USING IRIDIUM SATELLITE WITH A SENQUIP DEVICE

1. Introduction

Senquip devices have integrated Wi-Fi and 4G LTE connectivity options. In applications where neither of these options are available, a satellite modem can be connected to a Senquip device. This Application Note details the connection of an Iridium EDGE satellite communications device to a Senquip ORB. The concepts presented in this application note can be applied to any serially connected modem that uses AT-commands to communicate with the host. The Iridium EDGE satellite modem was chosen because:

- Hardware-Ready Device for simple, low risk integration

- Low-Cost Device for affordable customer adoption

- Satellite Add-On for truly global coverage

- Ready-To-Install for quick time-to-market

- Robust Power Supply

Figure 1 - Iridium Edge Satellite Modem

In this Application Note, it is assumed that the reader has an account on the Senquip Portal, has scripting rights, and that the device is connected to the Senquip Portal. It is also assumed that the Iridium modem is configured to send data to the Senquip Iridium endpoint.

2. References

The following documents were used in compiling this Application Note.

Reference	Document	Document Number
A	Iridium Edge Fact Sheet	Edge_FS_QX_QUE16_0007_23MAY17
B	Iridium Short Burst Data Developer Guide	MAN0025 Release 3.1
C	Iridium Edge User Manual	UM Edge V1.1 2017.06.07

D	ISU AT Command Reference	MAN0009, Version 5

3. Overview

The addition of a satellite modem to a Senquip device enables communication where there are no Wi-Fi or 4G LTE networks. A high-level communications diagram is given in Figure 2. In the figure, a Senquip ORB is measuring data on a machine, possibly over CAN Bus. The ORB parses received data into useful information and then sends it to the Iridium Edge modem via RS232. The Iridium Edge modem sends the data to an Iridium satellite that then forwards it to a ground based receive station. The station sends the data to the specified server via the internet.
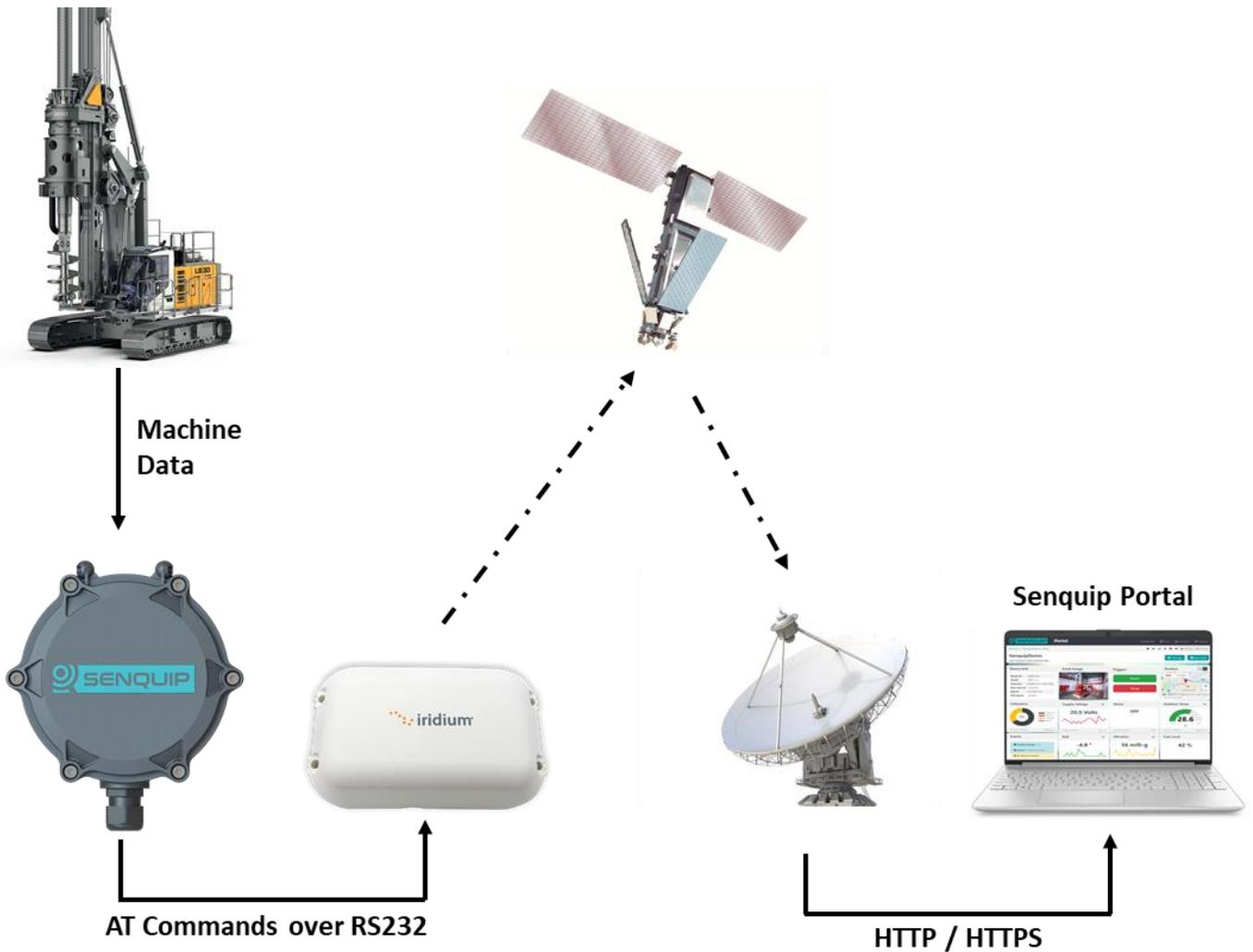

