版本	0.11
日期	2025-04-30
固件	1.0.28

D21 eStation 开发者手册

本文档仅适用于 ETAG ESL Gen 3.0 和 DSL Gen 1.0 系统集成。

技能支持：huanghaipengonline@hotmail.com

历史版本

版本	日期	概括	作者	审阅
0.1	2024-08-07	文件初始化	黄海鹏	黄海鹏
0.2	2024-08-13	增加设备侧的 Topic QoS 等级描述	黄海鹏	黄海鹏
0.3	2024-08-14	修正了 TaskResult 数据结构定义的错误	黄海鹏	黄海鹏
0.4	2025-01-05	增加安全通信描述，修改了 Pattern 定义	黄海鹏	黄海鹏
0.5	2025-01-07	增加基于 Bytes 数组的推送版本接口	黄海鹏	黄海鹏
0.6	2025-01-08	对 Bytes 数组的推送接口的补充说明	黄海鹏	黄海鹏
0.7	2025-02-17	增加 Topic Alias 增加 DSL 类型任务接口、OTA 接口	黄海鹏	黄海鹏
0.8	2025-03-01	Pattern 部分勘误，Topic 名称调整	黄海鹏	黄海鹏
0.9	2025-03-25	修正了密钥通信的错误	黄海鹏	黄海鹏
0.10	2025-04-10	修正了 DLS 类型任务接口数据结构定义 增加 RGB 组合颜色表	黄海鹏	黄海鹏
0.11	2025-04-30	补充 DSL 类型任务常亮、OTA MD5 值的说明	黄海鹏	黄海鹏

目录

1 介绍

1.1 背景

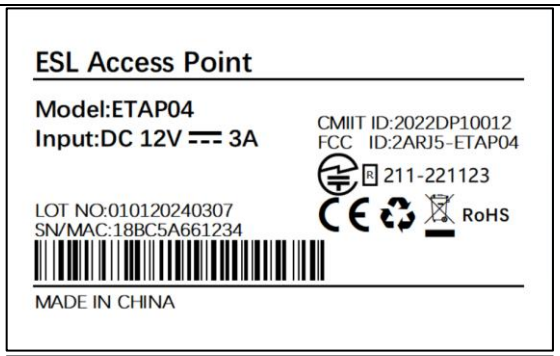
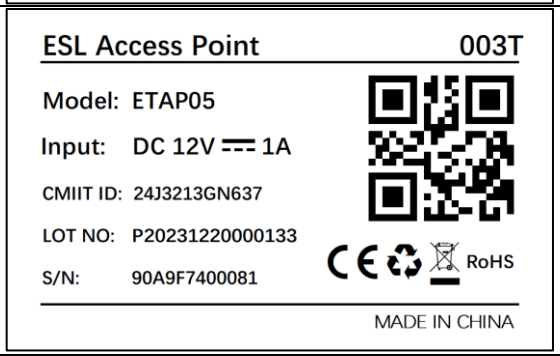
eStation IoT 设备，类型代码 AP05，是为系统集成商设计的，用于快速开发他们的业务项目，使用 MQTT v5 协议。

AP04 与 AP05 的区别：在开始使用 eStation 之前，请仔细了解 eStation AP04 和 AP05 之间的区别。它们都是用 MQTT 作为通信协议，但协议的 Topics 和数据结构定义略有不同。

AP05 标准版与开发者版本的区别：AP05 为系统集成上提供的开发者版本与为 eRetail 3.2/4.0 提供的标准版本都遵循同一的 4 位 ID 编码，但是它们的协议 Topics 和数据结构定义不同。本文只涉及到开发者版本的内容。

你可以通过设备背面的铭牌来进行区别。

AP04 Developer Edition、AP05 和 AP05 Developer Edition 的外观区别如下:

eStation Developer Edition (AP04)	 
eStation (AP05)	

eStation Developer Edition (AP05)	

