FNV2 Analytics System Design HLD 1.1

Document Status	IN-PROGRESS
Delivery Target	FNV2
Tracking JIRA	*https://jiramg.ford.com/browse/ANLYT-575*
Completion	100%
Document Version	1.1
Owner	Niemi, Jonathan (J.)
Reviewer(s)	
Approver(s)	Martin, Daryl (D.)
Approve Date	

1. Glossary

1.1. Acronyms

Acronym	Description
FNV-ATL	Fully Network Vehicle Analytics
VDR	Vehicle Data Recorder
ECG	Enhanced Central Gateway

ECU	Electronic Control Unit, i.e. APIM (SYNC), ECG, TCU
Fast ECU	An Electronic Control Unit connected to ethernet
SOA	Service Oriented Architecture
TCU	Telematics Control Unit
GEID	Global Event Id - a randomly generated 128 bit id that forms the unique key of this event.
VIM	Vehicle Information Manager
TGW	Things Gone Wrong
PII	Personally Identifiable Information

1.2. Terminology

In addition to the above acronyms this document makes use of some terminology that a basic understand of will help readers follow along.

Term	Description
logs	Logs are defined as the artifacts which are collected by our agents when a diagnostic event is reported. Logs are predefined on the platform, though they are not restricted solely to log files. The output of a shell script or command (i.e. 'pidin') is also a log in this context.
log sets	A log set is simply a predefined grouping of logs.

1.3. Major changes since 1.0

- Removed SYNC only deployment option
- Analytics Events are now sent first to the local DiagnosticsSlave which is responsible for store/forward to the AnalyticsAgent on the master ECU.
- SOA topics greatly reduced after development noticed amount of boilerpate required for each

2. Summary

The intention of FNV2-ATL is to provide a means with which clients (i.e. applications/processes running on **Fast** ECUs) can:

- Report generic key/value type events which may be delivered with relatively high frequency but of small size ~100 bytes each. These are referred to as Analytics Events.
- Report critical application failures, specifying a set of logs/data points to be collected and delivered to the Ford servers for analysis. These are Diagnostics Events.

At a high level, the platform is responsible for:

- Providing a library which clients can use to send events to our services/agents
- Collecting logs **at time of failure**, when a diagnostics event is received
- Storing undelivered logs and analytics events until they are either:
    - Delivered
    - Pruned
- Decorating events with both ECU specific and vehicle wide data
- Securely and efficiently uploading events to the FNV Analytics cloud
- Storing metadata related to diagnostics event, prioritizing delivery in a fair but prioritized way.

2.1. Purpose

The purpose of the FNV2-ATL program is to:

- Improve software quality earlier in the development process
- Drive down TGWs
- Provide insight into how our customers are using their vehicles
- Facilitate data-driven decision making based on real world usage data generated by our customers
- Reduce licensing costs by brining real usage / failure / other data to the negotiating table.

2.2. Scope

The FNV2 release will build upon the components that were successfully developed and deployed as a part of the SYNC 3.2v2 program. The following components will be carried forward with minimal changes (porting to FNV2 specific frameworks etc)

- **FNVAnalyticsLib** - A shared library clients use to interact with our agents. Net-new work is required to introduce a more user-friendly c++ API to complement the existing C API.
- **AnayticsAgent** - service responsible for managing all analytics events, decorating, persisting, uploading etc.

The Diagnostics Agent will require a fair amount of rework in the FNV2 timeframe. It will be split into two logical entities, a Diagnostics Master and a Diagnostics Slave. Logically this makes sense, the Slaves will run on each ECU performing similar, but in some cases ECU specific functions. The Master on the other hand is more of a supervisor, responsible for cooridnating the Slaves.

- **DiagnosticsSlave** - Responsible for collecting logs on individual ECUs, transmitting to cloud
- **DiagnosticsMaster** - responsible for coordinating slave event transmission, tracking event status, facilitating multi ECU events.

Some of the major architectural changes required include:

- Integration with ECG-CM (ConnectionManager)
- Migrating each componet to be fully POSIX compliant such that it can run on any POSIX compliant OS
- Use of common FNV2 libraries such as FNV-Log and other libraries that are part of FNV::Base (Timers, Threads etc.)
- Leverage platform level solutions to boilerplate type code, such as ECG-IPC-ALM for IPC.

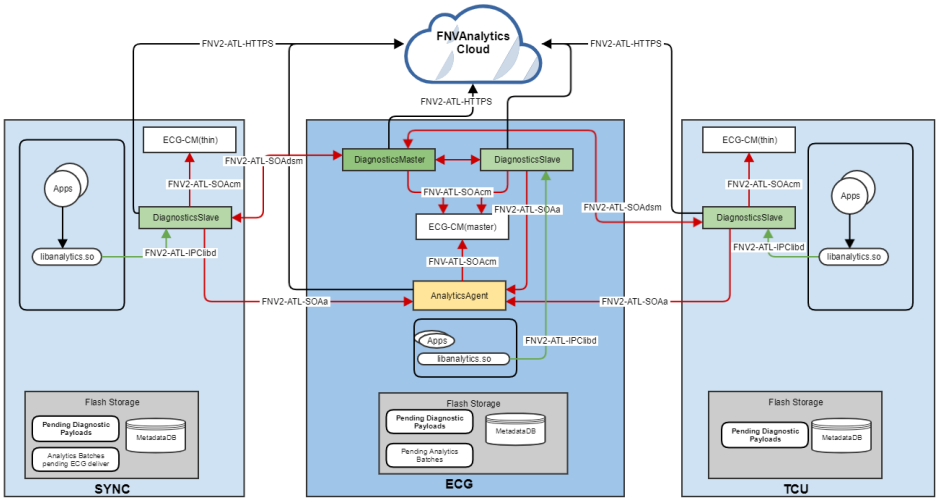
3. Requirements

Refer to https://www.eesewiki.ford.com/display/analytics/FNV+Analytics+Requirements

4. High Level Design

4.1. Block Diagrams

The FNV2 Analytics platform is comprised of a number of components, each responsible for a specific facet of the overall solution.



4.1.1. ECG + TCU + SYNC Deployment

This diagram summarizes the overall design of the FNV2-ATL solution in the most common scenario where all three in-house Fast ECUs are present in the vehicle. In this case the ECG will act as the Master ECU and host both the AnalyticsAgent, DiagnosticsMaster as well as a Diagnostics Slave.

4.2. Components

4.2.1. AnalyticsAgent

The Analytics Agent is a process that will run on the master ECU.

The Analytics Agent is responsible for receiving, processing and storing analytics events generated by all ECUs. It is also responsible for transmitting events to the Analytics Cloud when ConnectionManager informs us we can transmit. We chose a centralized model for the Analytics Agent because it deals with frequently published smaller events and is, for the most part, ECU agnostic. Any ECU specific information needed by analytics will be published to it by a DiagnosticsSlave. More information on this process is available in the Dimensions section of this document.

4.2.2. Diagnostics

As mentioned, Diagnostics is broken up into two compoents, the Diagnostics Master, and Diagnostics Slave.

4.2.2.1. DiagnosticsSlave

The DiagnosticsSlave is responsible for performing ECU specific tasks that cannot be accomplished without some degree of platform specific implementation. It is responsible for collecting logs from the ECU at the time of failure and storing them to flash. It is also responsible for collecting and maintaining a set of ECU specific metadata that will be used by both diagnostics and analytics to decorate events prior to delivering them to our infrastructure. Each Slave will be responsible for uploading the diagnostic events they generate, however, they will only do so when told to by the DiagnosticsMaster.

4.2.2.2. DiagnosticsMaster

The DiagnosticsMaster resides only on a single ECU (the same as the AnalyticsAgent). It's primary responsibility is for arbitrating the upload of events by each ECU. It is also responsible for maintaining a master list of all diagnostic events that have been generated by each ECU and their current delivery status. The DiagnosticsMaster will obtain an OAuth token from our cloud on behalf of each DiagnosticsSlave and send it to them when it is their turn to send any event. Slaves will not be able to independently obtain OAuth tokens as the API key required to request one is known only to the Master.

4.2.3. FNAnalytlicsLib

The FNV Analytics library is the main interface with which clients will construct and send analytics and diagnostics events. The library will be deployed as as shared library on each ECU.

The API will provide both a c++ Builder based model for creating events as well as a C based API.

4.2.4. Diagnostic Event Metadata Storage

When a diagnostic event has been generated by a client, metadata relating to that event will be generated and persisted. Each ECU will hold a local copy of the metadata for each event it has generated. The Diagnostics Master, which is responsible for deciding which events can be sent and when, will contain a record of the status of all events, from every ECU.

The metadata shall be stored in a SQLLite database, the following table describes the information we will store about each event. Because it will be possible for the master and slave databases to get out of sync a synchronization method will need to exist to align them. This is discussed in a later section of this document.

