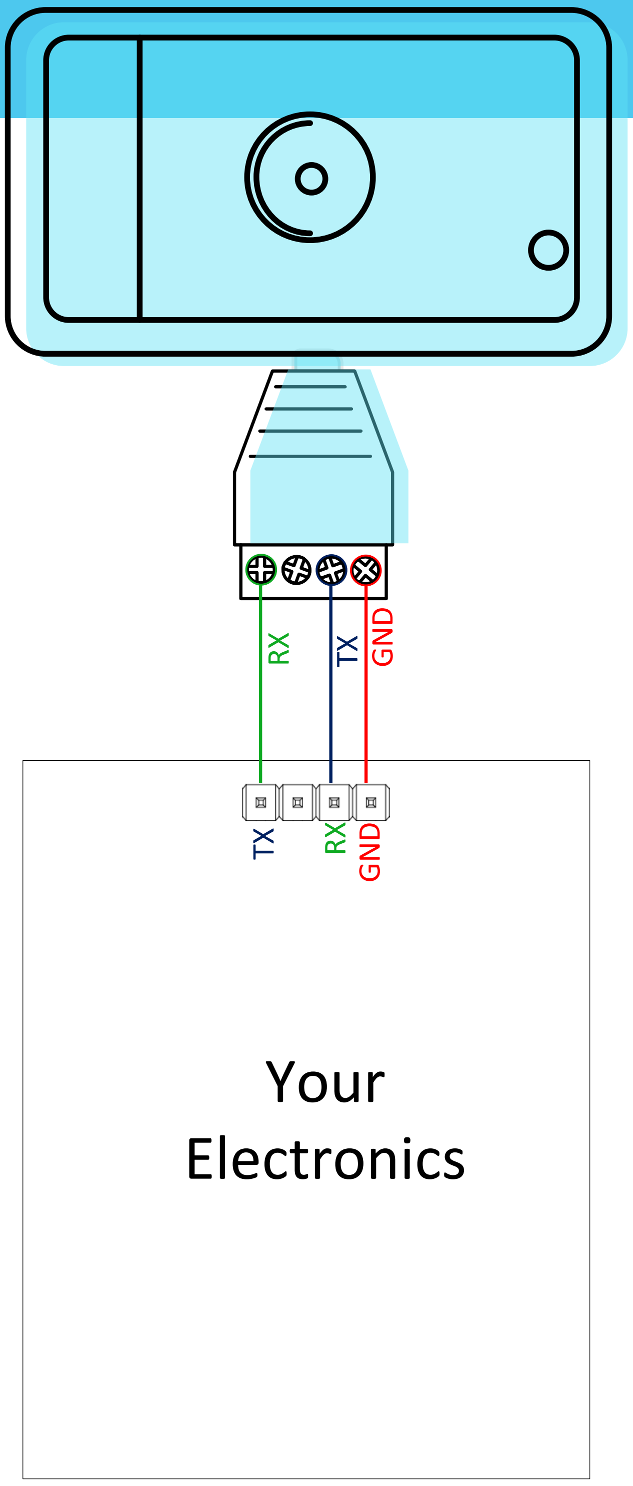
Proteus

DB

Introduction to Proteus Digital Bridge

Proteus DB provides a simple and convenient way to connect your digital systems to the cloud. With a json data structure, you can send data on multiple nodes in a single shot to the cloud. You can also set your own key value pairs on the cloud to update specific values on your system.



Connections

Proteus DB comes with a 4 contact terminal block interface. This provides contacts for Ground, TX and RX lines for the UART communication line between your board and Proteus. Leave the 4th wire disconnected.

Voltage Rating

Proteus DB runs on a 3.3V system. It is best if your UART lines operate on 3.3V as well. Proteus can handle upto 5V DC, though it is recommended to have a level shifter to ensure reliable and stable communication.

Data

Data must be send in JSON format through the uart. The keys are mandated by Proteus. You can set your values which can be letters, numbers or a combo. To ensure sanity of data transmission, encoding and decoding , we strongly recommend against use of special characters in the data payload. See sample cases in next page.

Emails, Calls, Text Alerts

You can add your phone numbers, multiple emails to the notification list. No matter where you are, you can be in the know when one or more of your data nodes go to alarm.

Testing Your Sensor

After the installation is complete, depending on the type of load you are monitoring, read further on how to configure the settings, and test your sensor.

Baud Rate

For your convenience, Proteus supports multiple baud rates that can be configured via the web interface. The default baud rate is set at 19200. Other options include 4800, 9600, and 38400.