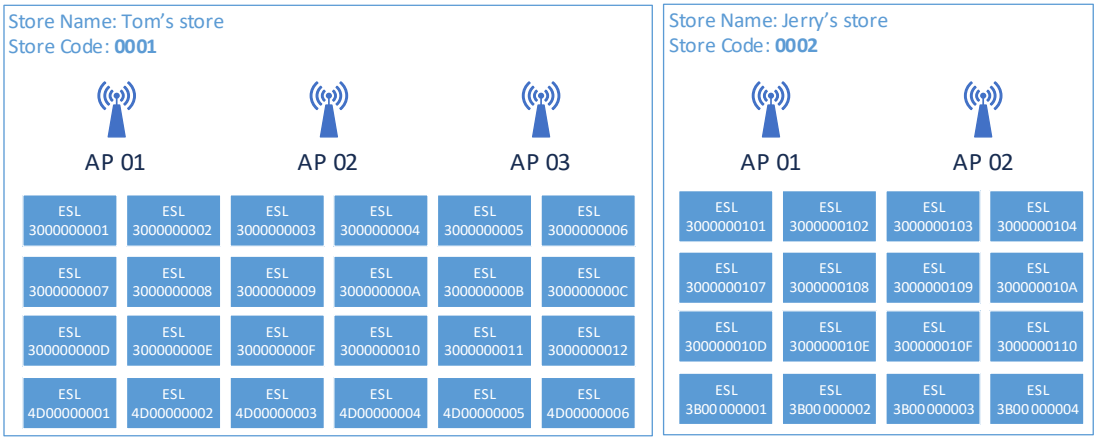
eStation Developer Edition 在 GitHub 中有一个演示项目。地址是: https://github.com/andersonhwang/eStation_Developer_Edition.

备注：该项目尚在开发中，目前已完成.NET 版本的 Console 和 WPF MVVM 内容。

在你开始使用 eStation 前, 你应该了解以下关键点:

1. **AP**: 与 ESL 相连的射频接入点。
2. **AP ID**: AP05 使用一个 4 位 ID 作为其全局唯一标识。该 ID 可以从张贴在设备背面的铭牌上获取到。
3. **ESL**: 电子货架标签，这里特指是使用低功耗蓝牙通信协议（BLE 5.0）的 ESL Gen 3.0。
4. **DSL**: 带有导轨供电的电子货架标签，当前仅限 2.4 寸的 TFT 价签。
5. **MQTT**: eStation (AP05) 使用 MQTT 通信协议。
6. **Base64String**: 将图像转换为 Base64String。有关 Base64String 的更多信息，请参考 Base64 Encode and Decode - Online.
7. **MessagePack**: eStation (AP05) 使用 MessagePack 数据格式来减少数据包的长度。有关信息包的更多信息，请参考 MessagePack: It's like JSON. but fast and small. (msgpack.org).

例如，你的项目有两个商店，像这样:



数据结构应该是:

Store Code	AP ID	ESL ID
0001	01	300000000001, 300000000002, 300000000007…
	02	300000000003, 300000000004, 300000000009 …
	03	300000000005, 300000000006, 30000000000B…
0002	01	300000000101, 300000000102, 300000000107…
	02	300000000103, 300000000104, 300000000109…

注意: 通常，你的应用程序应该记住与每一个 ESL 成功通信的 AP，但物理上，ESL 不存在与任何 AP 绑定关系。ESL 只是逻辑上尝试记住成功向其发送数据的最后一个默认 AP 的 ID。

1.2 eStation 的系统结构

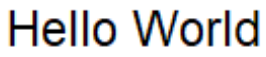
基本上，你的项目将包含 ESLID 和图像数据(Base-64String)的任务发送到 eStation，eStation 将使用 2.4GHz 射频(RF)将图像数据发送到确切的 ESL，并且 eStation 将结果返回给你的项目。换句话说，eStation 位于应用程序和 ESL 之间。

本文档将描述两个方面: 你的应用程序侧和 eStation 设备侧.

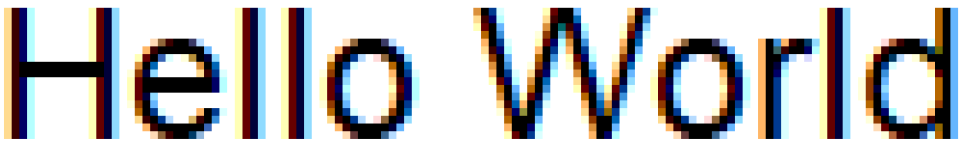
请记住，如果你的应用程序和站点不在同一个专用网络中，你应该添加一个强密码来保护连接，并且添加一个 X.509 证书。

1.3 有关图片

由于电子 ESL 屏幕是纯黑白或黑白红（黄）的屏幕，没有灰阶。假如你的图像看起来像:



放大 800%是这样的:



去除灰度部分后，它看起来像:

字体越大，锯齿状越清晰。

不推荐使用 ESL 来显示图像，如果需要显示，需要开发人员对图像进行抖动，以达到灰阶显示的效果。

一个用 C#实现图像抖动的算法是: Even more algorithms for dithering images using C# - Articles and information on C# and .NET development Topics • Cyotek

例如，一个图像看起来像:

使用抖动和不使用抖动图像效果如下:

2 使用 eStation

如前所述，不存在与 eStation 的代码级的集成。你应该管理应用程序中的连接和通信。

有 10 个 Topic，4 个 Topic 从 eStation 设备侧发布，6 个 Topic 从应用程序侧发布。

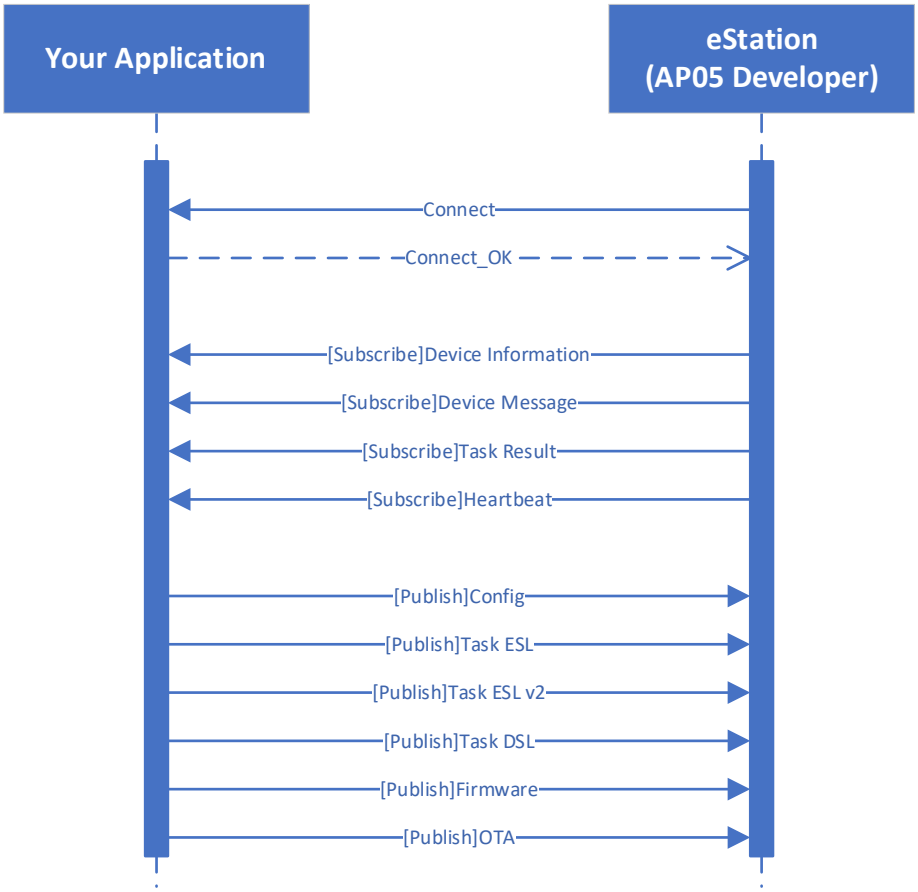
从你的应用程序侧，eStation 设备将订阅以下 Topic（Topic Alias）：

1. **config**(0x01): 你可以使用此 Topic 配置 eStation。
2. **taskESL**(0x02): 你可以发布带有此 Topic 的 ESL 任务（图像数据为 Base64 字符串格式）——此 Topic 已经不推荐使用，仅保留向前兼容。
3. **taskESL2**(0x03): 你可以发布带有此 Topic 的 ESL 任务（图像数据为 Bytes 数组格式）——推荐使用此 Topic。
4. **taskDSL**(0x04): 你可以发布带有此 Topic 配置 DSL 任务（当前仅限 2.4 寸 TFT 导轨供电价签）。
5. **firmware**(0x05): 可以使用此 Topic OTA eStation，或者预存 ESL DSL 的固件，供 0x05 ota 的 Topic 使用。
6. **ota**(0x06): 你可以使用此 Topic OTA ESL 和 DSL。

