Numurus SDK – NEPI-Bot Interface Control Document

Table 1: Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Changes |
| DRAFT-A | 11/13/18 | J. Maximoff | Initial Draft (OoT-specific) |
| DRAFT-B | 1/3/19 | J. Maximoff | Updated for consistency with other SB2 and OoT docs and to include feedback from the rest of the team. |
| DRAFT-C | 2/5/19 | J. Maximoff | Updated to bring in line with latest HPP/LPP ICD and current feature set. Clarified JSON structure and added example JSON files for data and config. |

Contents

[Purpose 3](#_Toc371418)

[General Interface Specifications 3](#_Toc371419)

[General File Format 4](#_Toc371420)

[Filesystem Structure 4](#_Toc371421)

[Concurrency 5](#_Toc371422)

[Archiving Serviced Files 5](#_Toc371423)

[File Descriptions 5](#_Toc371424)

[sys\_status.json 5](#_Toc371425)

[<node\_id>\_meta.json, <node\_id>\_std.json, <node\_id>\_change.json 7](#_Toc371426)

[action\_seq\_<INDEX>.json 9](#_Toc371427)

[task\_< INDEX>. json 10](#_Toc371428)

[smarttrig\_cfg\_<INDEX>.json 11](#_Toc371429)

[Common SmartTrigger Config. Params. 12](#_Toc371430)

[Accel-Mag-Thresh SmartTrigger Configuration 13](#_Toc371431)

[Accel-Freq-Interval 13](#_Toc371432)

[Gyro-Mag-Thresh SmartTrigger Configuration 13](#_Toc371433)

[Heading-Interval SmartTrigger Configuration 13](#_Toc371434)

[Batt-Charge-Thresh SmartTrigger Configuration 14](#_Toc371435)

[Geofence-Prox-Thresh SmartTrigger Configuration 14](#_Toc371436)

[sensor\_cfg\_<INDEX>.json 14](#_Toc371437)

[Configurable Sensor Gain 16](#_Toc371438)

[GPS Config Params 16](#_Toc371439)

[smarttrig\_rule\_<INDEX>.json 17](#_Toc371440)

[proc\_node\_cfg\_<INDEX>.json 18](#_Toc371441)

[geofence\_cfg\_<INDEX>.json 20](#_Toc371442)

[Coordination Scripts 21](#_Toc371443)

# Purpose

This document specifies the interface between the Numurus SDK (ROS-based application set) and the Numurus NEPI-Server (on-device cloud server), designated hereafter as the Interface. The initial draft covers the complete interface for the DARPA Ocean-of-Things (OoT) program, but the intention is to provide sufficient interface flexibility to support other projects and programs.

The NEPI-Server must support cloud communications via a number of network, data link, and physical layers (OSI layers I, II, and III). The Interface is designed to abstract this variability from the NumSDK by presenting a single standard software interface to the NumSDK. Similarly, the Interface is designed to abstract the internal workings of the NumSDK, including the ROS IPC framework, from the NEPI-Server.

# General Interface Specifications

The interface is file-based, consisting of a set of predefined filesystem paths, predefined file formats, and basic application controls (presented as executable shell scripts) available to the NumSDK and NEPI-Server for communication and coordination. In general, the NEPI Server is the master application, able to start and stop the NumSDK processes as necessary to reload updated configuration via Interface-specified utility scripts. Additional coordination capabilities are generally restricted to existence or non-existence of various filesystem nodes.

The following diagram illustrates the role of the NEPI-Server including top-level functional description of the interface (arrows leaving the NEPI-Server block).