Column	Description	"Type" - SQLite is typeless	Size (bytes)
geid	The GEID of the event	TEXT	32
created	When the event was created	INTEGER	6
logs_collected	When the logs were collected, or empty if not yet collected	INTEGER	6
log_location	Where the logs are located	TEXT	~64
uploaded	When the event was uploaded to the server	INTEGER	6
parent_geid	The GEID of the parent of this event	TEXT	32
ecu_id	The ECU where this event originated	TEXT	~8
parent_ecu_id	The ECU where the parent event originated	TEXT	~8
creator_id	The creator (i.e. process name) of this event	TEXT	32
event_type	The type of event	INTEGER	2
event_severity	The severity of this event	INTEGER	1

failures	The number of times this event has failed to upload	INTEGER	1
throttle_time	When the first event with this fingerprint was last throttled	INTEGER	6
status	0 - pending ack from master 1 - pending delivery 2- rejected by throttle 3 - rejected by throttle, no metadata sent 4 - rejected by fingerprint 5 - rejected by failure streak 6 - event has been pruned	INTEGER	1
fingerprint	The event's fingerprint	TEXT	16

4.2.4.1. Storage Requirements

Each row in the metastore DB could require up to ~200 bytes. The metadata DB will be purged on OTA update and Master Reset. The number of events that occur is not expected to exceed more than a few per day on each ECU. If we assume even 20 events total per day, this will amount to aproximately 1.5MB per year. For this reason we will likely want to prune the database once we have reached a certain number of events. This value can be configurable via OTA Configuration per ECU.

4.2.4.2. Pruning Algorithm

When the Slave and/or Master starts up, it should compare the current number of records in its database against the maximum value. If within 5% of the maximum, 20% of the rows should be dropped according to the following criteria:

a. Prefer events that have been sent
b. Prefer events by increasing severity (DEBUG->UKNOWN->WARNING->SERIOUS->CRITICALCATASTROPHIC)
c. Prefer old events

4.2.5. Master-Slave Diagnostics Architecture

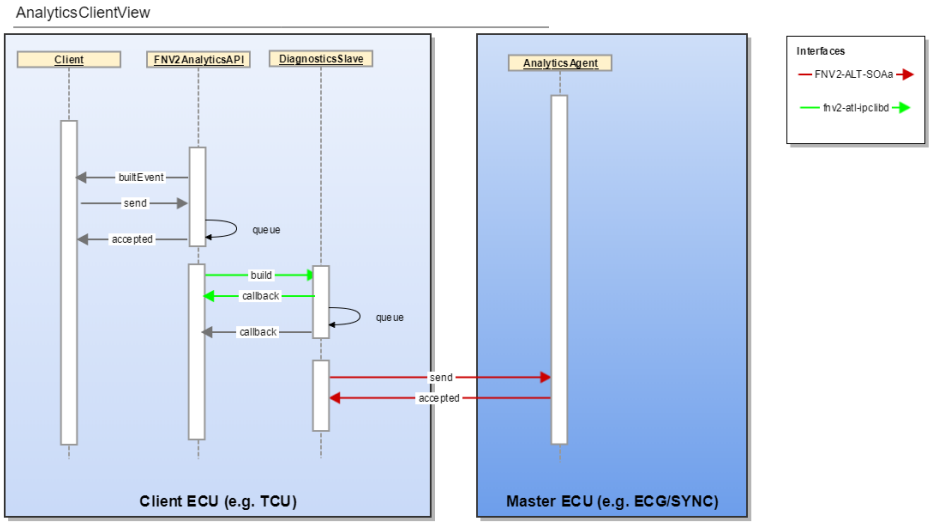
A master-slave architecture was chosen because it allows us to spearate some of the system wide responsibilities from those that are ECU specific. For example, the master is responsible for scheduling, broadcasting system changes (consent etc), and retrieving the OAuth token for each ECU.

The assumption is that we will also operate on 2nd party ECUs (i.e. cluster, HUD) and will require the ability to tightly control them. A master agent will give us this capability by controlling the OAuth token (the ECU cannot send without it), and the scheduling algorithm.

4.3. Sequence Diagrams

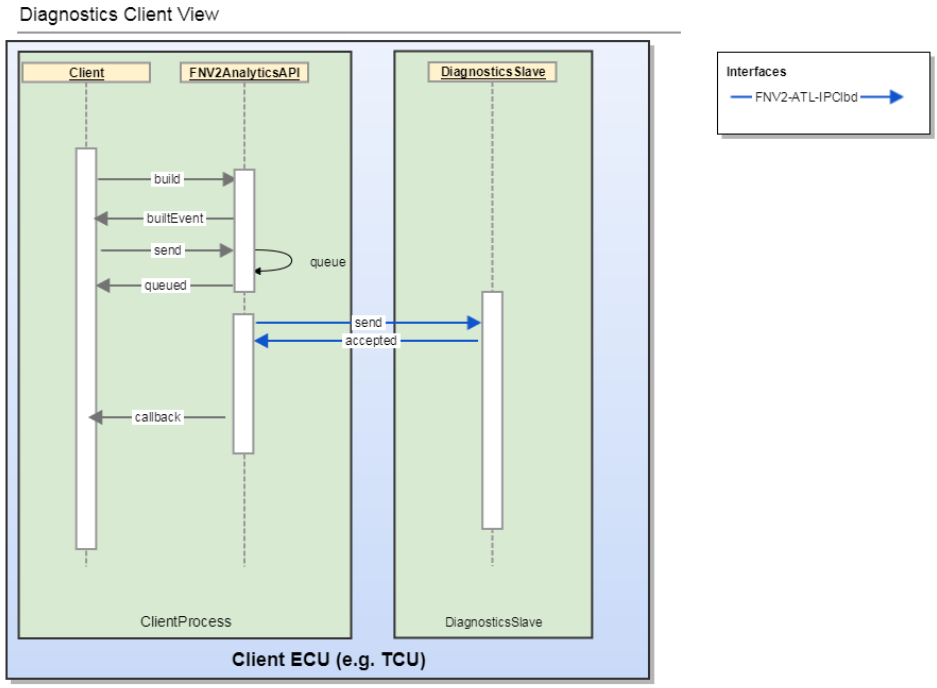
4.3.1. Client creates and sends Analytics event ⭐

This diagram outlines, from the client's point of view, the process of building and sending an event using our library. The event is first sent over IPC to the DiagnosticSlave, which is responsible for queueing the event on the local ECU and sending it to the master ECU. If the master is not online, the slave will persist the event to disk until the master is available.



4.3.2. Client creates and sends Diagnostics Event

The following diagram outlines, from the client point of view, the process of building and sending a Diagnostics Event. Similarily to above, the client's job is done once the event has reached the Diagnostics Slave.

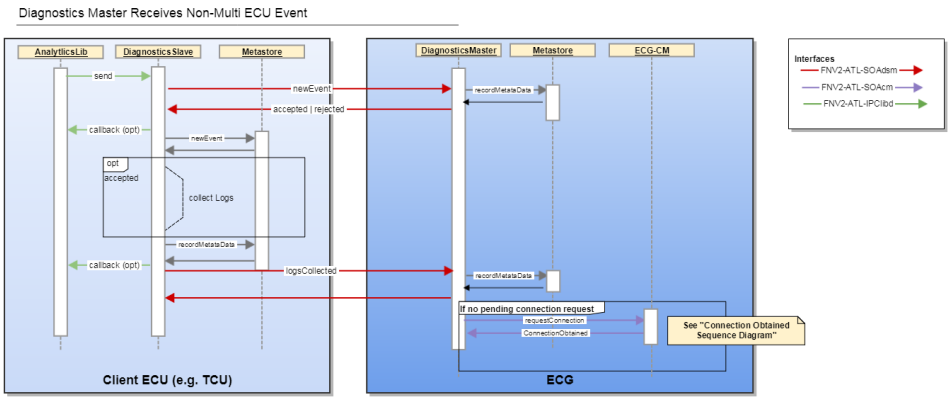


4.3.3. Diagnostics Master Notified of event

This sequence diagram carries on from the previous one, whereby a client has created a new DiagnosticsEvent and sent it to the DiagnosticsSlave.

You'll notice multiple messages being sent from the Diagnostics Slave to the DiagnosticsMaster, and corresponding callbacks to the Analytics library. This is done to track the state of the event not only in the DiagnosticsSlave, but the Master as well. This will ensure that the master always has an up-to-date view of every diagnostics event. It will be able to quickly identify orphaned events (events that were created, but logs never collected) and potentailly fire off another diagnostics event to root-cause.

When events are created through our API clients can optionaliy register a callback in order to track the event's state as it changes. For example, it may need to clean up any artifacts that were generated as part of the log capture once it knows that the logs have been collected.



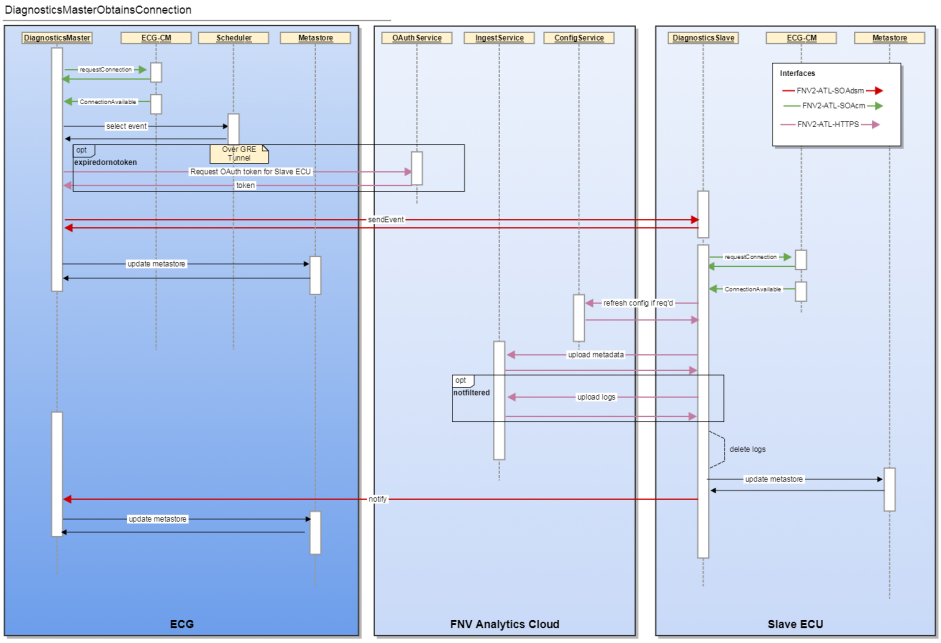
If the above scenario, if the Diagnostics Master cannot be reached (i.e. the Master ECU is in a low power state) then the Slave will process the event as usual and will later be required to bring the master database up-to-date when it becomes available. It is responsible for collecting logs unless the DiagnosticsMaster tells it not to (if it cannot reach the master, it must collect logs).

