First Responder UAS Data Gatherer Challenge (UAS 6.0): Guidance for Stage 3

Revision 2.0

March 26, 2025

Disclaimer

Any references to commercial entities, products, services, or other non-governmental organizations or individuals in this document are provided solely for the information of individuals using this document. These references are not intended to reflect the opinion of NIST, the Department of Commerce or the United States, or its officers or employees. Such references are not an official or personal endorsement of any product, person, or service, nor are they intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose. Such references may not be quoted or reproduced for the purpose of stating or implying an endorsement, recommendation, or approval of any product, person, or service.

Information, data, and software referenced in this document is “AS IS.” NIST MAKES NO WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT AND DATA ACCURACY. NIST does not warrant or make any representations regarding the use of the software or the results thereof, including but not limited to the correctness, accuracy, reliability or usefulness of the software. You are solely responsible for determining the appropriateness of using and distributing the software and you assume all risks associated with its use, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and the unavailability or interruption of operation. This software is not intended to be used in any situation where a failure could cause risk of injury or damage to property. NIST SHALL NOT BE LIABLE AND YOU HEREBY RELEASE NIST FROM LIABILITY FOR ANY INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE), WHETHER ARISING IN TORT, CONTRACT, OR OTHERWISE, ARISING FROM OR RELATING TO THE SOFTWARE (OR THE USE OF OR INABILITY TO USE THIS SOFTWARE), EVEN IF NIST HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Introduction

The First Responder Uncrewed Aircraft Systems (UAS) Wireless Data Gatherer Challenge (UAS 6.0) tests the plausibility of using a UAS as a communication relay device to gather information from sensors or Internet of Things (IoT) devices and transport that information to a centralized Command Server. This document provides guidance on how to best prepare for the UAS 6.0 Stage 3 live event. Please see the Official Rules for further guidance. Thoroughly review this document and refer to it often ahead of the live event. Contestants are also encouraged to continue testing in their home environment ahead of the live event to increase the probability of success in the live event.

Stage 3 Overview

This stage will consist of the live contest at Muskatatuck Training Center (MuTC) from April 7-10, 2025, in Butlerville, Indiana. Stage 2 winners will travel to the MuTC site no later than April 7 in order to compete in Stage 3. All contestants will be required to complete a UAS safety check and compliance review prior to demonstrating their solution's capabilities. Contestants who pass the safety check and compliance review will be evaluated based on their performance across test methods and mission trials within operational scenario mockups. They will be evaluated by a panel of subject matter experts (SMEs) and judges, and based on those scores, first-, second-, and third-place contestants will receive cash prize awards from judges. Additionally, Stage 3 offers up to 8 Best-in-Class Awards.

Live Contest Preparation

Travel Logistics and Schedule

For the most up-to-date information on Stage 3 travel logistics and schedule, refer to https://firstresponder.ezassi.com/#/challenges/details/4. Please reach out to psprizes@nist.gov and jcarlson@ensembleconsultancy.com with questions prior to or during the event.

Shipping

If contestants need to ship equipment to the site prior to arrival, shipments can be sent to the following address:

Muscatatuck Urban Training Center
ATTN: Jason Carrier
4230 East Administration Drive
Butlerville, IN 47223

Please include "NIST PSCR UAS 6.0 Challenge" in the return address and notify Haley Molchan at hmolchan@indiana.edu of the shipment's estimated time of arrival.

UAS 6.0 Test Procedures and Evaluation

Stage 3 test procedures will contain many of the same test metrics, thresholds, and apparatuses as those used in the Stage 2 evaluation. As in Stage 2, contestants will have multiple avenues and a wide spectrum of potential to score points. Contestants should not discount a test or see it as a failure if they can not fully accomplish all of the tasks or finish a test. Some contestant solutions may score higher in some areas and lower in others. While it is possible to exceed in every category, the degree of difficulty will be greater, and the demand will be higher to fully accomplish all tasks. One of the goals of the competition is to push technology limits and see where it falls short. These limits represent key research data points that may be used to identify gaps in public safety UAS technology and to further advance innovation.

Unlike Stage 2, contestants will not need to construct test apparatuses and do not need to film their flights. Additionally, a leaderboard will not be available for score comparison. Scoring values, will not be revealed in Stage 3; however, the scoring weights for each test will be as follows:

Test 1: Positive Aircraft Control - 15%
Test 2: Endurance - 15%
Test 3: Collision Avoidance - 20%
Test 4: Survey Acuity - 50%

A similar methodology will be used for raw score data collection. Score outputs and results may be released after the competition as a publication or report.

Contestants must complete either Test 1 or Test 2 before any other test. Event staff will coordinate this, and the tests will be scheduled so that one of the first two is completed before a contestant attempts Tests 3 or 4. Contestants will be scheduled to run each of the 4 tests at least once.

If a contestant is unable to continue competing due to a crash, UAS failure or any other reason, this will not disqualify them from receiving a prize. The contestant’s cumulative score up to the crash or failure event will be considered their “final” score for the event. Example: A contestant accomplishes three tests and crashes in the fourth final test. The cumulative score for that contestant will be the cumulative results from Tests 1 through 3 and any points scored in Test 4 before the crash or failure.

Best-in-Class awards will be given in the following categories:

- Best Single Scenario Score (four)– Awarded for best score per test.
- Best Affordable Solution – Awarded to best overall scoring team with a total BOM of less than $10,000 USD.
- Best Highly Portable Solution – Awarded to best overall scoring team with a solution that fits within the U.S. Domestic carry-on baggage size limit of 56 x 36 x 23 cm (22” x 14” x 9”) and weighs no more than the maximum weight for portable equipment as specified in

MIL-STD-1472F Section 5.11 of 16 kg (35 lb). If multiple solutions meet or exceed these requirements, Judges may determine the single best overall solution in this category.
- First Responder's Choice Award (two) – Awarded to top two contestants who receive the greatest number of votes from first responders in attendance at Stage 3.

Each individual test will be given approximately one hour to complete provided any constraints described in the following descriptions:

Test 1: Positive Aircraft Control

- For Stage 3, Stage 2's Test 1: Positive Aircraft Control and Test 2: Download and Inspect have been combined.
- Contestants do not have to run the test twice; it is now simply called Test 1.
- Contestants must inspect all 20 buckets (four bucket stands, five buckets each) as described in Test 2 in the Stage 2 Guidance document or the PAY-4 test method.
- Pilots or a team assistant, such as a visual observer, will call out the alphanumeric values to the test administrator, and more points will be given for each level of concentric ring identified.
    - Unlike Stage 2, contestants may discern and call out ring positions up to the 5th Landolt-C ring.
    - More points will be awarded for each smaller ring identified, per bucket.
    - When calling out, you will first identify and call out the alphanumeric, then you will call out the position of each ring, e.g. top-center, right, bottom-left, left, etc.
- Contestants will have a full hour to run the test, but those who complete the test faster will receive more points (less time is better.)
- Data collection will occur with two Sensor Clients and scored as discussed in the paragraph at the end of this section.
- Points will be awarded for each photo delivered to the Command Server, up to 20 photos.
- Photos of Landolt-C rings only have to resolve to the second ring.
- The maximum time in flight allowed will be approximately one hour and/or one battery pack, whichever comes first.
- Battery swaps will not be allowed for this test.
- Contestants may choose to end the test at any moment, and the cumulated score will be tallied; however, no points will be awarded for the time metric.
- If the contestant is unable to finish the course due to low battery, the cumulated score up to that point will be tallied; however, no points will be awarded for the time metric.
- This test does not have to be flown autonomously, and no additional points will be awarded for autonomous flight.
- Points will be awarded based on bucket alignment, time in flight, and accuracy.