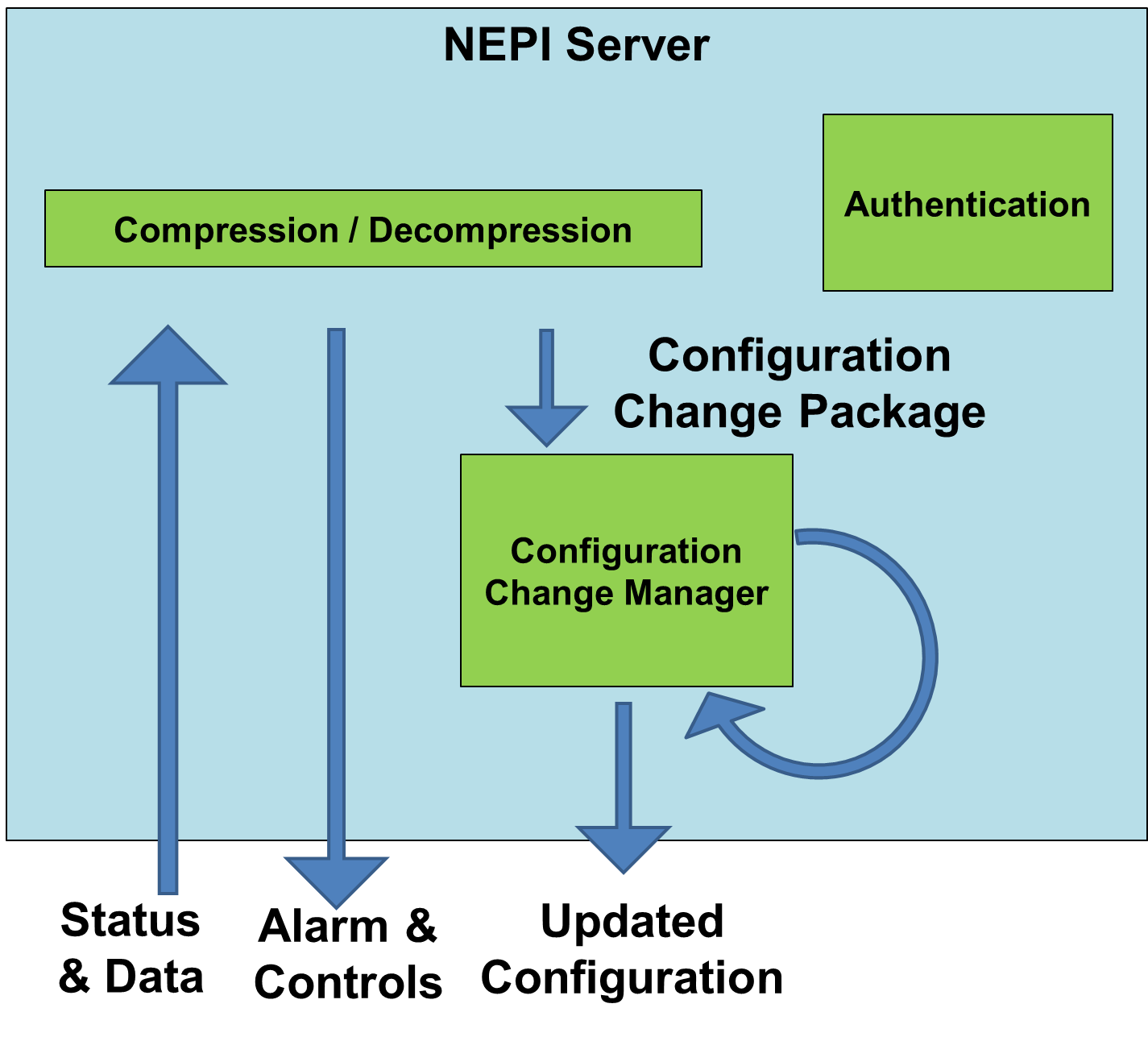


Figure 1: NEPI-Server Block Diagram

Subsequent sections describe these functional interface elements in greater detail.

# General File Format

All files (besides binary data) are JSON formatted, allowing adoption of a litany of 3rd-party FOSS serialization/deserialization libraries covering any modern programming language to be integrated for rapid development. Individual file formats are described in later subsections.

**Note: JSON is chosen over YAML as the interface represents a *data exchange* mechanism, not a configuration system; hence, many of the important features provided by YAML are not relevant in this application. YAML remains the preferred configuration file format for Numurus software components.**

Generally, binary data is provided separately from the JSON-formatted interface files (with implicit or explicit paths within the JSON to corresponding binary data). If binary data is required within a JSON interface file, Base64 encoding is to be used.

# Filesystem Structure

The following table provides paths to the various files involved in the interface. Dynamic components of filesystem names are denoted by angle brackets. In particular, many filenames include a timestamp, which is denoted in the table below by <DATE>, where DATE is in a YYYY\_MM\_DD\_HHMMSS.sss. format. Other filenames include a running index (denoted <INDEX>) to support a primitive configuration management mechanism. The index values increment (independently per root folder) throughout the mission, hence are non-repeating.

Table 2: Interface File List

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Interface Element** | **Root Folder  (relative to /home/nepi-usr/)** | **Sub-directory** | **Filename(s)** | | **Notes** | |
| Config. | cfg/ | bot/ | *TBD* | | Not part of interface, NEPI-Bot internal | |
| action/ | action\_seq\_<INDEX>.json | |  | |
| sched/ | task\_< INDEX>. json | |  | |
| trig/ | smarttrig\_cfg\_<INDEX>.json | |  | |
| sensors/ | sensor\_cfg\_<INDEX>.json | |  | |
| rules/ | smarttrig\_rule\_<INDEX>.json | |  | |
| proc\_nodes/ | proc\_node\_cfg\_<INDEX>.json | |  | |
| geofence/ | geofence\_cfg\_<INDEX>.json | |  | |
| Data | data/ | *<Date1>* | sys\_stat*. json, <Node\_ID>\_data.json, <Node\_ID>\_std.<ext>, <Node\_ID>\_change.json* | | Node IDs include those for any SDK node with data file output enabled. Processing Nodes). Associated binary data is stored with specific extension ( most often .json as an output of OpenCV FileStorage class). | |
| *<Date2>* | *…* | |
| *…* | *…* | |
| *<Daten>* | *…* | |