4.3.4. Connection Obtained

The DiagnosticsMaster and AnalyticsAgent will request a connection through ECG-CM whenever there are events to send. Once it receives a connection it will kick of a series of events.

a. DiagnosticsMaster selects an event to send based on scheduling algorithm
b. DiagnsticsMaster requests an OAuth token for that ECU from our OAuth service
c. DiagnosticMaster notifies the appropriate slave to send the event
d. Slave requests a connection through ECG-CM,**if it does not already have one**.
e. Slave checks for updated config, if there is an updated config it must parse and store it
f. Slave uploads metadata for event
g. Slave uploads logs for event, if required
h. Slave updates local metastore
i. Slave notifies master that event is uploaded
j. Master updates master metastore

This process will repeated for every event that hasn't been transmitted for as long as the connection and power are available.



4.4. Important Relationships

4.4.1. Diagnostic Event Parent-Child Relationship

It is possible to assign a child-parent relationship between diagnostics events. Often clients find it useful to add additional logs to a previously created event. By "appending" an event, and its logs, to an existing event, a link is formed between the two with the original event being the "parent" and the new event the "child". Each event will have a distinct geid, though when viewing the event though the web portal all of the children's logs will be included with the parent.

- A parent can have multiple children
- A parent cannot have grandchildren (no children of children)

4.4.2. Diagnostic and Analytics Event

Whenever a diagnostic event is sent to the Diagnostics Slave through our library, the library will also create a corresponding analytics event. The events are NOT linked due to PII concerns, however, if diagnostics is disabled or consent not granted, the analytics event will still be processed. The analytics library is responsible for creating the corresponding analytics event and sending it to the analytics agent.

4.5. Master Reset Handling

On each ECU the Diagnostics Slave is responsible for subscribing to the relevent Master Reset notification. Upon receipt the following actions will be taken:

- Purge all diagnostic log data stored on the ECU
- Purge all stored metadata and persisted state
- Default Configuration is NOT deleted
- VehicleId is NOT deleted
- Purge all pending analytics events stored by Slave (**new in 1.1**)

Additionally, on the the master ECU, the Analytics Agent will also subscribe to the master reset notification. Upon receipt it will take the following actions:

- Purge all stored analytics events
- Purge all persisted state
- Default Configuration is NOT deleted

4.6. Sleep Support

The primary transport used by Analytics will be Wi-Fi. In many cases, this will only be available at the user's house. In order to ensure each ECU has the opportunity to upload events, in particular those with large log files attached, the Diagnostics Master will need to tap into each ECU's power management capabilities in order to keep the master ECU awake for some amount of time after ignition off. It will use this time to determine if a connection becomes available after ignition off.

In addition to this, the Diagnostics Slave on each ECU must also listen for the ignition off Broadcast, and, upon receipt, vote to keep their ECU awake **if they have pending events**. This ensures that if the master is able to obtain a Wi-Fi connection and instructs a Diagonstics Slave to send an event, the Slave ECU is still awake. Should the Master fail to obtain a connection within a minute, it will Broadcast a "NO_CONNECTION" message to all ECUs at which point they can relinquish their vote.

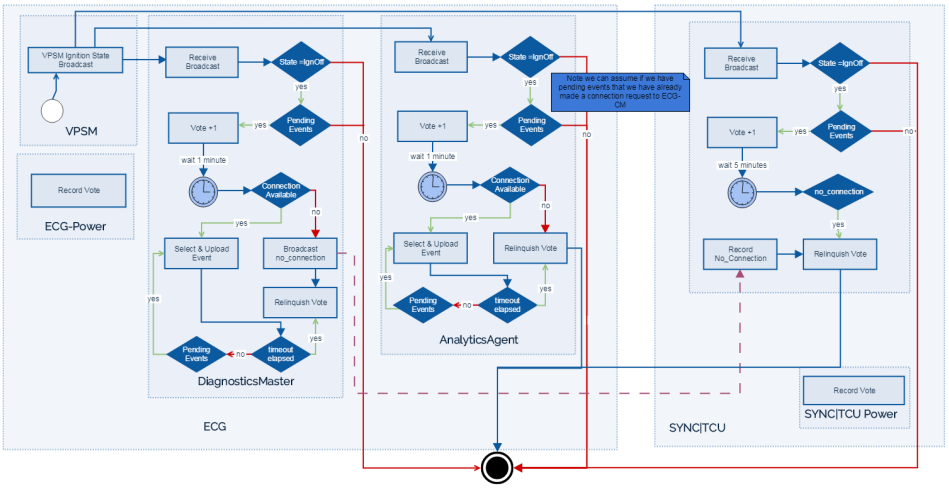
Details of the SYNC Power Management strategy are not yet finalized. This design, in regards to SYNC, is subject to change.

Should we not obtain a Wi-Fi connection within a suitable time period, ~1 minute, we will relinquish our vote, else we will proceed to upload.

	ECG Vote	SYNC Vote	TCU Vote
SYNC Undelivered Diagnostics Events	✓	✓	✗
ECG Undelivered Diagnostics Events	✓	✗	✗
TCU Undelivered Diagnostics Events	✓	✗	✓
Other Fast-ECU Undelivered Diagnostics Events	✓	✗	✗
Undelivered Analytics Events	✓	✗	✗

We will never vote if there are no events to send, which would include the case where the system is disabled due to lack of consent or other means.

All agents will subscribe to Ignition State change broadcasts through FNV2-SOAP-VPSM



4.7. Required System Resources

CPU Load - CPU Load is intended to be minimal. Intention is to not impact system performance. Our threads will run at normal priority. It will be an option to lower the priority of the thread performing log collection, if required.

Disk Usage - Each ECU is responsible for storing the data for the diagnostics events it generates. The amount of storage required is configurable per ECU. As this threshold is encroached upon old events will be discared in order to make room for newew events. See Pruning Strategy Section below. The Master ECU (ECG when available, else SYNC) will host the AnalyticsAgent which is responsble for storing all of the analytics data generated from each ECU. Since Analytics events are small and compress well this does not require a great deal of flash storage. The amount of disk space reserved for analytics events is also configurable.

Memory Usage - Memory Usage is expected to be minimal. It is possible that some logs may be more resource intensive to collect than others. As these are identified they will be documented as such so that clients are aware before deciding to include them in their payloads.

4.8. Threading Model

All three components follow a similar threading model in that they will have a main thread, a small thread pool to manage long running operations and a separate set of Messaging related threads to handle SOA and IPC.

4.8.1. Analytics Agent

Main Thread

- The main thread is responsible for application startup and shutdown, creating other required threads and the main event loop.

Worker Thread Pool

- A small thread pool is responsible for spawning threads to perform long running operations like File I/O and HTTPS upload . Past experience and anticpated usage has indicated that a single thread is more than capable of managing these tasks. A task queue is used to manage these operations.

Messaging Threads

- A SOA Messaging thread is used to receive messages from clients (both broadcasts and "requests"). In most cases this will result in an operation being queued to be handled by the worker thread.

4.8.2. Diagnostics Slave

Main Thread

- The *main thread* is responsble for application startup and shutdown, initializing other required threads and the main event loop.

Worker Threads

- Long running operations will be queued and handled by a thread from a small *thread pool.*

Messaging Threads

- *SOA Messaging Thread* is responsible for receiving messages from the Diagnostics Master and ConnectionManager
- An I*PC Thread* is responsible for receiving and reply to messages from the Analytics library.

4.8.3. Diagnostics Master

Main Thread

- The main thread is responsble for application startup and shutdown, initializing other required threads and the main event loop.

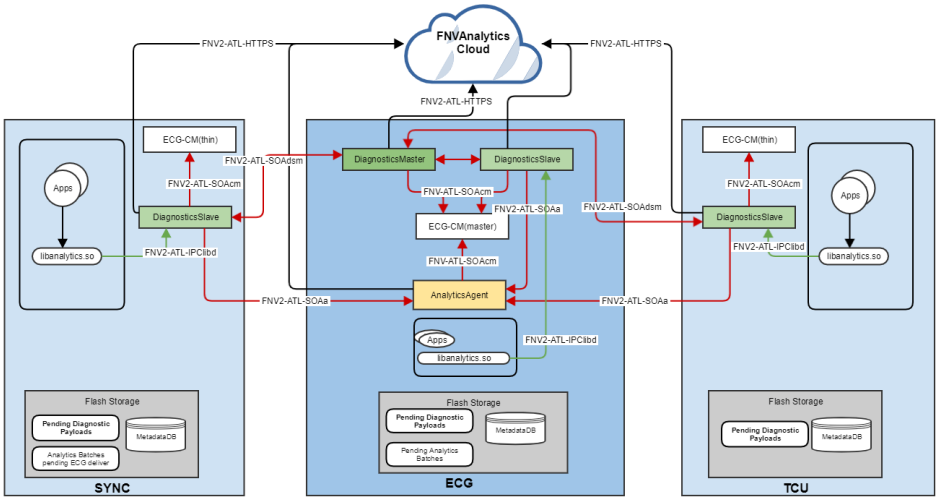
Worker Threads

- Long running operations will be queued and handled by a thread from a small thread pool.

