# NTX Embedded WM 18x API

Thank you for purchasing an NTX Embedded Wi-Fi module for Customer Connect Network Cloud. This API guide will help you understand how our module works and what you can do with it.

## About the WM18x

There are several models of the WM18x from NTX Embedded. API calls below work on all WM18x models unless otherwise noted.

The firmware on your WM18x is custom made for you. This document is the general API documentation for the WM18x. Your firmware may have special restrictions or additional calls. These differences will be provided to you by NTX Embedded as a separate document.

The WM18X has its own processor and is programmed for your application. It is designed to do as much of the work of communicating to the Internet or cloud services and doing Wi-Fi stuff as it can so your processor can do more important things. It is not a general purpose Wi-Fi module. As such the API allows only the most basic features needed to establish a connection to an access point and to send and receive the data. Everything else is handled for you by the NTX Embedded WM18x module. This is done both for security reasons and to minimize the work your system has to do.

## Customer Connect Network Cloud

This version of the API is designed to connect to the Customer Connect Network cloud services as an MQTT client. It implements some of the API as described in the Hounö documents:

- MQTT Messages (OvenEvents.docx), Version P16A, Dated 06-26-2018
- Oven Activation, Version 1.04, Dated 02-07-2018
- Firmware Update and WIFI Parameters, Version 1.03, Dated 06-26-2018

The Hounö hosted Customer Connect Network cloud services allow your device to securely connect to an MQTT broker and send periodic and live data from your device to the cloud for storage and/or display. It also has a mechanism to inform the device through the WM18x Wi-Fi module that files are available for download and the WM18x can retrieve those files and give them to the device. All of the hard work of connecting and maintaining the connection to the MQTT broker as well as packetizing the data properly is done by the WM18x module. The API calls below in the Network API section of this document are used to send and retrieve data from these cloud services.

## Connection Methods

The WM18x comes in several communication configurations including RS232, RS485, RS422.

It is also capable of communicating at various Baud rates. In order to prevent accidental changes to the serial settings, no API exists at this time to modify the serial settings on the fly. Your custom firmware for the WM18x will have the serial settings you specify that will communicate to your device.

The default serial settings are:

Baud: 115200; Data Bits: 8; Parity: None; Stop Bits: 1; Flow Control: None

## How to Read This Documentation

Anywhere the term Module is used it is referring to the NTX Embedded WM18x Wi-Fi module.

Anywhere the term Device is used it is referring to your electronics that are communicating to the Module using a serial connection.

Throughout the documentation actual values that are sent or received via serial communication between the module and the device will be **BOLD.**

Example: **NTX_CONFIG**

Anything that will be replaced by actual data will be indicated by {}.

Example: **NTX_AP**,{SSID},{password}

Where {SSID} and {password} must be replaced by the actual SSID and password to be valid. So if your SSID is Abcd and the password is 1234 the command would look like:

**NTX_AP,Abcd,1234**

All commands to the WM18X are formatted in the same way:

{Command},{Data}

Not all commands have data. If no data is sent, the comma is not needed.

Some parts of the {Data} may be optional. These will be indicated by square brackets [].

All commands to the WM18x should be terminated with a carriage return or line feed. Either one will cause the WM18x to accept the command and try to parse it.

## DEVICE STARTUP

When the WM18x powers on you will see the following message

NTX_RESTART,{version}

Example:

NTX_RESTART,XC20180921.01

The firmware version usually a 2 or 3 letter prefix indicating the custom build customer and the date in yyyymmdd format of the build. If a sub build number or special build is required it will be indicated by the numbers after the decimal. Not all versions will have a sub build number or decimal.

If the WM18x crashes it will automatically restart and send the NTX_RESTART,{version} message.

# AUTO CONNECT

The WM18x tries to automatically connect to an access point. It uses the last good SSID and password it stored to try to connect. If the last good SSID and password fails, it will try to connect using the previous SSID and password it stored. The module always keeps 2 sets of SSID and password, a primary and a backup. If the module has never been connected to an access point, it will not try to auto connect.

A backup SSID and password is stored so that the SSID and password can be changed remotely from the Customer Connect cloud. When the Customer Connect cloud sends a new SSID and password to the WM18x, it is stored in the backup SSID and password location. If the primary fails, the backup is tried. If the backup set successfully connects, it is moved to the primary position and the primary becomes the backup set. Only the backup is overwritten through API calls or from data sent from the cloud.

The module will continue to try to connect every 5 seconds or so until it successfully connects or is disabled by the device.