Test 2: Endurance

- The Endurance Test will be run only once in Stage 3.
- Contestants will not have to perform the loiter or hover-in-place test, only the lap test.
- The maximum time in flight allowed will be approximately one hour and/or one battery pack, whichever comes first.
- Battery swaps will not be allowed for this test.
- Contestants may choose to end the test at any moment, and their score will be calculated at the time of landing. There will be no penalty for landing early.
- Unlike Stage 2, this test allows autonomous flight; please see the autonomous flight section below for details.
- Scoring factors will include the number of laps flown, autonomy, and data collected.

Test 3: Collision Avoidance

- The Collision Avoidance Test will be run similarly to Stage 2 but will involve more maneuvers and multiple objects.
- Data collection will occur with two Sensor Clients.
- The maximum time in flight allowed will be approximately one hour and/or one battery pack, whichever comes first.
- Battery swaps will not be allowed for this test.
- This test allows autonomous flight; please see the autonomous flight section below for details. Scoring will reflect full, partial, or no autonomy features as in Stage 2.
- Scoring factors will include the number of laps flown and data collected.

Test 4: Survey Acuity

The Survey Acuity Test is considered the “final” or “simulated scenario” test event. As such, the most points and weight will be awarded in this portion of Stage 3. In this test, you will “hunt and gather” Sensor Clients and imagery data, and return it to the Command Server.

- Contestants are allowed to land and change or swap battery packs as many times as necessary.
- The UAS must return to the takeoff/landing area to swap packs.
- Contestants will be given approximately one hour to complete the test.
- No redos or remediations will be allowed for this test after the scheduled one-hour test.
- This test allows autonomous flight, please see the autonomous flight section below for details.

Landolt-C Targets (Survey Acuity):

- Landolt-C rings will be placed throughout the course, and points will be gained for each ring photographed.
    - Locations of the Landolt-C targets will not be revealed before the competition.
    - Pilots or a team assistant, such as a visual observer, will call out the alphanumeric values to the test administrator. More points will be given for each level or concentric ring identified.

        - Unlike Stage 2, contestants may discern and call out ring positions up to the 5th Landolt-C ring.
        - More points will be awarded for each ring identified.
        - When calling out, you will first identify and call out the alphanumeric, then you will call out the position of each ring, e.g. top-center, right, bottom-left, left, etc.
    - Landolt-C targets only have to be called out once. Subsequent passes over the same target do not have to be called out again.
- Points will be gained for each photo Landolt-C target captured and returned to the Command Server.
    - Subsequent passes over the same target do not have to be photographed again.
    - Photos of Landolt-C rings only have to resolve to the second ring.

Sensor Clients (Survey Acuity):

- The Survey Acuity Test will include more Sensor Clients, the locations and numbers of which will not be revealed or readily apparent.
- Scoring will account for the number of Clients found and data returned to the Command Server.
- Some Sensor Clients will be accompanied by a Landolt-C target while others may not.
- Likewise, not every Landolt-C target will have an accompanying sensor.

Autonomous Flight

Tests 2, 3, and 4 evaluate autonomous flight. Unlike Stage 2, human interactions will not be counted per instance but instead will fall into three categories: full autonomy, partial autonomy, and no autonomy. Below are examples of each:

1. Full autonomy – The pilot enters a flight plan, and the UAS completes the course without human interaction or intervention.
2. Partial autonomy – The pilot enters a flight plan but has to interrupt or make corrections to the plan or manually control the UAS mid-flight. However, the pilot is able to continue the flight plan after the intervention.
3. No autonomy – Pilot controls UAS for the duration of the flight.

Note:

- The autonomous flights must account for at least 20% of the flight time to be considered partial autonomy and 100% to be considered full autonomy.
- Takeoff and landing will not be evaluated as part of the autonomous flight.
- Scorers have the discretion to award points based on performance.
- Low battery behavior is handled the same as in Stage 2.

Data Collection

Data collection and analysis will be evaluated and scored differently in Stage 3. In this stage, NIST is looking for the following:

- Whole systems that can provide actionable data.
- Rather than counting individual JSON data points, the ability to collect, manipulate, process, and deliver data using diverse systems.
- Teams will be evaluated on their system's ability to interoperate with the NIST-provided Command Server and accurately collect data for the Sensor Clients
- Contestants may use their own server to display their system's capabilities; however, contestants must have a way to export data in the NIST-requested format for scoring purposes.
- While not required, contestants may stream pilot controller video (HDMI out) to either a local monitor, remote video management system or live-stream. This capability may give better preference for the First Responders' Choice Awards.

Event Scheduling

The following is a high-level draft schedule of the Stage 3 live event. A detailed schedule will be provided upon check-in. Schedules may be subject to change, and contestants affected by any alterations in the schedule will be notified by email and/or phone. Upon check-in, please ensure that contact numbers and emails are provided and up-to-date.

Week Schedule

Date	**Task/Event**
07 April 2025	**1:00 pm - Arrival** Registration Safety and static tests - Afternoon Technical evaluations - Afternoon
08 April 2025	Technical evaluations
09 April 2025	Technical evaluations
10 April 2025	Backup day for weather delays* Award ceremony** Teardown Depart

*A one-day delay may be implemented for weather-related events.
**The award ceremony may be held at FDIC on April 11th (TBD)

Daily Schedule

Time	**Task/Event**
8:00 am - 8:30 am	Daily safety briefing
8:30 am - 11:00 am	Technical Evaluations - Flight activity commences

11:00 am - 1:00 pm	Lunch/Break
1:00 pm - 5:00 pm	Technical Evaluations
5:00 pm	All flight activity stops

Test Station Scheduling and Timing

Each contestant will be given approximately one hour to complete a single test.

- A contestant's timeslot at the station begins and ends at exactly the time listed in the schedule.
- If a contestant arrives early and there is no other contestant at the station, the test administrator has the discretion to allow an early start, but the contestant will only be allowed one hour from the early start time.
- Please note that contestants must be at the station and staged to fly before their designated time to utilize the entire hour. The clock starts at the contestant's designated timeslot, regardless of whether or not they are there.
- If a contestant arrives late, their schedule will not be pushed. For example, if a constant arrives 15 minutes late, they will only have 45 minutes to complete the test.
- After a contestant completes a test (their hour is complete), the contestant must vacate the area within 15 minutes so that the next contestant can set up.
- Contestants will not be given extra time to land, so they must factor this into their test time.
- Similarly to Stage 2, contestants will have two minutes to complete the data download once landed.This will not be counted towards you total time.
- Points will be deducted for each minute that contestants are over time, rounded up to the nearest minute.
- Time will not be paused for any field repairs or battery swaps.
- Late arrivals past 30 minutes will not be permitted to test, and the contestant will have to coordinate with event staff to reschedule. Rescheduled testing for Tests 1 - 3 is not guaranteed, and Test 4 can not be rescheduled.

Each contestant will be given a single opportunity to compete in each test. Redos or retesting cannot be guaranteed for Tests 1 - 3, and no redos or tests will be allowed for Test 4. It is recommended that any remediation or field repair be performed within you time allotted window.

Rain and weather delays are built into the schedule to account for approximately one day's delay. Flights will not occur during rain, high winds (greater than 20mph), low visibility (less than 1 mile), or inclement weather. A fly/no-fly decision will be made after weather conditions improve. Contestants may also decide not to fly or reschedule if they do not feel comfortable with current weather conditions. Contestants who miss flights due to weather will be prioritized for rescheduling.

Event Safety Plan and Daily Briefings

An event safety plan containing standard operating procedures will be distributed before the event. This is not to be confused with the safety plan required by contestants for Stage 3 entry.

Daily safety briefings will be conducted each day at 8:00 am. These meetings are mandatory for any contestant or person attending, observing, competing or participating in the event. Most contestants will be scheduled to test each day of the event; however, in the event that a contestant is not required to come on site, they will not be required to attend the safety meeting for that day. If, for any reason, a contestant or other type of attendee plans to be on site, including practicing, they will need to attend the morning safety briefing.