Data Structure

You can send Data corresponding to one ore more elements in your system. Simply encode them in the correct format, send it to UART and Proteus will take care of it from there on.

When you have a single data node

Following is the data structure when you have just a single data node.

```
{
        "SensorList": [
          {
                "id": "101",
                "data": 110.5,
                "unit": "V",
                "name": "AC Line Voltage",
                "status": "OK",
                "desc": "Normal"
          }
        ]
}
```

Contents inside the red dashed box represents data corresponding to a single data element that you want to monitor. Let’s dive deep into this single node.There are few keys that in each node that are required (indicated in RED), and few that are optional (indicated in GREY).

```
"id": "101",
"data": 110.5,
"unit": "V",
"name": "AC Line Voltage",
"status": "OK",
"desc": "Normal"
```

For example: If you are monitoring a node that just has a Normal and Alarm Status, then you don’t need to provide any data or units. Your data node could look like

```
{
        "SensorList": [
          {
                "name": "AC Line Voltage",
                "status": "OK",
                "desc": "Normal"
          }
        ]
}
```

The ID key is also optional that you could use to keep track of your monitored nodes.

Mandatory Keys

There are 3 mandatory keys required for each sensor node you wish to monitor.

A. name
This key defines the name of your node. Proteus uses for display on the cloud, form alarm messages as well as for verbose data and alarm logs.

Name can be any alpha numeric string.

B. status
This key defines whether this specific sensor node is in Alarm or OK state. Hence only two states are supported for this key, “Alarm” or “OK”.

C. desc
This key is a descriptive word (preferably one word) that makes more sense to the status key. For eg: Let’s take the case of the AC Line Voltage Monitor. When status is “OK” you can set “desc” as “Normal” or “OK”. If the Line voltage is reading 105.4, You can set “desc” as “Low”. Alternatively, if the voltage is reading 117.2, you can set “desc” as “High”.

Proteus uses the desc key along with name key to form alarm messages and display tags on the cloud.

Non-Mandatory Keys

There are also optional non- mandatory keys that you can use for each sensor node.

A. id
This key defines an internal ID for your node. Proteus currently does not rely on this ID.

id can be any alpha numeric string.

B. data
This key defines numeric data if a particular sensor node generates numeric data. If present, will be used for display, logs, graphs, alarm messages, etc.

C. unit
This key defines the unit for the numeric data. If present, will be used for display, logs, graphs, alarm messages, etc.

D. kvPairs
In an attempt to be purely client agnostic, Proteus also lets you define your own variables and their values that can be passed back to your system. This means that you can set key value pairs on the cloud and update their values on the go. Any changes on these values will prompt the cloud to sync them back to your system. You can use them to adjust thresholds, timers, delays or pretty much anything you want.

When you connect a new sensor, the cloud has no clue to your system variables that you may want to update. While you can always manually add key value pairs on the cloud, Proteus goes one step further. You can include a kvPairs key with any one sensor node that contains all variables for your system, just once. Proteus will use this to initialize the variables and values so you don't have to do it for each sensor. Eg:

"kvPairs":"var1=val1&var2=val2&…"

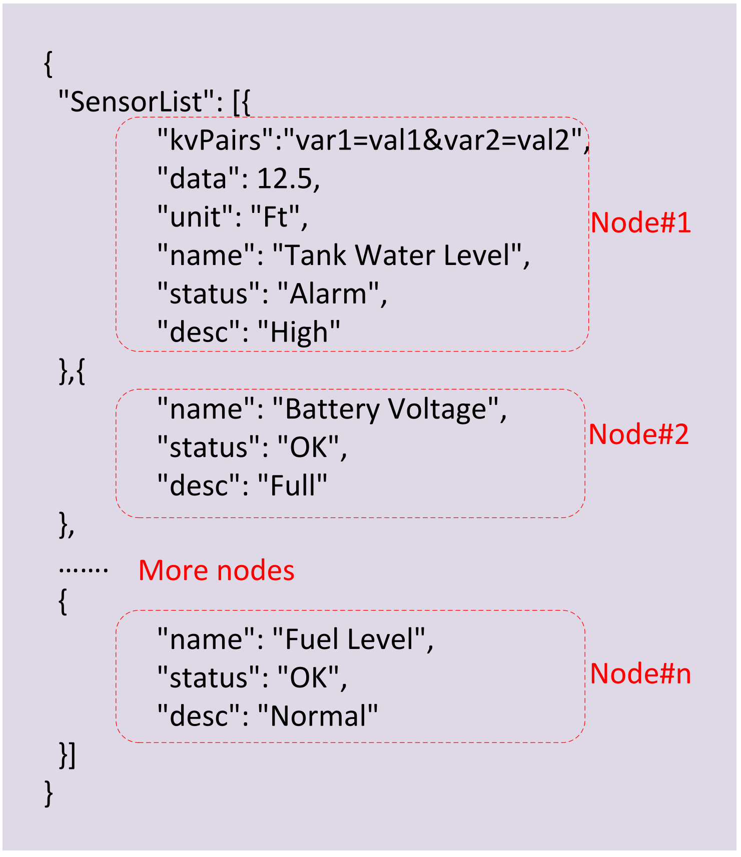
E. Proteus may add support for additional nodes in future, and ignores any nodes that it does not expect.