Messaging Threads

- The SOA Messaging Thread is responsible for receiving broadcasts from the Diagonstics Slaves

5. Interfaces



5.1. Cloud

5.1.1. FNV2-ATL-HTTPS

Interface Name	FNV2-ATL-HTTPS
Description	Both of the Diagnostics Agents as well as the AnalyticsAgent will connect "directly" (more accurately via GRE tunnel provided by ECG_CM) to the SYNC Analytics cloud in order to download configuration and upload events.

Actors	**Diagnostics Slave** • Requests a connection through ConnectionManager when instructed to do so by DiagnosticsMaster • Uploads log files for the ECU it is running • Downloads configuration **Diagnostics Master** • Authenticate with SYNC Analytics cloud and retrieve an OAuth token for each ECU that has logs to send • Uploads log files for the master ECU • Downloads configuration **Analytics Agent** • Uploads Analytics events from all ECUs to the cloud • Downloads configuration from the cloud **FNV Analytics Cloud** • Authenticates analytics components • Consumes Diagnostics events / log data • Consumes Analytics events • Distributes configuration to Slaves/Master/AnalyticsAgent
Interface Type	Compressed JSON over HTTPS

5.2. Interprocess / Intermodule Communication

5.2.1. FNV2-ATL-SOAa

Interface Name	FNV2-ATL-SOAa
Description	The client library will speak directly to the Analytics Agent on the master ECU.
Messages	

Topic	Message	Type	Publisher	Subscriber	Type	Payload Contents	Description
/service/request/fnva/<ecuname>/analytics	NEW_EVENT_REQ	RPC	DiagnosticsSlave	AnalyticsAgent	RPC	Contents of event (likely to be JSON)	Events are sent atomically from the slave to the Analytics Agent
'/service/request/fnva/<ecuname>/analytics	NEW_EVENT_RSP	RPC	AnalyticsAgent	DiagnosticsSlave	RPC	Result of Send	Response code sent back to slave after receiving event
/service/data/fnva/status	AGENT_STATUS	Broadcast	AnalyticsAgent	DiagnosticsSlave	Broadcast	analytics agent available	Broadcast when the agent comes online
/service/data/fnva/status	ANALYTICS_CONSENT	Broadcast	AnalyticsAgent	AnalyticsLib DiagnosticsSlave	Broadcast	analyticsconsent:enhanced:enabled:disabled	Broadcast when analytics consent status changes

Interface Type	FNV2-SOA

5.2.2. FNV2-ATL-IPClibd

Interface Name	FNV2-ATL-IPClibd
Description	The Diagnostics library will communicate with the Diagnostics Agents (both Slave and Master, depending on the ECU) over a IPC channel leverage FNV2-IPC-ALM

Actors	**FNVAnalyticsLib** • Sends events to DiagnosticsSlave **DiagnosticsSlave** • Receives events from clients and processes them
Interface Type	FNV2-IPC-ALM

5.2.3. FNV2-ATL-SOAdsm

Interface Name	FNV2-ATL-SOAdsm
Description	This is the SOA interface between the DiagnosticsMaster and DiagnosticsSlave instances.
Messages	Topic

Topic	Type	Publisher	Subscriber	Payload Contents (Gist)	Description
/service/request/fnva/<ecuname>/diagnostics	RPC	Diagnostics Slave	Diagnostics Master	Event metadata	Published by Slave to notify Master of new event
/service/response/fnva/<ecuname>/diagnostics	RPC	Diagnostics Master	Diagnostics Slave	Result of Event send	Published by Master in response to receiving a new event from Slave
/service/request/fnva/<ecuname>/diagnostics	RPC	Diagnostics Master	Diagnostics Slave	GEID of event to send OAuth token	Published by Master when it wants the Slave to upload an event
/service/response/fnva/<ecuname>/diagnostics	RPC	Diagnostics Slave	Diagnostics Master	Result of upload request	Published by Slave in response to Master's request to upload an event
/service/request/fnva/<ecuname>/diagnosticss	RPC	Diagonstics Master	Diagnostics Slave	Event metadata	Published by Master when it receives a multi ECU event from a Slave
/serivce/response/fnva/<ecuname>/diagnostics	RPC	Diagnostics Slave	Diagnostics Master	Result of multi ECU event request	Published by Slave in response to receiving a new Multi ECU event
/service/request/fnva/<ecuname>/diagnosticss	RPC - No response	Diagonstics Slave	Diagonstics Master	GEID of pruned event	Published by Slaves when they prune an event
/service/data/fnva/status	Broadcast	Diagnostics Master	Diagnostics Slaves	diagnosticsconsent • disabled by user • enabled by user • disabled by region • enabled by region	Published by Master when consent status changes
/serivce/data/fnva/status	Broadcast	Diagnostics Master	Diagnostics Slaves	diagnosticsmaster: online|offline ecuname:<ecuname>	Published when the Diagnostics Master's online status changes
/serivce/data/fnva/status	Broadcast	DiagnosticsMaster	Diagnostics Slaves	ignition_off_no_connection	Published by Master 5 minutes after ignition off indicating it did not obtain a network connection

Interface Type	FNV2-SOA

5.3. ConnectionManager

5.3.1. FNV2-ATL-SOAcm

Interface Name	FNV2-ATL-SOAcm
Description	Each of the DiagnosticsSlaves, DiagnosticsMaster and AnalyticsAgent require a network connection to the SYNC Analytics cloud. We will leverage ECG-CM components/libraries to facilitate this.

Actors	**DiagnosticsSlave** • Requests a connection **ECG-CM** • Informs DiagnosticsSlave whene a suitable connection is available
Interface Type	FNV2-SOA (via ECG-CM library)

6. Use Cases

6.1. Diagnostics Log Types

Log types are defined as constants inside the client library. They are predefined. It is not possible to elect to add an arbitrary file when creating a diagnostics event. It is important that we vet the contents of logs to ensure they comply with ourPII requirements. The following table summarizes some of the log types we expect to define on each ECU, but is by no means exhaustive. As development proceeds, more logs will be added. A "living" list will be made available in our Progammer Guide.

Log	Description	ECG	TCU	SYNC
LOG_TOP_INFO	The output of the 'top' command, or equivalent	✅	✅	✅
LOG_CORE_INFO	The mini core file from a crashed process	✅	✅	✅
PAS_DEBUG_LOG	The Panasonic Application Logs	❌	❌	✅
LOG_SCREENSHOT	A screenshot, only relevant to bug reports	❌	❌	✅
LOG_WIFI_INFO	Basic information about the WI-Fi Connection	❌	✅	✅
LOG_FLASH_INFO	Information about disk usage on the system	✅	✅	✅
LOG_PIDIN_LOGS	the output of various combinations of pidin (or equivalent)	✅	✅	✅
LOG_NETWORKING_LOGS	Networking information (i.e. ifconfig/netstat/route)	✅	✅	✅
LOG_MEM_INFO	Snapshot of memory on the system (meminfo or equivalent)	✅	✅	✅
LOG_OS_INFO	Basic OS/release information	✅	✅	✅
LOG_SYSINFO_LATEST	contents of sys_info log	✅	TBD	✅

6.2. Flash Storage

Being an embedded platform we must be cognizant of our flash storage utilization at all times. All event data and payloads are compressed on disk. In addition, pruning strategies exists to ensure we never exceed our allocated budget of flash usage.

6.2.1. Analytics Storage

Undelivered Analytics events will all be stored on the master ECU. Because they are small and compress well they occupy very little space compared to Diagnostics Events. Our storage strategy will be as follows:

a. For each ECU, keep a running file of events (JSON) open to which new events are added. Again, these are all stored on the Master ECU.
b. Once this file reaches a configurable size (100KB by default), they will be compressed.
c. The total amount of flash storage required will also be configurable. As we approach this value, old archives will be pruned to make space (see Data Pruning section below).

Events are discarded once they have been either uploaded or pruned.

6.2.1.1. Analytics Storage on Diagnostics Slave (v1.1)

Analytics events are sent first to the Diagnostics Slave and onto the DiagnosticsSlave if it is online. Events can be stored on the DiagnosticsSlave under the following circumstances:

a. Events received by the Slave while the AnalyticsAgent (ECG) is not online are stored
b. Events that otherwise cannot be sent to the AnalyticsAgent (for example due to a SOA failure) are stored. Care should be taken to ensure that no 'poison pill' type events are persisted, that is, if an event fails to get to the analytics agent for a consistently repeatable reason (i.e. data corruption, validation failure), it should not be stored.

It is anticipated that these events will be stored in a sqlite database.

6.2.2. Diagnostics Storage

Diagnostics events which are pending delivery are stored on the ECU that generates them. Diagnostics events can grow to become quite large, depending on the logs they contain. Log files are collected at the time of failure and then compressed and written to disk. A fixed amount of total flash space can be configured. The Diagnostics Slave is responsible for ensuring it is never exceeded.