## Concurrency

As an IPC mechanism, files are generally unsafe due to concurrent access issues. File integrity is maintained by requiring that both NumSDK and NEPI-Server use Linux system file locks for any access to the interface filesystem nodes. This requires additional logic to ensure that these processes will wait/poll for exclusive access whenever a required file is locked by another process.

## Archiving Serviced Files

Each file in the interface represents a unidirectional communication transaction between NumSDK and NEPI-Server, with one of these two components acting as a sender and the other acting as receiver. As the receiver processes each of these transactions, it must move the file from the root folder to an *archived* subfolder. Subsequent deletion of archived files for filesystem maintenance is TBD.

# File Descriptions

The following tables detail the contents and format of individual files specified in Table 2. Each file is JSON formatted using the simplest possible hierarchical structure – generally files consist of a single JSON object with multiple string/value pairs, with exceptions noted below.

Absence of a specified/expected element should be treated as a data error and handled gracefully. Forward compatibility should be maintained by simply ignoring unexpected and unknown elements.

## sys\_status.json

The *sys\_status* contains general system state and status information to be uploaded to the Cloud. One *sys\_status* file is generated per data collection event (scheduled or event-triggered) and stored alongside the resultant data files in a timestamped directory. All fields are mandatory.

Table 3: sys\_status contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **JSON Type** | **Range/Format** | **Description/Notes** |
| *timestamp* | number | Unix Epoch with millisecond resolution | Represents the wakeup time (when status initially collected) and should match the containing directory’s name. |
| *serial\_num* | string |  |  |
| *sw\_rev* | string |  | Specifies overall sw\_rev for the system. Independent of configuration index values. |
| *navsat\_fix\_time* | number | [0.0, Unix Epoch Max] | Unix Epoch Time for last satellite fix |
| *latitude* | number | [- 90.0,90.0] | Positive is north of equator |
| *longitude* | number | (-180.0,180.0] | Positive is east of prime meridian |
| *heading* | number | [0.0,360.0) | Magnetic North |
| *batt\_charge* | number | (0.0,100.0] | Charge remaining (percentage) |
| *bus\_voltage* | number | TBD | Main bus voltage |
| *temperature* | number | TBD | Main temperature, degrees C |
| *trig\_wake\_count* | number | [0,inf) | Number of times Zynq has awoken due to sensor detection triggers |
| *wake\_event\_type* | number | 0: alarm, 1: trigger | Specifies the event (trigger or alarm) that initiated the data collection. |
| *wake\_event\_id* | number | *Schedule line item* for alarms, *smarttrigger\_id* for trigger | Interpretation depends on the value of the *wake\_event\_type* field |
| *task\_index* | number | [0,inf) | Most recent task schedule update index. |
| *trig\_cfg\_index* | number | [0,inf) | Most recent SmartTrigger configuration index |
| *rule\_cfg\_index* | number | [0,inf) | Most recent SmartTrigger rule modification index |
| *sensor\_cfg\_index* | number | [0,inf) | Most recent sensor configuration index |
| *node\_cfg\_index* | number | [0,inf) | Most recent Node configuration index |
| *geofence\_cfg\_index* | number | [0,inf) | Most recent Geofence configuration index |
| *state\_flags* | number | 32-bit mask. Contents TBD | E.g., temperature and storage warnings, drag line state |



## 

The following example file shows a typical sys\_status.json.

Table 4 - Example sys\_status.json

|  |
| --- |
| {  "timestamp": 1549481009.373,  "serial\_num": "SN\_OOT\_0001",  "sw\_rev": "20190405\_2",  "navsat\_fix\_time": 1549480005.276,  "latitude": 33.186302,  "longitude": -118.526740,  "heading": 17.52,  "batt\_charge": 67.4,  "bus\_voltage": 10.6,  "temperature": 32.6,  "trig\_wake\_count": 346,  "wake\_event\_type": 0,  "wake\_event\_id": 7,  "task\_index": 41,  "trig\_cfg\_index": 12,  "rule\_cfg\_index": 17,  "sensor\_cfg\_index": 2,  "node\_cfg\_index": 26,  "geofence\_cfg\_index": 6,  "state\_flags": 1025  } |

## <node\_id>\_meta.json, <node\_id>\_std.json, <node\_id>\_change.json

The data files are generally split into a meta-data component, an associated *standard* data component, and an optional *change* data component, all residing in the timestamp-identified subfolder of *data* . The timestamp is associated with the time of data collection/processing (node-type dependent).