## ONLINE STATUS

The WM18x indicates the current online status of the module by automatically sending the following message to your device:

**NTX_ONLINE,**{status}

Possible {status}values:

0 – Not connected to an access point

1 – Connected to an access point but not connected to the Customer Connect cloud

2 – Connected to the access point and the Customer Connect cloud

Any time the online status is less than 2, any data commands (NTX_LC_xxx) sent to the WM18X will be discarded. The WM18x does not queue up data to be sent. It is the responsibility of the device sending the data to the WM18x to store offline data if it is not connected if needed. The WM18x will not inform you if the data is being discarded.

Any time the connection status changes, the WM18x will send the NTX_ONLINE,{status} message. Your device should be listening for this command at all times and should take proper action.

The NTX_ONLINE,0 or 1 message will be sent about every 10 seconds. Once the status reaches status 2, the status messages will not be sent unless the status changes or status is requested (**NTX_ONLINE**).

## Other Automatic Messages

There are other messages the module may send from time to time. These messages may be sent at any time but they will not interrupt a message/response sequence in process unless noted below. In other words if you send an API call like **NTX_CONFIG**, the **NTX_CONFIG**,{} response will always be the next message you can expect to get back.

**NTX_SEND_CNT** – This message prompts the device to send the counters using **NTX_LC_SVC_CNT**. This is initiated from the Network cloud portal by a user. This message is sent only once. If it is missed or disregarded, the user will have to make the request again before **NTX_SEND_CNT** is resent to the device.

**NTX_SEND_FCNT** - This message prompts the device to send the fault counts using **NTX_LC_FSTAT**. This is initiated from the Network cloud portal by a user. This message is sent only once. If it is missed or disregarded, the user will have to make the request again before **NTX_SEND_FCNT** is resent to the device.

**NTX_FILE** – indicates that a file is waiting to be sent from the module to the device. This message is repeated every few seconds until it is handled. See **NTX_FILE,1** below.

## Storing Data When Offline

Sometimes in the field, routers go down, the internet service is interrupted or other factors cause a disruption in the ability of a device to send data to the cloud services. Your data is very important. Both the WM18x module and the Network cloud have mechanisms for detecting a disconnection, attempting to automatically re-connect and informing the user that a disconnection has happened.

It is important that you take advantage of these methods to minimize down time. However there are circumstances where some time will elapse before the connectivity issue is resolved. In these cases, if data continuity is important you, it is recommended that, if your device is capable, you build in the option to store data on your device if the module is offline. After the module comes back online, you can send historical data with a date and time stamp when the action actually happened through the module so that the data is shown in the proper order in the cloud portal. See **NTX_LC_ACTIVITY** for more details.

## GENERAL API CALLS

**General API Call Summary:**

- NTX_AP – set the access point name and password
- NTX_CONFIG – show the current configuration
- NTX_DISABLE – Turn Wi-Fi Off
- NTX_ENABLE – Turn Wi-Fi On
- NTX_FORCE_RESET – force the module to reset
- NTX_GET_TIME – get the time
- NTX_GET_TIME – get the date
- NTX_LISTAP – return a list of access points
- NTX_ONLINE – check the online status
- NTX_RSSI – get the Wi-Fi signal strength
- NTX_SET_DEVICESERIAL – set the device serial number
- NTX_WIFI_VERSION – get the Wi-Fi version

**NTX_AP**,{SSID},{password}

{SSID} – the name of the access point to connect to. (MAX 75 characters)

{password} – the password for the SSID. (MAX 75 characters)

Example: **NTX_AP,**MySSIDName,MyPass#ord

Response: **NTX_ONLINE**,{online status}  See NTX_ONLINE for possible values of {online status}

Notes:

1. Every NTX_AP call will cause the WM18x to disconnect from the current SSID if it is connected and try to connect using the new SSID name and password.
2. The previous SSID and password will be retained. If the WM18x connects to the new SSID, the old SSID and password will become the backup. If the WM18X does not connect to the new SSID, the new SSID and password will be stored as the backup SSID and password.
3. Making a call to NTX_AP will trigger at least one NTX_ONLINE message
4. Sending the NTX_AP call more than once will cause an interrupt of any process going on including another NTX_AP call. It is recommended that you wait from the NTX_ONLINE response to the NTX_AP call before sending another. It is further recommended that you call NTX_CONFIG after the NTX_ONLINE response. The Status in the response to the NTX_CONFIG will be TRYING as the module works on getting a connection with the AP. You can poll the NTX_CONFIG call until the status is CONNECTED or FAILED. (See Workflow)

**NTX_CONFIG**

Example: **NTX_CONFIG**

Response: **NTX_CONFIG**,{MAC},{SSID},{Connect Status:IP Address:channel},{NetworkSerial}

{MAC} = The MAC address of this module in the form of

{SSID} = The SSID that the module is connected to or is trying to connect to

{Connect Status:IP Address:channel} = The connect status, IP Address if connected and channel connected to. Possible values of connect status are:

- GOT IP
- TRYING
- FAILED

GOT IP indicates that a DHCP address was obtained.