6.2.3. Storage requirements

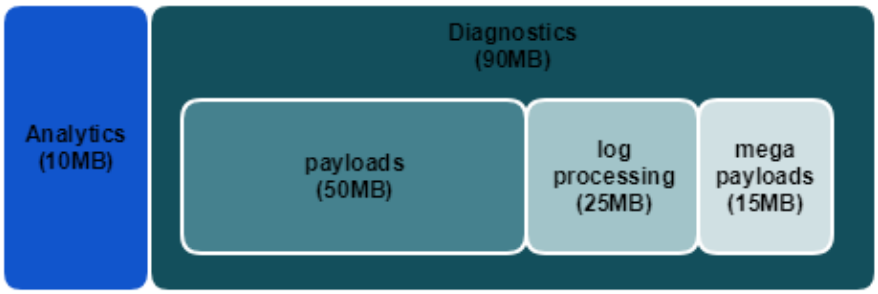
Diagnostics payloads are by far the heaviest user of flash storage. How much space is allocated to diagnostics on each ECU is up to the ECU, however, more space will lead to more events being uploaded with full logs as there is less chance of them being pruned. How much space each event uses will be entirely dependent on the logs that are being collected.

In addition to the flash storage required to store the payloads of pending events, a scratch area is required where the Diagnostics Agent can store temporary files and perform the compression. Again, the amount of scratch space that is allocated can impact the size of files that can be added to each event. Our recommendation is that each ECU reserve at least as much space as the largest file they expect to add to their events.

6.3. Data Pruning

In order to ensure we do not monopolize the available flash storage on an ECU a pruning strategy is in place which allows us to prune events / payloads using **configurable values**. It is not necessary, nor recommended, that each ECU has the same storage requirements.

As a reference, for SYNC 3.2v2, we allocated 50MB for undelivered payloads, 25MB for scratch space (i.e. intermediary files that are generated while collecting logs) and 15MB for megapayloads.



6.3.1. Diagnostics Pruning

We allocate X MB of storage (50MB for SYNC)) for diagnostic payloads. Once we reach a point where a new payload would result in us exceeding this number we aggressively prune payloads until we have freed up Y MB (15MB in the example above) resulting in a low water-mark of (X-Y) MB utilized storage (35MB). Once this threshold is met we will process the incoming event.

When an event is pruned, the Diagnostics Slave is responsible for notifying the Master that this has occurred. The master must then record this fact in its database (set state to pruned) so that it is no longer considered for delivery by the scheduler.

6.3.1.1. Selection Algorithm

The selection of events to prune in diagnostics is based on several factors:

- Severity of the event
- Age of the event

When selecting events to purge, we will first erase all events each severity grouping (DEBUG->UNKNOWN->WARNING->SERIOUS->CRITICALCATASTROPHICBUGREPORT), then by age.

6.3.2. Analytics Pruning

Analytics events can be physically stored by both the DiagnosticsSlave as well as the AnalyticsAgent.

Analytics events are pruned in one of two ways:

- Once a batch of events hasn't been sent for more than a configurable number of days (i.e. 30 days for SYNC)
- If a new batch (default 100KB in size) would result in us utilizing over (analytics.max_storage_size) KB of storage we will discard batches until we have > 100KB of storage available before storing new events.
- Oldest batch is deleted first

6.3.2.1. Pruning by DiagnosticsSlave ⭐

The DiagnosticsSlave must also be able to prune events that it has stored either because the ECG is offline and cannot accept the event, or because the event failed to be delivered to the Agent (perhaps due to a SOA failure). We anticipate storing pending events in a SQLite database on each ECU. Because these events can vary in size, and measuring the size of SQLite database is difficult, particularly given the unstructured nature of analytics events. Care must be taken to ensure that the pruning algorithm selected strikes the right balance between limiting flash usage and over-aggresive pruning.

6.3.3. Pruning & Storage Configuration

Storage and pruning is configurable for each ECU via our OTA Configuration mechanism. The following properties are used to control these values. It is worth re-iterating that **ECUs are configured individually.**

Property	Description
analytics.batch_size	The maximum amount of space each ECU's active batch of events can occupy
analytics.max_storage_size	The maximum amount of combined space each ECU's events (active + compressed) can occupy on the master ECU
analytics.expiry_days	The number of days a batch can be persisted for before being pruned
diagnostics.max_storage_size	The maximum amount of space all diagnostics log data can occupy
diagnostics.low_water_mark	The amount of space that should be used after pruning diagnostics events
diagnostics. working_space_size	How much space should be reserved for diagnostic event processing
diagnostics. mega_payload_size	The maximum space to be reserved for mega payloads. If set to 0 no space will be used.

6.4. Event Fingerprinting and Throttling

A new concept being introduced to the Diagnostics platform for FNV2 is the concept of fingerprints. The idea is that when building events through our API clients are able to identify key attributes which, when combined with other dimensions, will form a fingerprint that can be used to classify an event at a very granular level. This will allow us to take certain actions on the server side, such as reject events matching specific fingerprints (imagine a use-case where we have received 1,000,000 events having the same fingerprint, we are likely at a saturation point where further events with matching fingerprints provide diminishing return.

The fingerprint shall be be the a 128bit hash of all of the identified key fields concatenated together.

Note that this only applies to Diagnostics Events, Analytics events do not have the concept of fingerprints or throttling (due to their smaller size).

6.4.1. Sample Fingerprints

event_id	Client added elements	System Added

bluetoothDisconnected	phoneModel phoneOS	creator_id event_id
LwProcessCrashed	processName	creator_id event_id

6.4.2. Fingerprint Based Rejection and Throttling

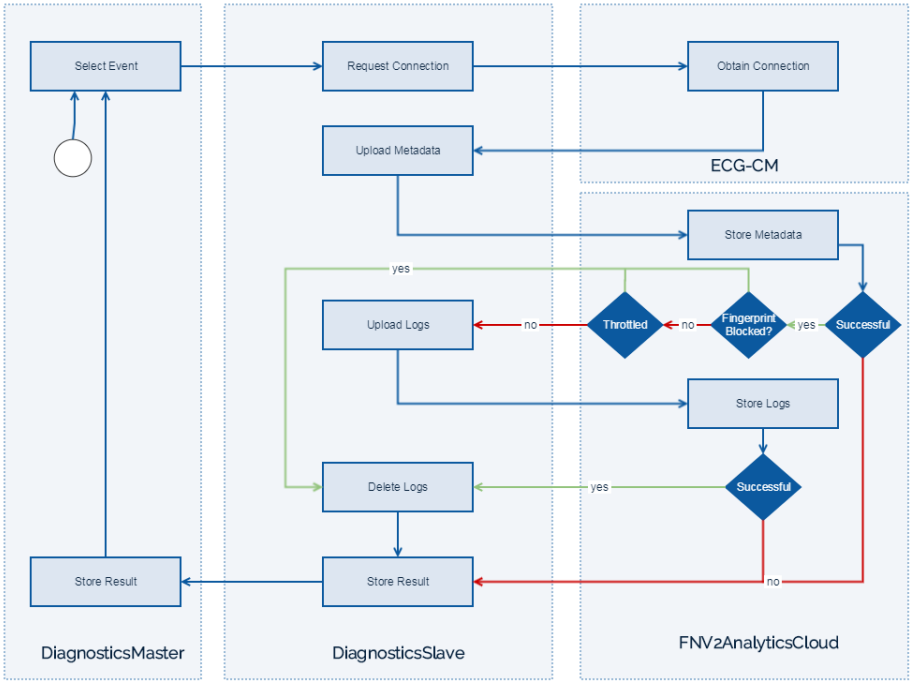
The intent of fingerprint based rejection is to provide a means to reject certain events based not only on the traditional means (i.e. creator, event_type) but also on specific elements of each event. Before upload an event to the server, each Diagnostics Agent is responsible for sending its metadata, including the fingerprint. The server will store this metadata regardless, however, it will respond to this upload indicating whether or not the agent should also upload the associated logs. If yes, the agent will proceed to do so and then remove them from the ECU, of not, they will be immediate removed from the ECU.

6.4.2.1. Throttling

Throttling is slightly different, while fingerprinting is meant to act as more of a long-term rejection strategy, throttling is meant to solve the problem of run away crashes and errors. We dont, for example, want a process to crash 50 times in a minute and generate 50 (more-or-less) identical events. Throttling is also based on event fingerprint, however, follows the below algorithm:

- Define a window to be 24 hours
- If more than one event with the same fingerprint is generated within the window, the server shall only request logs from the first.
- Accept metadata from 11 additional events (12 total) within the window
- After 12 total events received from the client, the client should no longer bother even sending metadata until

Note that event if a diagnostics event is throttled, the corresponding analytics event will still be sent.