从 eStation 设备侧，你的应用程序应该订阅以下 Topic（Topic Alias）：

1. **infor**(0x80): 当 eStation 与服务侧建立连接后，将会上报设备的相关信息。
2. **message**(0x81): 当事情发生时，eStation 将返回一些消息代码。
3. **result**(0x82): eStation 将在与 ESL 通信后返回结果。
4. **heartbeat**(0x83): eStation 将定期返回当前状态。

注意：你可以不指定 Topic Alias（即默认 0），但如果指定了 Topic Alias，则会优先判断 Topic Alias。



Note: Publish/Subscribe is from your application side

2.1 连接 eStation

eStation 作为 IoT 设备客户侧工作，它将在通电后尝试连接到你的应用程序(服务侧)。

eStation 的默认参数为:

默认目标服务器地址	192.168.1.92:9081
默认用户名	test
默认密码	123456

注意:请尽快更改用户名和密码。

连接到一个站点后，你可以使用配置 Topic 对其进行修改(请参考 2.7 Publish Configuration Information).

但是，如果你无法连接到 eStation，或者你忘记了它的目标服务器地址，或者你忘记了用户名/密码，你可以按重置按钮几秒钟，在你看到 REST 和引导显示后，它将恢复到默认的目标服务器 IP 地址，用户名和密码。

注意:如果你重置了 eStation，但它仍然无法连接到你的应用程序，请检查你的防火墙和网线连接。

2.2 接收 eStation 信息

接口描述：

Topic Alias	**0x80**
Topic	{Prefix}/{ID}/infor
Prefix	Topic 前缀，默认为 “/estation” （不含引号）
ID	eStation 的 ID
QoS Level	ExatclyOnce
PayloadSegment	eStationInfor 对象的字节数，MessagePack 序列化

eStationInfor 属性是:

#	属性	类型	注意
0	ID	Char (4)	唯一编码，4 位 ID
1	Alias	String	昵称，当为 2 位数字是，数码管将显示它
2	IP	IP String	IP 本地地址
3	MAC	MAC String	MAC 地址
4	ApType	Int	固定 4
5	ApVersion	String	主程序固件版本
6	ModVersion	String	模组固件版本
7	DiskSize	Int	磁盘空间，MB
8	FreeSpace	Int	可用空间，MB
9	Server	String	目标服务器地址，可以为 IP 地址或域名
10	ConnParam	String 数组	连接参数
11	AutoIP	Bool	是否自动获取 IP
12	LocalIP	IP String	固定 IP 地址
13	Subnet	IP String	子网掩码

14	Gateway	IP String	网关
15	Heartbeat	Int	心跳，秒，最少 15

2.3 接收 eStation 消息

接口描述：

Topic Alias	**0x81**
Topic	/estation/{ID}/message
ID	eStation 的 ID
QoS Level	AtMostOnce
PayloadSegment	eStationMessage 对象的字节数，MessagePack 序列化

eStationMessage 属性是:

#	属性	类型	注意
0	Code	MessageCode	枚举类型

MessageCode 的枚举值如下:

1. OK 常规性应答
2. Idle 空闲
3. Result 通信结果返回
4. Heartbeat 心跳监听反馈
5. ModError 通信模块异常
6. AppError 主程序异常
7. Busy 设备繁忙
8. MaxLimit 当前数据队列已经达到上限（待发数据超过 10000 条或内存大小超过 512MB）
9. InvalidTaskESL 不正确的 ESL 任务数据
10. InvalidTaskDSL 不正确的 DSL 任务数据
11. InvalidConfig 不正确的配置数据
12. InvalidOTA 不正确的 OTA 数据

2.4 接收 eStation 结果

接口描述：

Topic Alias	**0x82**
Topic	/estation/{ID}/result
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	TaskResult 对象的字节数，MessagePack 序列化 TaskResult 对象包含 TagResult 对象的列表

TaskResult 属性是:

#	属性	类型	备注
0	Port	Int	MOD 串口号
1	WaitCount	Int	缓存中的总任务计数
2	SendCount	Int	发送任务计数（在射频模块中）

3	Message	MessageCode	见 2.3 MessageCode 定义部分
4	Tags	List<TagResult>	ESL 结果列表

TagResult 属性是:

#	属性	类型	备注
0	TagID	String	ESL ID
1	RfPower	Int	RF 功率，dBm, -256 表示无数据
2	Battery	Int	电池电量，V， 0 表示无数据
3	Version	String	电子 ESL 固件版本，保留备用
4	Status	String	状态，保留备用
5	Token	Int	Token，用于数据闭环，参见 2.5 令牌
6	Temperature	Int	温度，℃，0 表示无数据
7	Channel	Int	信道，保留备用
8	UtcTime	Byte 数组	UTC 时间，保留备用，仅适用于 DSL
9	TimePercent	Byte 数组	时间可信度，保留备用，仅适用于 DSL
10	Count	Byte	计数器，保留备用，仅适用于 DSL

注意:工作站内部有一个队列，当应用程序将任务数据发送到工作站时，如果射频模块空闲，它将立即发送到射频模块，或者如果射频模块正在工作，它将等待直到射频模块空闲。因此，应用程序和 eStation 之间的通信是异步的。

2.5 接收 eStation 心跳

接口描述：

Topic Alias	**0x83**
Topic	/estation/{ID}/heartbeat
ID	eStation 的 ID
QoS Level	AtMostOnce
PayloadSegment	ApHeartbeat 对象的字节数，MessagePack 序列化 ApHeartbeat 对象包含 TagHeartbeat 对象的列表

ApHeartbeat 属性是:

#	属性	类型	备注
0	Id	String	AP ID
1	ConfigVersion	Int	配置版本（默认为 0，保留使用）
2	ApVersion	String	基站版本
3	ModVersion	String	蓝牙模组版本
4	Message	MessageCode	消息，参见 2.3 MessageCode
5	Message	String	消息（扩展字段）
6	WaitCount	Int	当前排队等待的价签个数
7	SendCount	Int	当前正在通信中的价签个数
8	Tags	TagHeartbeat 列表	价签心跳信息

TagHeartbeat 属性是:

#	属性	类型	备注
0	TagId	String	价签 ESL/DSL ID
1	RfPower	Int	RF 功率，dBm, -256 表示无数据
2	Battery	Int	电池电量，V， 0 表示无数据

3	Version	String	电子 ESL 固件版本，保留备用
4	Status	String	状态，保留备用
5	Token	Int	Token，用于数据闭环，参见 2.5 令牌
6	Temperature	Int	温度，℃，0 表示无数据
7	Channel	Int	信道，保留备用
8	UtcTime	Byte 数组	UTC 时间，仅适用于 DSL
9	TimePercent	Byte	时间可信度，仅适用于 DSL
10	Count	Byte	计数器，仅适用于 DSL
11	Factory	Byte	工厂代码，为 OTA 使用
12	Color	Byte	屏幕颜色代码，为 OTA 使用
13	Size	Byte	屏幕尺寸代码，为 OTA 使用
14	Type	Byte	屏幕类型代码，为 OTA 使用

注意：心跳（heartbeat）与结果（result）数据结构定义相似，区别在于：心跳（heartbeat）是监听到标签心跳信息的内容，而结果（result）是你的应用侧下发的数据后返回的结果。前者是主动发生的，后者是被动发生的。

2.6 发布配置信息

接口描述：

Topic Alias	**0x01**
Topic	/estation/{ID}/configure
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	eStationConfig 对象的字节数，MessagePack 序列化

eStationConfig 属性是:

#	属性	类型	备注
0	Alias	String	昵称，当为 2 位数字是，数码管将显示它
1	Server	String	目标服务器地址
2	ConnParam	String	连接参数
3	Encrypt	Bool	True: 启用 TLS12 安全协议，False：不启用 默认不启用，参见 2.9.2 X.509 证书
4	AutoIP	Bool	是否自动获取 IP
5	LocalIP	IP String	固定 IP 地址
6	Subnet	IP String	子网掩码
7	Gateway	IP String	网关
8	Heartbeat	Int	心跳，秒，最少 15

2.7 发布 ESL 任务（Base64 版本）

注意：这是一个过时的 Topic，推荐使用 2.8 Bytes 版本的 Topic。

接口描述：

Topic Alias	**0x02**
Topic	/estation/{ID}/taskESL

ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	一个 ESLEntity 对象的列表数组，MessagePack 序列化

ESLEntity 属性是:

#	属性	类型	备注
0	TagID	String	ESL ID
1	Pattern	Int, Pattern	模式代码 参见 3.1 Pattern
2	PageIndex	Int, PageIndex	页面索引 参见 3.2 PageIndex
3	R	Bool	红色 LED 灯
4	G	Bool	绿色 LED 灯
5	B	Bool	蓝色 LED 灯
6	Times	2 字节	LED 灯闪烁次数，秒
7	Token	2 字节	Token，将在 2.3 Token 中返回
8	CurrentKey	8 字节	当前密钥，如果使用默认密钥则保持空
9	NewKey	8 字节	新密钥，如果不需要设置则保持空
10	Base64String	String	图像在 Bitmap 格式下的 Base64String 值

覆盖：对于一个 ESL ID，如果你的应用程序没有等待其前一个任务结果反馈，并继续向其发送任务数据，则 eStation 将放弃其前一个任务。

尺寸：eStation 会对推送的 ESL 任务数据中的图像数据（Base64String）进行尺寸（长度）检查，如果长度不正确（或实际的图像尺寸不符合价签屏幕的尺寸），那么无法成功更新图片。因为无法根据图像数据的 Byte 数组倒推出图像的实际尺寸进行补齐，如 300X400 的图像和 200X600 的图像 Bitmap 数据长度都是一致的。

有关价签类型与屏幕分辨率的关系，请参阅 3.1 ESL Gen 3.0 型号列表。

上限：使用 Base64 字符串方式推送数据，eStation 的任务缓存上限很低（以 4.2 三色价签测试为例，可能约在 800 个左右），如果需要提升上限，请使用 2.7 Bytes 版本推送数据。

密钥：有关当前密钥和新密钥的更多信息，请参阅 2.12 安全通信.

2.8 发布 ESL 任务（Bytes 版本）

接口描述：

Topic Alias	**0x03**
Topic	/estation/{ID}/taskESL2
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	一个 ESLEntity2 对象的列表数组，MessagePack 序列化

ESLEntity2 属性是:

#	属性	类型	备注
0	TagID	String	ESL ID
1	Pattern	Int, Pattern	模式代码 参见 3.1 Pattern
2	PageIndex	Int, PageIndex	页面索引 参见 3.2 PageIndex
3	R	Bool	红色 LED 灯
4	G	Bool	绿色 LED 灯
5	B	Bool	蓝色 LED 灯

6	Times	2 字节	LED 灯闪烁次数，秒
7	Token	2 字节	Token，将在 2.3 Token 中返回
8	CurrentKey	8 字节	当前密钥，如果使用默认密钥则保持空
9	NewKey	8 字节	新键，如果不需要设置则保持空
10	Bytes	Byte 数组	图像在 Bitmap 格式下的 Bytes 数组
11	Compress	Bool	是否压缩，默认为 True， GZip 压缩算法

尺寸：该 Topic 对图像尺寸的要求与 2.7 发布 ESL 任务（Base64 版本）一致.

上限：使用 Bytes 版本的 Topic，eStation 的上限暂定为 20000 个（这已经满足了绝大多数场景了），如果此上限的任务会被丢弃，并通过 2.3 接收基站消息中的 Topic 返回 MessageCode.MaxLimit。

数据：将需要推送的图像转换为 Bitmap 格式（RGBA 模式）后读取的 Bitmap 的 Bytes 数组，使用 GZip 压缩算法。

2.9 发布 DSL 任务

接口描述：

Topic Alias	**0x04**
Topic	/estation/{ID}/taskDSL
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	一个 DSLEntity 对象的列表数组，MessagePack 序列化

DSLEntity 属性是:

#	属性	类型	备注
0	TagID	String	ESL ID
3	R	Bool	红色 LED 灯
4	G	Bool	绿色 LED 灯
5	B	Bool	蓝色 LED 灯
6	Period	2 字节	LED 闪烁次数, -1~3600, 默认 3600，单位次
7	Interval	2 字节	亮灯间隔, 100~10000, 默认 1000，单位 s
8	Duration	8 字节	亮灯时长, 50~100, 默认 50，单位 ms
9	Token	8 字节	Token，将在 2.3 Token 中返回
10	HexData	Byte 数组	Bin 文件数据流