Figure 2 - Network Diagram

The Iridium base station needs to know what server (endpoint) to send the data to. This is pre-configured by your Iridium service provider. The endpoint for Iridium data for Senquip can be requested from support@senquip.com.

4. Wiring and Configuration

This section outlines how to connect an Iridium Edge satellite modem to a Senquip ORB and how to configure the serial interface on the Senquip ORB to communicate with the Iridium Edge.

The Iridium Edge used in this example shipped with an interface cable that connects to a DIN fitting on the modem. Table 1 describes the physical interface of the Iridium Edge, the cable colours and their connection to the Senquip ORB. Only power, ground and the RS232 interface are used. The *Power Down* input could be later be connected to the ORB and used to power down the modem. The *Network Available* and *Power Detection* outputs are of limited utility as this information is available via the RS232 interface.

Table 1 - Iridium Edge Connections

Iridium Edge	Wire Colour	Signal Description	Senquip ORB	Signal Description
1	White	Spare		
2	Brown	Ground	2 or 4	Ground
3	Green	RS232 input	7	RS232 Tx
4	Yellow	RS232 output	6	RS232 Rx
5	Pink	9-32V DC power in	1	Power
6	Grey	Optional Power Down input		
7	Blue	Network Available output		
8	Red	Power Detected output		

The Senquip ORB can operate with a DC power source from 10-to 75V but the supply on the Iridium Edge is limited to a maximum of 32V and so a supply between 10V and 32V should be used. A 1A fuse is inserted after the power supply which is likely to be ignition on a vehicle.
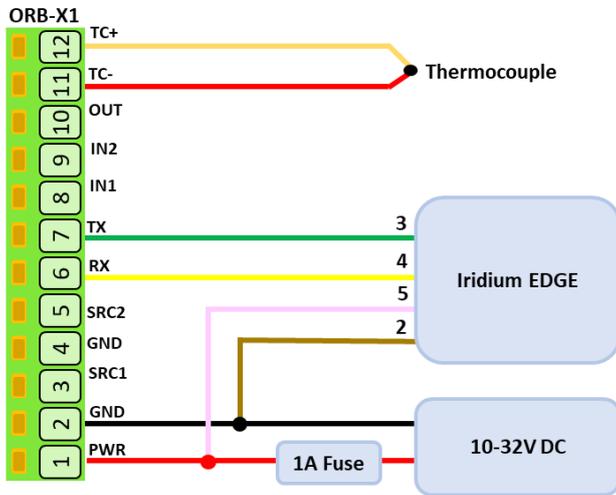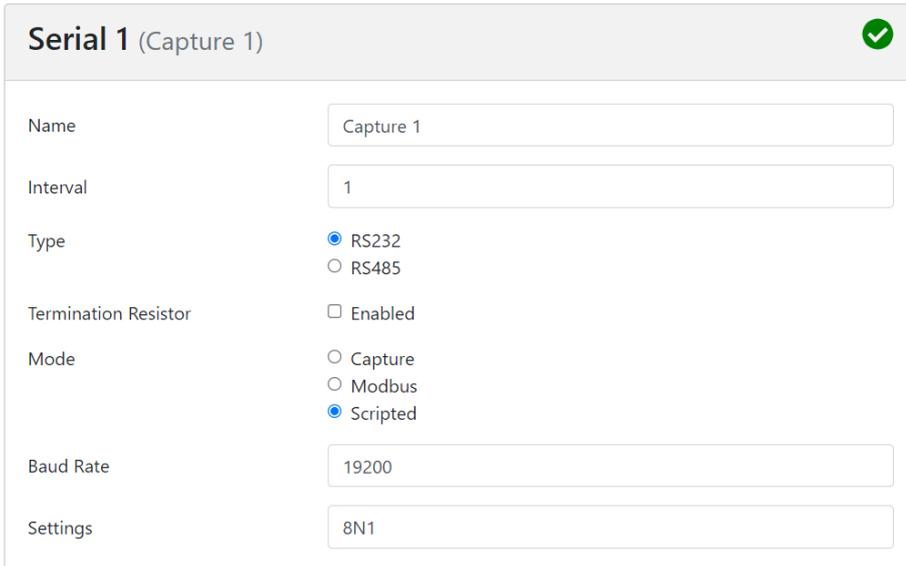
Figure 3 - Iridium Edge Wiring Diagram