6.5. OTA Configuration

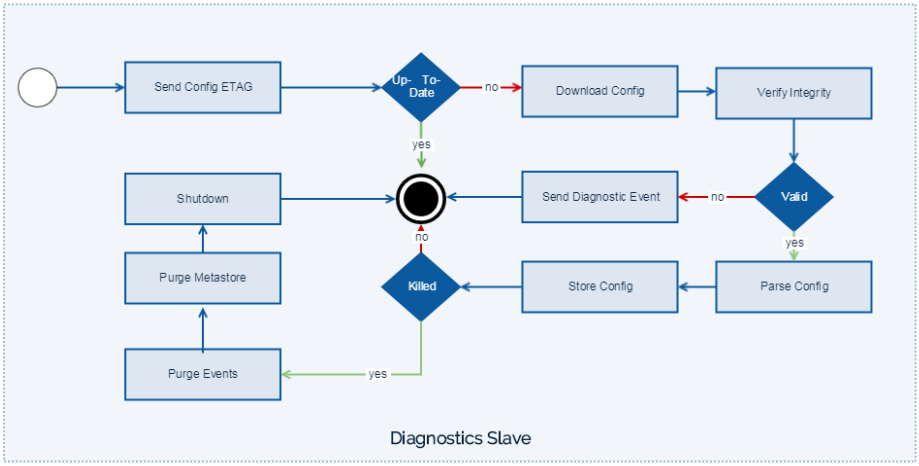
Diagnostics and Analytics offers a comprehensive configuration mechanism allowing us to tune certain elements of the platform. Clients pull down configuration prior to uploading data, with changes taking effect on the **subsequent boot-cycle**. The only exception to this is are the two properties which control whether or not analytics and diagnostics are enabled. If they are disabled, they are disabled immediately rather than on the next boot cycle.

Each of the DiagnosticsSlave/Master and AnalyticsAgents are responsible for managing their own configuration including retrival and persistance.

When requesting a configuration file, clients must provide an ETag for their current Configuration file. The server will only send back a new file if it determines that the version the client is currently using is stale.

The specific configuration which is returned is determined by a number of inputs - VIN11, ECU Name, ECU version as well as some additional ECU specific elements, for example, with SYNC we include whether or not the module is NAV based.

A separate OAuth client_key will be created on each ECU and associated with the configuration server endpoint. This will allow slaves to request configuration themselves when thy noticed that theirs has expired (likely to be during startup).



6.5.1. Local Storage

Configuration files are downloaded by the Diagnostics Slave/Master and Analytics Agent. Each is responible for persisting their downloaded copy locally somewhere in Flash storage. Files are ~ 5KB in size. This configuration must not be deleted during either Master Reset or OTA Update.

6.5.2. Default Configuration

Each ECU will be built with a default configuration file on the filesystem. This will seed all configurable properties with reasonable defaults.

6.5.3. Configurable Properties (not yet finalized)

Property	Platform	Description
agent_enabled.<ecu_name>	Master	Master kill switch to enable/disable either analytics or diagnostics on each ECU. Alternatively, omitting the ecu will disable the feature entirely
send_log_headers_over_cellular	Diagnostics	Determines whether or not metadata can be sent over cellular
max_payload_size_kb_over_cellular	Diagnostics	Maximum diagnostics payload that can be sent over cellular
send_data_while_raoming	Diagnostics	Whether or not data can be sent while roaming.
allow_raw_memory_capture	Diagnostics	Controls whether or not ramdumps are enabled
send_large_memory_captures	Diagnostics	Determines whether or not large memory captures are enabled.
event_id_filter	Diagnostics|Analytics	(whitelist|blacklist) of event_ids that (can|cannot) be sent
logset_override	Diagnostics	Dynamically modify the set of logs that comprise a logset (the logs must already exist on the platform)
logworthy_log_override	Diagnostics	Dynamically modify the set of logs each event_id collects
batch_event_size	Analytics	How large a batch of analytics events can be before it is queued for delivery
batch_event_expiry	Analytics	How long a batch of analytics events should be written to before it is queued for delivery

batch_stale_time	Analytics	How old a batch can be before it is pruned
disk_quota	Analytics	The disk space analytics is permitted to use
refresh_interval_hours	Master	How frequently clients should attempt to refresh the configuration automatically

6.5.4. Periodic Refresh

The master Configuration file will include a duration with which clients are expected to periodically update their configuration files. This doesn't necessarily need to be the same for each ecu.

It will be the responsiibility of the Diagnostics Master to tell the Slaves when to refresh their config. The reason the Master must control this is because an OAuth tokenD is required to fetch the configuration, and only the Master can retrieve them (the Slaves cannot request a token from the Master). This will hinge on two things:

a. Network Connectivity
b. The interval must have expired.

6.6. Master ECU Detection

A key aspect of this proposed architecture is the ability to determine which ECU is the "master". In cases where an ECG present (expected to be all cases at this time), it will be the ECG. Should a time come when ECG is not present, the master ECU would be SYNC. It is necessary then, for SYNC to know that an ECG exists when starting in order to know whether or not it should start the Analytics Agent and Diagnostics Master. Once this determination has been made the Diagnostics Master on SYNC can persist this value in its local config.

6.6.1. Diagnostics

If SYNC starts up, and has NOT yet determined if ECG is present (i.e. most likely initial boot), the DiagnosticsMaster must start, however, it must not enable any SOA messages or otherwise make itself available until after it has ensured the ECG is not present. The DiagnosticsMaster will both subscribe and publish to the **/service/data/fnva/status** topic. The Diagnostics Master will publish to this topic when it comes online **on the ECG**. If the Master running on SYNC does not receive this broadcast within 30 seconds (The ECG is expected to boot in 3-4 seconds), it will assume it is the master and publish to this topic itself. If, it does receive it, it will shut down.

If, later on, the Master ends up receiving this broadcast, it is responsible for purging its metastore of all events other ECUs may have sent to it (retaining its own) and dying. This is not expected to happen, but should be considered none-the-less. Since the expectation that the local config value indicating which ECU is master is only expected to be absent on initial boot, very few, if any, events should even exist at this point.

6.6.2. Analytics

Similarly the AnalyticsAgent will also start on SYNC and attempt to read its local configuration to determine if the ECG is present. If no file is found, it should wait for up to 30 seconds a Broadcast from the ECG's Analytics Agent indiciating it is available **/service/data/fnva/status.** If it does not receive this broadcast, it will send it itself thus becoming the Agent. If it does receve it, it will shut down. The Client library (details in API HLD) will queue events until it receives this broadcast itself so there should be nothing to purge.

6.7. Consent / Feature Availability

Consent is an important aspect of our overall solution. There are several tiers of consent which must be considered. For FNV2 Diagnostics collection will always require Customer Consent in order to be enabled on production vechiles, it will be off by default.

On Pre-production vehicles (identified by the signing of the master ECU), or those with a token deployed via Hancock, diagnostics is enabled by default, and user content setting does not have any impact on this.

6.7.1. Diagnostics Consent matrix

See the below matrix for details of the various consent combinations as they relate to diagnostics. Note that when the feature is disabled regionally the other values are not important as this supersedes all values.

	Diagnostics User Consent	**Regional Consent**	**Diagnostics Server Enabled**	**Pre-ProductionVehicle | Hancock Token**	**Impact on Diagnostics**

Explanation	Users can opt-in or out of diagnostics through a CCS setting on the SYNC module	In certain jurisdictions we require the ability to disable diagnostics completely **If disabled regionally no CCS setting or bug reporter should be shown in the UI.**	We can remotely disable Diagnostics. Consider this an emergency release valve.	Preproduction vehicles are those not signed with prod key, or those that have a special token deployed through **FNV2-FDTM**	
Combination 1	Enabled	Enabled	Enabled	No	Enabled Diagnostics functions normally
Combination 2	Enabled | Disabled N/A - When disabled regionally no UI is present to the user to enable diagnostics.	Disabled	Enabled|Disabled	Yes|No	Disabled Diagnostics is completely disabled. No UI is presented to user in this scenario.
Combination 3	Enabled	Enabled	Disabled	No	Disabled Feature will appear to be enabled to user, but system will behave as though no user consent is given.
Combination 4	Disabled	Enabled	Enabled|Disabled	No	Disabled
Combination 5	Enabled | Disabled	Enabled	Enabled	Yes	Enabled By Default, user Consent has no impact.
Combination 6	Enabled | Disabled	Enabled	Disabled	Yes	Disabled

6.7.1.1. Diagnostics Consent Changes - Enabled to Disabled

When Diagnostics transitions from an enabled to disabled state the following willl occur:

a. DiagnosticsMaster sends broadcast over **FNV2-ATL-SOAdsm** topic **/service/data/fnva/status/consent** with a payload indicating "disabled", the Slaves must subscribe to this
b. DiagnosticsMaster updates config with new consent value
c. DiagnosticsMaster purges stored logs
d. DiagnosticsMaster cancels any pending ConnectionRequest
e. DiagnosticsMaster relinquishes any outstanding votes
f. DiagnosticsSlave **repeats** steps 2-5 itself
g. DiagnosticsSlave deletes persisted analytics events
h. DiagnosticsSlave rejects any subsequent events.

6.7.1.2. Diagnostics Consent Changes - Disabled to Enabled

The behaviour when Diagnostics consent changes from disabled to enabled is similar to application start:

a. DiagnosticsMaster sends broadcast over **FNV2-ATL-SOAdsm** topic **/service/data/fnva/status/consent** with a payload indicating "enabled", the Slaves must subscribe to this
b. DiagnosticsMaster updates config with new consent value
c. DiagnosticsSlave updates config with new consent value
d. DiagnosticsSlave accepts any subsequent events.

6.7.2. Analytics Consent Matrix

Analytics behaves slightly differently. We will operate in a "anonymous" mode by default, where we guarantee that no PII is being sent to our servers. Users can optionally elect to provide consent to enable enhanced analytics data collection which will allow us to add some PII, like VIN. Also, unlike Diagnostics, Analytics is not affected by the type of signing on the master ECU. It is also possible to disable Analytics entirely.

