SmartEdge Industrial IoT Gateway User Guide

AVNET
Reach Further™

OVERVIEW

PROGRAMMERS REFERENCE Version 1.0 – Covers initial release

PROGRAMMERS REFERENCE Version 1.1. – Covers version 1.1 release

PROGRAMMERS REFERENCE Version 1.0

AVNET IMAGE

- Based on Raspbian stretch lite image April 2019
- Enables support for TPM 2.0 for trusted boot
- IoTConnect Cloud solution
- Our github site is https://github.com/Avnet/smartedge-iio-gateway for the linux source code.
- Our github site for u-boot is https://github.com/Avnet/smartedge-iiot-gateway-uboot
- Our github site for custom files is https://github.com/Avnet/smartedge-iiot-gateway-custom.
- Option to enable secure boot to ensure only signed boot files can be executed. Find files and how to information on our github site. (Important! Once secure boot is enabled only the AVNET u-boot image can be executed on the gateway. The zImage/linux kernel can be isn't affected by this change).

What is Included with image

- Default user name and password are "avnet" for username and "avnet" for password.
- Normal Raspbian stretch lite utilities such as "raspi-config, and raspi-gpio", etc.
- Various TPM 2.0 compliant utilities can be found in /usr/local/bin.
- U-Boot is Raspberry Pi secure, if using Raspberry Pi secure boot feature. It launches kernel after filling out TPM 2.0 PCR 0 value. This completes the Trusted Platform implementation.
- TPM 2.0 PCR 0 element contains the value of the kernel (/boot/zImage) that is calculated during u-boot.
- Access to github for various software source code and components.
- Default behavior requires a configuration set using the Mobile App "Avnet IoTConnect"
- For a list of preinstall packages, run "dpkg –l" for current debian installed packages. For Python 2.7 installed packages run "pip list".
- Custom utilities are provided for system control and are found in /usr/bin/*.sh or /usr/bin/*.py
- Image startup features are contained in /etc/rc.local.* files and various scripts operate from these files.
- Buttons and REST API for the Mobile App are in the python file "/usr/bin/server.py". For Python SDK information see below.
- NodeRed base line installation, not started. Look at the online NodeRed documentation for Raspberry Pi.
- The IoTConnect SDK will be restarted if it finds a problem, this is controlled by the file "/usr/bin/startup.sh"
- To reboot the gateway use "reboot.sh" from the terminal or your application. This eliminates any hangs do to software reboot, by enabling the onboard microcontroller watchdog feature to remove the cpu power after 30 seconds.
- To reboot the gateway using the RST button, do a short press on the button. This calls "/usr/bin/switch.sh" and the device will reboot in the default mode.

- All other .sh files in /usr/bin are used by others in the system, however some may be helpful to review. A complete list and functional description is in the Appendix.
- An IoTConnect SDK OTA software update method is available.
- Minicom and Tio are included as terminal programs.

IOTCONNECT

- The Avnet image comes configured with our IoTConnect Cloud solution
- A 30-day free trial is provided
- After the 30-day trial, or when you are ready, you can use the Mobile App to reconfigure your gateway so that it works on your new machine name.
- Please visit the web portal (www.element14.com/gateway) for details on how to continue accessing your device over the cloud beyond the free trial period.

RST Button and upper LED

The gateway will indicate the current status using the following:

- The default mode is blinking RED/GREEN once every second. This indicates that the gateway is in configuration mode (Access Point) and the Mobile App can be used to setup the device.
- Solid GREEN indicates that the configuration was successful and we have a WiFi or Ethernet(SKIP button on Mobile App) connection and are waiting for the SDK to become active
- Blinking GREEN/OFF once every second indicates the SDK is communicating your sensor data to the cloud.
- The RST button can be used to hard reset or put the gateway into default mode. A short press will cause a hard reboot. Use a long press (greater than 10 seconds) to go into default mode with a hard reboot. The LEDS will blink RED/GREEN in default mode.

HOW THE SOFTWARE WORKS

WHAT HAPPENS DURING BOOT

The boot process can be observed on a HDMI screen. A USB keyboard can be used to log into the system and run commands.

The boot process starts with some initial messages from the bootloader indicating the system is powering up. The next set of messages will be coming from the linux kernel. Just before the login prompt you will see a message indicating "My IP address is x.x.x.x" followed by the name of your device "IotGateway xxxxxxx" where xxxxxxx is the 8 digit serial number of the device. Once the kernel boots, all the operations for the gateway can be found in /etc/rc.local. These operations are required to use the mobile app and setup the device to talk to the IoTConnect cloud. Since this is a Trusted Platform using TPM 2.0 the boot up PCR list can be utilized, so that your system setup can be verified as trusted. Run the command "tpm2_pcrlist" and see SHA256 values for PCR 0 and 10. For more information on using the TPM tools installed on your system see the following web site https://github.com/tpm2-software/tpm2-tools/wiki/How-to-use-tpm2-tools . Our specific linux kernel version is "Linux raspberrypi 4.14.79-v7+" and can be verified by using the following terminal command "uname –a". The specific date stamp for the image creation can be found using the following terminal command "cat/usr/bin/ImageDate.txt".

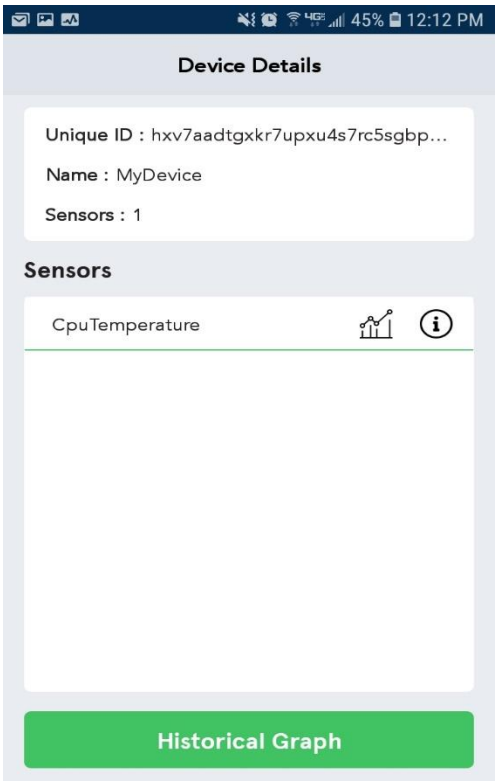
You will also be able to use this setup as your linux terminal. When you see the following enter the text below.

raspberrypi login: avnet

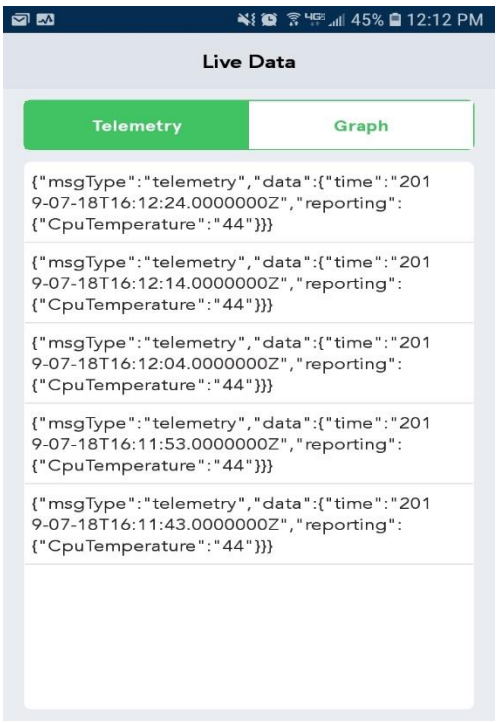
Password: avnet

HOW TO SETUP THE GATEWAY

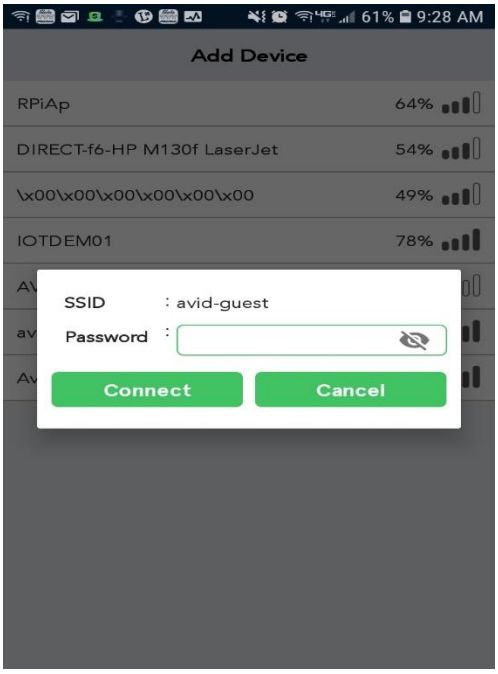
Avnet IoTConnect app is available at the Google and Apple app stores. The app is required to setup the Gateway. The app is used to setup your user account for the 30-day trial period. It will also configure the gateway's network connection to the IoTConnect cloud system. Once you complete this process, the device will be listed in the mobile app. Tapping the icon for the device will open a device screen which will look similar to this one:



Tapping on the CpuTemperature icon, will present live Telemetry data from the device. This screen will look similar to this one.



NOTE: Only WPA Access points are allowed from the gateway. Here is a screen shot from the mobile app selecting a wifi access point for the gateway.



WHAT'S NEXT

Now the device is communicating to the IoTConnect cloud. The SDK can be customized for sensors used, commands sent from cloud, complete the setup of the device on the web portal, etc. The next sections will go over these and other various things.

ADDITIONAL CLOUD SETUP ON WEB PORTAL

Once the mobile app set up is complete, then login into the web portal using a standard browser. The following web address should be used https://avnet.iotconnect.io/login . The same username and password should be used that were used on the mobile app. Once logged in your device can be found on Device tab on the left side. Also documentation is available under the Resource tab. Clicking on the device tab will show a list of the devices that are configured for the logged in user. To accomplish more advanced options like OTA updates, Command Setup, and other options available to an Administrator, the device will need to transition from the 30 day trial period to your own system. Once that's done full access will be provided using the administrative account on the new system to do other important tasks.

The following procedure can be used to do OTA updates on the gateway.

1) On a linux machine make a directory called updates.
2) Put a file in that directory called install.sh. Make sure the permissions are set to 777 on the file. See appendix for an example.
3) Add anything you want do to in that script file.
4) Include supporting files in that directory, this would be the files you want installed.
5) Compress the updates directory contents into a .tar.xz.gz file
6) Rename this file to "install".
7) Setup the web portal for the OTA and point the file for OTA to your install file.
8) Make sure the device is on and sending data to the cloud, this is important as the OTA will not get sent unless the device is connected to the cloud.
9) Issue a FORCED OTA update to your device.
10) The software will be installed. Double check on the gateway that the files were installed.

