRUE or FALSE to define whether the LUC is wanted in the current Lifecycle |
| **RequestNodeRestart()**  Public | The method is used by other applications to request a restart of the node. | int32 [in] RestartReason  The passed value will be based upon the enum NsmRestartReason\_e.  int32 [in] RestartType  This parameter will define the type of shutdown that is requested as part of the restart process. The acceptable values for this parameter are  NSM\_SHUTDOWNTYPE\_NORMAL  and  NSM\_SHUTDOWNTYPE\_FAST  int32 [out] ErrorCode |
| **SetAppHealthStatus()**  Public | This method will be used by the Node Health Monitor to report to the NSM if an application has failed and if it is running again. Internally the NSM will use this information to keep a count of the number of failed applications within the current lifecycle. Additionally it will unset any sessions that the failing application may have had active. It will also be possible for the product node state machine to make a decision on what to do with this information, i.e. even reset the node or reset the node if too many applications have failed | string [in] AppName  This parameter can be used to give the name of the application that has failed (this must be the applications systemd unit name)  bool [in] AppRunning  The AppRunning will define whether the application is currently running or not  int32 [in] ErrorCode |
| **SetApplicationMode()**  Public | The method is used by other applications to set the application mode. | int32 [in] ApplicationModeId  This parameter will be based upon the NsmApplicationMode\_e  int32 [out] ErrorCode |
| **SetBootMode()**  Public | The method has been introduced, because the property ‘BootMode’ can only be read by other applications. Nevertheless there are some exceptions where the property should be set by a restricted set of applications which will be handled within this method. | int32 [in] BootMode  The passed value will not be boundary checked to allow for product extensions. The actual value that a client should use will be defined in another common header that is currently not available  int32 [out] ErrorCode  Returned error code for set. |
| **SetNodeState()**  Public | The method is used by other applications to set the NodeState. When this method is called to change the current NodeState a signal will be sent to notify registered consumers of the new state | int32 [in] NodeStateId  The passed value will be based upon the enum NsmNodeState\_e  int32 [out] ErrorCode  The passed value will be based upon the enum NsmErrorStatus\_e |

## Defines

|  |  |  |
| --- | --- | --- |
| **NSM\_SHUTDOWNTYPE\_NORMAL** int  Private  Static Const | Client registered for normal shutdown | *Default:* 1 |
| **NSM\_SHUTDOWN\_TYPE\_FAST** int  Private  Static Const | Client registered for fast shutdown | *Default:* 2 |
| **NSM\_SHUTDOWN\_TYPE\_RUNUP** int  Private | The shutdown type "run up" can not be used for registration. Clients which are registered and have been shut down, will automatically be informed about the "runup", when the shut down is cancelled. | *Default:* 0x80000000 |

### NSM\_\_pau8CoreSessionName\_t

The constant string array defines the names of the sessions which are mandatory within the Platform. The strings defined here are used as first parameter for filtered registration to the signal ‘SessionStateChanged’.

A separate method is provided for adding new sessions dynamically to the list of sessions monitored and distributed by the NSM

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NSM\_\_pau8CoreSessionName\_t |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **DiagnosisSession** string  Public | This session will be used by the Diagnosis SW to indicate the status of authenticated Diagnosis sessions. | *Default:* |
| **HEVACSession** string  Public | This is used by certain OEM's and will also need to be a wake-up reason as it defines we start to a point where we can control the HEVAC | *Default:* |
| **HmiActiveSession** string  Public | This is an open session state that can be used differently for different products.  Traditionally it is expected that this session state will be set by the HMI when they are completely running and all graphics have been rendered on the appropriate layer.  This could for instance be used to determine when to enable the display or to switch from a Splashscreen to the real HMI to ensure that the user does not see the HMI before it is completely ready. | *Default:* |
| **NetworkActiveSession** string  Public | This session will be true/active when there is Network Activity and the Head Unit is in a user perceived on state and therefore is directly reliant on the Network. | *Default:* |
| **NetworkPassiveSession** string  Public | This session will be true/active when there is Network Activity but the Head Unit is in a user perceived off state and therefore is not directly using the Network | *Default:* |
| **PDCSession** string  Public | This is a product decision but it is likely that the PDC active will overrule a Poor and non critical failure. | *Default:* |
| **PermanentModeSession** string  Public | Permanent/Entertainment mode is normally active when the user has started the target via the Power On button and clamp state is not active. This mode would normally allow the target to run for a configurable period of time before an automatic shutdown will occur. | *Default:* |
| **PhoneSession** string  Public | Indicates that a phone call is in progress and would normally delay the NSM from delaying the system shutdown | *Default:* |
| **RVCSession** string  Public | This is a product decision but it is likely that the RVC active will overrule a Poor and non critical failure | *Default:* |
| **SWLSession** string  Public | When SWL is in progress we would need to handle reboot requests and recovery requests differently | *Default:* |
| **MemoryLowSession** string  Public | This session will be used by the Node Resource Manager to indicate that we are in a low memory state. Responsible applications can register for this event and reduce their memory overheads | *Default:* |

### NsmApplicationMode\_e

The enumeration defines the different values for the application mode.