Positioning of the nodes.

It is important to note that you must maintain the position of the nodes consistently. For example, if you have Water Level Data as the first node, and Battery Level as the second node, It is recommended that you maintain the order of the nodes between consecutive transmission. This is because Proteus is blind to the content of your data. It looks at the position of each node, and extracts specific keys for display and alerts purposes. If you switch positions of the data nodes, display and alerts would still work fine, but data logging and graphing functions would get mixed up as they rely on the position information as well.

When you have a multiple data nodes

It is easy to add more data nodes to the payload. Simply add them enclosed in paranthesis as a json node array. See example below..



```
{
 "SensorList": [{
        "kvPairs":"var1=val1&var2=val2",
        "data": 12.5,
        "unit": "Ft",
        "name": "Tank Water Level",
        "status": "Alarm",
        "desc": "High"
 },{
        "name": "Battery Voltage",
        "status": "OK",
        "desc": "Full"
 },
 …….
 {
        "name": "Fuel Level",
        "status": "OK",
        "desc": "Normal"
 }]
}
```

Testing your data payload

There are different ways to test your sensor payload. You can test them locally by printing on to your system output stream for one. To see the actual data being sent, you can also short out the TX and RX on your system and read on TX line what you send out on your RX line.

Once you have your UART lines connected to Proteus, Proteus will do a sanity check on the data. If it finds any structural errors, it will not be able to parse the data and will send an Error String back to your system. This would look like "Error=0x01". More error codes will be added in future.

With data being sent properly, you can check the data either on the local web server from Proteus by simply typing the IP address of the sensor in a web browser. The home page will display each node by its name and status. If a node contains data, then the data will be displayed. If not, the status will be displayed.

Tank Water Level	12.5 Ft
Battery Charge	Full
Fuel Level	Half

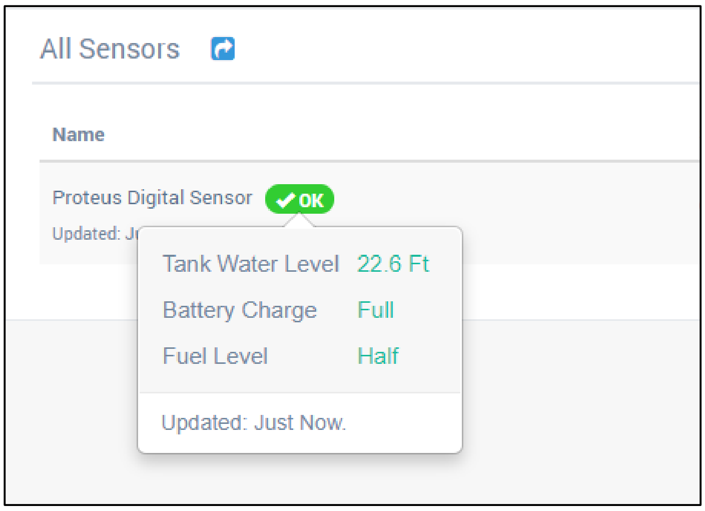
Want to see the JSON Data, Simply click CTRL + "." (Period) Key to view the raw payload displayed on the screen.

Hello Message, Data Frequency

Proteus does not initiate a data request to the UART client. It is the client who must send data to Proteus. The client can decide the frequency at which data is sent. Until new data arrives, the last received payload will be treated as the live data. Also, Proteus does not store your data locally. This means that if there is a power outage or a restart due to network error or any other reason, or even a normal power on event, Proteus will send "ERROR=0x00" to the client. When the client gets this message, It means Proteus has just turned on or rebooted, and has no data so far. The client can treat this as a request to send a data payload.

Cloud Interface

The cloud interface is very similar to the other sensors with minor differences in the data presentation. Since there could be one or more sensors present on each unit, the bubble indicator will not show data from each sensor, but rather show the status of the sensor in general. If any of the sensor nodes are in "Alarm", the bubble will indicate an alarm state, and if all sensors are "OK", then the bubble will indicate OK.



