Installing the Data Monitoring Application on Your SmartServer IoT

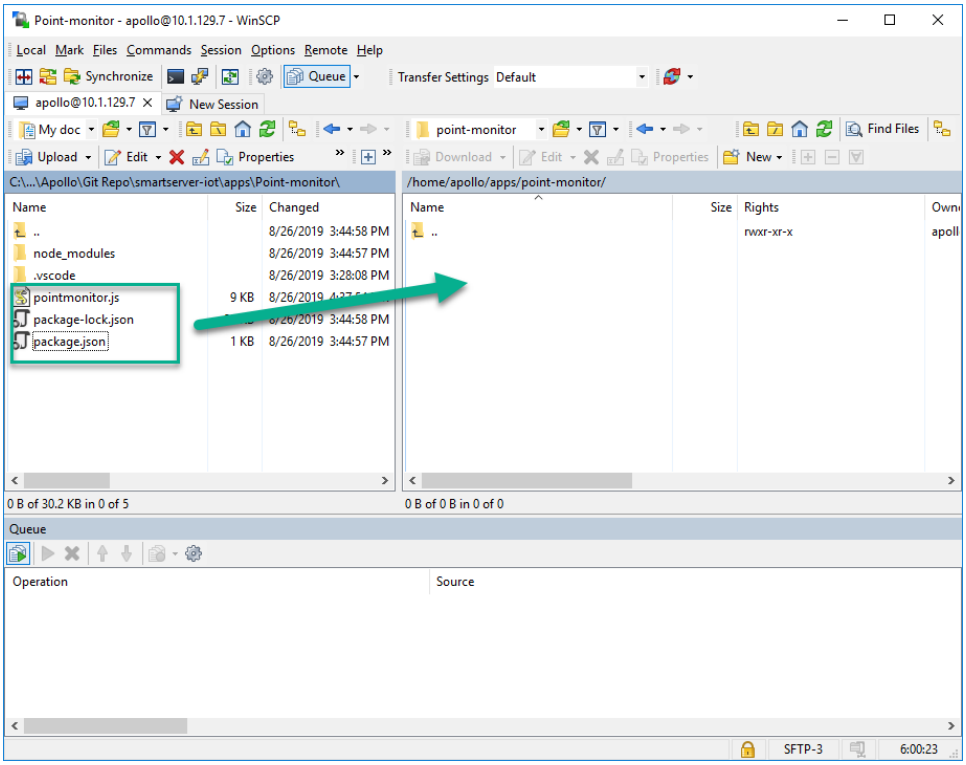
In this section, we will move the application developed in the previous section and set it up to run as a service on the SmartServer IoT. Here is what needs to be done.

- Move the application and npm package files to a location on the SmartServer IoT.
- Install the application with npm.
- Run the application from an SSH session.
- Set the application up to run as a service under **supervisorctl**.

Moving the Application to the SmartServer

In this step, you will copy this application to the folder `/var/apollo/data/apps/point-monitor`.

1. Connect to the SmartServer as user apollo using **winscp** or other sftp client.
2. Create a new folder called **/var/apollo/data/apps/point-monitor** as shown in the following screenshot.
3. Copy these files from the point-monitor project directory to the folder you just created.



Installing the NPM Packages

You have moved the application files (**pointmonitor.js**) along with the **package.json** file that describes the modules used by the application. The following procedure only works if your SmartServer IoT is connected to a network that can reach the internet. This is required for npm to complete installation of dependent modules.

1. Use **putty.exe** to connect to the ssh console as user apollo.
2. Type: `cd /var/apollo/data/apps/point-monitor`
3. Type: `npm install`
4. After about 30 seconds, **npm install** will add 73 packages that comprise the **mqtt.js**, and **collections.js** modules used in this example.

```
apollo@smartserver-17q3kd2:~/point-monitor$ npm install
npm WARN point-monitor@1.0.0 No repository field.

added 73 packages from 54 contributors and audited 313 packages in 21.898s
found 0 vulnerabilities
```

5. Run the application by typing: `node pointmonitor.js`

6. The application will report operation as observed in the debugger to the console as shown above.
7. Type **<cntrl>+c** to terminate the application.

Setting Up pointmonitor to Run as a Service

Many of the software modules that make up the SmartServer IoT, are managed by **supervisorctl**. In this section, we will create a **conf** file to run the **point monitor.js** application under **supervisorctl**.