TRYING indicates that the module is still trying to connect to the access point.

FAILED indicates connection to the access point was denied or failed.

NetworkSerial is the unique serial number Network knows this module by. By default the serial number will be the HEX values of the MAC address of each module without any colons between segments.

**NTX_DISABLE**

This API call causes the module to disconnect from any API, stop attempting to connect and stop all transmission. No API call will work when disabled, until the module is Enabled. This mode also puts the module into low power mode.

Some customers do not like the idea of a Wi-Fi device in their devices. The module cannot always be easily physically disconnected from the device so this method can be called on power up from the device and it will put the module into a mode where it is not able to connect to a router or to be connected to in any way.

**NTX_ENABLE**

The module is enabled by default. A call to NTX_ENABLE when the device is already enabled does not have an effect.

**NTX_FORCE_RESET**

Example: **NTX_FORCE_RESET**

Response: None (Module resets, see Startup above)

This call is to be used in case something on the module goes wrong and a soft reset of the module is required.

**NTX_GET_TIME**

Example: **NTX_GET_TIME**

Response: **NTX_GET_TIME**,{hh:mm:ss GMT}

The time is current GMT time.

**NTX_GET_DATE**

Example: **NTX_GET_DATE**

Response: **NTX_GET_DATE**,{yyyy:mm:dd GMT}

The date is current GMT date.

**NTX_LISTAP**

Example: **NTX_LISTAP**

Response: **NTX_LISTAP,**{AP1,AP2,…,APn}

{AP1,AP2,…,APx} is a comma delimited list of the SSID for all access points detected.

Notes:

1. If no access points are found, **NTX_LISTAP,NO_AP** will be returned
2. The strongest RSSI access point will be listed first.
3. Duplicate access point names are not included.
4. Only 10 detected access points will be listed at a time.
5. There is currently no way to get the next page of access points

**NTX_ONLINE**

Example: **NTX_ONLINE**

Response: **NTX_ONLINE**,{online status}

See Online Status above.

This method is to be used when an immediate check of the online status is needed.

**NTX_RSSI**

Example: NTX_RSSI

Response: NTX_RSSI,{RSSI value}

{RSSI value} – range -30 to -90

See the **NTX Embedded WM18x Signal Strength and Quality Guide** for more information and strategies for maximizing signal strength.

**NTX_SET_DEVICESERIAL**

Example: **NTX_SET_DEVICESERIAL**,{serial number}

Response: None

{serial number} = Max 25 printable characters. This is the number that you will see on Let's Cook. It is initially used as the friendly name for this device.

Sends a serial number of the device to the module. The serial number is required to initiate a connection to the Network service. See connection workflows.

**NTX_WIFI_VERSION**

Example: **NTX_ WIFI _VERSION**

Response: **NTX_ WIFI _VERSION**,{version_number}

This call is for informational purposes only. You may want to display the Wi-Fi firmware version or use it for diagnostic purposes.

## NetworkAPI

These API calls are specifically designed to send or receive data to the Network cloud services. Since they are making a call out to the Internet the response may be asynchronous so many of them are designed with no immediate response from the module. If you see Response: NONE, the device should make the call and move on to the next call without waiting for a response. Always observe call timing minimums.

This API refers to a document called the NetworkMatrix (shortened to “Matrix File” below). This is a document that should be created and negotiated between you and Houno that specifies the specific data you are sending to the cloud service. It identifies all the IDs you will send as data to this API that are then sent to the Network cloud.

It is likely that you will work with Houno to customize the Network cloud pages to fit your needs. In some cases you may change the name of tabs or screens, you may change where data is shown or you many change the labels or icons associated with data. In the API calls below we try to reference what we know about where and how data is displayed so you can easily test after you use the API. However, as you customize the Network cloud interface please be aware that the Module does not dictate how or where data is displayed in the Network cloud pages.

The API for the Network cloud is designed to minimize code changes on the Module. There may be cases where changes to ask Houno to make to the Network cloud require changes on the module to properly work.