From the Iridium Edge User Manual, we note the specifications of the RS232 interface as:

- Baud Rate: 19200 b/s

- Data Bits: 8
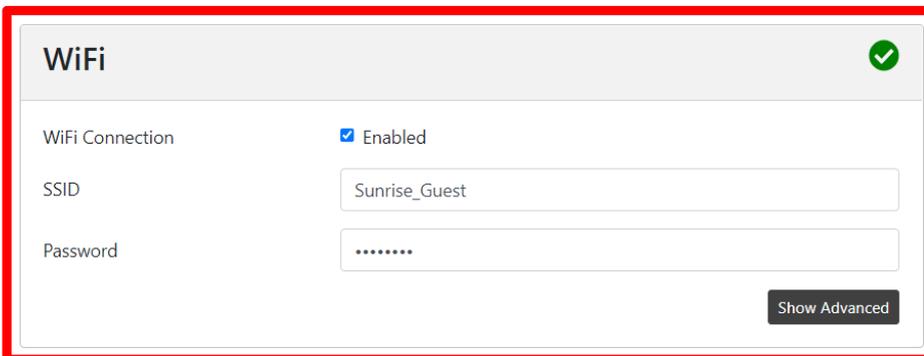
- Parity: None

- Stop Bits: 1

As shown in Figure 4, the serial port settings for the Senquip ORB are set to match those of the Iridium Edge using the Senquip Portal. Note that the serial port *Mode* has been set to *Scripted*. In *Scripted* mode, the sending and receiving of the serial port is fully controlled by a script.

Figure 4 - Senquip ORB Serial Settings

Under Network settings, the Iridium Edge option should be enabled, and the Iridium IMEI number inserted.

Figure 5 - Iridium Edge Setup

In this example, the base interval is set to 5 seconds, and a thermocouple has been enabled. The base interval and peripherals enabled have no bearing on the operation of the Iridium Edge modem and it is expected that the user will configure the Senquip device according to their requirements.

5. Writing a Script to Control the Iridium Edge Modem

This section describes the example script given in Appendix 1 to control the Iridium Edge device.

The Iridium Edge modem is controlled using AT commands. AT commands are instructions used to control a modem. AT is the abbreviation of ATtention. Every command line starts with "AT". The "AT" is the prefix that informs the modem about the start of a command line. The following AT commands are used:

AT+SBDWT: Write a message to the modem buffer. Modem responds "OK" if the messages was successfully loaded or "ERROR".

AT+SBDIX: Initiate sending the message.   Returns "SBDIX:" and a status code. Status codes can be found in the ISU AT Command Reference. The script presented takes codes 0, 1, 2 as successful.

A/: Send the last message again. Used extensively by the script to retry.

The script presented in Appendix 1 is a state machine that keeps track of the current state of the Iridium modem and tries to advance to the next state, ending in a successful transmission of a message. This script should be taken as an example only and it is expected that the user will improve it and tailor it to their application.

Three global variables are created:

iridium: keeps track of the current state of the modem.

- State 0: Ready
- State 1: Loading message into buffer
- State 2: Sending message

rxData: A buffer in which to store data received from the modem

wdog: A watchdog timer that increments every second based on a timer. If after 100 seconds, the state machine has not advanced to the next state, the transaction is cancelled and the state returns to 0. In all testing so far, the watchdog has not activated.

Four functions are defined:

iridiumSend(): Call this function to initiate the sending of a message. This function checks whether the modem is busy and if not, starts the state machine. It either returns "busy" or "ok".

Timer.set(): Defines a 1 second repeating timer that calls the function contained within it. The function increments the watchdog timer.

SERIAL.set(): Defines an interrupt to call the function contained within it when a serial character is received. This function adds the received character to the serial buffer and checks on next actions based on the state of the modem.

SQ.set_data_handler(): Defines the function that is called each time the Senquip device completes a measurement cycle. In this function, a timer is incremented and on 5-minute intervals, it attempts to send a simple Iridium message. The message is sent in JSON format and consists of the thermocouple (tc1) and input voltage (vin).

