SNP Service

SNP Service Structure

Subscribes to SNP.Inbound topic and receives packets from there. Routes packets through service out to active MQ, out UDP ports to MDE or MDI, through remote procedure calls for gRPC, through a tcp ip connection for Camstar or through a sql querry to the Engineering Database.

Using ActiveMQ is the best way to send Control signals ( create edit delete machine).just copy the packet from here ( replace the weird quotes with normal quotes) then change the machine to fit your needs.

ENG\_DB

ActiveMQ

MDE/MDI

Camstar

Producing App

Producing App

Producing App

Producing App

gRPC

Pac-Lite Service

Stomp, JMS, MQTT, Etc.

UDP

SQL Queries

gRPC Calls

TCP/IP

MQTT

ActiveMQ

SNP PLC

SCADA Box

Producing App

Producing App

SNP Packets

Packet Structure

6 Byte

Json Payload

Reserved

Dev\_ID

Packet Type

Packet Header

1 Byte

5 Byte

1 Byte

1 Byte

246 Byte

MQTTHeader

* MQTT Header: (6 bytes): See MQTT documentation for details. See <<http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/>>
* Packet Header: Depends on the application (ex., SNP uses header 1 (soh signal not ascii 1, Scada Box Environment monitoring uses header 2)
* Packet type: used to differentiate how it is forwarded. Each documented packet will have one attached (this is stored in a byte as a character). See section ”Packet Type” for details on the different packets.
* Next is the device id. This differentiates different SNP machines or other devices to the service. (one way to ensure that yours is unique is to use your unique mqtt identifier as your device id. SNP machines do this.
* Reserved : 5 reserved bytes, if you are no using these bytes leave them as the number 32 (ascii space) doing this as opposed to leaving it null allows the packet to be treated as a string!
* JSONPayload: Up to 246 bytes of Json encoded data. This data will be explained more below. However it is of note that the only section that cares about this data is after the packet is differentiated from all others. This means that If you are setting up your own packet in the service you can change this encoding to anything you want ( as long as you document it here).
* 260-byte max packet size with a max payload of 246 bytes (~5% packet overhead)
* All Packet Timestamps are generated Server Side

JSON Data Explanation

{pair1 id, pair1 data,….}

{“FirstName”:”Devlin”,”LastName”:”Paddock”,”isAlive:true”,”age”:20,”PhoneNumber”:”FakeNumber.number”}

Pair1 id = “FirstName”

Pair1 value= “Devlin”

JsonPacket[‘FirstName’] returns Devlin

A json string is characterized by certain features. First it is encapsulated in open brackets {} Next each key value pair is Comma separated ,Next each key value pair is made up of a “key”:value pair section. The key will always be in quotes as a string while the value may be either a number Boolean string, an array or another json segment all together. Above we can see the number 20 in age has no quotes as it is not a string while the string Devlin is encapsulated as a string. How this then works is when decoded you will have a json object we will call receivedPacket. You will then be able to call

receivedPacket[“FirstName”] or however your specific language implements it to return the value Devlin, or receivedPacket[“age”] to return the number 20.

Why did we pick json?

Due to the restrictions on MQTT for plc’s it was important to pick a structure that had relatively low overhead ( this removed standards like XML from the table) and with easy formatability /readability ( this removed formats like Yaml from the table due to more confusing scoping with tabs instead of clear cut quotes and brackets).

Next we wanted something that would interface well with more classful designs in higher language but be easy enough to format for very low level languages that it would be accessible to risc or PLC controllers(this was the main reason for avoiding Encoding schemes that would Pack all data into packed byte arrays and separate them later). We also wanted something that was scalable in its ability to not send certain data, for example with the ShortTimeStatistics Packet realistically only 1 to 2 errors will be available at once so we wanted something that wouldn’t have to send 31 0’s and the 2 1’s to designate that error. With Json there is a simple check for if the field was not passed and if so we can assume the bit was not high and send much fewer fields. (this was another factor against using packed byte arrays). Finally we wanted to avoid having different functions and documentations for each machine. To avoid this the packets alone need to have all information to control how the packet is processed. Json lends itself well to this by allowing you to get a collection of all the keys and process it according to the keys rather than putting a packed byte array into a function and processing it in a defined pattern.