**NetworkAPI Call Summary:**

- NTX_LC_ACTIVITY – Send activity events
- NTX_FILE – File transfer information and notifications
- NTX_LC_FSTAT – Send the fault counts
- NTX_LC_LIVE – Send live card data to Let’s Cook
- NTX_LC_META_ADDRESS – Send location data
- NTX_LC_META_DEVICE – Send device data
- NTX_LC_STATE – Send the state of the device
- NTX_LC_STATUS – Return the status of Network connectivity
- NTX_LC_SVC_CNT – Send the counters

**NTX_LC_ACTIVITY**

Example:
NTX_LC_ACTIVITY,{date_time},{activity_id},{group}[,{data_1_name},{data_1_type},{data_1_value},{data_2_name},{data_2_type},{data_2_value},…,{data_n_name},{data_n_type},{data_n_value}]

Response: None

{date_time} if this activity is live/realtime data send -1.

If this is offline data, meaning stored, the {date_time} must be in the form of

{yyyy}-{mm}-{dd}T{hh}:{mm}:{ss}Z

%d-%02d-%02dT%02d:%02d:%02dZ

And all dates and times must be in GMT/UTD.

{activity_id} – integer – the id assigned in the Matrix file to the event you are reporting

{group} – integer - each activity is assigned to a group in the Matrix File. You must send the group associated with the activity_id or the data may not properly display in Let's Cook.

{data_x_name} – string - the name of the data you are sending for this activity.

{data_x_type} – string - the type of data value you are sending. Possible values are int, str, bool, dec

{data_n_value} – various – the value associated with the data_n_name

Activities are single events that happen on the device that you want to record in the cloud data.

Examples of Activities might be the device being turned on, a door opening, a cook starting, a motor coming on, a temperature hitting a threshold, the volume being changed, or the user navigating to a certain screen. Activities can have data associated with them that is then displayed in the cloud. For instance if the event is a cook starting, you could send the recipe name or the time with the event. If the volume is changed, you could send the new volume as the data. Data can be simple or complex. If your event is a motor turning off, you could send the duration it was on and the position of the motor when it stopped. Activities should be used only for discrete actions that are not likely to happen many times a second.

Activity events should be sent to the Customer Connect Cloud as they happen. The device should queue the activities and send them one at a time. Observe the timing requirements of the module.

Notes:

- Only one Activity can be sent at a time
- A maximum of 5 sets of data can be sent with each activity
- Each occurrence of an activity should be sent only once. If the same occurrence of an activity is sent more than once, even with the same date code, it will be recorded each time as a separate activity.
- Activity data is optional

**NTX_FILE**

The WM18x module has the ability to receive files from Customer Connect. These files can be menu updates, images, firmware updates or just about anything. These files can be retrieved from Customer Connect when they are available or just when needed.

The process of moving a file across a serial connection is slow so the Device and the Module will not be sending and receiving regular data information while this is happening. In other words, if you need to constantly send temperature up to the cloud, it will have to stop while the file is being sent from the module to the device. Often, file transfers only happen when the device is in an idle or off state.

When a file is available, the module will send a message to the device every 20 seconds or so: NTX _FILE

When the device is ready to receive the file, it should send NTX_FILE,1.

The module will respond with NTX_FILE,{file_type},{file_size},{Name}

{file_type} is defined in the Matrix File. This can be used to decide the file name and/or how to use the data.

{file_size} is in bits. Files must be smaller than 1GB at this time.

{Name} is the name assigned to this file. It does not necessarily represent the filename. It is just a name that goes with the data. It can be used any way needed.

The device should then prepare to receive the file in chunks of 32767 bits at a time through the serial port.

The device then sends: NTX_FILE,2 and the module responds with the file data in chunks of 32767. The last chunk will be smaller than 34767 bits.

Once the last chunk is sent, the module and the device should return to normal operations.

**NTX_LC_FSTAT**

Example: **NTX_LC_FSTAT**,{fault 1 id},{fault 1 count},…, {fault n id},{fault n count}

{fault 1 id} – in integer defined in the Matrix file that tells Network which fault is being counted.

{fault 1 count} – an integer

Response: None

The NTX_LC_FSTAT call sends fault count data to the Customer Connect cloud.

Notes:

1. The faults for a device are defined in the Matrix File
2. The fault ID can be a numeric or alphanumeric ID but should never be more than 5 characters

3. IMPORTANT: One entry for each fault id in the Matrix File must be represented in each call to NTX_LC_FSTAT even if the count is 0. In other words if you have 10 fault IDs, all 10 fault IDs and counts must be sent with every call of NTX_LC_FSTAT. Any fault IDs that are not sent will not be shown in the Network data page.
4. Faults can be sent no more than once an hour. We suggest sending them only once a day.
5. A maximum of 15 faults are allowed

**NTX_LC_LIVE**

Example:
NTX_LC_LIVE,{live_data_1_name},{live_data_1_type},{live_data_1_value}, … ,{live_data_n_name},{live_data_n_type},{live_data_n_value}

{live_data_x_name} is the name of the live data as defined in the Matrix File. Use the exact value in the Matrix File

{live_data_x_type} can be: int, str, bool, dec. If the data type is a Boolean send the string true or false (lower case). If the data type is a decimal, it is limited to 2 decimal places.