Event Check-in

Contestants shall ensure NIST has final versions of the following documentation prior to the start of Stage 3 in order to compete. For reference, see the requirements listed on pages 12-14, Criterion 1, of the Official Rules document.

Required Submissions

If any of the following materials have changed from contestants' Stage 2 submissions, please submit final versions of the following materials to psprizes@nist.gov by no later than April 1, 2025 (last minute exceptions may be made depending on circumstances).

1. Photo or scan of Part 107 Certificate
2. Bill of Materials spreadsheet
3. Safety Plan Video or Documentation
4. Lost Comms Behavior Video or Documentation
5. Photo or scan of Insurance Policy

Vehicle Inspection

A visual inspection of the aircraft will be conducted to ensure flightworthiness and that the system matches the bill of materials (BOM). Contestants will have the opportunity to explain any changes or updates and demonstrate safety features during this time.

Stage 3 Wireless Data Gatherer System Guide

The goal of this section is to present contestants with enough information to properly prepare their UAS solutions for the Stage 3 Live Contest. A contestant's UAS type, level of autonomy, coordination with the ground station, data management, and means of exporting sensor data to the NIST Command Server or other third party systems (i.e., Contestant Command Server) are all part of the challenge. Your creativity and ability to solve the public safety use case should also be considered for Best-in-Class awards.

This guidance document and the API definitions are designed to assist with implementing the HTTP server; however, the goal of this Challenge is to find sensors, gather data, and share the data with first responders quickly and in a comprehensible form.

NIST Hardware for Stage 3 Contestants

NIST is mailing one FoxNode Sensor Client (see Figure 1) to each contestant for testing in their home environment shortly after Stage 2 prize awards and prior to the Stage 3 Live Contest.

Overview: FoxNode Sensor Client to Drone Server

This section describes in detail the process and interactions between the Stage 3 FoxNode Sensor Client (supplied/designed by NIST) and the Drone Server (supplied/designed by contestant); also referred to as the sensor or Client (FoxNode Sensor Client) and Server (Drone Server).

The FoxNode Sensor Client

The FoxNode Sensor is called a “Client” because it does not act as a Wi-Fi connection point. Rather, it seeks a connection with Wi-Fi networks, specifically the Wi-Fi network on the contestants’ UAS. It is not broadcasting or presenting as a connection point with an SSID, but instead, it is looking for a Wi-Fi network (from the Drone Server) to which it can connect. In this Client mode, it is very similar to a web browser on a PC that wants to connect to a Server that is accessible via Wi-Fi.

The FoxNode Sensor Client is the Adafruit ESP32-S2 TFT Feather – 4MB Flash, 2 MB PSRAM, and the brains of the sensor. (Note - as of January 2025, the model currently available on the market is the ESP32-S3 TFT Feather – 4 MB Flash, 2MB PSRAM, whose form factor and software are compatible with the ESP32-S2, which is currently out of stock.) The sensor module is built around the ESP32 microprocessor which has native Wi-Fi support. The board has a PCB antenna on the back side and a TFT color 240x135 pixel display on the front side. The board has a USB-C connector for power and supports a serial port on that connector. There is a four-pin I2C bus connector that connects to a small sensor board. There are two push buttons to reset and boot the device. There is also a battery connector which is not used in this challenge. The processor on the board runs at 240 MHz.

The sensor board includes sensors for Temperature, Humidity, Light, Pressure and a 3-axis accelerometer for orientation. There is also a Real Time Clock (RTC) with battery (SuperCap) backup. The ESP32 board and the sensor board are in a small chassis that has a window to see the display. The display shows information about the status of the board to help with debugging problems.

In Stage 3, the sensors placed in the field are powered by a 10 amp hour battery so they can operate all day long. The battery is also in a holder with the Client attached to the battery (see Figure 1).

Figure 1: Photograph image of a FoxNode Sensor Client comprising a holder, battery, and Adafruit ESP32-S2 TFT Feather – 4MB Flash, 2 MB PSRAM.

At the Stage 3 Live Contest, Sensor Clients will be strategically placed at each testing station for the four scored tests. UAS solutions will be allowed to ascend to the maximum 400-foot FAA limit with permission; however, most flights shall attempt to maintain lower altitudes below 120 feet, permitting obstacle clearing, such as treetops, maintaining visual line-of-sight of the aircraft, or as determined by the test station moderator or test requirements. As such, the UAS may not be able to “talk to” all sensors from one location, and the location of the sensors will not be known in advance. It is possible that some sensors may move locations during the time that data is being gathered. Additionally, some sensors will provide GPS location in the JSON output while others will not. This scenario is consistent with sensors being low cost (no GPS) and the moving sensor is analogous to a first responder collecting ambient conditions as the responder moves locations.

The Sensor Clients will look for the Wi-Fi network on the Drone Server flying overhead. They will look for the pre-defined SSID that the Server broadcasts from its Wi-Fi networks. When the Clients see the Server’s Wi-Fi network, they will join that network. The Clients will have a fixed IP address and, as such, will not use DHCP. The Server will also have a fixed IP of 192.168.40.20.

The Drone Server

Stage 3 contestants’ UAS solutions will have as part of the airframe a Server and a Wi-Fi network. The Wi-Fi network will have a fixed SSID so that the Clients can find and connect to

the Server network. The UAS will need to “fly around” and find the sensors. Communication between the Client and the Server is done using HTTP protocol. The Client will be using a POST method when sending information to the Server, and the Server will then respond with various information. This is similar to using a web browser where the Client (such as a laptop computer) sends information to the Server which then replies with information for display on the laptop screen.

The nature of the Server on the UAS solution is a decision made by contestants. The ESP32 Feather board can be used with the limitation that only one Client can connect to the HTTP system (socket) on the UAS solution at a time. A more robust server, such as a Raspberry Pi or similar, that can support multiple Clients connected to the HTTP system may be able to communicate with multiple Sensor Clients at the same time. The tradeoffs in server architecture are a large part of the challenge.

Sensor Client to Drone Server Communications

The nature of Client-Server communications is defined by using HTTP protocol with the POST method. This differs from a simple TCP full-duplex connection in that the Client is driving the communication process because it is the only “end” of the connection that can initiate contact with the Server using the POST method. After the Client gets the reply from the Server, it will stop trying to talk to it and go quiet for a period of time or until the signal from the Wi-Fi network gets too weak because the Uncrewed Aerial Vehicle (UAV) has flown away from the sensor. The actual mechanism is for the Client and Server to pass JSON-encoded data objects using API schemas, as outlined in the Stateful Communications description in Appendix C.

A Sample Client to Server Communications Exchange can be found in Appendix B.

A sample Stateful Communications Description can be found in Appendix C.

The Drone Server Perspective on Talking to the FoxNode Sensor Client

The Server is running an HTTP server. The Server itself is unaware of the UAS solution seeing a valid SSID at a strong signal level and also connecting to the Server Wi-Fi network. With fixed IP addresses for the UAS solution and all of the Clients, the HTTP server is unaware of the Client until it does an HTTP POST. The POST content, a JSON-encoded data object, is provided to a program that determines what response is provided back to the Client. Once that reply is sent, the HTTP server is idle.

Consider the aforementioned Client-to-Server interaction in light of the state diagram and the API JSON definitions. Consider printing these and referring to them often in preparation for the Stage 3 Live Contest. Now consider the Server's perspective.

A POST is received from the Client. In the POSTed data object, the core set of data identifies which FoxNode Sensor Client is talking to the Drone Server. The Client command is “hi”, indicating that the Client is connecting in anticipation of providing sensor values to the Server. The Client has also provided the Server with information on the time range over which it has sensor data, the number of samples it has, and other information that may be of interest to the Server. The Client may have optionally provided its location.

The Server now prepares its *Response DataObject*. The reply to the Client is a srep of "sval". The Server is requesting that the sensor values be sent from the Client. Additionally, a starting time and ending time for the sensor values are supplied. The Server sends the JSON-encoded Response Data Object back to the Client.