Also, the install files will be left in the SDK area in case you need to further debug/analyse failures.

CONFIGURING THE SDK

When you first get your Gateway, you should configure it with the Mobile App Avnet IoTConnect, which is available at the App store for IOS and Android. Once the gateway is configured, you should be able to login to the web portal at "avnet.iotconnect.io" and see your device. The factory configuration sends "CpuTemperature" to the cloud. The IoTGateway uses an access point with WiFi configuration:

- SSID: IoTGateway_xxxxxxx"
- WPA PSK: IoTConnect

Modifying the SDK can be done by following these steps

1) Connect a HDMI screen and USB keyboard to the device, then power the device on. Now reboot the device and log in. Once you have logged , the command line prompt "avnet@raspberrypi:~ $ will be available. The user home directory is /home/avnet.
2) Use the following command to get to the SDK directory. "cd /usr/bin/IoTConnectSDK_Py2.7/Testing/sample"
3) The 2 files you will need to edit for your configuration are "IoTConnectSDK.conf" and "user_functions.py". They are found in the SDK directory.

4) The .conf file contains sections for “CloudSDKConfiguration”, “CloudSystemControl”, and “CloudSDKDefaultObject<xxx>” entries. For the 30 day trial period you only need to setup the section for “UserName” and “PassWord” in “CloudSystemControl” which will be the same credentials you used to create your account with the Mobile App “Avnet IoTConnect”.
5) For the “CloudSDKDefault options, you should add the section names [CloudSDKDefaultObject<xxx>] sequentially numbered and update the “DefaultObjectCount” in the “CloudSystemControl” section.
6) Working examples for Objects are commented out in the “IoTConnectSDK.conf” file.
7) The file “user_functions.py” is used to add any custom python code to the system. There is a requirement that the first two fucntions remain avaialble. They are “def user_callbackMessage(msg):” which is used for COMMANDS sent from the cloud. The second function is “def user_Initialize():” This function gets called when the SDK starts and you can put any initialization code here.
8) There is an example called “def ThisIsMyFunction():” that is commented out and simply prints to the console.
9) Now if you want a simple test do the following:
    a. Add a section to the “IoTConnectSDK.conf” file
    b. [CloudSDKDefaultObject2]
    c. # Predefined objects can go here. NUMBER/STRING/OBJECT
    d. Value=NUMBER
    e. Name=TestUserFunction
    f. UsePythonInterface=ThisIsMyFunction

10) Then uncomment the function in “user_functions.py”, for the items can will be sent to the cloud.
11) Now issue the following command “reboot.sh”
12) The new information you just added will show up on the cloud.

Changing the default device parameters on the cloud

To change the device parameters on the cloud, edit the IoTConnectSDK.conf file and modify the parameters for Display_Name and Template_Name to the desired settings. Note: do not use quotes. Then remove the device and template for that device from the cloud . The device will be re-enrolled using the new settings, there is no need to reboot for this change to take effect. After the change, the new template can modify on the cloud. This includes using the OTA feature.

Debugging the SDK

After you get the basic idea and want to be sure your changes are working you can do the following.

1) Plug in the HDMI screen and USB keyboard, power off /on , then log on.
2) Go to the SDK area described earlier.
3) Run commands “sudo pkill startup” and “sudo pkill python”. Now you are ready
4) Run this command to see the SDK output. “sudo –E python example.py”
5) Now you should be able to see what’s going on and track down odd behavior, such as you have a syntax error in “user_functions.py”, etc.

Included in the SDK are options for CpuTemperature, CpuFrequency, DigitalIN1, DigitalIN2, DigitalIN3, and DigitalIN4. There are also options for Reboot, FactoryDefault, SetDigitalOutput1 , SetDigitalOutput2, SetDigitalOutput3 , SetDigitalOutput4 , ClearDIgitalOutput1 , ClearDigitalOutput2 , ClearDigitialOutput3, ClearDigitalOutput4 , that are setup as cloud commands.

SYSTEM OPERATIONS

On boot up, the gateway will look up the TPM unique ID and Endorsment key. It will then construct the BSSID for the gateway access point mode and startup access point mode services. The gateway’s BSSID will be broadcast so the mobile app can connect to it and the RED/GREEN leds will be blinking. Then the API is started that communicates with the mobile app for

gateway configuration. After the mobile app configures the gateway the contents of /etc/rc.local will change and the gateway operating state will shift from Access Point mode to SDK running. The GREEN/OFF leds will be blinking at this time. The following services will be started at this time. The python SDK will be started and the API for the LEDS/BUTTON/REST will be started. Now date will be send to the cloud. The system will restart the SDK if there is a problem.

You can control the system services to stop the SDK or the LED/BUTTON/REST service. There are also scripts that will put the gateway back to the default state, cleanly reboot with no hangs.The system has access to the on board micro for hardware reset which the reboot.sh command uses. The gateway Broadcom hardware driver on /dev/watchdog0 is also available for any application needs

1) To stop the LED/BUTTON/REST service run command “sudo pkill server”
2) To stop the SDK, run command “sudo pkill startup” then “sudo pkill example”
3) To put the gateway back to the default state run the command “switch_only.sh”
4) To cleanly reset run “reboot.sh”

SYSTEM CUSTOMIZATION

The gateway has the potential to be customized in almost any fashion.

The standard Pi HAT connector can be used to add already available cards, or you can develop and use your own HAT card. Standard HAT cards can be easily added to the system, all you need to do is edit the file in /boot/config.txt and add an overlay for your standard HAT card. If you make your own card and need a custom kernel driver , then the w kernel source files from our github site, are required. A cross compiler for ARM is also required. We use cross-gcc-8.3.0-pi_2-3 tool set. Any toolset with GCC 8.3 or greater that compiles to arm linux gnueabihf- standards will work. Follow your standard developer instructions for adding your driver/device tree overlay to the kernel source tree.

The digital inputs and outputs in any fashion. There are hooks in the SDK for sending the Digital Inputs to the cloud. If you need cloud interface to the digital outputs, you must use the cloud command feature and add hooks in your SDK configuration (IotConnectSDK.conf and user_functions.py). The digital outputs are available on /sys/class/gpio/gpio201, gpio203,gpio205, and gpio207. The inputs are on /sys/class/gpio/gpio200, gpio202, gpio204, and gpio206. They all must be exported in your application to use them. If they are used in the SDK (IoTConnectSDK.conf) then the ones specified are exported automatically.

The RS-232/RS-485 port is pre-configured for RS-485 operation. On board jumpers are used to setup the port for RS-232/485 and half/full duplex in the case of RS-232 or RS-422. This port is available for anything you need, like a serial terminal or an industrial based RS-485 device. The port is found on /dev/ttySC0. See the Appendix for default jumper position and other jumper configurations for this port.

The internal CAN bus is also available but disabled in the image. It can be enabled by removing the comments dealing with the can0 section in /etc/network/interfaces. Utilities "CANSEND and CANDUMP" have been included and the can0 device can be used by the network stacks. Note: there is a CAN bus termination jumper inside the gateway next to the CAN connector. See the Appendix for more information.

The u-boot portion of the code is also available on the github site. If the intent is to generate a custom boot loader, we would recommend that the secure boot feature is never enabled. If it is the u-boot generated will need to be digitally signed by Avnet. Also, a completely different solution can be loaded, however if it replaces any files in the /boot directory and if secure boot is enabled it will not work. If secure boot is enabled it can NEVER be disabled. If a different solution is used Avnet will have to digitally sign the appropriate files before you can enable secure boot.

There is a battery backed up RTC on the gateway. For systems that do not have internet connectivity (I.e. closed system) then this can be set then enabled with a few standard linux commands to keep the date/time even if the power fails. Normally this system is on the internet and the correct time information is provided using network time keeping protocols.

There is a PCIe connector on board and accepts most PCIe USB devices and is primarily used for cellular modem connections. Note: If you decide to use a cellular modem there will be customization needed control data calls. Standard linux tools that are available can be used for dialer customizations.

The gateway has the NodeRed project baseline installed on the system. By default the service is not started. Additional NodeRed modules will be provided for the device, as they are currently being developed by Avnet. For more information on NodeRed and how to use it, use the following web site. https://nodered.org/docs/getting-started/raspberrypi

The ssh terminal service is disabled by default. You can use the raspi-config tool to enable it. The setting is under the interface options section. Also add an empty file called "ssh" to the /boot directory, then reboot.

The raspi-gpio tool can be used to see the default GPIO configuration for the gateway. If custom hardware is used on the HAT connector you should use the device tree overlay to setup and HAT connection GPIO. Then use the raspi-gpio tool to ensure the GPIO's are configured correctly.

DOWNLOAD A NEW IMAGE OR UPDATE FILES ON THE DEVICE

There are several options to update files on the device. Also, the micro usb port near the center of the board can be used to download a new image or update files on the device. It can also be used to save your current gateway image.

To download a new image or update files on the device, get the custom files from our github site. The rpiboot directory contains the needed files to use on your PC linux system for this feature. Note: The rpiboot must be complied for your version of linux. Run "make" to do this. There are 2 scripts that can be used after the code is compiled. The scrip rpi.sh is the non secure method to do the updates. First plug in the micro usb to the PC, then power on the gateway. The gateway will show up as a new /dev/sd* file system and you can mount either the /boot area or the /rootfs area if you intend to update files. To update a new image use the linux "dd" command. To save the current image use the linux "dd" command.

The gateway USB port can be used to update files on the device. Put your files on a usb stick , then plug it into the gateway. On a gateway terminal you will have to mount the usb device as it does not auto mount for security reasons. Then simply copy the files from the USB stick.

Here's a link that describes the procedure. Please use our version of usbboot called RPIBOOT in the github area.

The production image can be downloaded from this site.

https://docs.avnet.com/amer/smart_channel/smartedge/

https://www.raspberrypi.org/documentation/hardware/computemodule/cm-emmc-flashing.md

PROVISIONING MULTIPLE DEVICES

Multiple gateways can be provisioned without the Mobile Application. To accomplish this task the gateway should be plugged into the Ethernet. Then the REST API's can be used for this purpose. See appendix for REST API's supported. The process would look something like this. Note: These features will be available in future images, they can be downloaded from the github custom URL specified above.