Response: None

Live data is intended to show some event that is happening right now that is interesting to the user to observe in near-real time. Examples of live data would be the recipe name and count down timer while cooking or the actual temperature during warmup or the countdown timer on a cleaning cycle. The data sent as live data is not stored in the cloud so it cannot be referenced later and charted over time.

Live data shows up on the back of the live card in Let's Cook. It is sent only if the device is being actively monitored and is not stored in Let's Cook. The device should always send live data to the module when but it will only be sent to the cloud if the device is being monitored.

Notes:

- Live data can be sent in any order.
- Because the amount of live data effects how quickly it can be transmitted, we suggest keeping the amount of live data to a minimum.
- Live data should not be sent more than once a second
- Live data does not need to be sent all at once. It can be updated piece by piece if needed.
- Maximum number of pieces of live data per call - 5

**NTX_LC_META_ADDRESS**

Example: NTX_LC_META_ADDRESS,{friendly_name},{device_info_1_name},{device_info_1_value}, … , {device_info_n_name},{device_info_n_value}

{friendly_name} is a string that will show up as the name of this device. Often the serial number is used but it can be any name up to 35 characters

{device_info_1_name} is a string – use the exact value in the Matrix File.

{device_info_1_value} is a string. Max 35 characters.

Response: None

Address meta data is usually information like city, state, country, address, zip etc. Meta data will be defined in the events matrix.

Notes:

- Meta data is displayed in several places throughout the Network screens
- Maximum number of address meta data sets per call is 5
- You may make more than one call to NTX_LC_META_ADDRESS if you have more than 5 sets of data

**NTX_LC_META_DEVICE**

Example: NTX_LC_META_DEVICE,{device_id}{device_info_1_name},{device_info_1_value}, … , {device_info_n_name},{device_info_n_value}

{device_id} – string – the serial number of your device. Maximum of 20 characters.

{device_info_1_name} - string – use the exact value in the Matrix File

{device_info_1_value} – string – the value associated with the device_info_name

Response: None

Device meta data is usually information like version number, model number, serial number etc. Meta data will be defined in the events matrix.

Notes:

- Meta data is displayed in several places throughout the Network screens
- Maximum number of device meta data sets is 5
- You may make more than one call to NTX_LC_META_DEVICE if you have more than 5 sets of data

**NTX_LC_STATE**

Example: NTX_LC_STATE,{state_name},[{data_1_name},{data_1_type},{data_1_value}…]

Response: None

{state_name} is the name of the state assigned in the events matrix.

[{data_x_name},{data_x_type},{data_x_value}] is optional.

{data_1_name} – string – the string identifier of the data about to be sent.

{data_1_type} – string – the type of the data_x_value about to be sent

{data_1_value} – the value

The state of the device shows up on the front and back of the live card in Let's Cook. It is sent only if the device is being actively monitored and is not stored in Let's Cook. The device should always send it when the state changes to the module but it will only be sent to the cloud if the device is being monitored.

Because Network does not store the state, if you change screens or refresh the cloud, the state in Network may revert to STANDBY. This is the default state if it has not received a state recently. We suggest you send the current state as often as makes sense. It should never be sent more often than once a second. A more reasonable period is about once every 10 seconds.

Notes:

- Not all states have data so it is shown as optional.
- If you sent a state that is identified in the Matrix file as needing data, without the data, the state may not display properly in Let's Cook.
- Maximum number of data sets per state is 10
- If you also want to collect state changes as activities you need to send an activity as well as a state API call to the module.

**NTX_LC_STATUS**

Responds with the status of the connection to the Network cloud.

Example: NTX_LC_STATUS

Response: NTX_LC_STATUS,{lc_status}

{lc_status} = 0 – Not connected to the NetworkMQTT broker

{lc_status} = 1 – Connected to the NetworkMQTT broker

Notes:

1. A connection status of 1 may mean that the device is fully connected or may be in pending or approved but not put into a group. The module does not know which of these modes the device is in only that it is connected or disconnected from the Network server.
2. Use the Network online portal to determine the status of the device connection.

**NTX_LC_SVC_CNT**

Example:
NTX_LC_SVC_CNT,{counter_1_id},{counter_1_type},{counter_1_value}, … , ,{counter_n_id},{counter_n_type},{counter_n_value}

{counter_x_id} is the integer id of the counter as defined in the events matrix.