Each possible value will indicate that a different level of functionality is expected to be provided in the current and subsequent lifecycles.

This value will be persistent and will be used to define the "run level" that systemd should enable.

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmApplicationMode\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmApplicationMode\_NotSet**  Public | Initial state | *Default:* |
| **NsmApplicationMode\_Parking**  Public | Value indicating that we should provide “Parking” functionality | *Default:* |
| **NsmApplicationMode\_Factory**  Public | Value indicating that we should provide “Factory” functionality | *Default:* |
| **NsmApplicationMode\_Transport**  Public | Value indicating that we should provide “Transport” functionality | *Default:* |
| **NsmApplicationMode\_Normal**  Public | Default value indicating that we should be fully operational | *Default:* |
| **NsmApplicationMode\_Swl**  Public | Value indicating that we should provide “SWL” functionality | *Default:* |
| **NsmApplicationMode\_Last**  Public | Last value to identify valid boot modes. | *Default:* |

### NsmDataType\_e

This enum defines the different data available within the NSM and will be used by the NSMc when requesting to read data

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmDataType\_AppMode** byte  Public |  | *Default:* |
| **NsmDataType\_NodeState** byte  Public |  | *Default:* |
| **NsmDataType\_RestartReason** byte  Public |  | *Default:* |
| **NsmDataType\_SessionState** byte  Public |  | *Default:* |
| **NsmDataType\_ShutdownReason** byte  Public |  | *Default:* |
| **NsmDataType\_BootMode** byte  Public |  | *Default:* |
| **NsmDataType\_RunningReason** byte  Public |  | *Default:* |

### NsmErrorStatus\_e

The enumeration defines the different error level used as return values

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmErrorStatus\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmErrorStatus\_NotSet** byte  Public | Initial value when error type is not set | *Default:* |
| **NsmErrorStatus\_Ok** byte  Public | Value when no error occurred | *Default:* |
| **NsmErrorStatus\_Error** byte  Public | This value can be used to state that an error occurred handling the request | *Default:* |
| **NsmErrorStatus\_Dbus** byte  Public | Error in D-Bus communication | *Default:* |
| **NsmErrorStatus\_Internal** byte  Public | Internal error (memory alloc. failed, etc.) | *Default:* |
| **NsmErrorStatus\_Parameter** byte  Public | A passed parameter was incorrect | *Default:* |
| **NsmErrorStatus\_WrongSession** byte  Public | The requested session is unknown | *Default:* |
| **NsmErrorStatus\_ResponsePending** byte  Public |  | *Default:* |
| **NsmErrorStatus\_Last** byte  Public | Last error value to identify valid errors | *Default:* |

### NsmRestartReason\_e

The enumeration defines the different restart reasons.

The restart reason will only be updated by the Node State Manager during the system startup phase. The NSM will use the value that it stored persistently in the previous lifecycle when its interface RequestNodeRestart was called.

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmRestartReason\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmRestartReason\_NotSet**  Public | Initial value when reset reason is not set | *Default:* |
| **NsmRestartReason\_ApplicationFailure**  Public | Reset was requested by System Health Monitor | *Default:* |
| **NsmRestartReason\_Diagnosis**  Public | Reset was requested by diagnosis | *Default:* |
| **NsmRestartReason\_Swl**  Public | Reset was requested by SWL application | *Default:* |
| **NsmRestartReason\_User**  Public | Reset was requested by an user application | *Default:* |
| **NsmRestartReason\_Last**  Public | Last value to identify valid reset reasons | *Default:* |

### NsmRunningReason\_e

This is an enumeration to define all the possible WakeupReasons. These will be accessible via the Node State Manager to interested applications.