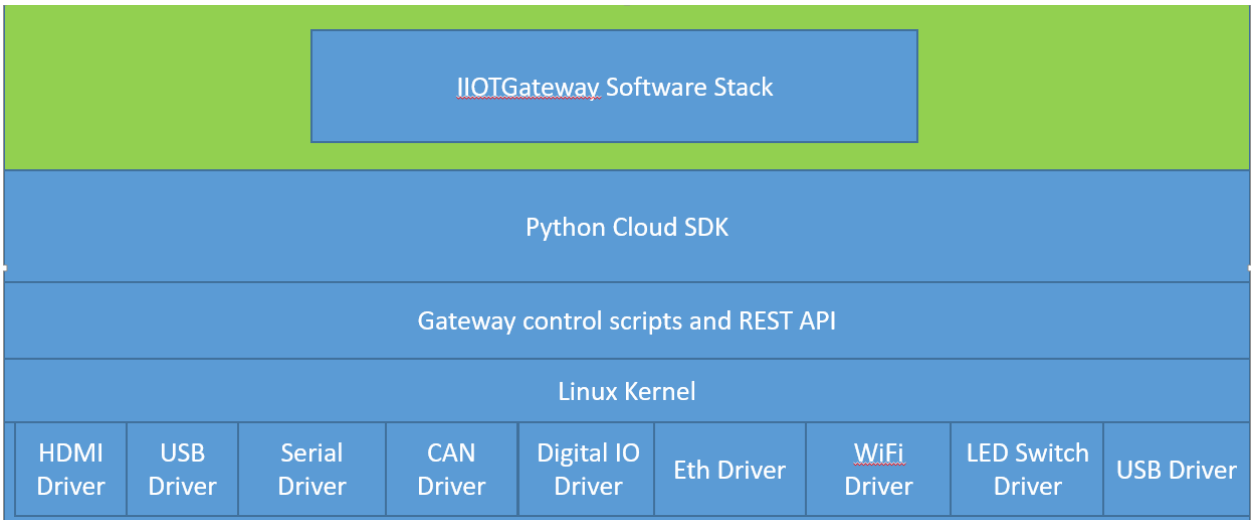
1) Find the IP addresses for the connected devices.
2) Issue the /DeviceID REST command to get the device serial number. You can use this to select the files needed for that device. If you are unsure just issue the REST command to all IP addresses and if it responds then proceed with that Device ID.
3) Issue the /WiFiGetWPAConf to obtain the current wpa_supplicant.conf file. This can be used if the current settings need to be preserved.
4) Issue the /WiFiIWList to get the currently available access points. This can be used if to determine what access points are available. If the connection settings are already known, they can add them to wpa_supplicant.conf.
5) Issue the /WiFiSetWPAConf to set the new wpa_supplicant.conf file.
6) Issue the /IOTGetIOTConnectSDK.conf command to get the current SDK configuration file.
7) Modify the SDK configuration file, the use the IOTSetIOTConnectSDK.conf file on the gateway.
8) Issue the /CloudAttach command to setup the WiFi connection, if configured. It will also start the SDK.

APPENDIX

The production image can be downloaded from this site

https://docs.avnet.com/amer/smart_channel/smartedge/

GATEWAY SOFTWARE STACK



JUMPER CONFIGURATIONS

The default jumpers for the RS-232/RS-485 are

J22 – pin 1 jumpered to pin 2 . Selects RS-485

J22 – pin 7 not jumpered. Selects half duplex operation.

All other RS-232/RS-485 jumper configurations are found near the beginning of this document.

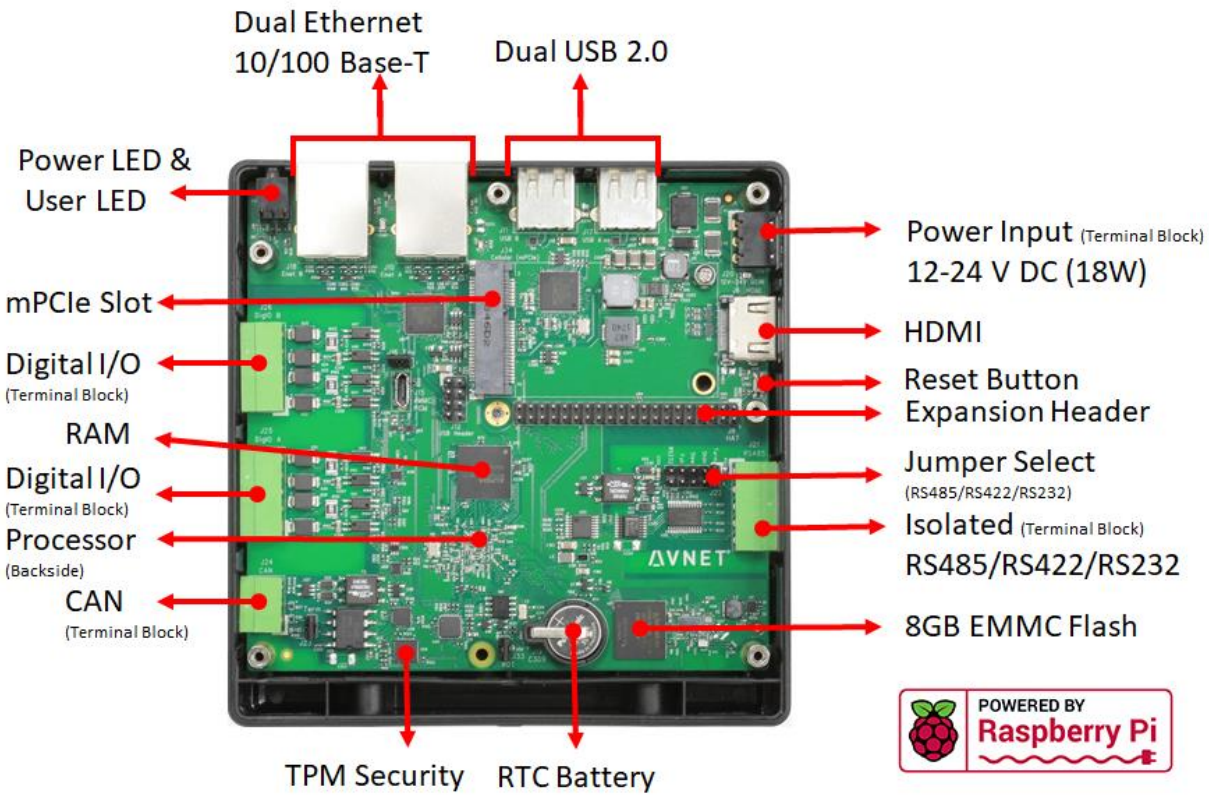
The default jumper for the CAN bus termination resistor is enabled.

For how to configure the CAN bus termination please reference CAN bus documentation found on https://www.ni.com/en-no/innovations/white-papers/09/can-physical-layer-and-termination-guide.html

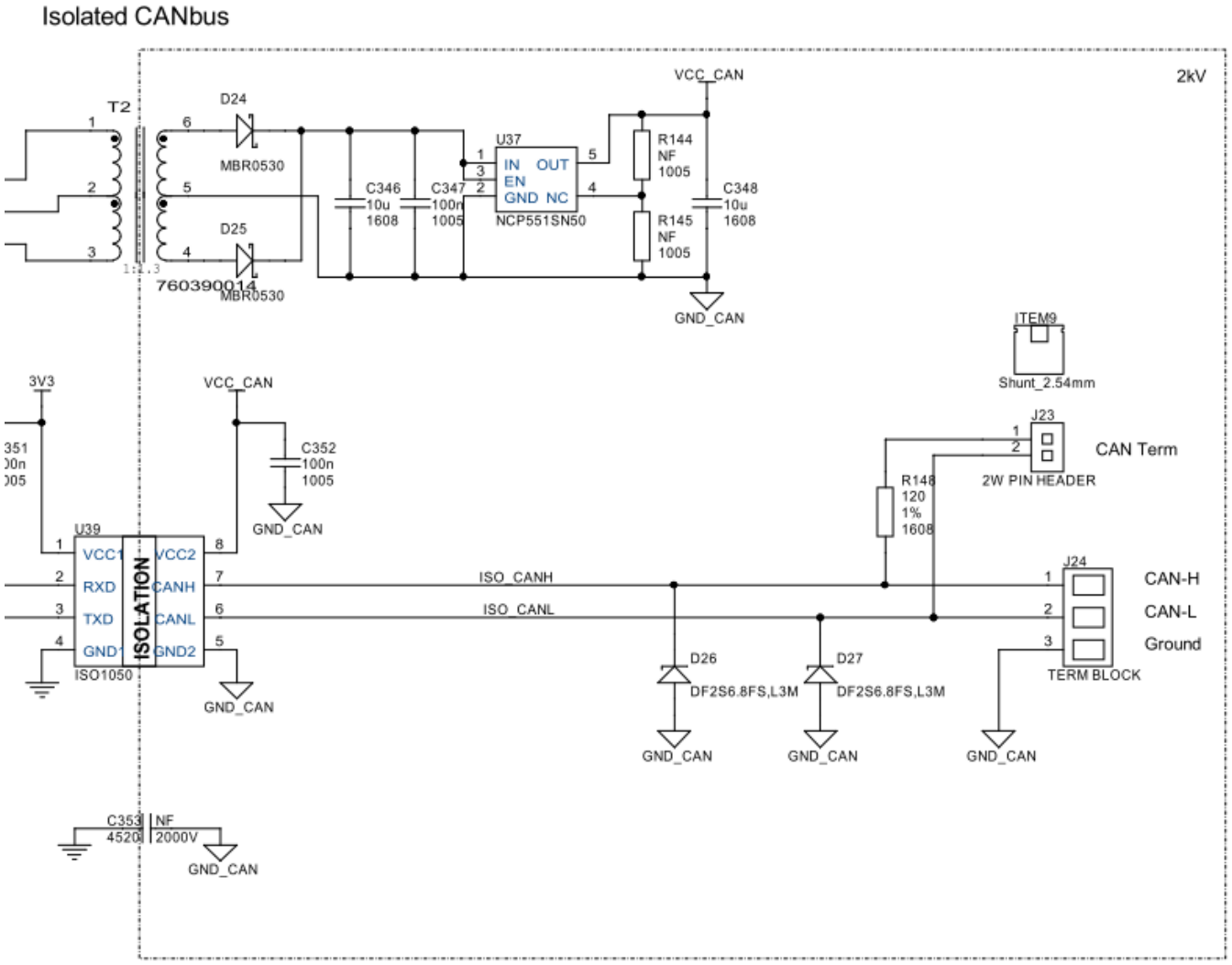
The default WDT jumper is not enabled. This feature is currently not enabled and should never be jumpered.

The jumper near the onboard USB micro connector J16 should be left in the default shipping position. This jumper is used with the usb port for complete image updates using the rpiboot utility. More information can be found later in this appendix.