**Section: Packet Type**

|  |  |  |
| --- | --- | --- |
| Packet Header | Packet type/Name | Info |
| 1 Simply Not Pac | 1/ Index Summary | Required fields {Machine, Good, Bad, Empty, Indexes, UOM, and NAED};  Optional Fields {};  Triggers a throughput packet to Camstar and a packet to the ENG Database for recording the information  Sent every 15 minutes by SNP. Summarizes what has happened in those 15 minutes in terms of good bad empty and number of indexes run.  Sample  {"Machine": "HIL-XS-FIM", "Good":"32" , "Bad":"32", "Empty":"4", "Indexes":"68", "UOM":"EA", "NAED":"31474"} |
| 1 | 2/ Downtime | Required fields { Machine,Status,Time,MReason,UReason, NAED,Code};  Optional Fields { };  Triggers a downtime packet to Camstar and a packet to the Eng. Database for recording the information sent out when no index is received for 60 seconds or a user/machine error stops it.  Sample  {"Machine": "HIL-XS-FIM", "StatusCode":"2" , "MReason":"Preventing Wrenching", "UReason":"oiling up wrench launcher", "NAED":"31474","Code": "1300"} |
|  | 3/ Short Time Statistic | Required fields {Machine, Good, Bad, Empty, Indexes, UOM, Other };  Optional Fields {Any Errors you are reporting};  a packet to the Eng. Database for recording the information. Sent out each index by SNP. Can send out UDP Packets as well ( disabled as we are phasing MDE out)  Sample  {"Machine": "HIL-XS-FIM", "Good":"0", "Bad":"1","Empty":"0","Attempt":"1","Other":"1","Head\_number":"1"} |
| 1 | 4/Product Change | Required fields {Machine, NAED};  Optional Fields { };  A packet to Camstar allowing BOM Enforcement scanning to work. Must be sent out on every product selection.  Sample  {"Machine": "HIL-XS-FIM”,”NAED”:”03301”} |
| 1 | 5/Gas Analyzer | Required fields { Line,Head\_number,InternalWaterPercent,ExternalWaterPercent,InternalPresureReading,DaysToRetain};  Optional Fields {};  A Packet sent to SQL to record Gas Analyzer data. If the table is missing for the line it is created and any data older than DaysToRetain is deleted. Next the data is added with a timestamp generated server side. DaysToRetain.  Sample  {"Line": "XS", "Head\_number":"1", "InternalWaterPercent":"1", "ExternalWaterPercent":"1", "InternalPresureReading":"1", "DaysToRetain":"14"} |
| 1 | 252/ Delete Machine | Required fields {Machine,Line};  Triggers the removal of the databases and Machine Info Entry for whichever machine is entered.  Will not remove the database. That must be done manually. currently  Sample  {"Machine": "HIL-XS-FIM", "Line":"XS"} |
| 1 | 253/ Edit Machine | Required fields { Machine,Theo,Line };  Optional Fields {Errors};  Edits the Machine Info entry for the machine to match the rest of the statement and the SNP id it comes from. Will also addon any optional Errors onto the SQL Database table. They must be unique or the command will fail.  Cannot rename machine must delete then remake.  Sample  {"Machine": "HIL-XS-FIM","Theo":"4800", "Line":"XS","Errors": "Error21,Error22","Engineer": "Devlin Paddock"} |
| 1 | 254/ Create Machine | Required fields { Machine,Theo,Line,Engineer,Plant};  Optional Fields {Errors};  Creates SQL databases for the machine entered and naming scheme. Also adds a machine info entry with the above information and the SNP id it comes from Will add any errors in the csv Errors section as a column in sql for short time statistics to report to.  Sample  {"Machine": "HIL-XS-FIM","Plant": "Hillsboro","Engineer": "Devlin Paddock","Theo":"4800", "Line":"XS","Errors": "Error1,Error2"} |
| 2 Environmental Monitoring Program | 1/Index Summary Packet | Required fields {Temperature, ChangeOver5Seconds, Location};  Optional Fields {TimeStamp, Humidity, FlowRate };  Triggers a recording packet to sql to record the data shown above^ if the Time stamp is missing it will be taken server side and inserted. If humidity or flow rate are missing they are assumed 0.  Send every 10 seconds by the BaseTec Machines  Sample  {"Temperature": 87.5, "Location":"XSCabinet" , "ChangeOver5Seconds":"2.12", "Humidity":"76.6", "FlowRate":"2112.2", "TimeStamp":"19-11-11-08:31:26"} |
| 2 | 2/Warning Packet | Required fields {Message, Urgency, Location, Timestamp};  Triggers a recording packet to sql to record the data shown above^ if the Time stamp is missing it will be taken server side and inserted.  Send every 10 seconds by the BaseTec Machines  Sample  {"Warning": "Help Im on fire, and the flow rate is really low i wonder if the cooling system broke.", "Location":"XSCabinet" , "Urgency":8, "TimeStamp":"19-11-11-08:31:26"} |
| 3 Service Control Packets | 1/Logging Level | Required fields {LoggingLevel};  Optional Fields {IntTimeInSeconds};  Sets the logging level of the service. If intTimeInSeconds is assigned and not 0 it will revert to the previous loggingLevel after that time has passed.  Sample  {"LoggingLevel": 5, "IntTimeInSeconds":"320"}  0 is none 1 is only system errors 2 is Camstar errors 3 is packets received and all errors 4 is all errrors packets received and additional info 5 is all available info plus an output of the actual packets being received. |
| 3 | 2/ Silence/Send | Required fields {Sendbool};  Optional Fields { IntTimeInSeconds };  Sets weather the service will output to any outputs. If 0 it will not send outputs if 1 it will. If IntTimeInSeconds is assigned and not 0 it will revert to the vrevious logging level after that time has passed.  Sample  {"SendBool ": 1, "IntTimeInSeconds":"320"} |
| 3 | 3/Deafen/Listen | Required fields {ListenBool};  Optional Fields {IntTimeInSeconds };  Sets weather the service will listen to any inputs. If 0 it will not listen to inputsif 1 it will. If IntTimeInSeconds is assigned and not 0 it will revert to the previous logging level after that time has passed. All control packets will still be heard even when deafened for hopefully obvious reasons  Sample  {"ListenBool ": 1, "IntTimeInSeconds":"320"} |
| 254GenericPackets | 1/SQLCommand | Required fields {Message};  Optional Fields { };  Runs the Message as a SQL Command against the database it is hooked up to ( currently ENGDB) useful for arbitrary inserts and other non standardizeable sql requests.  Sample  {"Message": “Insert into ChainStretch ([Head],[Stretch],[Timestamp],[B32\_Output],[B31\_Input]) values (1,1.12,TimeStamp,1,0)”} |
| 254 | 2/ Camstarservice | Requiredfields {Service};  Optional Fields {};  Runs the Service passed in to the Camstar server with SNP’s privileges and connection strings.  Sample  {"Message": “<\_\_service \_\_serviceType="ResourceSetupTransition"><\_\_utcOffset><![CDATA[-04:00:00]]></\_\_utcOffset><\_\_inputData><Availability><![CDATA[1]]></Availability><Resource><\_\_name><![CDATA[Resource]]></\_\_name></Resource><ResourceGroup><\_\_name><![CDATA[]]></\_\_name></ResourceGroup><ResourceStatusCode><\_\_name><![CDATA[Unscheduled]]></\_\_name> //if down send down</ResourceStatusCode><ResourceStatusReason><\_\_name><![CDATA[]]></\_\_name></ResourceStatusReason></\_\_inputData ><\_\_execute /><\_\_requestData ><CompletionMsg /><ACEMessage /><ACEStatus /></\_\_requestData ></\_\_service > |