{ counter_x_type} can be: “count” or “hours”

{counter_x_value} is an integer

Response: None

Counters are shown in the Service Counters tab of Let’s Cook. All counters must be sent every time. If you only send 3 counters in a single call to **NTX_LC_SVC_CNT**, you will only see 3 entries in the Service Counters tab.

Notes:

- The units of the service counters must be worked out in advance with the Customer Connect cloud services system.
- Maximum number of counters that can be sent is 10

# Workflows:

## Connection

When your device first establishes the serial connection to the WM18x, you should wait for the **NTX_RESTART** message before sending anything to the module. **NTX_RESTART** indicates that the module is communicating with your device properly.

Next, wait for the **NTX_STATUS** message. This message will indicate if the module is communicating with the AP and with Let's Cook.

If your device takes longer to come up and start communicating serially and this message is missed, it is suggested that you make calls to **NTX_ONLINE**

From the results of the **NTX_STATUS** or **NTX_ONLINE** message, you should be able to decide what to do next.

Since the WM18x connects to the last known access point automatically, after the initial connection is established there is very little for the device to do on startup after the initial connection is established.

In most systems, there is a UI to set the SSID and password. Initially you should call **NTX_CONFIG** so you can show the MAC address in the UI. This is important because the MAC address is used to uniquely identify the module to the Network cloud. Your device's serial number may change but the MAC will not change so it can always be used to find the device if you are unsure of the serial number.

When a device connects for the first time, weather after a power cycle or when the module status transitions from offline to online, it is recommended that the device, get the current time and date and then send the current status (NTX_LC_STATUS) and meta data (NTX_LC_META_DEVICE). This makes sure Customer Connect has everything it needs to show the current status of the device.

After connection to the access point is successful, the module is ready to connect to the Network system. To start this process it needs the serial number of the device. Use NTX_SET_DEVICESERIAL,{serial number} to set it.

First Time Connection Check List:

- Wait for **NTX_RESTART** and or **NTX_ONLINE**
- If NTX_ONLINE value is 0, Send **NTX_AP**,{SSID},{password}
- Wait for **NTX_ONLINE**,1
- Send **NTX_SET_OVENSERIAL**,{oven serial}
- Wait for **NTX_ONLINE**,2
- Log into Network and select the new device in the pending list and either confirm it or put it in a group. Until this is done, the device will not report NTX_ONLINE,2. As soon as the device is confirmed online, the module will get a message and inform the device it is ready to receive data.
- Once NTX_ONLINE,2 is received

    - Send and handle the response for **NTX_DATE** and **NTX_TIME** if needed
    - Send **NTX_LC_META_DEVICE**,{…}
    - Send **NTX_LC_META_ADDRESS**,{…}
    - Send **NTX_LC_STATE**,{…}
- Continue to monitor the serial connection and send data as needed

Second Time Connection Check List

- Wait for **NTX_RESTART** and or **NTX_ONLINE**
- If NTX_ONLINE value is 0, Send **NTX_AP**,{SSID},{password}
- Wait for **NTX_ONLINE**,1 or 2
- If NTX_ONLINE,2
    - Send **NTX_SET_OVENSERIAL**,{oven serial}
    - Send and handle the response for **NTX_DATE** and **NTX_TIME** if needed
    - Send **NTX_LC_META_DEVICE**,{…}
    - Send **NTX_LC_META_ADDRESS**,{…}
    - Send **NTX_LC_STATE**,{…}
    - Continue to monitor the serial connection and send data as needed

## File Download

In Customer Connect – send a file to a device

Module sends – NTX_FILE every 20 seconds to the device. The device can ignore this message for as long as it wants.

Device sends – NTX_FILE,1 which prompts the module to return the file header info and get ready to send the file which may include downloading it depending on the size.

Module sends – NTX_FILE,{header}

Device sends – NTX_FILE,2

Module sends {data packet 1}{data packet 2}{…}{data packet n}

## Monitoring The Module

The device should have three modes of monitoring the serial port.

Mode 1: Send and Receive. Certain API calls above always respond to the API call. In this case it is guaranteed that when the device sends an API request, the response will come in before anything else. It is always possible that something goes wrong so even in this mode a timeout should be set. If no response comes in within a few seconds, the call should abort or repeat. Likewise if a response comes in that was unexpected, the call should abort or repeat.

Mode 2: Listening. In several cases, the module may send information to the device based on input from the cloud like a request to refresh the statistics or a change in connectivity state. For each of these, the device needs to be listening and needs to handle them when they come in.

Mode 3: File Transfer. In this case, there is no end of data terminator character, it is all based on the buffer size. The rest of the modes should be suspended while in this mode.