SCHEMATIC REFERENCES

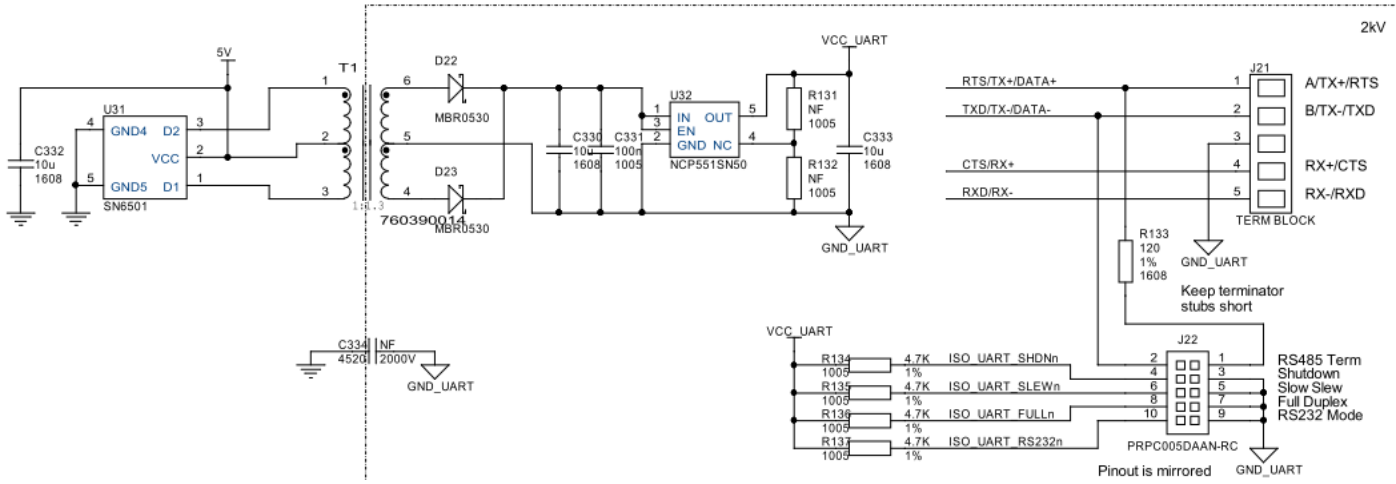


Here is the CAN bus schematic for the gateway

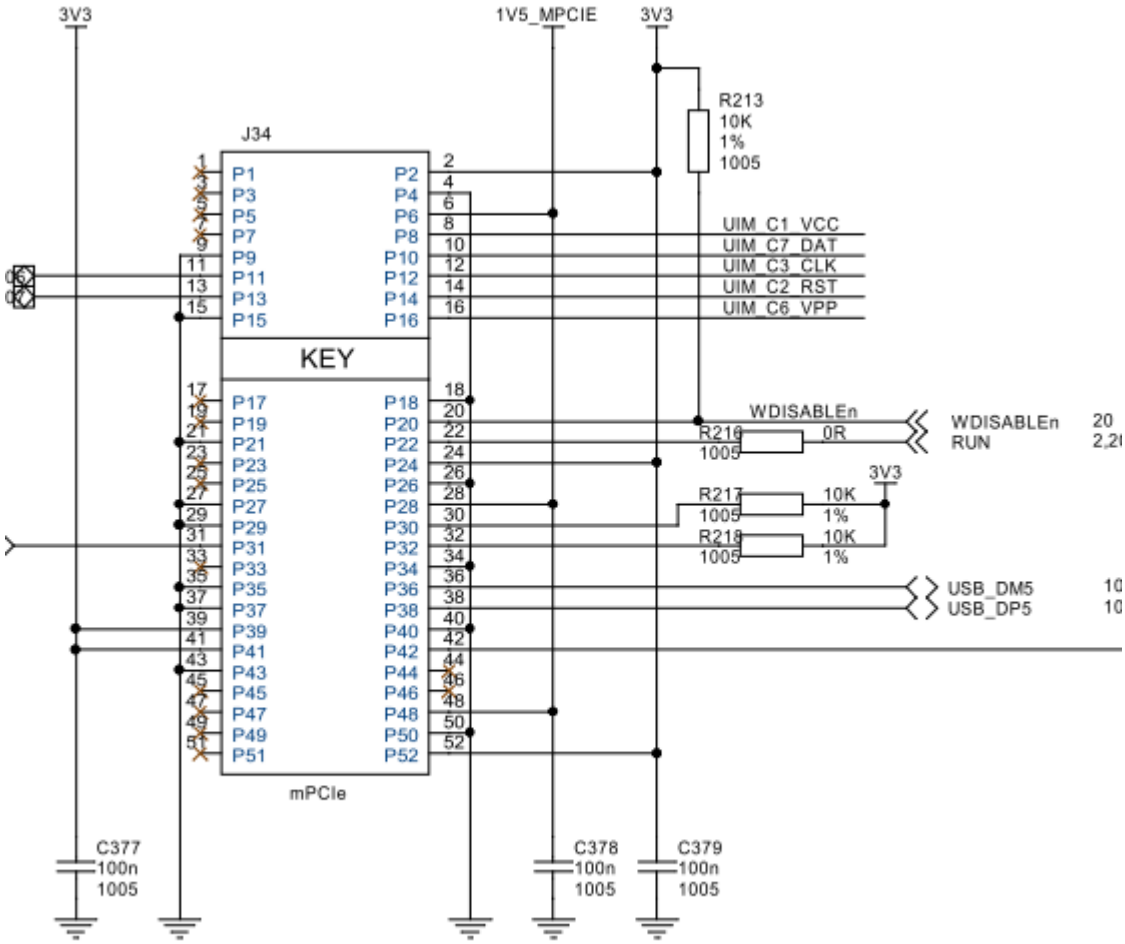


Here is the RS-232/RS-485 schematic for the gateway

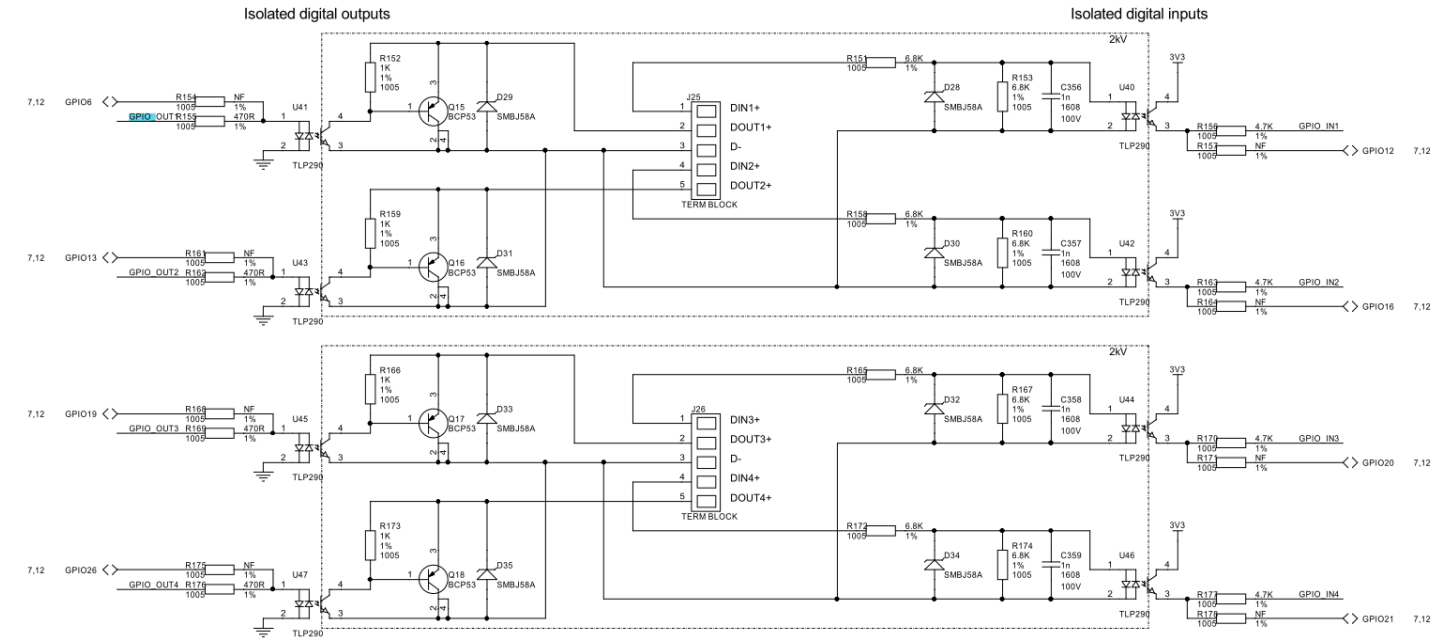
Isolated RS232/RS422/RS485



Here is the PCIe schematic for the gateway



Here is the digital IO schematic for the gateway



OTA update example install.sh

install.sh
echo “Installing software updates”
sudo cp rc.local /etc/rc.local
ehco “Installation Complete!”

Note: must include new rc.local and install.sh in the compressed installation file. See above text for description.

1 REST API DOCUMENTATION

- Can be found in github custom area under documentation

ERRATA

1) For the initial release, the file /usr/bin/ImageDate.txt is missing. The release date was 07/19/2019
If you wish to use the feature to create your own template name and device name then first edit the file wifi_client.sh and change this line

cat provisioning.txt | tr –d “\r\n” | cut –d ‘:’ –f2 | cut –b 1-424 | tr –d “\r\n” >tmp3.txt

To

cat provisioning.txt | tr –d “\r\n” | cut –d ‘:’ –f3 | cut –b 1-424 | tr –d “\r\n” >tmp3.txt

2) For the initial release, do not use “apt upgrade” until you do the following. Login in to the gateway with the terminal and type “sudo apt-mark hold raspberrypi-kernel” , “sudo apt-mark hold raspberrypi-bootloader”, and “sudo apt-mark hold raspberrypi-sys-mods”.
3) The initial release has a bug related to a file called /usr/bin/IoTConnectSDK_Py2.7_Testing/sample/sensor_data.txt. If this file becomes corrupted, then multiple readings will be sent to the cloud . If you see readings on the cloud from a prior timestamp you have this condition. Login to the terminal and remove the file with “sudo rm /usr/bin/IoTConnectSDK_Testing/sample/sensor_data.txt.
4) The above file sensor_data.txt can grow substantially if the gateway is offline for long periods of time. Is one option if you want long periods of offline. Insert a USB memory stick and mount it. Use the following command “mkdir /tmp/usb”, “mount /dev/sda /tmp/usb”. Next create a file on the usb stick usind “sudo touch /tmp/usb/sensor_data.txt. Then remove the file sensor_data.txt if it exists and make a symbolic link to the usb by this command “ln –s /tmp/usb/sensor_data.txt /usr/bin/IoTConnectSDK/sample/sensor_data.txt
5) DO NOT LOAD GUI to this device.
6) Some customers report that on reboot the gateway is not booting properly. We have found no root cause yet and are investigating . Please send the following information if this occurs. First attach the gateway for image update procedure. Then send the following information, Output of “sudo ls –l <mount point>/var/log”. “sudo ls –l <mount point>/usr/bin/IoTConnectSDK_Py2.7_Testing/sample/”. Also send the following file test.tar. Create test.tar by doing the following. “sudo tar –cvf test.tar /<mount point>/boot”, then “gzip test.tar” the output file should be test.tar.gz that we would like sent back.
7) Some customers report the device bricking. To fix do the following.