Field Information

* Machine
  + Key Machine
  + Value string, should be the name of the machine it is interfacing with. For SNP this is used to name tables and communicate with a Camstar resource named the same. This must include HIL, Line, Machine. Ex HIL\_XP2\_BAM,
* Line
  + Key Line
  + Value string, not used by SNP however recorded as part of the machine information and may be used by other applications as it is recorded to an sql database.
* Good
  + Key Good
  + Value int, a count of the number of good products produced.
* Bad
  + Key Bad
  + Value int, a count of the number of Bad products produced.
* Empty
  + Key Empty
  + Value int, a count of the number of Indexes that the head was empty during.
* Indexes
  + Key Indexes
  + Value int, a count of how many times the machine has indexed
* UOM
  + Key UOM
  + Value string, Unit of Measure for the NAED produced.
* NAED
  + Key NAED
  + Value string, The NAED of the product being produced.
* Time
  + Key TIME
  + Value string, The time whatever occurred, in the format yy-mm-dd-hh:mm:ss
* Status
  + Key Status
  + Value int, current status of the machine. 0 being unscheduled downtime 1 being PM and 2 being running
* MReason
  + Key MReason
  + Value int, Reason the machine went down. Either in the form of a description or error code.
* UReason
  + Key Good
  + Value int, UReason User description as to why the machine when down.
* Any Errors
  + Key Error/Column name
  + Value int, 0 for false 1 for true reports whatever the error is under sql for the error name in the short time statistics page and to mde as the mde error bit it is set to use.
* Theo
  + Key Theo
  + Value int, Theoretical index count of a machine.
* Head\_number
  + Key Head\_number
  + Value int, Reports what head produced the last part for short time statistics.
* IntTimeInSeconds
  + Key IntTimeInSeconds
  + Value int, The number of times in seconds a setting is to last. If 0 it is ignored. It is also an optional field any time it is missing or 0 it is assumed permanent and will only be changed by another packet being sent or the app.config being changed.
* Sendbool
  + Key Sendbool
  + Value int, 1 allows packets to be processed and outputs dispatched. 0 The service will deque packets but will not process or dispatch any messages.
* LoggingLevel
  + Key LoggingLevel
  + Value int, 1: will only report errors 2: will only report errors and connection attempts 3: medium level reporting and diagnostic 4:Everything Currently 5:anything in the future that gets added that is deemed mostly useless but possibly needed.
* ListenBool
  + Key ListenBool
  + Value int, 1 will set the service to listen to the configuared input while 0 will disable input for the designated time. Does not affect control packets
* Code
  + Key Code
  + Value int, the value of the code that caused the crash.
* Message
  + Key Message
  + The exact SQL that will be run on the server. Currently selected database is not always the same so include a using.