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmRunningReason\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmRunningReason\_NotSet** byte  Public | Initial value when reason is not set. This value should be set on the rundown of the target | *Default:* |
| **NsmRunningReason\_WakeupCan** byte  Public | Wake up because of CAN activity | *Default:* |
| **NsmRunningReason\_WakeupMediaEject** byte  Public | Wake up because of 'Eject' button | *Default:* |
| **NsmRunningReason\_WakeupMediaInsertion** byte  Public | Wake up because of media insertion | *Default:* |
| **NsmRunningReason\_WakeupHevac** byte  Public | Wake up because of user uses the HEVAC unit in the car. Even if the HEVAC actually causes activity on the CAN bus a different wakeup reason is required as it could result in a different level of functionality being started | *Default:* |
| **NsmRunningReason\_WakeupPhone** byte  Public | Wake up because of a phone call being received. Even if this is passed as a CAN event a different wakeup reason is required as it could result in a different level of functionality being started | *Default:* |
| **NsmRunningReason\_WakeupPowerOnButton** byte  Public | Startup because user presses the "Power ON" button in the car. Even if this is passed as a CAN event a different wakeup reason is required as it could result in a different level of functionality being started | *Default:* |
| **NsmRunningReason\_StartupFstp** byte  Public | System was started due to a first switch to power | *Default:* |
| **NsmRunningReason\_StartupSwitchToPower** byte  Public | System was started due to a switch to power | *Default:* |
| **NsmRunningReason\_RestartSwRequest** byte  Public | System was restarted due to an internal SW Request (i.e. SWL or Diagnosis) | *Default:* |
| **NsmRunningReason\_RestartInternalHealth** byte  Public | System was restarted due to an internal health problem | *Default:* |
| **NsmRunningReason\_RestartExternalHealth** byte  Public | System was restarted due to an external health problem (i.e. external wdog believed node was in failure) | *Default:* |
| **NsmRunningReason\_RestartUnexpected** byte  Public | System was restarted due to either an unexpected kernel restart. This will be the default catch all when no other reason is known | *Default:* |
| **NsmRunningReason\_RestartUser** byte  Public | Target was reset due to user action (i.e user 3 finger press) | *Default:* |
| **NsmRunningReason\_PlatformEnd** byte  Public | Last value (127) to identify where the platform defines end (product will start from here on) | *Default:* |

### NsmSeat\_e

This can be used to identify, where needed, the car seat that is applicable for the data item being referenced

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmSeat\_NotSet**  Public | Initial state | *Default:* |
| **NsmSeat\_Driver**  Public | This is the default setting and refers to the driver seat | *Default:* |
| **NsmSeat\_CoDriver**  Public | This refers to the front passenger seat | *Default:* |
| **NsmSeat\_Rear1**  Public | This refers to 1 of 3 generic rear seat locations | *Default:* |
| **NsmSeat\_Rear2**  Public | This refers to 1 of 3 generic rear seat locations | *Default:* |
| **NsmSeat\_Rear3**  Public | This refers to 1 of 3 generic rear seat locations | *Default:* |
| **NsmSeat\_Last**  Public | Last valid state | *Default:* |

### NsmSessionState\_e

The enumeration defines the currently foreseen session states

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmSessionState\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmSessionState\_Unregistered**  Public | Initial state, equals "not set" | *Default:* |
| **NsmSessionState\_Inactive**  Public | Controlling application has registered session state as being inactive | *Default:* |
| **NsmSessionState\_Active**  Public | Controlling application has registered session state as being active | *Default:* |

### NsmShutdownReason\_e

The enumeration defines the different shutdown reasons.

The Node State Manager will update shutdown reason based on Lifecycle events in the system and will store it persistently for anyone interested in the value in the next lifecycle

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmShutdownReason\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmShutdownReason\_NotSet** byte  Public | Initial value when shutdown reason is not set. | *Default:* |
| **NsmShutdownReason\_Normal** byte  Public | Standard value when a shutdown occurred due to no clamp signal | *Default:* |
| **NsmShutdownReason\_SupplyBad** byte  Public | Shutdown initiated because of bad supply. | *Default:* |
| **NsmShutdownReason\_SupplyPoor** byte  Public | Shutdown initiated because of poor supply. | *Default:* |
| **NsmShutdownReason\_ThermalBad** byte  Public | Shutdown initiated because of bad thermal state. | *Default:* |
| **NsmShutdownReason\_ThermalPoor** byte  Public | Shutdown initiated because of poor thermal state. | *Default:* |
| **NsmShutdownReason\_SwlNotActive** byte  Public | Shutdown initiated after software loading. | *Default:* |
| **NsmShutdownReason\_Last** byte  Public | Last value to identify valid shutdown reasons | *Default:* |

### NsmNodeState\_e

This enumeration defines the different node states. There is only one node state active at the same time. The NodeState system itself and other system components need to know the NodeState to decide whether certain actions can be performed in the current state.