1. Type `sudo nano /etc/supervisor/conf.d/pointmonitor.conf`
2. The following lines need to be added to this file:
   a. Type `<cntrl>+o` to write the file.
   b. Type `<cntrl>+x` to exit nano.

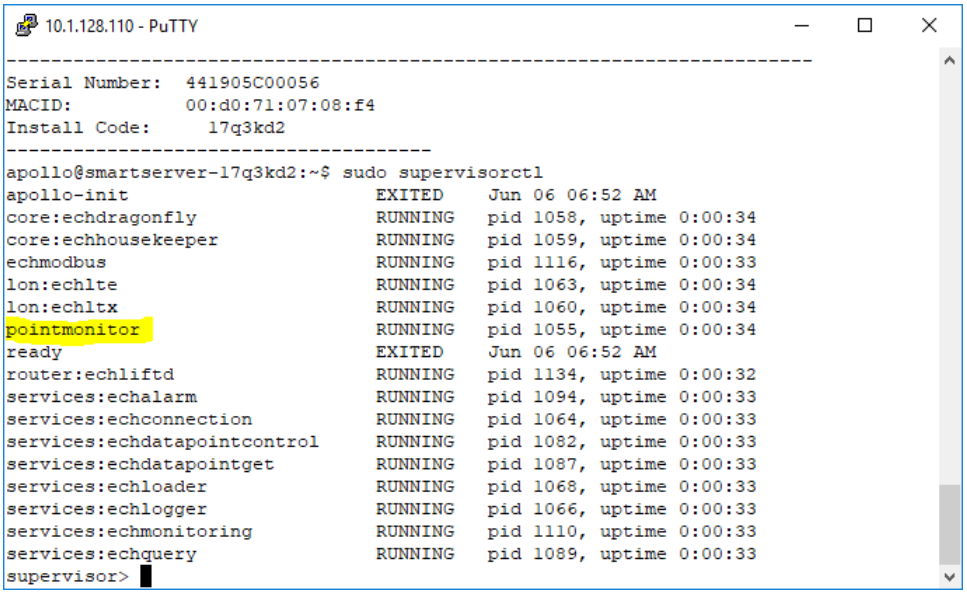
pointmonitor.conf

```
;pointmonitor
[program:pointmonitor]
command=node /var/apollo/data/apps/point-monitor/pointmonitor.js
priority=500
autostart=true
startsecs=10
autorestart=unexpected
exitcodes=0
stopsignal=TERM
user=apollo
stdout_logfile=/var/log/supervisor/stdout-pointmonitor.log
stderr_logfile=/var/log/supervisor/stderr-pointmonitor.log
stdout_logfile_backups=2
stdout_logfile_maxbyte=1MB
stderr_logfile_backups=3
stderr_logfile_maxbyte=1MB
```

Now you can reboot the SmartServer IoT, and the **pointmonitor.js** application will run as a service in the following steps, you will understand how to confirm this.

1. After the SmartServer IoT reboot, use **putty.exe** to connect to the SSH console.

2. Type `sudo supervisorctl`



3. The **pointmonitor** application is show in the list of services running on your SmartServer IoT along with the other services and modules that implement the SmartServer IoT. Type `exit` to return to the bash shell.

4. If you want to monitor the standard output stream for the **pointmonitor** application running as a service, type `tail -f /var/log/supervisor/stdout-pointmonitor.log`

5. You can use `sudo supervisorctl` to issue commands to stop and restart the **pointmonitor** application.

This application as written is reporting the results of every network variable poll request that is done by the **lon:echlte** engine. Here are some explorations you should try.

1. You can control the state of the **pointmonitor** application by typing `sudo supervisorctl` and using **supervisorctl** commands like `stop pointmonitor` or `start pointmonitor` .

2. Modify the **monitorSpecs** object array so the monitoring object for the points of interest generate data events on '**change**' as shown in the following source code fragment.

monitorSpec Object Array set to Monitor on Change

```
// Edit monitorSpecs object array according to your requirements to monitor the target data points of
interest
var monitorSpecs = [
    {
        pid: "9FFFFF0501840460",
        nvList: [
            {ptPath: 'LightSensor/0/nvoLuxLevel', ms:{rate: 20, report: 'change', threshold:1,
inFeedback:false}},
            {ptPath: 'TempSensor/0/nvoHVACTemp', ms:{rate: 20, report: 'change', threshold:0.1,
inFeedback:false}},
            {ptPath: 'Switch/0/nvoValue', ms:{rate: 20, report: 'change', threshold:0, inFeedback:
false}},
        ]
    },
    {
        pid: "9FFFFF0501840450",
        nvList: [
            {ptPath: 'LightSensor/0/nvoLuxLevel', ms:{rate: 20, report: 'change', threshold:1,
inFeedback:false}},
            {ptPath: 'TempSensor/0/nvoHVACTemp', ms:{rate: 20, report: 'change', threshold:0.1,
inFeedback:false}},
            {ptPath: 'Switch/0/nvoValue', ms:{rate: 20, report: 'change', threshold:1, inFeedback:
false}},
        ]
    }
];
```