These files include the *node\_id* in the filename. The *node\_id* consists of a 3-character type identifier (each processing pipeline node type has a unique identifier, followed by a numeric index to differentiate separate instances of a given pipeline node\_type. An example node\_id is *spc\_4*, indicating type SPC (spectrogram) and instance ID 4.

Table 5: Data File Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **JSON Type** | **Range/Format** | **Description/Notes** |
| *node\_type* | string | 3-char identifier | Matches the type identifier component of the *node\_id* in filename. Inserted here for convenience. **Mandatory** |
| *instance* | number | [0, 15] | Matches the numeric index from the *node\_id* in the filename. **Mandatory** |
| *timestamp* | number | Unix Epoch Timestamp. Millisecond resolution | The timestamp of the start of data collection. May differ from the containing folder’s timestamp due to differences in when *sys\_status* info is captured and when data is acquired. **TBD Policy about sending these timestamps topside – probably will not for bandwidth limited applications.** |
| *heading* | number | [0.0,360.0). Mandatory | Current reading. Magnetic North. **Mandatory** |
| *quality* | number | [0.0,1.0]. Mandatory | Data-dependent indicator of quality. Used for prioritizing data. **Mandatory** |
| *node\_id\_score* | number | [0.0,1.0] | Node\_id-dependent indicator of value (provided by a config. param for all pipeline nodes). Used for prioritizing data.  **Mandatory** |
| *data\_file* | string | <node\_id>\_std.json | The *standard data product* associated with this *node\_id*. This file is JSON, but may contain Base64-encoded binary.  **Mandatory** |
| *change\_file* | string | <node\_id>\_change.json  or “NC”: No computed change. | The *change data product* associated with this node\_id. This file is JSON in TBD format. **Optional. If not present, this node\_type does not provide change data and NEPI-Bot should operate solely on its standard data product for all downstream processing.** |

The following example file shows the meta-data component of the output from the sole temperature sensor (processing node).

Table 6 - Example <node\_id>\_meta.json

|  |
| --- |
| {  "node\_type": "tmp",  "instance": 0,  “timestamp”: 1549481009.251  "heading": 16.86,  "quality": 0.77,  "node\_id\_score": 0.25,  "data\_file": "tmp\_0\_std.json",  "change\_file": "NC"  } |

The format and contents of the *<node\_id>\_std.json* and *<node\_id>\_change.json* are intentionally not specified, though they are generally emitted by the *cv::FileStorage* class in OpenCV. It is expected that NEPI-Bot treats these files as opaque, with no regard for their specific contents.

## action\_seq\_<INDEX>.json

Action Sequences represent a sequence of modes and/or controlled actions that are executed in response to either a SmartTrigger or a Task Schedule Alarm. The config. file structure for action sequences is provided in the table below. When SDK processes the file, the action sequence specified by *action\_seq\_id* is completely overwritten with the new configuration.

Table 7 - Action Seq. Config

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Range/Format** | **Description/Notes** |
| *action\_seq\_id* | | number | [0, 31] | Identifies which of the 32 available action sequence line items this file configures. |
| *actions* (JSON array of action objects. Up to 8 action objects in array) | *action\_id* | number | Enum: See Table 9 - Action IDs [0, uint\_max) | Specific action id |
| *max\_duration* | number | Max duration allowed for this action in seconds |

Table 8 - Example action\_seq\_cfg\_<INDEX>.json

|  |
| --- |
| {  "action\_seq\_id": 5,  "actions": [{  "action\_id": 1,  "max\_duration": 20  },  {  "action\_id": 2,  "max\_duration": 600  },  {  "action\_id": 5,  "max\_duration": 1000  }  ]  } |

The following table provides the action\_id enum values.

Table 9 - Action IDs

|  |  |  |
| --- | --- | --- |
| **Name** | **Enum Val.** | **Description** |
| WiFi Mode | 0 | Power up WiFi chip and wake HPP to conduct WiFi session |
| Collect/Process Mode | 1 | Wake HPP to collect data, process, and save data for subsequent upload |
| Uplink Mode | 2 | Wake the HPP to execute the NEPI-Bot task to upload prioritized data and download new configuration and commands |
| Drag Line Release | 3 | Release the drag line. If the drag line has already been released, this action has no effect. |
| Delay | 4 | Simple action to add a pause state to an action sequence. Delay time is controlled by the *max\_duration* parameter of the action’s data structure. |
| Scuttle | 5 | Perform the mission-terminating scuttle operation |

## task\_< INDEX>. json

Individual files represent task schedule line-item changes or additional line items. Index increments from 0 over full mission timeline. The task schedule entry files may specify either a new task item or a modification to an existing item. These entries may specify one-shot events or periodic events.