The first "POST and Reply" transaction has happened between the Client and Server.

The Sensor Client, which has been awaiting the reply to its POST, now processes the Server's reply. It sees the "sval" reply and creates the sensor readings JSON object. Once that object has been JSON encoded, the Client will POST it to the Server. The Client command is "rval", indicating that the Client is replying with sensor values in it. The Client will fill in all the other data that was sent when the Client was saying "hi" to the drone on initial contact.

The Server receives the POST, stores the sensor values, and if all goes well, sends back a reply of "buby" indicating that it has successfully received, processed, and stored the sensor readings it requested. It does not need anything else from the Client. It wants to end the conversation.

The 2nd "POST" and "Reply" transaction has happened between the Client and Server.

Upon receiving the Server's reply, the Client sends one more POST with the command "bye." It may or may not send the other data; typically, once the Server sees the "bye" command, it will not bother to look at that data.

Upon receiving the Client's "bye" command, the Server replies with the "done" command. This acknowledges that the Server understands that this communications session is done and that the Client will be disconnected from the Wi-Fi network and "going dark" for a minute or so.

The third and last "POST" and "Reply" transactions has happened between the Client and Server, and communication is complete.

If the Server did not need to collect sensor values, its reply to the Client's first POST would be "buby", indicating that all communications are done.

Impact of Drone Server Architecture

HTTP servers are familiar systems to programmers. One aspect of these Servers is that a typical server has many "sockets" open at a time. As such, the HTTP server can be in the process of communicating with a number of Clients simultaneously at any time.

The ESP32 HTTP "stacks" (IP stacks and HTTP server) only have one socket available when running as a stand-alone system. As such, if a Client is connected to an ESP32 server and another Client attempts to connect, it will receive an error. It is not clear if the ESP32, when running under its RTOS, can have more than one socket open. Errors returned when an HTTP server is out of sockets are in the 500 range, with 503 and 529 being most likely. Contestants using the ESP32 as the Server will need to determine how to deal with this situation.

If the Server on the UAS can have multiple sockets, such as Apache on a Raspberry Pi, then contestants must still be able to handle an "out of sockets" situation. Presumably a multi-socket server could handle a number of Clients that were in close proximity to each other concurrently communicating with the Server.

Contestants should also consider that different URLs could be used to provide Wi-Fi access to the Server for functionality other than just obtaining sensor values.

Encryption of Data

Encryption and authentication methods will not be implemented or evaluated in Stage 3. Design options are included for contestant reference or future use and implementation.

Use cases for sensor collection include scenarios where there would not be Internet access. As such, HTTP port 80 will be used. The current APIs do not deal with encryption. When writing code for this system, consider decrypting JSON strings from the Clients and encrypting JSON strings provided by the Sensor Client and to third party systems.

Should development continue beyond the UAS 6.0 challenge, a single pre-shared key may be considered for all the sensor nodes in small deployments or key distribution and management systems and/or certificate based public key infrastructure in larger deployments.

Basic Data Objects

fnid: FoxNode ID
This object is how a given FoxNode Client presents its ID and information for delivery to the Command Server. One of these would be part of a data dump, and it would most likely be included in any Client-Server request.

Data Elements – sr (‘s’ensor ‘r’eading)

Id	Unique ID. This is also colloquially referred to as the serial number. An id of zero is no unit id, an id of 999 is a broadcast value used in some of the APIs
Loc	Location. This is a GPS object
Tsc	Current timestamp for the real-time clock
Gps	GPS object, optional

sens: Sensor Readings
An object with values from all the sensors at a given point in time. When the FoxNode Client reads all of its sensors on whatever polling period is set, it records all those values. The timestamp for when the samples were taken is included just once for all the sensors.

Data Elements –

l	(lower case ‘l’) Lux values, integer, lumens
t	Temperature. If we go Centigrade, keep one digit to the right of the decimal point. If we go Fahrenheit, just simply int.
t2	Secondary temp measurement
c	Chip temperature
h	Humidity in percent.
p	Pressure in hPa Integer (NOTE: do the math and figure out if one or two digits to the right of the decimal point would give us “meter precision” altitude, albeit relative.
a	Accelerometer Object
ts	Timestamp for this set of readings.

Acel Accelerometer Data Set Object

This comprises values from the accelerometer. Note that by making this its own object, there is the possibility of added “software derived” additional data elements, such as derivatives of acceleration or a “bumped” Boolean

X - X sensor All sensors are units of milli G, integer
Y - Y sensor
Z – Z sensor

gps: GPS Reading

This one is different in that there are required data elements of latitude and longitude, and optional for elevation, accuracy, and number of satellites.

Data Elements:

Lt – latitude, in float-point format (sometimes noted as DD.ddddd). No more than 6 digits to the right of the decimal point. Remove trailing zeros if possible.
Lg – longitude
Ts – time that these readings were acquired

These are optional:

El – Elevation in meters, integer
Ac – accuracy in meters
Nsat – number of satellites

With the above data objects, a large object of sensor readings can be provided. That large object will be:

An id object
An array of sensor data measurement objects.

Data Objects Between Client and Server

The next task is to determine the data object used to communicate between the Client and the Server. As such, we need to first define the various conversations that the Client and Server will have with each other.

Dividing these conversations into two situations:

1) Talking to and getting sensor data
2) Dumping sensor data to a system, presumably at the launch point

For the first case, as the UAS flies, the Client sees the WiFi network at an acceptable signal strength. It then joins that network. Both the Server and the Client have fixed IP addresses. There are two ways to get the sensor data stored on the Server:

a) The Server sees that the Client is online and begins talking to it.
b) The Client, upon joining the WiFi network, connects to the Server via HTTP at a known port number and sends a message that is the equivalent of “Hello”. The data on the connection means that the Server is now aware of the Client. Note that the Client could

also provide information about the earliest timestamp it has for a collection of sensor readings.

c) With the HTTP TCP connection still up (keep alive setting on the TCP connections being in the 20 second range here is critical), the Server then replies to the Client with a command to send the sensor data for all readings at a “since” timestamp or greater to the Server.
d) The Client then sends the JSON encoded object with a single FoxNode ID data object, and an array of sensor values, all JSON encoded.
e) The Server then sends back an acknowledgment, and both systems drop the TCP connection. The Client, which is running a state table to know where it is in this process, then waits for the WiF signal strength to go below a threshold, enters a one-minute timeout, and goes back to looking for the Server WiFi , circling back to step (a).

The challenge with this is “can the system use HTTP to connect, but then use the connection like a classic full-duplex communications channel?”. Since this is a somewhat atypical use of HTTP, the port number should be non-standard on the Server (perhaps 560, which is not used a lot).

For the second case, as the UAS flies, the Client sees the WiFi network at an acceptable signal strength. It then joins the network on the Server. Both have fixed IP addresses. Through some means, either as part of the WiFi net driver or perhaps the Client sending a UPD message, the Server becomes aware of the presence of the Client. The Server then opens a TCP connection on the Client, and the two can talk. Note that the notion of HTTP is gone in this case, so it comes down to the relative ease of having an open TCP port on the Server and the network protocol having the ability to move the connection to another port number. As soon as the TCP connection is set up, everything goes back up to step (b) above, as the two systems send small JSON packet-encoded data objects to each other. In summary:

1) The Client sends a “Hello” data object, with its ID packet and another data object that is the oldest timestamp for the data it has.
2) While the TCP connection behaves like a full-duplex communications system, the conversation tends to go back and forth between the two ends of the connection mimicking a half-duplex communications channel. As such, once the Server receives the “Hello” data object, it sends a request for the Client to provide samples that are newer than a supplied timestamp. Note that the Server is responsible for remembering the last timestamp loaded from the Client. Also, recall that at some point later in time, either on the Server or when the sensor samples are dumped at the launch point, any duplicate readings are easy to remove.
3) The Client then sends the samples to the Server. Since all the samples are enclosed in an array in the data object, once the object is parsed and stored, both sides know the data has been delivered.
4) The Server then sends back an acknowledgment that it received the data. One second later it disconnects from the TCP connection, The Client disconnects after the reception and processing of the acknowledgment.
5) The Client then waits for the drop in signal strength so it can then go into its “blind timeout” mode.