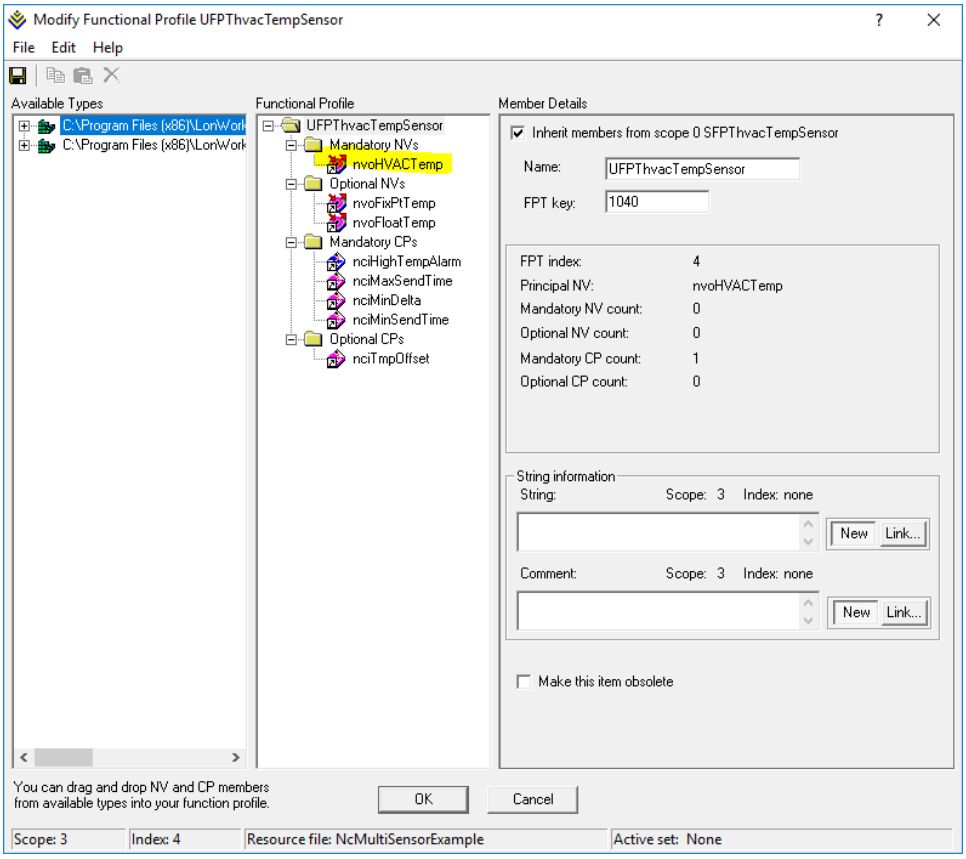
3. Observe how the application responds when the monitored devices are power down. You will see monitor rate adjustments being made to leave only a single active point. This type of behavior is most critical in a bandwidth limited channel, like a power line. You could improve this application by leaving only a single actively polled point when a device goes down, and setting up the monitoring when the device health changes to "normal".

4. Modify the application to work with your own LON devices.

5. Bonus points if you can modify the application to publish an update to `../rq/dev/lon/+/if/Lamp/0/nviValue`

6. This last point is a detail you need to understand. An XIF file defines the implementation name for network variables on a devices interface. When dealing with interfaces in the IAP/MQ, you need to use the names as defined by the functional block definition in the resource file. For example, if you use the Open CT browser to browse the temperature sensor function block. You will see the network variable listed by its program name as defined in the XIF file (**nvoTemperature**), but the topic to access this point in IAP/MQ is **nvoHVACTemp** as defined in the resource file as shown here. The actual program name for the variable is part of the **label** property within the **lon.cfg** object within the **datapoint**

object.



You have successfully run this application as a service. You will need to remove or change the **conf** file extension of the `/etc/supervisor/conf.d/pointmonitor.conf` file to prevent the application from running as a service.

Congratulations! You have completed the programming tutorial for the SmartServer IoT.