Table 10: task\_cfg contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Element Name** | **JSON Type** | **Range/Format** | **Description/Notes** |
| *line* | number | [0, 255] | Task schedule line number. If this number already exists in the task schedule, this file represents a modification to that existing task. Otherwise, this is a new task. |
| *action\_seq\_id* | number | [0, 31] | Specifies the action sequence associated with this task. |
| *start\_time* | number | [0.0, Unix Epoch Max] | Time of first execution, seconds since Epoch |
| *period* | number | [0.0, float\_max) | Seconds between task executions. Not present if this is a one-shot task. (WiFi Mode is always treated as a one-shot task). |
| *max\_repetition* | number | [0, int\_max) or -1 to repeat indefinitely | Max number of times a periodic task can repeat. Not present for non-repeating tasks. |

Table 11 - Example task\_cfg\_<INDEX>.json

|  |
| --- |
| {  "line": 12,  "action\_seq\_id": 14,  "start\_time": 1549480005.276,  "period": 7200.0,  "max\_repetition": -1  } |

## smarttrig\_cfg\_<INDEX>.json

Each SmartTrigger type has a unique ID (see Table 14: SmartTrigger IDs) and a fixed number of instances (differing per SmartTrigger type) as specified in Table 14: SmartTrigger IDs and Counts. Each SmartTrigger config file identifies the type and specific instance, and provides a variable length collection of param/value pairs.

Table 12 - SmartTrigger Configuration

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Range/Format** | **Description/Notes** |
| *smarttrig\_type* | | number | Enum: See Table 14: SmartTrigger IDs and Counts | Specifies which sensor this configures. |
| *instance* | | number | Zero-indexed. See Table 14: SmartTrigger IDs and Counts |  |
| *params*  (JSON array of param *objects.* Max array size is smarttrigger-dependent. Seine individual tables below) | *param\_id* | number | Enum - See individual param tables below | Specifies which (smarttrigger-dependent) param\_id is addressed by this object |
| *value* | number | See individual param tables below | Value to set for this param\_id |

The following example demonstrates the 6th instance of the *Geofence-Prox-Thresh* being reconfigured such that it is enabled and the trigger is asserted whenever the float is within 2km of any polygon in the Geofence region.

Table 13 - Example smarttrig\_cfg\_<INDEX>.json

|  |
| --- |
| {  "smarttrig\_type": 8,  "instance": 6,  "params": [{  "param\_id": 0,  "value": 1  },  {  "param\_id": 2,  "value": 2000.0  },  {  "param\_id": 3,  "value": 255  }  ]  } |

Table 14: SmartTrigger IDs and Counts

|  |  |  |  |
| --- | --- | --- | --- |
| **SmartTrigger Name** | **Enum Value** | **Instance Count** | **Description** |
| Accel-Mag-Thresh | 0 | 4 | Threshold trigger for accelerometer. Magnitude of 3-axis acceleration. |
| Accel-Freq-Interval | 1 | 4 | Threshold trigger for accelerometer. Highest intensity Frequency of Z-axis (wave period). |
| Gyro-Mag-Thresh | 2 | 4 | Threshold trigger for gyro. Magnitude of rotational rate. |
| Heading-Interval | 3 | 4 | Trigger for detecting heading in a wedge of interest |
| Hotel-Mic-Thresh | 4 | 1 | Threshold magnitude for the OoT Hotel Microphone. Threshold value not configurable, implemented in hardware. |
| Txdr-Thresh | 5 | 1 | Threshold magnitude for the OoT Transducer Mission-Specific-Sensor. Threshold value not configurable, implemented in hardware. |
| SBand-Thresh | 6 | 1 | Threshold trigger for the OoT SBand RF receiver. Threshold value not configurable, implemented in hardware. |
| Bat\_Charge\_Thresh | 7 | 4 | Low-side threshold for battery charge |
| Geofence-Prox-Thresh | 8 | 32 | Threshold trigger for when within proximity of a geofence polygon |
| GPS-Fix-Failed | 9 | 4 | For tracking the rate of failed GPS attempts. |

Individual SmartTriggers have unique configurable parameters. The following subsections detail each of these.

### Common SmartTrigger Config. Params.

Every SmartTrigger includes a generic *enabled* configuration parameter that is always provided at *config\_param* enum value 0. When set to 0, the corresponding SmartTrigger will be ignored in all SmartTrigger processing. Each SmartTrigger configuration also includes a *trig\_stats\_interval*, which provides the sample count over which trigger assertion frequency is computed.