A description of Signal Strength Considerations can be found in Appendix D.

Note that for either scenario, the process relies on a TCP connection that behaves like a half duplex communications channel – only one side is sending at a given time. As such, after sending a message to the other end of the communications channel, the sender can start a timeout process. If it does not hear from the other side within 10 seconds for example, it drops the connection and goes to the state where it does the one minute timeout wait.

A number of parameters are used with this communication schema. They will comprise another of our basic data objects:

cp – ‘c’onfiguraton ‘p’arameters

a) WiFi signal strength sufficient to allow joining the Server network.
b) WiFi signal strength that the drone has to go below to start the blind time timeout.
c) The value for the blind time timeout in seconds.
d) The maximum age to keep sensor readings.

Optional values for future development:

1) The current Unix Timestamp time.
2) The current encryption/decryption keys (note that by putting that into the cp object, we can start with a common key for all Clients, but leave the door open for a specific Client.
3) A “password” for the cp data object, since it will be sent in the clear. Like Bluetooth, there is an assumption that comms are “clear” at the launch site or at a previous point in time where the encryption key was established.
4) A FoxNode ID. For boot up, if there is no password, a packet can be sent that will establish the ID number and the password. Once a password is set, there can be no change in the FoxNode ID unless the firmware is updated (or some other serious “reset back to the stone age of a freshly flashed unit” mechanism exists).

These configuration values will need their own API. Some key concepts for this API are:

1) An address, namely the serial number of the FoxNode.
2) A broadcast address of 999 that sends to all the sensors.
3) A concept that any of the above data elements may or may not be provided. As such, the time and only the time can be set in a broadcast packet. Or, only a change in the parameters a-d above can be sent to just one unit.

All this depends on the Server being able to create a bind to a UDP port so that a message can be sent to all the UAS at once with each UAS checking the address to see if it is being addressed or if this is a broadcast address. This UDP approach is needed because TCP (or HTTP on top of TCP) is a two-end point system and has no broadcast functionality.

Note also that some program, most likely on a laptop, will be needed to provide/set this information.

A description of How the Communication APIs Work can be found in Appendix E.

API JSON Definitions

The JSON data objects are used in controlling the communication between the Client and Server as well as passing the sensor values accumulating over time. Timestamps in the system are Unix timestamps. The FoxNode ID number is used to set the FoxNode IP address. It is the “base address” of 192.168.60.20 **+** the FoxNode ID number.

Note that all of the indexes for the data elements are four (4) or less characters. This can help software locate a given data element more easily if the string is turned into an integer.

A description of API JSON definitions can be found in Appendix F.

Wi-Fi Network Topology

At each Stage 3 testing station, there will be a Wi-Fi access point and charging power for contestants to connect their Contestant Command Server, UAS Solution, and other integrated devices. The testing stations will be connected through a mesh Wi-Fi network with video streamed to a central location for contest reviewing and judging.

Additional Live Contest Considerations

Altitude limits will be moderate in the Stage 3 live event. In real-world events, the incident plan has an Aviation Sector that assigns altitudes for all aircraft, including UAS. Often, there is an altitude for UAS actively engaged in Search and Rescue, UAS for first-pass damage assessments, and UAS flying at higher altitudes for imagery photos, with the end goal of a GeoTIFF file that can be used to assess damage or incident planning.

In the Stage 3 Live Event, contestants will be asked to fly at predetermined altitudes appropriate for each test. For any test procedure contestants will be asked to not exceed 120 feet Above Ground Level (AGL.) If a contestant has justification to fly higher than 120 feet, they shall consult with the test station moderator prior to taking off. If an altitude adjustment is required to clear an obstacle, to maintain visual line-of-sight, or for any other reason that would require them to ascend above 120 feet, the pilot must consult with the test station moderator before proceeding. The only exception would be in the event of an emergency in order to avoid a crash or collision.

By design, the Client's antenna characteristics and power output are such that even if a UAS hovered at the geometric center of the course at 300 feet, it could not talk to all of the Clients. There may also be times when, at the 120-foot altitude, there will be Clients close enough to each other that more than one sensor could try to talk to the UAS. It is also possible that Client data from “unimportant” devices may be picked up from adjacent test stations. It is important to consider signal strength when collecting Client data. Points will not be deducted for collection of extraneous data.

The Client number and location will not be made available to contestants. As such, there is the challenge of finding the Clients and then managing flying the UAS around the course to collect sensor values. This is a real constraint that would be seen in a wildland fire incident where

sensors were deployed by boots on the ground or aircraft. There is not the time to deploy sensors at pre-determined locations. Sensors may not have a GPS to extend battery life.

Summary

Talk with Challenge SMEs or local first responders about the scenarios of the challenge – ask them how they would use the data, what they would like to see, etc. Be sure to listen and avoid sharing one’s own opinions on what they should do. If making changes to a UAS solution, talk with them again for more feedback. Keep in mind the Best-in-Class and First Responder's Choice Awards as well as the challenge goal and objectives.

Appendices

Appendix A - Terminology, Acronyms, and Definitions

Above Ground Level (AGL)

Above Ground Level (AGL) refers to the height or altitude of an object, such as a drone or aircraft, measured from the ground directly beneath it. AGL indicates how high an object is relative to the earth's surface at a specific location, rather than relative to sea level (which is referred to as Mean Sea Level, or MSL).

Command Server

A computer that performs data processing from external data sources and displays it in an actionable format. For UAS 6.0 Stage 3, this computer can display and collect JSON data and UAV photos.

Data Collector

A middleware component carried on the UAV collects data from sensor modules or onboard devices, such as the UAV camera, and processes and/or stores it for delivery to the Command Server. This may also be referred to as the “Drone Server,” not to be confused with the Command Server.

Javascript Object Notation (JSON)

Javascript Object Notation is a text-based format for exchanging and storing data that is both human-readable and machine-parsable.

On-Screen Display (OSD)

The on-screen display used to control a drone provides real-time information to the pilot, typically shown on a monitor, mobile device, or built into the pilot controller's screen. This display offers essential telemetry data and controls to help manage and navigate the drone effectively.

Pilot Controller

A device that the uncrewed aircraft pilot uses to maintain control and remotely control an aircraft. Also commonly referred to as “pilot interface,” “supervisory controller,” “ground control station,” “teleoperation device,” or simply “radio controller.”

Uncrewed Aircraft System (UAS)

The uncrewed system consists of the pilot controller, remote aircraft, and any supplemental devices or controls used to influence or alter the aircraft's control.

Uncrewed Aerial Vehicle (UAV)

The aircraft itself or the flight component that is being controlled via remote means is also referred to as a “drone,” “aircraft,” “remote vehicle,” or “uncrewed flight vehicle.”

Remote Pilot in Command (RPIC)

In UAS 6.0, this is the FAA Part 107 certified individual who is remotely piloting the uncrewed flight vehicle.

Return to Home (RTH)

The Return to Home (RTH) feature on a drone is an automated function that brings the drone back to a predetermined location, typically the point where it took off (the home point). This feature is activated under certain conditions, such as:

1. Low Battery: When the drone's battery reaches a critical level, the RTH function is triggered to ensure the drone can safely return before the battery is depleted.
2. Loss of Signal: If the communication link between the drone and the remote controller is lost, the drone will automatically initiate the RTH sequence.
3. Manual Activation: The pilot can manually trigger the RTH feature via the controller or a mobile app, often by pressing a designated button.

During the RTH process, the drone typically ascends to a predefined altitude (to avoid obstacles) and then flies back to the home point. Once it reaches this location, the drone descends and lands automatically.