a) Create this file with contents below /etc/init.d/autofsck
b) Run “sudo systemctl enable autofsck”
c) Run “sudo systemctl start autofsck

```
# /etc/init.d/autofsck
### BEGIN INIT INFO
# Provides:                 bootservice
# Required-Start:    $local_fs $syslog
# Required-Stop:     $local_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:
# Short-Description:        OpenBSD Secure Shell server
### END INIT INFO


case "$1" in
start)
  sudo touch /forcefsck
  echo $!>/var/run/autofsck.pid
  ;;
stop)
  sudo kill `cat /var/run/autofsck.pid`
  rm /var/run/autofsck.pid
  ;;
restart)
  $0 stop
  $0 start
  ;;
status)
  if [ -e /var/run/autofsck.pid ]; then
    echo  autofsck is running, pid=`cat /var/run/autofsck.pid`
  else
    echo autofsck is NOT running
    exit 1
  fi
  ;;
*)
  echo "Usage: $0 {start|stop|status|restart}"
esac

exit 0
```

PROGRAMMERS REFERENCE Version 1.1

WHATS CHANGED IN VERSION 1.1

There were several substantial changes from version 1.0. Many were related to stability and system bricking issues. Many were feature additions. The following describes what has change from a system stability standpoint, new feature view, new SDK operational modes, new SDK features, and github changes.

➢ System Stability improvements for smartedge-iiot-gateway version 1.1

- ❖ The u-boot loader changed to include fsck.mode=force in the builtin command line to the kernel. This will always force a filesystem check on the gateway as it boots. This fixes any file corruption that may occur on the filesystem.
- ❖ Log file growth and syslog features have been enable. Now all files in /var/log will be rotated daily above the 50K size and up to 7 previous zip files will contain the last logs.
- ❖ Apt upgrade services have been modified to put raspberry pi components on hold. This will prevent the gateway from receiving any unintentional upgrades that would brick the gateway.
- ❖ Internet metrics have been added to properly route traffic on wlan0, wwan0, DHCP Ethernet and Static IP ethernets.

➢ New smartedge-iiot-gateway features

- ❖ Moved the SDK from /usr/bin to /opt/avnet-iot/IoTConnect
- ❖ Changed the shell scripts and dropped the .sh. New shell scripts can be found in /opt/avnet-iot/iotservices
- ❖ Removed the old state machine method. Now everything with the SDK is done using a systemd service.
- ❖ Added helper functions for the SDK features. They are found in /opt/avnet-iot/iotservices
- ❖ Added automatic updates for the ATTiny. Also, including both version 1.1 and version 1.3 of the attiny firmware
- ❖ Added Quectel modem driver along with REST API to configure the Cellular Access Point name.
- ❖ Now using our own reboot to enable the watchdog on power down if something hangs.
- ❖ Enable watchdog on powerup in ATTiny by using the jumper WDT on board the gateway. This gives you 240 seconds before the ATTINY shutsdown and restarts the CPU. There are new systemd services that handle disabling / enabling it . This prevents hangs in the kernel booting. Rebooting and using watchdog can be down with the watchdog services for start and stop. Start requires to pet the watchdog every 60 seconds on /dev/watchdog1.
- ❖ Out of box support for Omega Smart Sensor , Omega RS-485 Smart Sensors, and Omega ZW-REC WiFi product line. To enable ensure that the mobile app has set your username and password in /opt/avnet-iot/IoTConnect/sample/IoTConnect.conf. If not then either upgrade your mobile app or manually edit the file.
- ❖ Now the RST button has three modes. 1 – Short press reboots gateway, 2- a 10 to 15 second press causes the gateway to switch to mobile app configuration mode, 3 – Long press over 30 seconds returns gateway to factory default.
- ❖ LED's , Buttons, and Rest API are now 3 separate systemd services.
- ❖ New LED patterns can be found in the user guide.
- ❖ New state information can be found in the user guide.
- ❖ No special .txt files or other odd processes are used as in version 1.0 . Everything is done as needed by the particular feature.

➢ New smartedge-iiot-gateway version 1.1 SDK operational modes

- ❖ The SDK starts in Factory Default mode. This mode is used by the Mobile app to establish the gateway on the cloud. This mode can also be entered by running "switch_only" at the gateway command prompt or via cloud command if you have your username/password in the .conf file
- ❖ Once the mobile app establishes the gateway on the cloud, the gateway enters the SDK running state.
- ❖ At any time you can issue the "iotstat" command to see the current mode. For Factory default you will see Button, Led, and Rest running. For SDK running you will see Button and example.py running.
- ❖ The button service can be disabled to prevent unwanted RST button activity once the gateway is deployed.
- ❖ Run iotstop to stop the running SDK
- ❖ Run iotstart to start the SDK
- ❖ Run iotrestart to restart the SDK. This is useful if you have changed the .conf file.
- ❖ Run iotview to view the log if you have set enabledebug = 1 in the .conf file

- To run the SDK and view the output on the console, run “iotstop” set enabledebug = 0 in the .conf file and then you can run “iotrun”
- Run “switch” to reboot, also available on the cloud command section
- Run “switch_only” to reenter Factory Default
- Run “switch_only_configs” to enter Mobile App configuration mode.
- New system services are ledservice, buttonservice, restservice, iotconnectservice, quectel, watchdogstart, watchdogstop, hwclock-start, attinyupdate, and smartedgehalt.

New smartedge-iiot-gateway version 1.1 SDK features

- Automatically generate Templates, Attributes, Cloud commands and Cloud rules based on the IoTConnectSDK.conf file.
- Redirect sdk output to any file with enabledebug = 1 in .conf file
- Auto generate cloud entries for Omega sensors, Omega sensors support hot plug unplug and cloud updates are applied as needed.
- Added support for gateway Digital IO
- Added support for gateway cloud command to do various things, see .conf file for details.
- Added support for several new gateway Attributes, see .conf file.
- Added support for local rules generation, see .conf file for details.
- Added support for new subscription based mobile app.
- Added new REST API’s, see rest api documentation.
- New IoTConnectSDK.conf.default holds the factory default configuration and will delete and .conf file if entering Factory mode.
- SDK now gathers data at intervals controlled by the .conf file then sends to the cloud. Also a priority send channel is implemented if you set pushdataalways = 1 in the .conf file
- SDK now supports proper floating point. See “precision” entries in the .conf file.
- SDK supports queuing data on Polled or OnChange, see .conf file.
- SDK supports adding descriptions, unit to cloud Attributes.
- SDK supports sending priority on heartbeat rate if sensor not changed.
- SDK supports optionally using version 1.0 SDK user_functions / IoTConnectSDK.conf file or version 1.0 complete SDK. Requires manually doing or if using the upgrade version1.1 from version1.0 process it’s done with a command line parameter.
- SDK when running controls the LED state, otherwise its controlled by ledservice.
- SDK support for Omega Smart Sensors obtains all information like Name, Inputs/Outputs/Precision/ and descriptions from the sensor, can be changed by using the Omega PC configuration tool outside of the scope of this document.

New smartedge-iiot-gateway version 1.1 github layout and changes

- The github area for the custom sdk features now has a branch containing the version11 SDK. Please use the install.sh installer script to switch to version 1.1. It can have an optional parameter called importoldsdk or useoldskd. For importoldsdk it copyies your user_fucntions.py and IoTConnectSDK.conf file only. For useoldsdk, the original SDK found in /usr/bin/.... will be used.
- New scripts have been added to upgrade from 1.0 to 1.1, can be done on gateway command line or OTA from cloud.
- One change to u-boot for forcing filesystem check on every boot.
- Several kernel changes , the following have changed, mcp251x driver, all attiny_ drivers. Mcp251x fixes reset issue. Attiny_mfd is the i2c mutilfunction interface for attiny_btn, attiny_wtd, and attiny_led.
- Updates to the rpiboot area.

SMARTEDGE-IIOT-GATEWAY VERSION 1.1 SYSTEM DETAILS.

There following will explain the details of version 1.1 for the above new features. Most of the foundation of the version 1.0 section still applies. There is also a new mobile app for this release that supports the 30 day trial.

➢ Smartedge-iiot-gateway system operation and changes related to version 1.0

In version 1.0 the operating system was left completely open for apt upgrades. This would allow you to easily brick the smartedge-iiot-gateway. Now the raspberrypi-kernel is marked on hold for the apt upgrade function because then our custom kernel drivers would not work. The raspberrypi-bootloader is marked on hold for the same reason as well as the raspberrypi-sys-mods. This greatly improves the stability and overall bricking issue from the previous version.

In version 1.0 the filesystem could become corrupted and the gateway would not boot properly. Now the u-boot forces the kernel to do a filesystem check each boot to ensure there are no issues. The original version also had several issues that would quickly fill up the filesystem making the gateway unusable. These issues were resolved by properly implementing logrotate and limiting the growth of auth.log (I.e. removed "sudo" from all most all internal operations). Now on a daily /weekly/monthly basis logrotate will save and compress 7 logs for each .log file. This will happen each time the cron job runs and each log file is greater than 50K. The original SDK logging has been reduced to primarily initialization and SensorName/Value/Timestamps send to cloud.

In version 1.1 the IoTConnect SDK component had several issues. First was a problem related to a corruption in the offline data file sensor_data.txt. This file is used to save cloud attributes while there is no internet. Once the internet is restored these values are sent to the cloud. Once the file had a corruption the old samples would repeatedly be sent and the file wasn't deleted upon error detection. This has been resolve with a newer IoTConnect SDK. The original SDK had an issue with multithreading calls to sdk.SendData. This has been resolved.

Internet metrics have been added for all the interfaces which fixes some issues with version 1.0. Now for eth0 and eth1 they are assigned the default 201 and 202, wlan0 is assigned the default of 301, wwlan0 is assigned to 401. If eth0 and/or eht1 are plugged into a static IP network with no DHCP server, they are assigned a metric of 501 and 502. This fixes issues with original version not picking the correct path to the internet.

In version 1.1 the IoTConnect SDK can be found in /opt/avnet-iot/IoTConnect/sample. If you upgrade from version 1.0 to 1.1 with the install.sh script then the original SDK will still be available for reference. The upgrade script also has options for using some or all of the version 1.0 SDK.