The values shown in this list are those that are mandatory for the Platform but it will be possible for the product to extend the list as needed by rebuilding the Node State Manager

*Connections*

| **Connector** | **Source** | **Target** | **Notes** |
| --- | --- | --- | --- |
| Realization  Source -> Destination | Public  Node State Manager | Public  NsmNodeState\_e |  |

Attributes

| **Attribute** | **Notes** | **Constraints and tags** |
| --- | --- | --- |
| **NsmNodeState\_NotSet**  Public | Initial state when node state is not set | *Default:* |
| **NsmNodeState\_StartUp**  Public | Basic system is starting up | *Default:* |
| **NsmNodeState\_BaseRunning**  Public | Basic system components have been started | *Default:* |
| **NsmNodeState\_LucRunning**  Public | All 'Last user context' components have been started | *Default:* |
| **NsmNodeState\_FullyRunning**  Public | All 'foreground' components have been started | *Default:* |
| **NsmNodeState\_FullyOperational**  Public | All components have been started | *Default:* |
| **NsmNodeState\_ShuttingDown**  Public | The system is shutting down | *Default:* |
| **NsmNodeState\_ShutdownDelay**  Public | Shutdown request active. System will shut down soon | *Default:* |
| **NsmNodeState\_FastShutdown**  Public | Fast shutdown active | *Default:* |
| **NsmNodeState\_DegradedPower**  Public | Node is in degraded power state | *Default:* |
| **NsmNodeState\_Shutdown**  Public | This is the state that the NSM will transition to at the point where it believes it has completed the shutdown process | *Default:* |
| **NsmNodeState\_Last** | Last valid entry to identify valid node states | *Default:* |

# Requirements

## Vehicle Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Alias** | **Originator (Name, Company)** | **Description** | **Additional Info** | **State** | **Assigned To** |
| VH-LICY-001 | Mark Hatle, Wind River | Lifecycle Management must override blocking consumers on shutdown | In shutdown situations, Lifecycle Management must timeout after a defined time duration and proceed with shutdown even if one or more consumers or devices fail to respond to a state change request. Special handling must be configurable for the Software Loading consumer to prevent potential system instabilities. | Released | NSM |
| VH-LICY-004 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The system functionality to store necessary data is depending on the shutdown mode. | Differentiation between normal shutdown (normal persistence) and fast shutdown, e.g., from diagnostic. In a fast shutdown situation we will not do normal persistence.   This will be a product decision on consumers will be called. | Released | NSM |
| VH-LICY-005 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | In diagnostic or flash sessions the lifecycle is controlled by diagnostic. | Note: Incoming messages over bus will be ignored, e.g. phone call, button pressed. Node internal exceptions from thermal management should not be ignored to save the ECU. | Released | NSM |
| VH-LICY-006 | [Fabien Hernandez](https://collab.genivi.org/wiki/display/~fabien.hernandez), PSA | Before a managed reset/reboot, the system provides a degraded mode in order to maintain main user services. | If diagnosis starts a reset/reboot procedure, system can give a little time to the user to finish actions. This time will be a product configurable value. | Released | NSM |
| VH-LICY-007 | [Fabien Hernandez](https://collab.genivi.org/wiki/display/~fabien.hernandez), PSA | The system has the ability to delay the shutdown depending on application state. | Rational:  Ongoing phone call and the driver asks for ignition off (key) | Released | NSM |
| VH-LICY-008 | [Fabien Hernandez](https://collab.genivi.org/wiki/display/~fabien.hernandez), PSA | The system has the ability to manage a timeout to shutdown depending on application state  . | Rational:  When a call is still ongoing a defined period of time (x minutes) after the driver asked for ignition off (key) the call is switch to the phone and the system is shut down. | Released | NSM |
| VH-LICY-009 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The system has to follow the global car operation modes. | Ignition on -> system on. Low battery -> system shutdown. | Released | NSM |
| VH-LICY-010 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | There must be a normal operation mode. | Full functionality for all user interactions with ignition on. | Released | NSM |
| VH-LICY-015 | [David Yates](https://collab.genivi.org/wiki/display/~david.yates), Continental | The system can be shutdown by a specific Clamp State notification |  | Released | NSM |
| VH-LICY-016 | [David Yates](https://collab.genivi.org/wiki/display/~david.yates), Continental | The system can be shutdown by the ECU after a product defined timeout when there is no active Clamp State |  | Released | NSM |
| VH-LICY-017 | [David Yates](https://collab.genivi.org/wiki/display/~david.yates), Continental | The system can be instructed to shutdown by an external tester | Specific diagnostic authentication and protocols will need to be handled before this Diagnosis job is accepted by the ECU | Released | NSM |
| VH-LICY-018 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | There must be a parking mode. | Reduced functionality (i.e. internal HU functionality only) during ignition off but key is present | Released | Lifecycle |
| VH-LICY-019 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | There must be an operation mode for transporting the cars. | No function for the customer (e.g. navigation, entertainment etc.) except safety functions (e.g. PDC) during key presence and ignition on. Different start-up and different functionality to building the cars possible. Only used in factories and service. | Released | Lifecycle |
| VH-LICY-020 | Christian Muck, BMW | There must be an operation mode for building the cars in factories. | Reduced functionality (i.e. Diagnosis, PDC, etc.) during key presence and ignition on. Different start-up and different functionality to transporting the cars possible. Only used in factories. | Released | Lifecycle |
| VH-LICY-021 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | There must be an operation mode for flashing devices. | This mode will be accessible through a protected interface that can only be accessed via authorized applications. | Released | Lifecycle |
| VH-LICY-025 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The ECU must handle an immediate power off. | When this occurs it is required that some critical persistency data is stored using the remaining battery charge | Released | NSM |
| VH-LICY-027 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | To provide Log&Trace data for problem analysis the DLT must be supported. | Each application/component should provide version information and build details in DLT at start-up time. | Released | Lifecycle |
| VH-LICY-028 | [Fabien Hernandez](https://collab.genivi.org/wiki/display/~fabien.hernandez), PSA | Accordingly to its state, the system could generate a reset/reboot |  | Released | NSM |
|  | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The system must support a real time clock and an uptime clock. |  | Open |  |
| VH-LICY-029 | [David Yates](https://collab.genivi.org/wiki/display/~david.yates), Continental | Lifecycle Management must be able to perform a fast system shutdown when requested | During a fast shutdown it is accepted that persistence data from that lifecycle will not be stored during the shutdown | Released | NSM |