Ongoing Monitoring Check List (Pseudocode)

1. Am I waiting on a response to some message I sent? Did the response come in? If so handle it. If not continue to wait for the response until I time out.
2. Did NTX_ONLINE,{} come in? If it did, does it indicate we are offline? If so change to storing the activity data offline. Show the user we are offline so they can fix it.
3. Did NTX_SEND_FCNT, NTX_SEND_META, NTX_FILE come in? If so handle them.

## Sending Offline Data

As discussed previously, it is recommended that if the module is offline, the device should store activity data locally if it can until the module comes back on line. A date and time stamp should be stored with each activity along with all the data needed to make the NTX_LC_ACTIVITY API call.

When it does come back on line, the activity data can be sent with the time stamp. The cloud will use the time stamp to put the data in the right order. You can send offline data mixed with real time data. We recommend a queue of some kind that is being filled with a mix of real time and offline data. Be careful to leave your program time to look for incoming messages and to do other work.

| | |
|---|---|
| Size | 18.0*25.8*2.8（±0.2）mm |
| SPI Flash | Default 32Mbit |
| Bluetooth | Bluetooth 4.2 BR/EDR and BLE standards |
| Wi-Fi | 802.11 b/g/n |
| Interface | UART、SPI、SDIO、I2C、PWM、I2S、IR、ADC、DAC |
| On-chip Sensor | Hall sensor, Temperature sensor, Capacitive touch sensor |
| IO Port | 22 |
| UART Baudrate | Support 300~4608000bps,De115200 bps |
| Frequency Range | 2412 ~2462MHz |
| Antenna | PCB Antenna, Max. gain 0dBi |
| Transmit Power | 802.11b: 17±2 dBm (@11Mbps)<br>802.11g: 14±2 dBm (@54Mbps)<br>802.11n: 13±2 dBm (@MCS7) |
| Receiving Sensitivity | CCK, 1 Mbps : -90dBm<br>CCK,11Mbps: -85dBm<br>6 Mbps (1/2 BPSK): -88dBm<br>54 Mbps (3/4 64-QAM): -70dBm<br>MCS7 (65 Mbps,72.2 Mbps): -67dBm |
| Power Dissipation | 300mA@3.3V |
| Security | WPA/WPA2/WPA2-Enterprise/WPS |

| | |
|---|---|
| Power Supply Range | 3.0V ~ 3.6V |
| Operating Temperature | -20℃ ~ 85℃ |
| Storage Environment | -40℃ ~ 90℃ , < 90%RH |
| Weight | 1.5g |

BOM

| Part Description | Quantity | Package | DigiKey Part Number | Reference |
|---|---|---|---|---|
| CAP CER 0.01uF 16V X7R 0603 | 2 | 0603 | 1276-1926-2-ND | C1,C3 |
| CAP TANT 100UF 20% 4V 1206 | 1 | 1206 | 399-3688-2-ND | C2 |
| CAP CER 0.1UF 16V 10% X7R 0603 | 1 | 0603 | 1276-1005-2-ND | C16 |
| DNP - CONN HEADER VERT 2POS .100 TIN | 0 | | A31112-ND | JP1 |
| FERRITE BEAD 30 OHM @100MHz 3A 0805 1LN | 1 | 0805 | 1276-6376-2-ND | L1 |
| RES 1.0K OHM 1/10W 5% 0603 SMD | 4 | 0603 | 311-1.0KGRTR-ND | R1,R2,R3,R4 |
| RES 10K OHM 1% 1/10W 0603 SMD | 1 | 0603 | RMCF0603FG10K0TR-ND | R8 |
| RES 10 OHM 1/10W 5% 0603 SMD | 2 | 0603 | 311-10GRTR-ND | R6,R7 |
| 2mm 10X1 pin header (use 2 pieces) | 2 | | 880-70-010-10-001101-ND | U1 |
| Wi-Fi + BT + BLE module, AI Thinker | 1 | | | U2 |
| TVS DIODE 24VWM 44VC SOT23 | 1 | SOT-23 | NUP2105LT1GOSTR-ND | U3 |
| PCB 2-layer FR-4 0.8mm, 25mm X 34mm | 1 | | | |