	Analytics Consent State	Regional Consent	Analytics Server Enabled	Impact on Analytics
Explanation	Users can opt-in or out of diagnostics through a CCS setting on the SYNC module	In certain jurisdictions we require the ability to disable diagnostics completely **If disabled regionally no CCS setting or bug reporter should be shown in the UI.**	We can remotely disable Diagnostics. Consider this an emergency release valve.	
Combination 1	Enhanced	Enabled	Enabled	Analytics data is collected including approved PII
	Enabled	Enabled	Enabled	Analytics data will be collected free of PII
Combination 2	Enabled | Disabled N/A - When disabled regionally no UI is present to the user to enable diagnostics.	Disabled	Enabled|Disabled	Disabled
Combination 3	Enabled	Enabled	Disabled	Disabled
Combination 4	Disabled	Enabled	Enabled | Disabled	Disabled

6.7.2.1. Enhanced Analytics Consent Changes - Enabled|Enhanced to Disabled

PreCondition: AnalyticsAgent subscribes to **FNV2-ATL-SOAdsm** topic **/service/data/fnva/status/consent**

a. AnalyticsAgent receives Broadcast
b. AnalyticsAgent updates local config
c. AnalyticsAgent purges stored events (no effective means to remove PII from stored events)
d. AnalyticsAgent no longer adds PII to events or accepts those that identify as containing it.

6.7.2.2. Enhanced Analytics Consent Changes - Disabled to Enabled

PreCondition: AnalyticsAgent subscribes to **FNV2-ATL-SOAdsm** topic **/service/data/fnva/status/consent**

a. AnalyticsAgent receives Broadcast
b. AnalyticsAgent updates local config
c. AnalyticsAgent accepts all events, and decorates with any PII we decide to add.

6.7.3. Additional Notes on Pre-production vehicles

In addition to enabling diagnostics by default on these vehicles, other features can be enabled as well. For example, the Bug Reporter application on this vehicles will allow users to attach a voice annotation, something we cannot do on consumer vehicles.

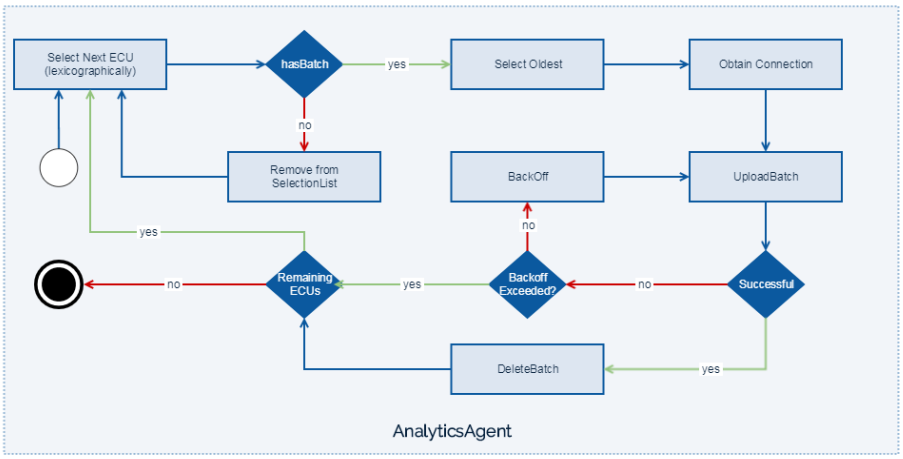
6.8. Event Scheduling

Network connectivity will not be available with any great degree of frequency in vehicles. We must plan for scenarios where when it does become available, we will have several diagnostics events and analytics batches waiting to be uploaded. In order to ensure both fairness and that diagnostics events of higher importance (i.e. Bug Reports) are published we will use the following scheduling algorithms.

6.8.1. Analytics Scheduling

Analytics events are batched together until they are ready to be sent. They are ready to be sent when one of the following conditions holds true:

Each ECU will have separate batches of analytics data, though all are stored on the Master ECU where the AnalyticsAgent resides. We will round robin through each of the ECU's batches, delivering the oldest of each first.



6.8.2. Diagnostics Scheduling

Diagnostics Scheduling is slightly more complicated the Analytics in that it takes more factors into consideration. Recall that the DiagnosticsMaster is ultimately responsible for telling the Slaves when and what they are able to send.

Also important to note is that Diagnostics events take a two phase-approach, metadata is uploaded first, followed by the payloads. Often times the server will respond indicating that it does not need the payload at all.

Consider the following generic algorithm which will be applied to the selection algorithm:

a. Round-robin through each ECU lexicographically *according to some criteria (see below)*
b. Select the oldest event from the current ECU
c. Instruct the DiagnosticsSlave on the current ECU to upload *something* (either metadata or logs)

The Diagnostics Schedule Algorithm becomes:

a. Apply Generic algorithm for **events of critical severity**, uploading each event's **metadata.**
b. Apply Generic algorithm for **remaining events**, uploading each event's **metadata**.
c. Apply Generic algorithm for **events of critical severity**, uploading each event's **logs**.
d. Apply Generic algorithm for **remaining events**, uploading each event's **logs**.

6.9. VDR Support

VDR (Vehicle Data Recorder) support is something we are considering and indeed will likely implement, however, the details are currently vague and assumed implementation challenges large. We have other requests to support a large-file streaming type of solution which would likely faciliate VDR as well. This work will be tracked in a separate HLD as it is non trivial.

6.10. Data Dimensions & Decoration

Almost as important as the contents of each diagnostics and analytics event are the dimensions which are added to it. Dimensions are additional attributes that provide additional context around the event. Dimensions can fall into three categories:

- **Static** - Data that will never change. Examples include
  - VehicleModel
  - VIN11
- **Slowly Changing** - Data that can change, but rarely does:
  - Language
  - SoftwareVersion
- **Frequently Changing** - Data that changes frequently
  - Speed
  - Gear

Dimensions will be decorated in the following way:

Diagnostics

- Regardless of the type of dimension, they will be added to every event.

Analytics

- Static - Per Upload, added at time of upload
- Slowly Changing - Per Batch, added at time of batch expiry.
- Frequently Changing - Per Event, added at time of event receipt

Note that if the value of a frequently changing dimension is critical and must exactly correspond to the value that the time the event was generated the client should add it itself. There could be delays that occur between the time the event is generated and dimension added by our service. If our agent notices that a dimension it typically adds is already populated for a certain event it will not clobber the value. This will be highlighted in the developer guide.

6.10.1. List of Dimensions

Similar to Log Types, this is a moving target and not yet finalized, however, there are some dimensions we know we will be collecting which are documented here.

6.10.1.1. Global - applied to every ECU's events

Dimension	Source	Type	DiagnosticsEvents	AnalyticsEvents	Description
VIN11	VIM	Static	Included	Yes - Per Upload	The first 11 digits of the VIN. Provides information about the vehcile but not the specfic vehcile.
VIN	VIM	Static	Included	If enhanced analytics enabled	The vehicle's VIN
VehicleId	Generated once Per vehicle by Master ECU.	Static	=VIN	Yes - Per Upload =VIN if enhanced analytics enabled	Randomly generated when not already present. Should **n ot** be deleted during master-reset.
isproduction	Based on signing key of Master ECU OR presence of specific token loaded onto vehicle.	Slowly Changing - in theory this could change between boot cycles if a token is loaded in between.	Included	Yes- Per Upload	Whether or not this is to be considered a production vehicle
synclanguage	VIM	Slowly Changing	Not Included	Yes - Per Upload	The language of the SYNC module
gear	VIM	Frequently Changing	Not Included	Yes - Per Event	The current gear of the vehicle
speed	VIM	Frequently Changing	Not Included	Yes - Per Event	The current speed of the vehcile
ignitioncount	If not available, will be tracked by DiagnosticsMaster.	Frequently Changing	Included	Yes - Per Batch	The total number of ignitions, reset w/ master reset
Igntionstate	VIM	Frequently Changing	Included	Yes - Per Event	i.e. off, acc, on

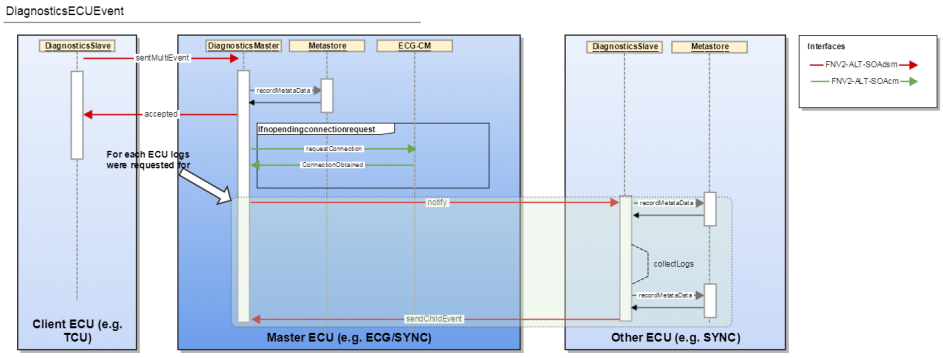
6.10.1.2. ECU Specific

Details of ECU specific dimensions will be hashed out as we on-board more and more teams from each. Examples of ECU specific dimensions would include things like FPNs and platform level software versions.