Table 15: Common SmartTrigger Config. Params

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| enabled | 0 | uint | 0 or 1 | N/A | If 0 (disabled), this SmartTrigger is unused and will never generate a wake-up event. |
| trig\_samp\_interval | 1 | uint | [0, 255] | samples | Total number of samples in an interval over which trigger level statistics are computed. |

### Accel-Mag-Thresh SmartTrigger Configuration

The Accel-Mag-Thresh SmartTrigger provides a SmartTrigger based on the magnitude of the acceleration vector from the embedded accelerometer. The following table provides the specific configuration parameter set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| mag\_thresh\_val | 2 | float | [0.0,float\_max) | g (gravitational force) | Threshold magnitude to assert SmartTrigger. |

### Accel-Freq-Interval

The Accel-Freq-Thresh SmartTrigger provides a SmartTrigger based on the frequency of directional changes in the Z-axis within a zero-centered hysteresis band. For example, a 1Hz rate represents the Z-axis acceleration crossing above +hyst\_radius and below -hyst\_radius over the course of 1 second.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| freq\_intv\_min | 2 | float | [0.0,float\_max) | Hz | High frequency interval limit to assert SmartTrigger. |
| freq\_intv\_max | 3 | float | [0.0,float\_max) | Hz | High frequency interval limit to assert SmartTrigger. |

### Gyro-Mag-Thresh SmartTrigger Configuration

The Gyro-Mag-Thresh SmartTrigger provides a SmartTrigger based on the magnitude of the rotational rate vector from the embedded gyroscope. The following table provides the specific configuration parameter set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| mag\_thresh\_val | 2 | float | [0.0,float\_max) | deg/s | Threshold magnitude to assert SmartTrigger. |

### Heading-Interval SmartTrigger Configuration

The Heading-Interval SmartTrigger provides a SmartTrigger based on the current heading of the SB2 as reported by the integrated magnetometer. The SmartTrigger is activated whenever the current heading falls within the specified range. It is re-armed whenever the heading falls outside the specified range for the specified quiescent period.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| heading\_min | 2 | float | [0.0,360.0) | deg. mag. north | Smaller heading in the wedge definition |
| heading\_max | 3 | float | [0.0,360.0) | deg. mag. north | Larger heading in the wedge definition |

### Batt-Charge-Thresh SmartTrigger Configuration

The Batt-Charge-Thresh SmartTrigger provides a means to identify a low charge state and take actions based on this recognition.

The configurable parameters are as follows:

Table 16 - Batt. Charge Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| low\_thresh | 2 | float | [0.0,1.0) | % of charge | Specifies the battery charge value below which this trigger is asserted. |

### Geofence-Prox-Thresh SmartTrigger Configuration

The Geofence-Prox-Thresh triggers whenever the LPP detects that it is within the threshold distance of the region(s).

Table 17 - Geofence Proximity Configuration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| low\_thresh | 2 | float | [0.0,inf) | m | Distance below which this trigger is asserted |
| polygon\_id | 3 | uint | [0,15]: individual polygons, 255: Entire geofence region (union of polygons) | index | Polygon of interest, 255 for all. |

## sensor\_cfg\_<INDEX>.json

Some system sensors have configurable parameters. These are modified through sensor configuration files, which specify updates to one or more sensor parameters. The sensor to be configured is identified by the filename through its sensor id (See Table 20 - Sensor IDs).

Table 18 - Generic sensor\_cfg format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Range/Format** | **Description/Notes** |
| *sensor\_id* | | number | Enum: See Table 20 - Sensor IDs | Specifies which sensor this configures. |
| *params*  (JSON array of param *objects.* Max array size is sensor-dependent. See individual tables below) | *param\_id* | number | Enum - See individual param tables below | Specifies which (sensor-dependent) param\_id is addressed by this object |
| *value* | number | See individual param tables below | Value to set for this param\_id |

The following example file sets the Hotel-Mic adjustable gain to 20dBm.