PCB Layout

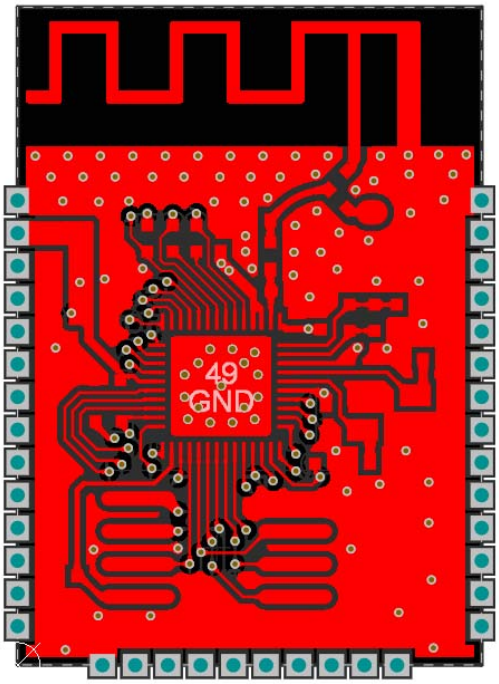


Figure 1 ESP32-S PCB Top Layer

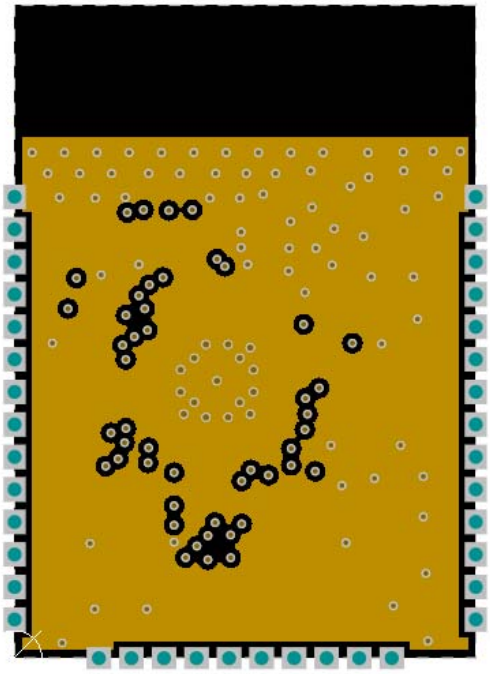

Figure 2 ESP32-S PCB GND Layer

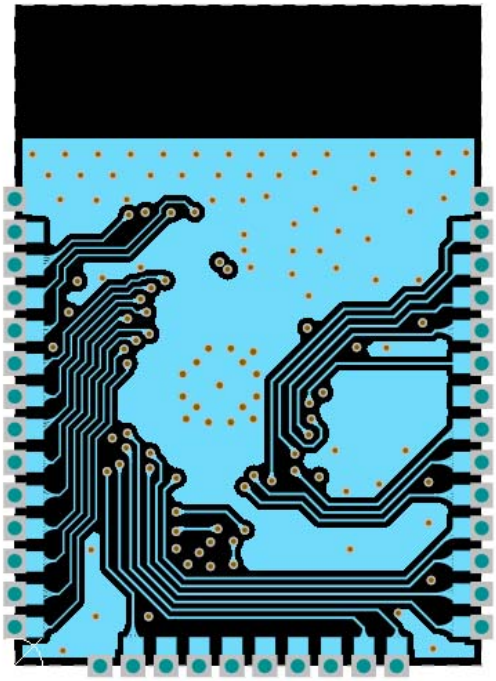

Figure 3  ESP32-S PCB POWER Layer


Figure 4  ESP32-S PCB Bottom Layer

# FCC Statements

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:
(1) This device may not cause harmful interference, and
(2) this device must accept any interference received, including interference that may cause undesired operation.

Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

**FCC Radiation Exposure Statement**
The modular can be installed or integrated in mobile or fix devices only. This modular cannot be installed in any portable device, for example, USB dongle like transmitters is forbidden.
This modular complies with FCC RF radiation exposure limits set forth for an uncontrolled environment. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter. This modular must be installed and operated with a minimum distance of 20 cm between the radiator and user body.

If the FCC identification number is not visible when the module is installed inside another device, then the outside of the device into which the module is installed must also display a label referring to the enclosed module. This exterior label can use wording such as the following: “Contains Transmitter Module FCC ID: 2AMU6WM181 Or Contains FCC ID: 2AMU6WM181”

When the module is installed inside another device, the user manual of this device must contain below warning statements:
1. This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions:
(1) This device may not cause harmful interference, and
(2) This device must accept any interference received, including interference that may cause undesired operation.
2. Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.
The devices must be installed and used in strict accordance with the manufacturer's instructions as described in the user documentation that comes with the product.

The host product manufacturer is responsible for compliance to any other FCC rules that apply to the host not covered by the modular transmitter grant of certification. The final host product still requires Part 15 Subpart B compliance testing with the modular transmitter installed.
The end user manual shall include all required regulatory information/warning as shown in this manual, include:
This product must be installed and operated with a minimum distance of 20 cm between the radiator and user body.