## SW Platform Requirements

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Alias** | **Derived from** | **Originator (Name, Company)** | **Description** | **Additional Info** | **Prio** | **State** | **Assigned To** |
| SW-LICY-001 | OEM | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The Node State Manager of the Automotive Controller must provide mapping between OEM-specific system states and GENIVI States. | The Lifecycle Manager API will distribute the OEM and GENIVI states as part of the interface. | P1 | Released | NSM |
| SW-LICY-002 | Internal | Mark Hatle, Wind River | The Node State Manager must send a notification to all registered consumers after a state change has completed |  | P1 | Released | NSM |
| SW-LICY-003 |  | Mark Hatle, Wind Rive | The definition and structure of Lifecycle API'smust be hardware independent |  | P1 | Released | Lifecycle |
| SW-LICY-004 |  | Mark Hatle, Wind River | Lifecycle Management API must provide functionality to get a list of all registered consumers |  | P1 | Released | NSM |
| SW-LICY-005 | VH-LICY-002 | Mark Hatle, Wind River | The Node State Manager must provide an API to get the current power state |  | P1 | Released | NSM |
| SW-LICY-006 | VH-LICY-002 | Mark Hatle, Wind River | The Node State Manager must provide an API to switch the power state |  | P1 | Released | NSM |
| SW-LICY-007 |  | Mark Hatle, Wind River | The Node State Manager must be accessible to multiple consumers |  | P1 | Released | NSM |
| SW-LICY-008 |  | Mark Hatle, Wind River | The Node State Manager must be able to distribute the node state to all consumers |  | P1 | Released | NSM |
| SW-LICY-009 | VH-LICY-027 | Mark Hatle, Wind River | All Lifecycle components must provide monitoring capabilities with DLT for the Test Framework |  | P1 | Released | Lifecycle |
| SW-LICY-011 | VH-LICY-006  VH-LICY-007  VH-LICY-008  VH-LICY-028 | Mark Hatle, Wind River | The Node State Manager must have the ability to trigger immediate or deferred system reboots |  | P1 | Released | NSM |
| SW-LICY-067 | VH-LICY-026 | [Christian Muck](https://collab.genivi.org/wiki/display/~christian.muck), BMW | The Node State Manager must be able to trigger the cancelling of an ongoing shutdown |  | P1 | Released | NSM |
| SW-LICY-035 | VH-LICY-022 | David Yates, Continental | The Node State Manager must provide an interface with which the Thermal Manager can write the Platform Thermal State |  | P1 | Released | NSM |
| SW-LICY-036 | VH-LICY-001 | David Yates, Continental | The Node State Manager must provide an API mechanism for a consumer to delay the system shutdown |  | P1 | Released | NSM |
| SW-LICY-037 | VH-LICY-001 | David Yates, Continental | The Node State Manager must provide a mechanism to configure how long a consumer can block the system shutdown |  | P1 | Released | NSM |
| SW-LICY-069 | VH-LICY-001 | David Yates, Continental | The Node State Manager must enforce that no consumer can block the system shutdown for more than the product configured time |  | P1 | Released | NSM |
| SW-LICY-040 | VH-LICY-023  VH-LICY-003 | David Yates, Continental | The Node State Manager must provide a public interface whereby the wakeup reason can be read |  | P1 | Released | NSM |