Table 19 - Example sensor\_cfg\_<INDEX>.json

|  |
| --- |
| {  "sensor\_id": 3,  "params": [{  "param\_id": 2,  "value": 20  }]  } |

Table 20 - Sensor IDs

|  |  |  |
| --- | --- | --- |
| **Sensor Name** | **Enum Value** | **Description** |
| Accelerometer | 0 | 3-axis accelerometer (linear acceleration) |
| Gyroscope | 1 | 3-axis gyroscope (rotational rate) |
| Magnetometer | 2 | 3-axis magnetometer |
| Hotel-Mic | 3 | Microphone with adjustable gain stage and threshold detection circuit |
| Txdr | 4 | OoT Transducer Mission Sensor with adjustable gain stage and threshold detection circuit |
| SBand-Radar | 5 | Threshold trigger for the OoT SBand RF receiver. Threshold value not configurable, implemented in hardware. |
| GPS | 6 | System GPS |
| Battery-Charge | 7 | Battery level sensor |
| Temperature | 8 | System temperature sensor |

Table 21 – Generic Sensor Settings

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **JSON Type** | **Value Range** | **Units** | **Description** |
| enabled | 0 | Number | 0 or 1 | N/A | If 0 (disabled), this sensor will not be powered (if on switched power), no data will be sampled, and no SmartTriggers derived from this sensor’s outputs can be activated. |
| samp\_period | 1 | Number | [sensor\_min – float\_max) | Hz | Sampling period for the sensor input in seconds. This may be either direct sensor data or the sensor trigger output depending on sensor. |

### Configurable Sensor Gain

A subset of sensors includes the ability to configure the sensor gain stage. This collection, namely the *Hotel-Mic*, *OoT-Txdr,* and *OoT-SBand* sensors can be configured according to the following table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **JSON Type** | **Value Range** | **Units** | **Description** |
| gain | 2 | Number | 20, 30, 40 | dB | Gain is selected from this small discrete set. Values outside this set will be rejected. |

### GPS Config Params

The GPS sensor includes configurable params as described below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Param. Name** | ***config\_param* Enum Value** | **Type** | **Value Range** | **Units** | **Description** |
| max uptime | 2 | Number | [0, uint\_max) | s | Max number of seconds to maintain GPS power before which a GPS fix attempt will fail, powering down the GPS and asserting the GPS-Fix-Failed trigger |
| max fix per session | 3 | Number | [0, uint\_max) | count | Max number of entries to add to the GPS data buffer per GPS sample period. Receiving fewer than this number does not affect GPS-Fix-Failed trigger, which asserts only if NO fix is acquired. The GPS will remain on until either the on\_time expires or the max\_fix\_per\_session count is achieved. |

## smarttrig\_rule\_<INDEX>.json

*SmartTrigger Rules* provide a means by which individual SmartTriggers are combined in software to aggregate triggers from multiple sources, perform trigger frequency checks to limit transient triggers, and execute specified actions based on the result.

The collection of SmartTrigger Rules are organized as a list, with each rule having a unique line number (akin to the task list). New rules should specify unused line numbers.

The following tables describe the organization of files used to add new or modify existing SmartTrigger Rules.

Table 22 - SmartTrigger Rule Config

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Value Range** | **Units** | **Description** |
| rule\_id | | Number | [0,63] | N/A | Rule set line number. If this number already exists in the rule set, this item represents a modification to that existing task and only the *enabled* field is required, all others optional. Otherwise, this is a new task and requires a set of premises, *action*, and *min\_period*. |
| enabled | | Number | 0: False, 1: True | N/A | Establishes whether this rule is active. |
| premises   (JSON *array* of premise *objects*. Up to 8 premises per rule) | smarttrig\_id | Number | Enum: See Table 14: SmartTrigger IDs and Counts | N/A | SmartTrigger type ID for this premise |
| instance | Number | See Table 14: SmartTrigger IDs and Counts | N/A | Instance of this smarttrigger type. |
| freq\_thresh | Number | [0.0, 1.0] | % (decimal rep. | Assertion frequency as a sample percentage over *smarttrig\_id’s trig\_samp\_interval* that must be exceeded for premise *n* to evaluate true |
| action\_seq\_id | | Number | [0, 31] | N/A | Action sequence to execute if complete rule premise evaluates true |
| min\_period | | Number | [0.0, float\_max) | s | Minimum period between successive executions of the *action* from evaluation of this rule |

The following example file shows a SmartTrigger Rule being enabled, which performs the following logical rule check

IF  
 *Accel-Freq-Interval* instance *2* is asserted with at least *10%* frequency   
 AND  
 *OoT-Txdr-Thresh* (sole instance) is asserted with *100%* frequency,  
 THEN  
 *Execute action sequence 17* AT MOST  
 every 1 hour

Table 23 - Example smarttrig\_rule\_<INDEX>.json

|  |
| --- |
| {  "rule\_id": 10,  "enabled": 1,  "premises": [{  "smarttrig\_id": 0,  "instance": 2,  "freq\_thresh": 0.1  },  {  "smarttrig\_id": 5,  "instance": 0,  "freq\_thresh": 1.0  }  ],  "action\_seq\_id": 17,  "min\_period": 3600  } |

## proc\_node\_cfg\_<INDEX>.json

The SDK’s configurable data processing capability is composed of a collection of processing nodes arranged into a series of disjoint directed tree structures (i.e., a *directed forest* in graph-theoretic terms), each of which is rooted at a SmartSensor (raw data producing node). Each path through the forest is referred to as a processing pipeline. Configurability is provided by parameters that control the individual behavior of the nodes and parameters that control the directed interlink between nodes.