Dimension	Source	Type	DiagnosticsEvents	AnalyticsEvents	Description
uptime		frequentl y changing	Yes	Yes	The total uptime, since boot, of this ECU
build		slowly changing	Yes	Yes	The currently build of the software running on the CCPU of the ECU. This differs from the FPN in that the FPNs are not assigned until late in the process and only to release builds.

6.11. Multi-ECU Events

Multi ECU events give ECUs to ability to create DiagnosticEvents that include logs from multiple ECUs. For example, SWUM may want to raise a DiagnosticEvent when a OTA download fails, and include logs from both the ECG and TCU. The only difference to the client when creating these events is that they define logs from multiple ECUs, the event is still send to the DiagnosticsSlave as usual, and then onto the DIagnosticMaster. The DiagnosticMaster is then responsible for notifing the corresponding DiagnosticSlave on each required ECU of this event. Those ECUs will then create child events of the original parent with the required logs.



6.11.1. Multi-ECU validity period

It is possible that an ECU will create a Multi-ECU event request logs from an ECU that is not currently powered on, and may not come online for some time. As is always the case, the usefulness of log data is directly proportional to the amount of time that has passed between the fault and time of collection. Collecting logs from an ECU 7 days after a fault occurs, for example, is likely not to be beneficial and would be a waste of bandwidth. For this reason we define a validity period of 24 hours for Multi ECU events. If a participant in a Multi-ECU event cannot collect logs within 24 hours, it will not bother to do so at all.

6.12. Multi Module Bug Reporter

One immediate application of Mutl-ECU events is the ability to enhance the current Bug Report application introduced in SYNC3.2v2. For FNV2 Analytics bug reports will trigger a multi-ECU diagnostics event and collect logs from all three of the fast ECUs (TCU, ECG and SYNC).

6.13. ECG in low/no-power mode

A strategy is required in order to handle the scenario where an ECU sends a diagnostics event at a time when the ECG is powered-down or otherwise not able to respond. This is not necessarily a corner-case as the TCU is not powered down nearly as frequently as ECG. There are two scenarios to cover:

6.13.1. Analytics Event sent from Client Library ⭐

If the ECG is not online to receive analytics events, the DiagnosticsSlave is responsible for persisting the event on the local ECU. When the ECG comes online (announced via Broadcast) the DiagnosticsSlave will 'replay' each of the persisted events in order to deliver them to the AnalyticsAgent.

6.13.2. Diagnostics Event sent from Slave to Master

Diagnostics Slaves replicate their metastore database to the master when they receive an event. We will implement a means to synchronize these databases to cover the scenario where the master ECU is powered-down and unable to accept the event from the slave. If an event's status (in the metadata store) is equal to 0 then that means it has not yet been acknowledge by master. When the master comes online it will send out a Broadcast indicating that it is currently available at which point Slaves will be able to synchronize. In this context, synchronization is merely the act of the Slave sending to the Master each event it has not ack'd.

6.14. ECU Replacement

Though not expected to be a common use-case, we need to ensure we gracefully handle the scenario where an ECU is replaced. There are two distinct use cases to consider, specifically with respect to diagnostics.

6.14.1. Master ECU Replacement

In this scenario a new Master ECU will come online with an empty database and default configuration. In this scenario, the master database will no longer be in sync with the slaves. In this case the synchronization process described above will be initiated, and the slaves will automatically send any events that they think the master has not acknowledged. The important thing to note is that the master database will no longer have the historical record of all events each ECU has sent.

6.14.2. Slave ECU Replacement

In the scenario where a Slave ECU is replaced, it willl come online with an empty database and default configuration. In this scenario the master ECU metastore will contain more information than the slave ECU metastore, but this does not pose any problems. The Slave ECU will simply beginnormal operation.

6.15. ECU Update

Similar to ECU Replacement, though more likely, is the situation of ECU Update. When an ECU is OTA updated, we will purge it's metastore and undelivered events. There is no need to purge the Analytics Batches as they use very little space compared to Diagnostics. This will put the ECU in a situation similar to the one it finds itself in after replacement, though the configuration file will be retained.

6.16. Network Selection

Our current plan is to stick to using Wi-Fi (specifically SYNC Wi-Fi) for all uploads. This doesn't mean we have ruled out cellular completely, however, because some of our payloads can be quite large we do not want to monopolize it. We may consider leverage the cellular modem under the following circumstances:

1. To upload the metadata associated with diagnostics events
    - Given that diagnostic events will be generated a relatively low rate, and the metadata will be quite small, this should not consume too much bandwidth.
2. To upload critical diagnostic events in full, such as User triggered bug reports.
    - These could be large (~1-2MB) but if a user is going to the trouble of filing a bug report we should do our best to try to deliver it.
3. A hybrid model whereby after some amount of time has passed with no Wi-Fi connectivity we resort to using cellular (for 1 & 2 above).

6.17. On Demand Flash Release API

We will require the ability for an ECU to request that we purge our persisted data, the interface is TBD (will be available in version 1.2 of HLD).

Upon receiving this request, we will take the following actions:

1. Purge Diagnostic payloads
2. Update metastore to indicate events have been purged
3. Purge analytics events from both Analytics Agent (on ECG) and Diagnostics Slaves

7. Performance

It is critical that the analytics and diagnostics components do not hinder application performance beyond any expected means (ideally there should be no impact). All requests to send events to either of our agents through the library shall be delivered asynchronously, unless explicitly requested to be performed as a blocking call by the client (a use-case reserved for exceptional circumstances, i.e. catastrophic failure where the application is no longer in a predictable state and some assurance is required that the event has been delivered).

Specific KPIs have not yet been identified.

8. Security Features

On the vehicle itself we will benefit from our use of the SOA and IPC frameworks and the inherent security they provide. Request to upload data to our cloud are all made over a secure TLS connection, the agents themselves must request an OAuth token from our server and identify themselves by a clientId. OAuth tokens shall only be granted to the DiagnosticsMaster which is responsisble for distributing it to other components that wish to upload (i.e. the Diagostics Slaves and AnalyticsAgent). We are working with the Ford Cyber security and Red (pentest) teams in order to ensure our solution conforms with Ford's security requirements.

8.1. TLS

All communication to the SYNC Analytics cloud is over a secure TLS1.2 channel.

8.1.1. Certificate

All three of the DiagnosticsSlave, DiagnosticsMaster and AnalyticsAgent talk directly with our cloud, they must each include a copy of FNV Analytics's Root CA Certificate. Some ECUs (TCU and ECG) offer secure key storage, which we may be able to leverage. The exact Root Certificate we use will be that of the SSL Termination point of the FNV Analytics cloud. All signs point to this being a GlobalSIgn certificate though we are still working through deployment details with IT.

We will disable SSL renegotiation

There is an open issue to track the specifics of certificate and API Key management.

8.1.2. Cipher Suites

Per guidance from the Ford cyber security team we will only suppor the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384_P256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256_P256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA_P256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA_P256

8.1.3. Other Mitigations

We will use the following options to further strengthen our TLS connection

- Server Name Indiciation
- Disable SSL Renegotiation

8.2. Authentication

Clients authenticate with the Analytics cloud using an OAuth based mechanism. They pass in a client API key which is embedded and obfuscated within the code using a modified SHA based algorithm. Clients pass this key, along with their client_id (reconn) into our authentication server. The server then validates the supplied credentials, and, if valid, returns an HMAC signed OAuth token which must be passed to our service in order to retrieve configuraton and upload events.

Only the DiagnosticsMaster and AnalyticsAgent, both of which reside on the master ECU, are able to request OAuth tokens from the authentication service. The DiagnosticsSlaves will be given an OAuth token when the DiagnosticMaster schedules them to send an event.

8.3. Transmission Protocol

While the exact schema has yet to be determined, we do know that Analytics Events are sent over the (TLS) air as compressed JSON files. Similarily, Diagnostics metadata will also be delivered via JSON while the Diagnostics payloads themselves will simply be comprised of zip files.

8.4. Penetration Testing

We will be leveraging the Ford Red team to complete a penetration test of our components in order to ensure they comply with Ford Standards.

9. Testing Strategy

A variety of testing techniques will be utilized in order to ensure the quality of our product. We have a dedicated testing team will that coordinate and design this, at a very high level we will employ the following:

- Unit tests
- Automated testing
- Both bench based and in-vehicle testing

10. Dependencies and Risks (if applicable)

FNV2-ATL has dependencies on the following compoents:

- FNV::Base
- FNV2-ALM-IPC
- FNV2 SOA Middleware
- FNV2-CM
- ECG-POWER
- TCU-POWER
- SYNC-POWER
- FNV2-FDTM

There are no known risks as this time.

11. References

- https://www.eesewiki.ford.com/display/ecg/FNV2-ALM+Interprocess+Communication+%28FNV2-ALM-IPC%29+HLD
- FNV2 Vehicle Information Manager Architecture (FNV2-VIM)

12. Open Questions / Issues

Question	Resolution	Assigned to	Notes
How do / should we integrate with CCS. I suspect the answer is "yes"		Jonathan Niemi	https://www.eesewiki.ford.com/pages/viewpage.action?pageId=8026233
Open question how do we determine which ECUs are present	Resolved	Jonathan Niemi	See section 6.6
How to handle case where ECG is in low-power mode and cannot accept events.	Resolved	Jonathan Niemi	See section 6.13

13. Revision History

Date	Version	Changes
April 9th, 2017	0.1d	Initial Draft for internal review

Approved Version	Link to Approved Document	Link to Review Comments