| SW-LICY-041 | VH-LICY-004 | David Yates, Continental | The Node State Manager must provide a configurable dynamic shutdown procedure based on the shutdown reason and node session state | Based on the target configuration consumers might not be called before initiating a system shutdown | P1 | Released | NSM |
| SW-LICY-043 | VH-LICY-004  VH-LICY-017 | David Yates, Continental | The Node State Manager must provide a mechanism for fast shutdown whereby only a small subset of components are informed about the shutdown procedure |  | P1 | Released | NSM |
| SW-LICY-044 | VH-LICY-017 | David Yates, Continental | The Node State Manager must provide an interface whereby applications in the system can request a system shutdown | For instance this would be used by SWL, Diagnostics, System Health Monitor etc. | P1 | Released | NSM |
| SW-LICY-045 | VH-LICY-016 | David Yates, Continental | The Node State Manager must provide an interface whereby applications in the system can request a system restart | For instance this would be used by SWL, Diagnostics, System Health Monitor etc. | P1 | Released | NSM |
| SW-LICY-046 | VH-LICY-016 | David Yates, Continental | The Node State Manager must provide an interface that can be used by an external component to notify about changes to the Clamp State |  | P1 | Released | NSM |
| SW-LICY-047 | VH-LICY-012  VH-LICY-013  VH-LICY-014  VH-LICY-015 | David Yates, Continental | The Node State Manager must provide an interface whereby the wake-up reason can be provided by an external component |  | P1 | Released | NSM |
| SW-LICY-049 | VH-LICY-011  VH-LICY-016 | David Yates, Continental | The Node State Manager must support a mode whereby the Head Unit will remain operational for a configurable period of time regardless of clamp state. | This will be activated by the user pressing the Power On button on the Unit | P1 | Released | NSM |
| SW-LICY-054 | VH-LICY-018  VH-LICY-019   VH-LICY-020  VH-LICY-021 | David Yates, Continental | The Node State Manager must provide an interface whereby the application mode can be updated | The following modes must be supported :  Parking Mode  Transport mode  Factory/Production mode  SWL Mode | P1 | Released | NSM |
| SW-LICY-057 | VH-LICY-005 | David Yates, Continental | The Lifecycle Management must support different levels of functionality dependent on active node sessions | NOTE: the active node sessions agreed in the requirement actually refers to the "Node Application Mode", i.e. Transport, SWL etc. | P1 | Released | Lifecycle |
| SW-LICY-058 |  | David Yates, Continental | The Node State Manager must provide an interface to allow applications to change the Node Boot Mode |  | P1 | Released | NSM |
| SW-LICY-059 | VH-LICY-012 | David Yates, Continental | The Node State Manager must store the lifecycle wake-up reason |  | P1 | Released | NSM |
| SW-LICY-060 | VH-LICY-006 | David Yates, Continental | The Node State Manager must provide a product configurable shutdown delay value | This value will be used to delay all requested system shutdown/resets | P1 | Released | NSM |
| SW-LICY-064 | VH-LICY-013  VH-LICY-014 | David Yates, Continental | The Bootloader must provide an API whereby it can receive the Wake-up reason from the Automotive Controller |  | P1 | Released | NSM |
| SW-LICY-068 | VH-LICY-013 | David Yates, Continental | When the Node State Manager receives an indication of a media removal and the wake-up reason was "Media Eject" then it must initiate a system shutdown |  | P1 | Released | NSM |