Suggested enhancements:

- Only send Iridium messages when Wi-Fi and 4G LTE networks are not available

- Additional error checking for instance before using obj.tc1 we should be checking that it exists

- Reporting status codes when a message send fails

6. Creating your Message

Senquip promotes the use of Senquip devices to send data to any server. To facilitate this, the contents of the packets to be sent via the satellite modem can be created within a script.

Some notes:

- The Iridium Edge modem has a limit of 340 bytes for the message size.

- A timestamp need not be sent if the receiving portal logs the time that the message arrived an timing does not need to be precise. The Senquip Portal does this.

- The Senquip Device ID does not need to be sent when using the Senquip Portal as the device will be identified from the Iridium IMEI number.

Text or binary messages can be created.

When using the Senquip Portal, messages should be sent in JSON format using the same keys that are used when a message is sent via Wi-Fi or 4G LTE. In that way, the data sent will end up in the same columns in the database no matter which communications method was used. When using satellite, it is suggested that the number of pairs in the JSON packet be reduced to reduce the packet size to below 340 bytes. Check with your service provider whether you are being charged for the number of characters sent or the number of messages. If the latter, you may as well fill the messages. Be careful when creating messages that you limit the number of characters in each value in a key value pair. For instance, a value may be returned as 4 one time and as 4.12345 the next.

The number of messages sent when using satellite should be limited as the cost of airtime is relatively high.

7. Conclusion

An Iridium modem can be connected to the serial port of a Senquip device and be controlled through the use of a simple script. Messages can be sent to any server in any format including to the Senquip Portal.

The concepts presented in this application note can be used to connect any external modem that communicates via serial using AT commands.

Appendix 1: Source Code

```javascript
load('senquip.js');
load('api_config.js');
load('api_serial.js');
load('api_timer.js');

let iridium = 0;   // state of the modem
let rxData = "";   // buffer for data from modem
let wdog = 0;      // not sure this is needed

let timer = 55;    // do what you want here to determine when you want to send a message

// call this function to send a message via iridium - returns ok or busy
function iridiumSend(message) {
  if (iridium >0){return "busy";} // still trying to send a previous message
  if (typeof message === 'string') {
    let m1 = "AT+SBDWT="+message+"\x0D\x0A"; //load buffer message (max 120 bytes)
    SERIAL.write(1,m1,m1.length); // load buffer
    iridium = 1;
    wdog = 0;
    return "ok"; // received string ok
  }
}

// just in case we get stuck in a continuous loop - could flag a device error if the wdog times
out
Timer.set(1000, Timer.REPEAT, function() {
  wdog++;
  if (wdog >100){
    iridium = 0;
    wdog = 0;
  }
}, null);


// Configure the serial receive interrupt
SERIAL.set_handler(1, function(channel) {
  let r = SERIAL.read(1);
  rxData = rxData + r;   // Keep adding to data string as characters arrive
  if(rxData.indexOf("\x0D\x0A")>-1){  // look for <CR><LF>

    if (iridium === 1){    // check buffer loaded correctly
      if (rxData.indexOf("OK")>-1){ // return from modem contains "OK" then buffer is loaded
        SERIAL.write(1,"AT+SBDIX\x0D\x0A",10); // send the message
        iridium = 2;
        wdog = 0;
        debug1 = "1: "+rxData; // only for debug
      }
      else if(rxData.indexOf("ERROR")>-1){ // buffer did not load correctly
        SERIAL.write(1,"A/",2); // try load buffer again
      }

    }

    else if (iridium === 2){ // check the data was sent
      let t2 = rxData.indexOf("SBDIX:"); // check the SBDIX message state
      if (t2>-1){
```

```javascript
      let mo = rxData.slice(t2+7,t2+9); // mo is the status returned by the SBDIX message
      if (mo === "0," || mo === "1," || mo === "2,"){
        iridium = 0;  //successful - go back to idle state
      }
      else{
        SERIAL.write(1,"A/",2); // send the message again
      }
      wdog = 0;
    }
  }

  else{
    iridium = 0;
  }
  rxData = ""; // clear the incomming buffer
  }
}, null);


// The function that is called when a measurement cycle is completed
SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);

  timer++;
  if(timer > 60){ // send every 5 mins
    let msgObj = {
      ts: obj.tc1,
      vin: obj.vin
    };
    let message = JSON.stringify(msgObj);
    SQ.dispatch(1,iridiumSend(message));
    SQ.dispatch(2,message);
    timer = 0;
  }
  SQ.dispatch(4,iridium); // monitor for testing
  SQ.dispatch(6,timer);
  SQ.dispatch(7,wdog);

}, null);
```