Sensor Module

A processing unit or development board, such as an ESP-32 or Raspberry Pi, consists of a microprocessor, accessible and writable memory, optional storage, and optional networking components. The board often contains peripheral interfaces that allow the addition of external add-on modules or individual sensor components to provide additional data inputs. For UAS 6.0, this describes a processing board with Wi-Fi capability that provides environmental data from integrated or attached sensor components. This may also be referred to as an “IoT Sensor” or simply “sensor.”

Distance

In the Endurance and Autonomous Obstacle Avoidance tests, contestants will be expected to at least meet the stated distances by crossing the start and end lines. The proof is provided by observing the start/end boards as described in the test methods. It is suggested that contestants who rely on GPS or other positioning systems run past the lines enough such that any random errors in their measurements do not inadvertently cause them to run short due to positioning noise.

Horizontal Position

In the tests containing the start/end boards or bucket stands, contestants will be expected to show that they have achieved particular positions in the sky by observing the various targets and apparatuses, which are only visible from those locations. Observing the targets and apparatuses as prescribed is sufficient to demonstrate the necessary level of precision.

Altitude

Altitude is measured from the takeoff/landing pad. Contestants should use their onboard altimeter, which is expected to be accurate to within 10% of the real measurement. Any altitude measurements shall be expressed as AGL where possible.

Appendix B - Sample Client to Server Communications Exchange

A typical sequence of Client POST Sends and Server Replies might look like this:

> **(1 *Client sends*)** After connecting to the Server Wi-Fi network, the Client would send a POST with information about itself. This would include information such as its ID number, the number of sensor readings it has and the oldest timestamp for those readings, [*optionally*] its GPS information, [*optionally*] its battery status, and the like.
>
> The Server will then reply with information that acknowledges the reception of the POST from the Drone Server. It will also reply back to the Client with a request for what happens next. An example of this would be requesting that all of the sensor readings from a passed timestamp forward to another passed timestamp be sent to the Server. Other valid requests are to acknowledge the reception of values or to say “goodbye” to the Server.
>
> **(2 *Server replies*)** Having received the “hello” POST from the Client, the Server would respond by sending back a number of pieces of information. One piece would acknowledge the “hello” POST; another would request all the sensor sample values between two times be sent to the Server.
>
> At this point, a single POST transaction between the Client and the Server has been conducted. The Client, having received the reply from the Server will now start another POST transaction. This particular example describes a typical set of POST transactions:
>
> **(1 *Client sends*)** The Client now sends all of the requested sensor values that are within the times the Server specified.
>
> **(2 *Server replies*)** The Server acknowledges the receipt of the sensor values, saying it is done with the Client and a “goodbye”.
>
> There is one last step – the Client must acknowledge that the Server is done with it. So there is one last POST transaction:
>
> **(1 *Client sends*)** The Client sends a “goodbye acknowledgement”.
>
> **(2 *Server replies*)** The Server sends a “I’m done acknowledgement.”

While it may seem odd to acknowledge a “goodbye”, this is somewhat analogous to people talking on the phone. When you say “goodbye” to someone, you wait for them to say “goodbye” and then you both can hang up and know that you are done. The situation is the same here. After the Client gets the reply from Server, it will stop trying to talk to it and go quiet for a period of time or until the signal from the Wi-Fi network gets too weak because the UAV has flown away from the sensor. In all the above, when the idea of passing information or replying has been used, the actual mechanism is for the Client and Server to pass JSON encoded data objects using API schemas, as outlined below.

Appendix C - Stateful Communications Description

The above sequence where the Client provides sensor readings to the Server is an example of Stateful Communications. Here is an example where people exchange information over the telephone:

> **State 1:** An individual waits for the phone to ring.
>
> **State 2:** The individual answers the phone, and both individuals on the phone make sure they are speaking to the correct person. This is an “identification and authentication” step.
>
> **State 3:** Information is exchanged. The process is Half Duplex in nature in that typically only one party speaks at a time. Since the connection (the phone line) is full duplex (information can go in both directions at the same time), each exchange of information can be acknowledged, asked to be repeated, or interrupted at any time.
>
> **State 4:** One individual says “goodbye”, and then waits for the other individual to either say “goodbye” or “one more thing”. If the other individual says goodbye in response to the other saying “goodbye”, both hang up. If the other individual says “one more thing”, then both revert to State 3 and continue to talk (exchange information).

Nearly all communication systems that are not broadcast (radio or television) are stateful. The medium or channel used for communications can be “Full Duplex” (each end can listen and talk at the same time) or “Half Duplex” (only one end can talk at a time). A telephone is “Full Duplex” whereas radio traffic with responders is typically “Half Duplex” as the radios can not receive and transmit at the same time. If everybody is talking, no one is listening.

While technically a HTTP connection uses a TCP full duplex connection, the rules of the HTTP protocol result in making the connection look like a half-duplex channel. This is most certainly the case for the sensor system where the Client uses the HTTP POST method to talk back and forth with the Server.

State Definitions for the Sensor Client

The section above details the “states” of a telephone call. The section prior to that details a *series of transactions* that let a Sensor Client supply its sensor readings to the Drone Server. Those *series of transactions* are the states needed to get the Client readings sent to the Server.

At each state in the HTTP POST process, different information is exchanged between the Client and the Server as defined in the JSON encoded data objects by the API definitions. The software that describes this process is implemented with a “**State Machine**” approach. The state machine for the process of supplying the UAS solution with sensor readings is one of many state machines.

Another state machine manages the process of the Sensor Client “seeing” the drone Wi-Fi network SSID and joining it. This process is complex because the Client needs to ensure that the signal strength of the Wi-Fi network is above a certain threshold so that, as the UAS solution moves, communications can continue despite changes in the receiver signal strength. Signal strength will vary due to the UAS solution location, the sensor position, and the effects of

radiation patterns from antennas not being uniform. The Client needs to see good signal strength for a certain amount of time to ensure a high probability that the sensor values exchange process will not be interrupted due to low signal strength.

In this case, the two state machines need to have some sharing of information between them. The Client cannot start to POST to the Server until the “connection state machine” says that there is a good signal and the Server's Wi-Fi network has been joined.

The state machine C code for the FoxNode Sensor Client is provided so that contestants can understand in detail how the Clients are behaving. This should be valuable when debugging the Server software.

Here is a list of the State Machines in the Client:

1. Signal Strength Monitor
2. WiFi Connection
3. Sensor Dump to Server
4. Read Sensors at a defined interval
5. Manage LCD display

All of the state machines are in essence running at the same time, albeit on a single thread with one state machine having the opportunity to change state one after another on a 10 mS per cycle basis.

State machine functionality is often drawn in a graphical form with circles for each state and arrows showing the conditions/rules for transitioning to another state. The diagram becomes complex as the number of states in the machines increases. The source code does a reasonable job of explaining what the states and transition requirements are.