The scripts for version 1.1 can be found in /opt/avnet-iot/iotservices. Some of the linked systemd services are found in /opt/avnet-iot/services . These include the attinyupdate, bootservice, quectel, and smartedgehalt services.

Version 1.1 supports automatic attiny reprogramming if the version is out of date. The files and utilities for that operation are found in /opt/avnet-iot/utilities.

System states are now controlled by systemd services. There are 2 basic states. The first is FactoryDefault and the gateway is in this stated on first boot. This is the mobile app registration state. The gateway WiFi is in Access point mode at this point. The REST API is also available in this mode to support the mobile app and any Ethernet configuration if setting up more than one gateway. Also in this mode the button service , and the led service are running. The second state is SDK running. In this state the gateway is sending values to the cloud. The running state is systemd service iotconnectservice. The button service will be running and the led's are controlled by the SDK example.py file.

This version has a button service that is always running but can be disable once the gateway is in service to prevent unwanted RST button usage. The RST button now has 3 options. First short press triggers a reboot. A 10 to 15 second press triggers a switch to mobile app configuration mode and the REST API service is started and the gateway WiFi Access point mode is enabled. A greater than 30 second press triggers FactoryDefault. Note this overwrites you SDK .conf file with the default one.

Added Quectel PCIe/USB modem service that's always on unless disabled. To connect to the internet put the APN name in a file called /etc/apn.conf

Watchdog is now stopped when the SDK services load. It can be reenabled by “sudo systemctl start watchdogstart” then you must pet the watchdog at /dev/watchdog1 every 60 seconds. To disable run “sudo systemctl start watchdogstop”. Then 60 second pet is not required to keep gateway from rebooting.

New reboot command uses internal script in the SDK directory to prevent hangs on reboots. This changes led state to be driven by the ATTINY fast flashing green/off.

New system service smartedge-iiot-halt switches control to the ATTINY to blink fast red/off if you “sudo halt”. Meaning need to power cycle gateway.

New system service hwclock-start, now synchronizes the state between the RTC battery back up clock and system services for network timesync.

New rest system service for starting and stopping gateway RESP API. Can be used to configure via Ethernet so it can be used during operation for operational updates, reconfigurations, etc.

Now IoTConnect SDK automatically generates Templates on the cloud. Use IoTConnectSDK.conf to control names and other Template creation parameters.

The new SDK automatically generates Attributes for all your sensors. Use the IoTConnectSDK.conf file to see default examples. Most of the Attribute fields can be utilized with the .conf options.

Added autogeneration of cloud commands and rules, use the .conf file options.

New cloud data reporting method uses the settings in the .conf file to control sample rate, OnChange or Polled send, Priority send, and Heartbeat send. The Cloud sending thread looks for normal data to send at the defaults used in the .conf file High priority data gets sent every second. At heartbeat rate will send regardless of any changes.

The IoTConnect SDK supports debugging using .conf “enabledebug” then you can point debug output to any device or use the default /var/log/iot.log. Useful for debugging and not using excessive writes to the EMMC part.

Added Digital IO support for cloud, New attributes for monitoring/status, New cloud commands and one cloud rule example. Use the .conf file to see examples.

Added local rules abilities for offline processing, this will be used until IoTConnect provides edge/gateway support with the SDK.

Added new REST APIS, the documentation can be found with this one on the github sites.

Be aware if you factorydefault the gateway it copies the IoTConnectSDK.conf.default to IoTConnectSDK.conf, so if you want permanent changes put them in the .default file too.

CpuTemperature is the default out of box sensor. To enable the rest use the new mobile app or edit IoTConnectSDK.conf and fill in username/password with the mobile app signin information. HotPlug Omega SmartSensor support also requires proper username/password. Also any of the customer code based on this implementation would require username/password.

The SDK now supports proper floating point data to the cloud. Use “precision” in the .conf file to specify this.

Omega SmartSensors for USB/RS-485/and Omega Ethernet ZW series are fully supported and automatically setup with proper username/password in the .conf file. The attribute information, description, precision, etc are all retrieved from the SmartSensor. Omega documentation will be on the github.

SMARTEDGE-IIOT-GATEWAY TPM2.0 DETAILS.

There following will explain the details of TPM2.0 support since the initial release

The connection to the cloud is secured with TPM interface to Connect with DPS for cloud functions and Device Authentication processes (Device enrollment, Template creation, Attribute creation, etc). Documentation can be found on the github site.

The u-boot loader generates PCR0 for the kernel image "/boot/zImage", the user application is responsible for verifying this. Only on power up is the initial state well known. During power on boot the PCR0 is calculated and extended to PCR0 using the TPM2.0 chip. The the TPM2_RH_PLATFORM is set with a random number password so that a user can not modify any of the TPM contents we setup. Note for non power on reboots PCR0 is extended from the last PCR0 value as we can no longer clear the PCR's.

While the kernel boots it does a boot_measurement. Currently, Audit or EVM support is not enabled. Further information can be read from here. Audit could easily be added to the .config file for u-boot if you needed it(BOOTCOMMAND=). https://sourceforge.net/p/linux-ima/wiki/Home/

The TPM2.0 chip is verified as genuine by the factory the github custom area for version 1.1 has a file called manufacturing.sh that verifies the TPM is genuine from Infineon.

The user may use the TPM in any other fashion if they have specific requirements and the full linux tpm2-tools are implemented. https://github.com/tpm2-software/tpm2-tools

SMARTEDGE-IIOT-GATEWAY COMMAND LINE FUNCTIONS.

There are several commands that are included in the OS and our SDK that can be executed on the command line. The following is a description of these.

➢ Smartedge-iiot-gateway additional command line functions.

Name: attiny_flash

Synopsis: This script is used to program the ATTINY micro processor.

Description: The script can be executed manually, it is automatically run to verify the version of your ATTINY is up to date. The released version of the ATTINY was 1.1 . The latest version is 1.3. To verify which ATTINY firmware you are using do the following "dmesg | grep ATTINY". Results will be "... attiny_led_probe: ATTINY FW/HW REV X"

Options: The script can be modified to downgrade to another supported version, currently ver1.2 and ver1.3 are supported. Change the THISVERFW variable to manually select the version.

Return Value: Output will indicate success or failure and verification can be done by checking dmesg after reboot

---

Name: boot

Synopsis: Boot is used interally to prepare for gateway access point mode.

Description: This system service is only done once on factory reboot to gather the correct AP name from the serial number and setup the hostapd.conf file in /etc/hostapd/hostapd.conf

Options: none

Return Value: none

Name: button

Synopsis: This shell scrip is used internally by the bootservice.service in /etc/systemd/system.

Description: It runs the button python code that interfaces to the RST button. This is the code that runs the short press/long press/very long press functional checks with the ATTINY and executes the SDK scripts appropriatedly. This may be disabled by "sudo systemctl disable buttonservice" which prevents RST button from working once the gateway is deployed.

Options: None

Return Value: None

Name: checkpishrink

Synopsis: Internally checks to see if pishrink is expanding the filesystem.

Description: All SDK system services use this before starting and prevent starting if a filesystem expansion is occurring.

Options: None

Return Value: None

Name: cleanup

Synopsis: This is an internal function for preparing the gateway before generating a release image.

Description: This cleans up and puts the gateway in the factory default state. It will also create an avnet user ,if it doesn't exist.

Options: None

Return Value: Output on terminal for visual verification.

Name: getid

Synopsis: Internal function used to get unique ID from TPM

Description: Uses tpm_device_provision to return the UniqueID and EndorsementKey

Options: None

Return Value: Outputs UniqueId

Name: halt

Synopsis: Internal function that get called by smartedge-iiot-halt

Description: This makes the LED's blink fast RED/OFF until power is removed.

Options: None

Return Value: None

Name: iotrestart

Synopsis: Restarts running SDK

Description: This restarts the running SDK, it's useful if you edit IoTConnectSDK.conf and want to test the new settings.

Options: None

Return Value: None

Name: iotrun

Synopsis: Manually start the SDK

Description: This is useful when debugging new additions to the SDK, first run iotstop, then iotrun will run a single instance

Options: None

Return Value: SDK output to selected enabledebug files in SDK configuration file.

Name: iotsetup

Synopsis: One time setup for AP mode name.

Description: This generates /etc/hostapd/hostapd.conf which describes the AP mode access point settings ESSID, etc. and is used internally.

Options: None

Return Value: None

Name: iotstart

Synopsis: Start the IoTConnect SDK service.

Description: This can be used to start a stopped IoTConnect SDK.

Options: None

Return Value: None

Name: iotstat

Synopsis: List running SDK components.

Description: This lists the running SDK components as well as other python components.

Options: None

Return Value: If in FactoryDefault then Led, Button, and Rest, will be running, if running Example then Cloud SDK is running.

Name: iotstop

Synopsis: Stops running SDK

Description: This stops the running IoTConnect SDK.

Options: None

Return Value: None

Name: iotview

Synopsis: Real time view the IoTConnect log file.

Description: This shows in realtime the IoTConnect log file provided you have “enabledebug = 1” and have the log going to /var/log/iot.log.

Options: None

Return Value: Realtime SDK output on terminal.

Name: led

Synopsis: Used internally by the ledservice.service

Description: This runs the python code led.py controlling the led.

Options: None

Return Value: None

Name: reboot

Synopsis: Reboot gateway

Description: This safely reboots the gateway to prevent hangs using the watchdog timer if the kernel hangs during reboot. This is linked to the /sbin/reboot which replaces the standard reboot feature.

Options: None

Return Value: System reboots after up to 60 seconds.

Name: rest

Synopsis: Used internally by the restservice

Description: This runs the REST API python code rest.py.

Options: None

Return Value: None

Name: startup

Synopsis: Start the IoTConnect SDK

Description: This is used internally to start the IoTConnect SDK python code and adjust the services appropriately.

Options: None

Return Value: None

Name: startwd

Synopsis: Used internally by the watchdogstart.service

Description: This enables the watchdog , it must be petted before the 60 second timeout reboots the system.

Options: None

Return Value: None

Name: stopwd

Synopsis: Stop the watchdog

Description: This stops the watchdog. After that the watchdog does not need to be petted.

Options: None

Return Value: None

Name: switch

Synopsis: Same as reboot.

Description: Used internally by the buttonservice or cloud command to reboot the gateway.

Options: None

Return Value: None

Name: switch_only

Synopsis: Start FactoryDefault operation.

Description: Used internally by the buttonservice or cloud command to restore the gateway to FactoryDefault mode.

Options: None

Return Value: None

Name: switch_only_configs

Synopsis: Start AP mode and REST API.

Description: Used internally by the buttonservice or cloud command to start AP mode and REST API the gateway.

Options: None

Return Value: None

Name: tpm_device_provision

Synopsis: Returns UniqueID and EndorsementKey from TPM chip.

Description: Used internally by the SDK.