Interlinks between nodes are implemented as a publish/subscribe IPC mechanism, where subscribers must know who they are subscribing to, but publishers need not be aware of their subscribers. Hence, the interlinks are well-defined by assigning to each node a source node identifier (i.e., a publisher). Raw data producers that form the roots of the directed trees do not require a source node – their data comes directly from sensors by way of SmartDrivers.

Table 24 - Processing Node Cfg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Range/Format** | **Description/Notes** |
| *node\_type* | | string | 3 char lowercase string | Specifies which node type this configures. **Mandatory** |
| *instance* | | number | [0, 15] | The specific instance identifier of the node\_type. **Mandatory.   Note: Raw data producers are limited to a single instance (hence this element has value 0).** |
| *src\_node\_type* | | string | 3 char lowercase string | Specifies the data source (i.e., publisher node) for this node. **Ignored for raw data producers**. |
| *src\_instance* | | number | [0, 15] | The specific instance identifier for the src\_node\_type. **Ignored for raw data producers**. |
| *nepi\_output* | | number | 0: False 1: True | Determines whether this node’s output is saved to the /data folder, as detailed in data file section |
| *node\_score* | | number | [0.0, 1.0] | A score associated with data produced by this node. Provided in metadata files and used in NEPI-Bot data prioritization, but otherwise unused and unmodified by the processing node’s perspective. |
| *params*  (JSON *array* of param *objects*. The max length of the array is node-type dependent). | *param\_id* | number | Enum: node-type-specific | Identifies which node parameter this entry sets. Any valid *param\_id* that does not appear in the array is left unchanged. |
| *value* | number | value: param-id-specific | Provides the value for the specified *param­\_id*. |

The following example processing node config file sets the second instance of a spectrogram node to receive its input data from the sole instance of a transducer node, specifically disabling data output to the /data folder and setting the frequency bin count (first parameter) to 64 and skip\_count (second parameter) to 8.

Table 25 - Example proc\_node\_cfg\_<INDEX>.json

|  |
| --- |
| {  "node\_type": "spc",  "instance": 1,  "src\_node\_type": "txd",  "src\_instance": 0,  "nepi\_output": 0,  "params": [{  "param\_id": 0,  "value": 64  },  {  "param\_id": 1,  "value": 8  }  ]  } |

## geofence\_cfg\_<INDEX>.json

The geofence feature allows for configuration of geographic areas of interest, where specific actions may be taken and near which the GPS duty cycle is automatically increased. The geographic areas of interest are defined by polygons consisting of ordered lat/long vertices. These polygons are configurable through the file structure described in this section.

Table 26 - Geofence Cfg

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element Name** | | **JSON Type** | **Range/Format** | **Description/Notes** |
| *polygon\_id* | | number | [0, 15] | Identifies the specific polygon in the geofence set configured by this file. |
| *enabled* | | number | 0: False, 1: True | Specifies whether this polygon is actively tracked. Disabling unused polygons saves processing time and power. |
| *vertices* (JSON array of vertex *objects*. **Up to 64 entries per file**) | *lat* | number | decimal degrees, positive north, equator centered | Each lat/long pair is a JSON object in the array that specifies a vertex. The array entries (vertices) are ordered such that subsequent entries form a polygon edge, with the final entry presumed to form an edge with the first entry. |
| *long* | number | decimal degrees, positive east, prime meridian centered |

The following example file sets the 12th geofence polygon to be the Bermuda Triangle.

Table 27 – Example geofence\_cfg\_<INDEX>.json

|  |
| --- |
| {  "polygon\_id": 11,  "enabled": 1,  "vertices": [{  "lat": 25.761681,  "long": -80.191788  },  {  "lat": 18.465540,  "long": -66.105736  },  {  "lat": 32.307800,  "long": -64.750504  }  ]  } |

## Coordination Scripts

The following table specifies the collection of scripts available to the NEPI-Server to coordinate with NumSDK. All scripts are located in /opt/numurus/ros/nepi-utilities.

Table 28: Coordination Scripts

|  |  |  |
| --- | --- | --- |
| **Script Name** | **Args** | **Description** |
| *process\_updates.sh* | Directory (one or more):  *sched*  *trigger-cfg  rule-cfg*  *node-cfg  all* | Forces NumSDK to process the files in the specified folder(s), moving them to *archive* subdirectories upon completion. |