Typical Sensor Value Flow (after connection is made)

Once the Client has connected to the Server Wi-Fi network, this is the flow of POST states that provides the stored sensor values to the drone:

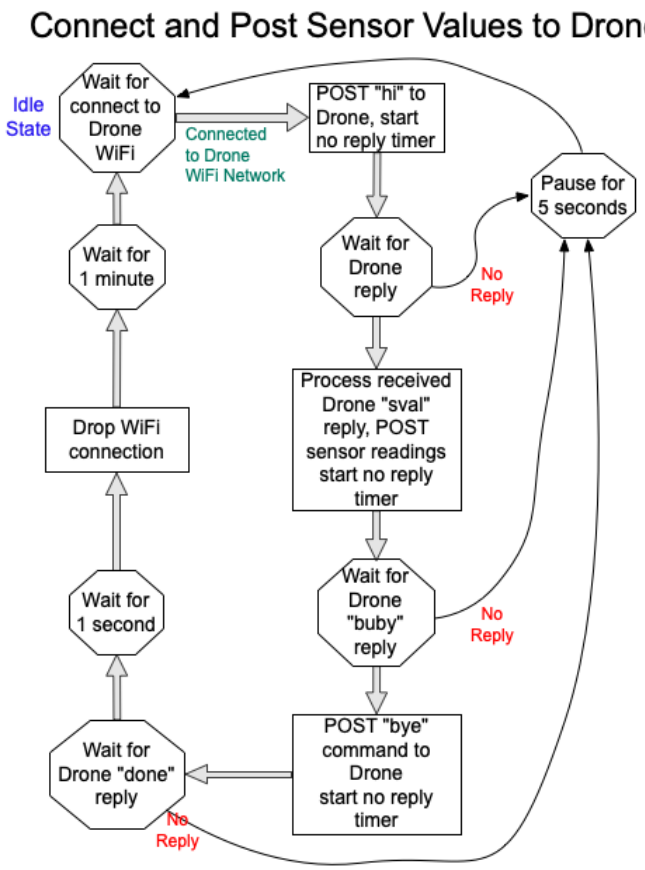


Figure 2. State machine functionality diagram depicting connecting and posting sensor values to a drone.

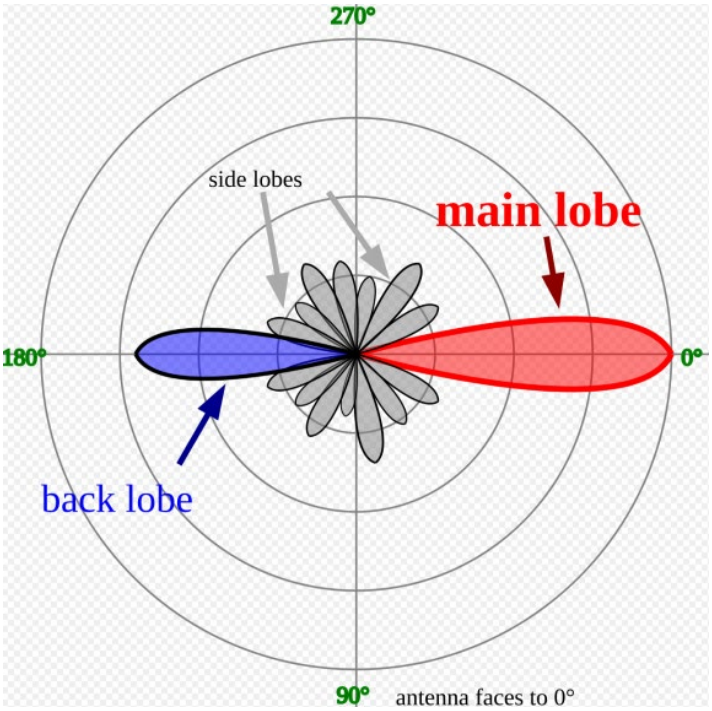
The actual state diagram is more complex than the diagram above (Figure 2). The State 3 state machine has to “talk to” / “work with” the State 2 machine in order to drop the WiFi connection, and the State 2 machine has to talk to the State 1 machine to “blind” the signal strength monitoring for various periods of time.

There are also global means to reset all the state machines. State machines also must manage and monitor global variables in the program.

Appendix D - Signal Strength Considerations

The Clients will be watching for a “good” receive signal strength from the UAS solution’s Wi-Fi network. They will need to see that strength for a few seconds to ensure that it was not due to a lobe of the antenna pattern. Only then will they connect to the Server. If, in the course of actively communicating with the Server the signal strength should drop too low, the Client will drop the connection to the WiFi net. As such, a very short HTTP Keep Alive value should be configured into the HTTP system.

The ESP32 module has the antenna behind the display. The antenna is also close to the battery pack. As such, the radiation pattern is anything but a classical pattern. Here is a “clean” antenna radiation pattern:



(source: https://morepcb.com/pcb-antennas)

The pattern from the Clients will be very distorted from this ideal donut pattern. This is why the strength of the signal needs to be “good” for a while. When flying, the signal strength will be changing quickly as the UAS moves. It may be necessary to fly slowly when locating the Clients. Flying through the side lobes it will be impossible to have good communications. To save battery power, the Clients do not continuously transmit. They are looking for the drone Wi-Fi signal.

Coordinates for a polygon of the “field” where the Clients are located will be provided prior to the event. Contestants may wish to have their software able to switch between their local test area boundaries and the provided final event boundaries. This will allow you to fly and search inside of a specified area both when developing your system and flying at the final event.

Appendix E - How Communications APIs Work

In the overview of Basic Data Objects, these data objects were defined:

1) fnid – the FoxNode ID
2) sr – a ‘S’ensor ‘R’eading
3) gps – GPS / location values
4) acel - Accelerometer reading values
5) cp – ‘C’onfiguration ‘P’arameters

For the APIs, another data object is created:

api – The data element that says what the data object is. It has to have this data element:

apid – This is a (preferably) 4 character or less way to say what API or “which element of the conversation” this data object is. For example, here are some possible apid for the above communications scenario:

helo – How either the Client or the Server introduces itself to the other side

send - how the drone requests sensor readings

data - identifies the data object with the sensor readings

ack – how the receipt of the sensor readings from the Client to the Server are acknowledged, or Acknowledge.

bye - how a connection is terminated, typically by the Server.

Cp - configuration parameters if desired that the Server is able to configure a specific sensor modules (advance capability, optional, only after everything else works)

The Client would then Ack and the Server would ‘bye’

If this overall plan looks implementable, then we can take the next step of creating a document that using the “pretty JSON” format shows what each of the data objects looks like.

Then, for each phase of the communications, we can show the JSON format for each of the apid (the API ID) that we have.

Finally, we can create a state table for the communications showing sequency, how time-outs are handled and the like.

The main tasks/decisions/existing software functionality to make or ascertain:

1) Can there be a UDP port open on the FoxNodes for configuration and time setting in addition to either the HTTP or TCP connection scheme ?
2) Do we want to use HTTP and keep alive or a “classic” TCP connection for the communications? Food for thought: If we go straight TCP, then we can leave port 80 open for classic “Post”/”Put”/”Get”/”Delete” interactions with the Drone Server at the

launch point. It would also mean that data could be collected from the Drone Server concurrently with the Drone Server communication with the Client Sensors. It would also mean that the two systems (data collection and data dumping) and independent processes.

Appendix F - API JSON Definitions

This section comprises all JSON data object definitions. There are data objects used in controlling the communication between the Client and Server as well as the data object used to pass the sensor values accumulating over time. Timestamps in the system are Unix timestamps. The FoxNode ID number is used to set the FoxNode IP address. It is the “base address” of 192.168.60.20 **+** the FoxNode ID number.

Note that all of the indexes for the data elements are four (4) or less characters. This can help software locate a given data element more easily if the string is turned into an integer.