Options: None

Return Value: None

Name: wifi_client

Synopsis: This sets up the wifi connection and runs the IoTConnect SDK

Description: Used internally by wifi_connect

Options: None

Return Value: None

Name: wifi_connect

Synopsis: This sets up the wifi connection

Description: Used internally by the rest api to setup and test Wifi connection then decide to launch SDK or simply revert to Access point mode.

Options: None

Return Value: None

Name: wifi_connect_only

Synopsis: This assumes Ethernet connection and starts SDK

Description: Used internally by the rest api to implement the SKIP feature in the mobile app.

Options: None

Return Value: None

SMARTEDGE-IIOT-GATEWAY SDK OPERATIONS

The IoTConnect SDK consist of 2 basic parts. The first part is the main engine and the second part is the configuration file. The following will describe how the main engine works. The details of the configuration file and how to use this framework to implement any custom solution will also be described.

➢ Smartedge-iiot-gateway main engine operations.

The main engine is found in /opt/avnet-iot/IoTConnect/sample/example.py. The python version that the SDK uses is version 2.7. The engine also uses 2 other files in that same directory. One the user_functions.py is there for custom python code. The end user can choose to use this or edit the example.py file directly for customizations.

The main() function is called to start the SDK. Several things occur at startup. This is an ordered list of what happens during the startup sequence.

1) The function tpm_device_provision is called and returns the uniqueId and Endorsement key from the TPM system. This is used to communicate with the IoTConnect Cloud.

2) The Raspberry Pi Serial number is obtained from /proc/cpuinfo and is used for default clout template name generation and default cloud device name.

3) Next the IoTConnectSDK.conf file is parsed for various operational information. Details will be given in the following section.

4) Next check for debug logging and enable it if specified in the .conf file.

5) Then automatically generate all specified cloud Attributes, cloud Commands, and Cloud Rules based on the contents of the .conf file. This also includes enrolling your device with the cloud if it has not been enrolled or was previously deleted.

6) Then the main engine connects to the cloud and starts processing data. All the low level functions for accomplishing this are part of the iotconnect-sdk that is a PIP python component on the gateway. This includes the initial connect and the sending of sensor data.

7) The main engine runs a thread that is responsible for the LED slow blink green/off pattern and the ledservice is stopped.

8) Then main engine is thread based so I sits in an idle loop after it finishes initialization and cloud connectivity.

Now the data processing begins. The data is sent from the function SendDataToCloud. This is happens periodically or on demand. Each sensor gets its own thread and puts its data on either the standard priority queue or the high priority queue. The data on the high priority queue will be checked and sent every second if needed. The standard queue data will be sent at a rate defined by the .conf file.

Each sensor task will take a reading from the device using the python interface specified in the .conf file for that sensor. That is done at the rate specified in the .conf file. The data can be reported using 2 methods. Polled will send after the configured time, and OnChange will send only when the data has changed. There is also a Heartbeat send that will send the data at that rate independently of Polled or OnChange.

9) Along with the sensor data, there may be several cloud commands that are added. These commands are processed with a callback from the iotconnect-sdk component. The callback function that processes the commands is called DoCommand.

10) There is also and OTA feature that is detected and executed from the DoCommand function

11) The SDK can now use the ATTINY watchdog by setting useiotwatchdog=< pet rate less than 60 seconds> in the .conf file

➢ Smartedge-iiot-gateway configuration file description and usage.

This is a detailed description of the IoTConnect.conf file. The syntax is basic .INI style. It can be changed at anytime and the new values will take effect by "iotrestart" or reboot if SDK is running.

The first section is the cloud configuration section . This contains the scopeid, env, and cpid for the cloud. It may be modified by version 2 of the mobile app when setting up your cloud instance.

```
[CloudSDKConfiguration]

scopeid = One0005A911

env = avnetpoc

cpid = B3A7D54220AD4397ABF35D7EC539FBA6
```

The next section is the cloud system control section . This contains several of the IoTConnect cloud URL's for the API that setups and verifies the gateway. Some of these may be modified by version 2 of the mobile app. See below for more specific details.

```
[CloudSystemControl]
```

This area contains the IoTConnect Cloud URL endpoints for doing the automatic gateway enrollment, template generation, Attribute generation, Command generation and Rule generation. These are also used for Plug and Play sensors.

```
# cloud urls for feature control

configuration_version = 1

http_auth_token = https://avnetauth.iotconnect.io/api/v1.1

http_auth_login = https://avnetauth.iotconnect.io/api/v1.1

http_device_create = https://avnetdevice.iotconnect.io/api/v1.1

http_device_template = https://avnetdevice.iotconnect.io/api/v1.1

http_rule_template = https://avnetdevice.iotconnect.io/api/v1.1

http_event_template = https://avnetevent.iotconnect.io/api/v1.1

http_user_template = https://avnetuser.iotconnect.io/api/v1.1

http_device_firmware = https://avnetfirmware.iotconnect.io/api/v1.1

# override default template name or guid and display name
```

This area can be used to override the default template information by name or GUID. The display name default can also be set here. The default template name is "zt<serial_number>" and the default display name is

"IoTGateway_<serial_number>". The Cloud entity ID can also be changed, see below. Edge and Gateway support are reserved for future implementation.

```
template_guid =

template_name =

display_name =

# cloud entity ID

entity_guid = 12F60889-BF0E-402B-A3B4-FFD38BA62D62

# next 2 are for future support

isedgesupport = 0

isgatewaysupport = 0

#

# user name and password from the register new account section of the mobile app. Verison 2 of the mobile app will
automatically fill the information.

#
```

This area is used to specify the username and password. Use the same values that were used when you registered your account with the mobile app. The next version of the mobile app will fill these in automatically. They are required to do the automatic cloud setup from the gateway.

```
username =

password =


# Cloud role for access to device
```

This area is used to specify the gateway role. Default is PiRole but can be changed to "Admin" for example. The cloud solution key can also be change from the default, see below.

```
role = PiRole

# Cloud solution key

solution-key = QkQ2OTg1MUMtM0M4Ni00NzZGLUJDODctOUJDRThDQ0E4QkU4LWFjY2Vzc0tFWS1pazZhbW8wbmJp


# Default SDK object count/command count/rulecount, etc. Also logging feature you could save logs on network or usbmemory.
If enabledebug=0 then log goes to /dev/null

#
```

This area is used to specify the gateway Attribute, Command, and Rule counts for the sections that follow. For example if you have 4 sensors set “defaultobjectcount = 4” and include 4 named DefaultObject sections.

```
defaultobjectcount = 1

defaultcommandcount = 0

defaultrulecount = 0

defaultomegarulecount = 0
```

This area is used to specify the gateway data transmission times, enable attiny watchdog feature, and enable debug feature.

```
# Rate in seconds that the sensor data is sent to cloud (PushAlways or Heartbeat can be used which are delevered every second if available) otherwise data is collected on a queue until the timeout)

sendtocloudrate = 20

# 0 - disables sdk attiny watchdog, otherwise pet rate is given and must be less than 60 seconds.

useiotwatchdog = 0

# 0 – disables debugging. 1 – enable debugging to specified debuglogfile.

enabledebug = 0

debuglogfile = /var/log/iot.log
```

This section is used to specify the Omega Sensor and RS-485 modbus serial port parameters. See below for descriptions.

```
#

# Default Omega sensor configuration

#

# usbsensorreport - used for all smart sensors can be Polled or OnChange

# usbsensorreportpolltime - Time in seconds between checking smart sensor.

# zwrecconnectstatic - Yes/No - Check for static IP in 192.168.200.*

# zwrecconnectdynamic - Yes/No - Check for IP's with zwrec as part of hostname

# zwrecconnectdynamicnames - Check for hostname on ethernet interfaces can be All(for all devices with "zwrec" as part of the hostname. It can also be a list of specific ZWREC devices such as "zwrec8820". It can also be set to None.

#

# zwrecconnectstaticip - Check for IP address on ethernet interfaces can be All(for all devices. It can also be a list of specific ZWREC devices such as "192.168.200.200".
```

```
#
# zwrecsensorreport - used for all ZWREC sensors can be Polled or OnChange
# zwrecsensorreportpolltime - Time in seconds between checking ZWREC sensor.
# rs485modbus - enable or disable modbus Can be Yes or No for Enable/Disable.
# rs485modbusslaveaddresses - This is a list of modbus slave addresses (i.e 1 2 10)
# rs485modbusbaud - modbus baudrate
# rs485modbusparity = modbus parity. Can be E - even, O - odd, N - none
# rs485modbusstopbits = modbus stopbits (1 or 0)
# rs485modbusdatabits = 8 or 7, etc.
# rs485modbusheartbeatcount =
# precision = floating point precision reported to cloud for ZW series
# enabledebug = 0 add to debug log if enabled
#
[OmegaSensorConfiguration]
usbsensorreport = OnChange
usbsensorreportpolltime = 10
usbsensorreportheartbeatcount = 10
zwrecconnectstatic = Yes
zwrecconnectdynamic = Yes
zwrecconnectdynamicnames = All
zwrecconnectstaticip = All
zwrecsensorreport = OnChange
zwrecsensorreportpolltime = 10
zwrecsensorreportheartbeatcount = 10
zwrecunits = unknown
rs485modbus = Yes
rs485modbusslaveaddresses = 1
rs485modbusbaud = 38400
rs485modbusparity = E
rs485modbusstopbits = 1
```

```
rs485modbusdatabits = 8

rs485mobbusheartbeatcount = 10

precision = 0

enabledebug = 0  # if 1 then send to currently defined debuglogfile with enabledebug = 1 above as well.
```