All Sensors

Name

Proteus Digital Sensor ✓OK

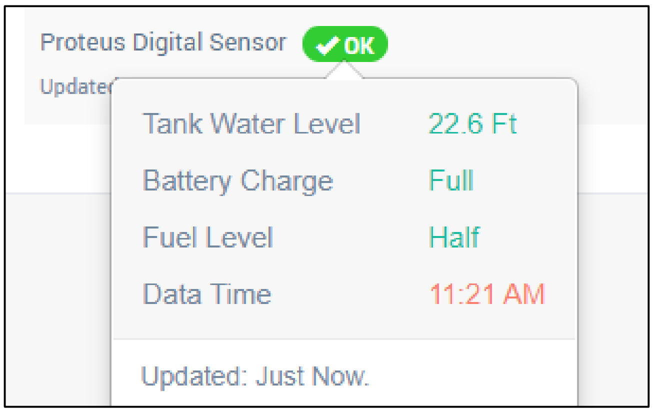
Updated: J

Tank Water Level	22.6 Ft
Battery Charge	Full
Fuel Level	Half

Updated: Just Now.

Uart Client Status

Proteus has built-in option to notify you if power or wifi goes down. You can customize the delay time after which you wish to be notified. Since the uart client is a completely independent system that Proteus has no control over, we have extended the sensor down monitoring option to the uart client. If the uart client stops sending data to Proteus for more than the time specified on the settings tab, you can get notified. You will also be able to see on the cloud the time stamp of when the last data was sent from uart client to Proteus.



Proteus Digital Sensor ✓OK

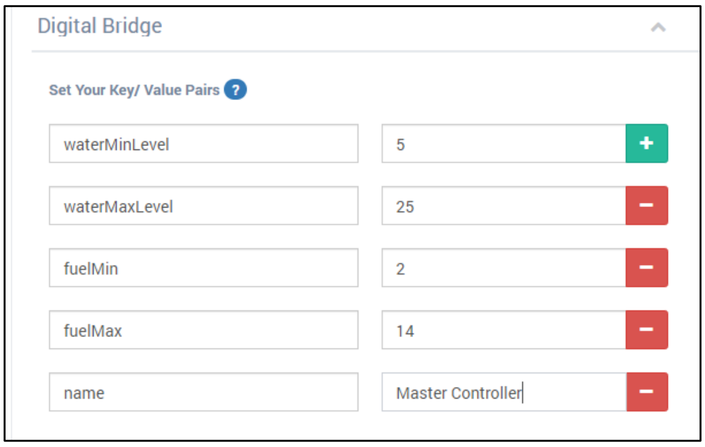
Update

Tank Water Level	22.6 Ft
Battery Charge	Full
Fuel Level	Half
Data Time	11:21 AM

Updated: Just Now.

Setting Key Value Pairs

You can set your own variable names and their values on the cloud. You update the values, save settings and these values will be synced with Proteus.


Digital Bridge

Set Your Key/ Value Pairs

waterMinLevel	5
waterMaxLevel	25
fuelMin	2
fuelMax	14
name	Master Controller

Proteus will then pass it own to the uart client formatted as a post data string.

```
kvPairs=1&waterMinLevel=5&waterMaxLevel=25&fuelMin=2&fuelMax=14&name=Master Controller&..
```

Programatically request for key value pairs

The uart client can also make requests to the Bridge. A request for key value pairs can be made by sending the following command. It is important to keep the array structure inside the json request.

```
{"Request":[
          {"kvPairs":1}
          ]
}
```

Proteus will respond back with the key value pairs in the same format as above.

Ping/ Minimal Data Mode

If your system is not generating any new data that you need to send to Proteus cloud, you can use the "Ping" Mode. In this mode, you can add a sensor array entry as follows.

```
{
 "SensorList": [{
        "name": "Ping",                  Ping
        "status": "OK"                   Packet
 },
 {
        "name": "Battery Voltage",
        "status": "OK",                  Optional
        "desc": "Full"
 },
 .......  More nodes
 {
        "name": "Fuel Level",
        "status": "OK",                  Node#n
        "desc": "Normal"
 }]
}
```

If ping packet is included, It is not necessary to include any other sensor information. The cloud will ignore any sensor specific data and only update the 'Live' status of the client.

More features

We plan to add more features moving forward, while keeping the system as agnostic as possible to the type, design and structure of the uart client. If you are looking for a specific feature, feel free to drop us a line at support@proteussensor.com.