JSON Definition for Sensor Client POST to Server

```
"core": {
        "ccmd: "hi"                 // client command
        "fox":   12,                // FoxNode ID number
        "fip":   "192.168.60.32",   // FoxNode Sensor Client fixed IP
        "fts":   1737235233,        // FoxNode current time stamp
        "fbv:    4.92,              // FoxNode battery voltage
        "fcnt:   123,               // Number of sensor sample in the client
        "ftso:  1737235000,         // 'o'ldest timestamp of sensor values
        "ftsr": 1737235180,         // most 'r'ecent time stamp of sensor values
        "wrxs": -30,                //'w'ifi receive ('rx') ‘s’trength in dBm
  [these next data elements are optional]
        "lat":   36.08914,          // latitude
        "lng":   -79.16151,         // longitude
        "elev":  186,               // MSL elevation in meters
}

   [ this typical large array of sensor values are returned when
     the Drone Server has asked for sensor reading. If no sensor
     values, this data element array is not sent]
"data": { another JSON object described below }
```

Valid commands from Client passed as “ccmd”

"hi"	First contact & info for the Server
"rval"	Sensor values reply, values are in the POST
“rlog”	Log file dump reply, values are in the POST
"bye"	Acknowledgement of Server sending "buby" after receiving sensor data

JSON definition for Server to Client in response to a POST message

Note: This is the Response Data Object

```
"serv": {
        "srep": "sval",             // Server reply to Sensor Client
        "sip":   "192.168.60.10"    // Server fixed IP
        "sts":   1737235234,        // Server current time stamp
```

```
        "stsb":  1737235050,        // beginning timestamp for values
        "stse":  1737235180,        // Ending timestamp for sensor values
}
```

Valid replies from Server back to Sensor Client as srep:

"sval"	Send sensor values for times "stsb" to "stse" inclusive
“ping”	Send the core data object again
“slog”	Send entire log file and delete contents
“conf”	Receive a configuration block
“fnoc”	Request a configuration block (fnoc is conf spelled backwards)
"buby"	Send request to disconnect to Sensor Client (think of getting off a plane, "bu-buy" from flight crew)
“done”	Reply back to the Sensor Client to acknowledge a bye command. Communication session is complete.

Most of the time the Server will use the sval reply so that it can have the sensor values sent to it. The ping request can be used to get the core data object sent again. This could be useful for seeing the receive signal strength of the drone Wifi at the Sensor Client. The drone could slightly adjust its location for better receive signal strength.

And finally, the reply can optionally have the standard AJAX data elements at the top level of the returned object.

status code: 0
status text: “OK”

Sensor values are returned with a base value for the timestamp (fts0). This is added to all the "relative times stamps" (rts) for each reading. This reduces the size of the JSON string. There is an array of objects that are associative arrays for each set of sensor values at the given timestamp. To make parsing software easier, the number of readings in the array of sensor readings is also included in the data object.

The use of short index values reduces the length of the JSON encoded values. For example, “**t**” is shorter than “**temperature**”.

Sensor readings JSON object:

```
"data": {
   “fox”:  12,                  // the FoxNode ID number
   "dlen": 42,                  // number of readings in this dump
   "ftso": 1737235000,          // base timestamp
   "dump": [                    // array of sensor values objects
            {                   // single reading object
                "rts": 20,      // relative time stamp
```

```
            "t": 52,          // temperature in degrees F, integer
            “c”: 80,          // ESP32 chip temperature F
            "h": 60,          // relative humidity, percent, integer
            "l": 430,         // light level, lumens, integer
            "p": 995,         // pressure, hPa, integer
            "x": 3,           // accelerometer X, mg, integer
            "y": 4,           // accelerometer Y mg, integer
            "z": 5,           // accelerometer Z mg, integer
        },
        { more readings },
          ....
        { last reading }
      ];
}
```

If a sensor does not have a valid reading, rather than have another data element for this, the value of the sensor is returned as a string instead of an integer. The string typically can be just “**F**” for false. In theory in a real system this could be a more descriptive string. For this Challenge, the single character string “**F**” will be used.

Base Area Sensor Dump

When performing the final challenge, the Drone Server will be flying around the test area collecting data from the Sensor Clients. The drone will need to return to the take-off area periodically and dump the sensor values it has. This is the dump of data that NIST will collect to score the contestants, and it is what would, in a real-world scenario, would be supplied to 3rd party systems. These 3rd party systems could use the data in several ways. Examples are displayed on a map, plotting the information, analyzing the information to provide real time warning and for Record Management Systems that document the incident.

This data object format must be used by all contestants. This lets NIST write the software to ingest the format once, not ten times. This is also realistic when you consider the world of APIs and data exchange standards that we live in. The data can be delivered as a JSON (preferred) or XML format file, it could be accessed from the drone over an HTTP GET or POST, it could be available in an Ajax format. It is suggested that contestants first create the data object and encode it into a text file and then focus on how that encoded data object text string would be delivered. Note that obtaining the data directly from the drone via WiFi, as opposed to having to land, power down and then extract a SD card, will improve times and scoring. The drone must return to the take-off area, but can remain at its 120 foot altitude during the dump. From a “time to run the course” perspective, if that data was only available via an SD card, the card would have to be extracted after each landing, read, the incremental file deleted, and then re-insert the card into the drone.

There are two possibilities for how the collected sensor data will be shared when the drone is back at the take-off site. A full dump or incremental dumps. Note that it is a goal of the challenge, and the real world is to dump at every return. This allows for quicker access to the

data as opposed to waiting for multiple flights and only then getting the full dump. It also allows for less loss of data should the drone crash or lose data. You would still have data from previous gathering flights. This is a very real-world concern.

The first is to dump a summary of all the data supplied thus far. This would include all the data from previous data gathering sorties. While this would grow to be a larger dump, it would allow 3rd party systems to overwrite what they had with the latest data. This requires more storage on the Server. In the case of this challenge where WiFi data rates could be used, the transmission time for the encoded data object is not an issue.

The second is to provide only an incremental dump of the data gathered during the last flight around the course to get sensor information. While this requires less storage on the drone, it also requires that the systems receiving the data are tasked with having to gather up all the incremental dumps and make a single, final data set.

The advantage of “over the air” sharing should be evident. It would be reasonable for the contestants to first get the data in a file on the Server and then focus on how to send that file electronically vs. having to remove the card. This can be a good incremental development strategy for a time that will pay off if the contestant is pressed for time to develop the software. A “GET” method would be a good way to provide this data

The data object will have some indication and summary information. Then there will be an array of sensors, each with an array of the sensor measurements. We will use the data object “data” from above. Only the dump with reconstructed “real” timestamps for each set of sensor values will be different. This allows for a lot of code re-use.

```
“bigDump”: {
        “team”: “Flying Data Miners”, // team name
        “foundCount”:  14,              // # of FoxNode Sensors found
        “sortieCount”: 8,               // # of gathering flights
        “earliestTakeOff”: <time_stamp>  // earliest take-off time
        “lastLanding”:   <time_stamp>          // last time of landing
        “firstDumpTime”: <time_stamp>       // time of first dump
                        // at take-off point
        “lastDumpTime”: <time_stamp>       // time of last dump at
                        // take-off point
        “totalSampleCount: 588,       // Total # of samples taken
                        // across all FoxNodes
        “collectData”: {          // all of the gathered data
          [
        “foxNodeId”: 8, // The ID of the foxNode for this data set
        “sValues”: [     // the array of sensor values. There are
                        // the ‘data’ objects that have been
                        // collected with one change:
                        // the ‘rts’ should now be a full Unix
                        // timestamp with an index of ‘ts’
         ]
           ],
                [
```

```
                ( more foxNode sensor data objects )
                ]
        ]
}
“sensorLocations: {             // this dataset is options
        [
        “sensorLoc”: {
        “fox”: 12,              // FoxNode number
        "lat":  36.08914,       // latitude
                "lng":  -79.16151,      // longitude
                "elev": 186,            // MSL elevation in meters
         }
],
[
        // one entry per sensor that was found
]
}
```

Admittedly, this is a somewhat complex object in that there is an array of Client data collections and for each Client there is another array of all the sensor values read. The very nature of the data is an array of sensors that hold an array of all the values over time. The organization and definition of this object has a minimal amount of dependency on other data elements in the object. The content of the “data” array stands on its own because it contains the sensor ID. That sensor ID is also included in each entry into the “collectData” object, which means software does not have to “dig down” into an array element to know which sensor’s data is in the array. Timestamps are attached to each sample which allows for the sample rate to be either a strict period, or with smart processing, only when values change enough to warrant saving them. For the challenge, whatever data was collected is the data that will be submitted in the ‘collectData’ object. Sensors will be collecting data at a constant rate.

A good name for the URL to get this info might be: 192.168.40.20/getBigDump<language-extension> where the Language Extension would be .php, .ph, etc.