This section and the ones that follow are used to specify the sensor information for the automatic cloud Attribute setup. The sections must be named "[CloudSDKDefaultObject<xx>]" where xx is a sequential number starting with 1. See below for a description of the required parameters.

```
#
# Default Objects Section
#
# Notes:
#   value = one of STRING or NUMBER
#   name = name of sensor on cloud
#   report = one of "Polled" or "OnChange"
#   reportpolltime = number of seconds to check sensor.
#   PushAlways = 1 - always send this data, 0 - Send on heartbeat only.
#   usepythoninterface = name of the python routine servicing your sensor.
#   description = Description of sensor as it appears on the cloud.
#   reportheartbeatcount = number of reportpolltimes before pushing value to cloud
#   units = units value as they appear on the cloud
#   precision = floating point report data precsion, 0- no precision for this object, 1-n = number of digits after decimal point
#
# edgeaggregatetype, edgetumblingwindow = future edge device enhancements.
#


[CloudSDKDefaultObject1]

value = NUMBER

name = CpuTemperature

report = OnChange

reportpolltime = 60
```

pushdataalways = 1

usepythoninterface = GetTheTemp

description = This monitors the interal cpu temperature and reports on change

edgeaggregatetype = Min

edgetumblingwindow = 1s

reportheartbeatcount = 1

units = Celsius

precision = 1

[CloudSDKDefaultObject2]

value = STRING

name = SystemHealth

report = Polled

reportpolltime = 60

usepythoninterface = SystemHealth

description = This monitors the SystemHealth

edgeaggregatetype = Min

edgetumblingwindow = 1s

units = String

reportheartbeatcount = 10

pushdataalways = 1

precision = 0

[CloudSDKDefaultObject3]

value = NUMBER

name = GetDigitalInput1

report = OnChange

reportpolltime = 10

usepythoninterface = GetDigitalInput1

description = Returns the state of Digital input 1

reportheartbeatcount = 100

units = on-off

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject4]

value = NUMBER

name = GetDigitalInput2

report = OnChange

reportpolltime = 10

usepythoninterface = GetDigitalInput2

description = Returns the state of Digital input 2

reportheartbeatcount = 100

units = on-off

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject5]

value = NUMBER

name = GetDigitalInput3

report = OnChange

reportpolltime = 10

usepythoninterface = GetDigitalInput3

description = Returns the state of Digital input 3

reportheartbeatcount = 100

units = on-off

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject6]

value = NUMBER

name = GetDigitalInput4

report = OnChange

reportpolltime = 10

usepythoninterface = GetDigitalInput4

description = Returns the state of Digital input 4

reportheartbeatcount = 100

units = on-off

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject7]

value = NUMBER

name = ProcessMonitor

report = OnChange

reportpolltime = 60

usepythoninterface = ProcessMonitor

description = Returns the number of python threads running

reportheartbeatcount = 100

units = integer

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject8]

value = NUMBER

name = FreeMemory

report = OnChange

reportpolltime = 600

usepythoninterface = FreeMemory

description = Returns the amount of free memory

reportheartbeatcount = 10

units = integer

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject9]

value = NUMBER

name = CpuUtilization

report = Polled

reportpolltime = 60

usepythoninterface = CpuUtilization

description = Returns the percent cpu utilization

reportheartbeatcount = 10

units = integer

pushdataalways = 0

precision = 0

[CloudSDKDefaultObject10]

value = NUMBER

name = FreeDisk

report = OnChange

reportpolltime = 600

usepythoninterface = FreeDisk

description = Returns the number of free bytes on the rootfs

reportheartbeatcount = 10

units = integer

pushdataalways = 0

precision = 0

This section and the ones that follow are used to specify the command information for the automatic cloud Command setup. The sections must be named "[CloudSDKDefaultCommand<xx>]" where xx is a sequential number starting with 1. See below for a description of the required parameters.

```
#
# Default Commands Section
#
# Notes.
#
# commandname = name of command as it appears on the cloud
# command = name of command to execute
# hasparameter = 1 - if parameter required, 0 - no parameter
# requiresack = 1 - if Ack required, 0 - no Ack required
# isiotcommand = 1 - if OTA command, 0 - if cloud command.
# usepythoninterface = python function called when cloud command is received.
#

[CloudSDKDefaultCommand1]
commandname = Reboot
command = Reboot
hasparameter = 0
requiresack = 0
isiotcommand = 0
usepythoninterface = RebootNow

[CloudSDKDefaultCommand2]
commandname = FactoryDefault
command = FactoryDefault
hasparameter = 0
```

```
requiresack = 0
isiotcommand = 0
usepythoninterface = FactoryDefaultNow

[CloudSDKDefaultCommand3]
commandname = SetDigitalOutput1Now
command = SetDigitalOutput1Now
hasparameter = 0
requiresack = 0
isiotcommand = 0
usepythoninterface = SetDigitalOutput1Now

[CloudSDKDefaultCommand4]
commandname = SetDigitalOutput2Now
command = SetDigitalOutput2Now
hasparameter = 0
requiresack = 0
isiotcommand = 0
usepythoninterface = SetDigitalOutput2Now

[CloudSDKDefaultCommand5]
commandname = SetDigitalOutput3Now
command = SetDigitalOutput3Now
hasparameter = 0
requiresack = 0
isiotcommand = 0
usepythoninterface = SetDigitalOutput3Now

[CloudSDKDefaultCommand6]
commandname = SetDigitalOutput4Now
```

command = SetDigitalOutput4Now

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = SetDigitalOutput4Now

[CloudSDKDefaultCommand7]

commandname = ClearDigitalOutput1Now

command = ClearDigitalOutput1Now

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = ClearDigitalOutput1Now

[CloudSDKDefaultCommand8]

commandname = ClearDigitalOutput2Now

command = ClearDigitalOutput2Now

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = ClearDigitalOutput2Now

[CloudSDKDefaultCommand9]

commandname = ClearDigitalOutput3Now

command = ClearDigitalOutput3Now

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = ClearDigitalOutput3Now

[CloudSDKDefaultCommand10]

commandname = ClearDigitalOutput4Now

command = ClearDigitalOutput4Now

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = ClearDigitalOutput4Now

[CloudSDKDefaultCommand11]

commandname = Enable SSH

command = EnableSSH

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = EnableSSH

[CloudSDKDefaultCommand12]

commandname = Disable SSH

command = DisableSSH

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = DisableSSH

[CloudSDKDefaultCommand13]

commandname = SwitchToConfiguration

command = SwitchToConfiguration

hasparameter = 0

requiresack = 0

isiotcommand = 0

usepythoninterface = SwitchToConfiguration

This section and the ones that follow are used to specify the rule information for the automatic cloud Rule setup or for the local Rule setup. The sections must be named "[CloudSDKDefaultRule<xx>]" where xx is a sequential number starting with 1. See below for a description of the required parameters.

#

Default Rules Section

#

Notes: This one is for fun and should be treated as an example.

#

name = name of rule as it appears on the cloud

ruletype = 1 - standard rule (smart rules not yet supported)

rulelocation = Cloud for cloud rules or Local for Cloud like standard rules.

They are run locally for speed or offline support.

sensor = name of sensor to monitor.

condition = string that appears on cloud for condition. If local rule use IsEqualTo, IsNotEqualTo, IsGreaterThan, IsGreaterOrEqual, IsLessThan, IsLessOrEqual

conditionvalue = used for local rules only.

commandparameter = 1 - command requires parameter, 0 - no parameter required.

severity = one of Critical, Information, Major, Minor, Warning for cloud usage

notificationtype = one of DeviceCommand, Email, Push, SignalR, WebHook, UI Alert, MQTT for cloud usage.

command = name of cloud command to execute.

#

[CloudSDKDefaultRule1]

name = SmartEdgeIoTTempToLow

ruletype = 1

rulelocation = Cloud

sensor = CpuTemperature

condition = CpuTemperature < 10

conditionvalue = 10

commandparameter = 0

severity = Critical

notificationtype = DeviceCommand

command = Reboot

[CloudSDKDefaultRule2]

name = SmartEdgeIIoTTempToLow

ruletype = 1

rulelocation = Local

sensor = CpuTemperature

condition = IsGreaterThan

conditionvalue = 10

commandparameter = 0

severity = Critical

notificationtype = DeviceCommand

command = Reboot

This section and the ones that follow are used to specify the rule information for the automatic cloud Rule setup for the Omega Sensor products. The sections must be named "[CloudSDKCustomOmegaRule<xx>]" where xx is a sequential number starting with 1. These are very specific to the actual real sensors plugged into the gateway. See below for a description of the required parameters.

#

Omega Rules Section

#

#

Uncomment to test/add custom omega rules

#

```
# Notes:
# You must know the naming format for USB SmartSensors.Use OmegaTools to obtain
#   <sensor_type><SensorName programmed into sensor><SensorID>
# sensor can be named device i.e. "SP_002_1_Input1_0x101133910109622"
# Output may be named any valid output ("command registered with system") in
#   the example below we use "SP_002_1_Output0_0x101133910109622Clear" which
#   is autogenerated when you plugged in that device.
# You can preset rules for devices not yet plugged in.
#


#[CloudSDKCustomOmegaRule1]
#name = OmegaCpuTemperatureHigh
#ruletype = 1
#rulelocation = Cloud
#sensor = CpuTemperature
#condition = CpuTemperature > 60
#conditionvalue = 60
#commandparameter = 0
#severity = Critical
#notificationtype = DeviceCommand
#command = SP_002_1_Output0_0x101133910109622Clear
#[CloudSDKCustomOmegaRule2]
#name = OmegaCpuTemeratureLow
#ruletype = 1
#rulelocation = Cloud
#sensor = CpuTemperature
#condition = CpuTemperature < 10
#conditionvalue = 10
#commandparameter = 0
#severity = Critical
```

#notificationtype = DeviceCommand

#command = SP_002_1_Output0_0x101133910109622Set

➢ Smartedge-iiot-gateway SDK examples

This section contains several examples of what is possible with the IoTConnect SDK. The functions to access the sensor / outputs must be written in python or provide a python interface for custom sensor /output functions. There are 19 default commands included, 10 default sensors, and 1 default rule included in the SDK.

Example: Add new Sensor Object for automatic cloud Attribute creation from defaults included with SDK.

Synopsis: This details what is required to add a sensor using the standard ones provided.

Description: Edit the configuration file and be sure the following entries are provided. From the defaults to add 1 additional sensor set "defaultobjectcount = 2". Ensure the username and password match the mobile app registration values. This will enable the System Health report to the cloud

---

Example: Add new Command for automatic cloud creation from the defaults included with the SDK.

Synopsis: This details what is required to add a command using the standard ones provided.

Description: Edit the configuration file and be sure the following entries are provided. From the defaults to add 1 additional command set "defaultcommandcount = 1". Ensure the username and password match the mobile app registration values. This will enable the Reboot command on the cloud for this gateway.

Example: Add new Rule for automatic cloud creation from the default example rule included with the SDK.

Synopsis: This details what is required to add a rule using the standard one provided.

Description: Edit the configuration file and be sure the following entries are provided. From the defaults to add 1 additional rule set "defaultrulecount = 1". Ensure the username and password match the mobile app registration values. This will enable the example rule "SmartEdgeIoTTempToLow" cloud rule. If the cpu temperature drops below 10 degrees C then reboot the gateway.

Example: Add your own Sensor for automatic cloud Attribute creation.

Synopsis: This details what is required to add your own sensor.

Description: Edit the configuration file and provide the following new section and entries. . From the defaults to add 1 additional Attribute add 1 to "defaultobjectcount". Then create a section "[CloudSDKDefaultObject< The number used for defaultobjectcount>" and add the appropriate entries. Now edit the user_functions.py and add the Python code function named in the "usepythoninterface=<name of python function>". Alternatively, you can add the python code to "example.py". Note: You can repeat this process for Commands and Rules.