数据：Bin 文件指为 2.4TFT 价签（DSL）特定生成的图像/指令文件。

常亮：Period=-1 时，为常亮。

2.10 发布 OTA 任务（固件）

接口描述：

Topic Alias	**0x05**
Topic	/estation/{ID}/firmware
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	一个 OTAData 对象的列表数组，MessagePack 序列化

OTAData 属性是:

#	属性	类型	备注
0	DownloadUrl	String	固件下载地址
1	ConfirmUrl	String	目标服务器地址
2	Type	String	OTA 类型：0-eStation, 1/2-MOD, 3-ESL/DSL
3	Version	Bool	True: 启用 TLS12 安全协议，False：不启用 默认不启用，参见 2.9.2 X.509 证书
4	Name	Bool	是否自动获取 IP
5	MD5	IP String	固定 IP 地址

注意：

1. 该 Topic 如果 Type=**0**，则会视为 eStation 的固件，下载校验完成后会立刻重启 eStation 并进行 OTA 动作，如果 Type=**1** 或 **2**，则被视为 MOD 固件，会立刻进行 MOD OTA 动作。如果 Type=**3**，则视为预存 ESL/DSL 的固件，用于 2.11 节中的 OTA 价签使用。
2. 当开始进行 OTA 后，请勿在此期间与 eStation 进行通信，直至确切的 OTA 结束。
3. M D5 值的格式，在早先版本（<1.0.28）中，格式为：00-11-22-33-44-55-66-77，在后续的版本（>=1.0.28）中，格式调整为：00111223344556677。

2.11 发布 OTA 任务（价签）

接口描述：

Topic Alias	**0x06**
Topic	/estation/{ID}/ota
ID	eStation 的 ID
QoS Level	ExactlyOnce
PayloadSegment	一个 OTATask 对象的列表数组，MessagePack 序列化

OTATask 属性是:

#	属性	类型	备注
0	Step	Int	类型：0-PreOTA; 1-OTA; 2-Clean
1	Firmware	String	固件文件名称
2	TagIDList	String 数组	需要更新的价签 ID 列表

注意：

1. 该 Topic 依赖于 2.10 节中的固件，否则无法找到固件并进行更新。
2. 该 Topic 需要至少连续执行三次（Step 依次为 0/1/2），并根据 2.4 节中的结果检查当前 Step 中所有的价签是否都已经通信成功再决定是否执行下一个 Step。
3. 该 Topic 中的 TagIDList 需要为同一型号的价签 ID，Firmware 为该型号的 2 位编码。

2.12 安全通信

2.12.1. 价签密钥

通常来说，如果其他项目使用他们的 eStation (AP)来控制你的项目 ESL，就会发生意料之外和不可接受的结果。在这种情况下，建议你为你的 ESL 设置一个密钥。

当你从销售经理那里收到新的 ESL 时，ESL 的默认密钥为空（11 个字节，即 FFFFFFFFFFFFFFFF），你可以保留它或设置当前密钥和新密钥。之后，你的通信应该使用新密钥。

1. 密钥固定长度为 22(11 字节)。
2. 只接受十六进制字符(0123456789ABCDEF)。
3. 如果当前密钥为空，eStation 将使用默认密钥来替换。
4. 如果 New Key 为空，工作站将不会更改密钥。如果 New Key 不为空，基站将更改密钥。
5. 非常重要:请安全存储你的密钥，如果你忘记了密钥，你将失去对的 ESL 的控制。
6. 仅固件版本在 35 之后的 ESL 支持密钥通信功能。

2.12.2. X.509 证书

如果在 2.7 发布配置信息中启用了 TLS12 安全协议，服务端可以使用自定义的 X.509 证书对通信数据进行加密。有关服务端如何使用 X.509 证书和使用 TLS12 协议，视开发者选取的服务端开发语言框架决定。

如使用 C#和 MQTTnet 框架开发服务端，则需要一个 pfx 证书，示例代码如下:

```
public async Task Run()
{
    var options = new MqttServerOptionsBuilder()
        .WithDefaultEndpoint()
        .WithEncryptedEndpoint()
        .WithEncryptedEndpointPort(PORT) // TCP port in your server side
        .WithEncryptionCertificate(CERTIFICATE_PATH) // Your certificate
path
        .WithEncryptionSslProtocol(SslProtocols.Tls12)
        .Build();
    _server = _factory.CreateMqttServer(options);
    _server.ValidatingConnectionAsync +=
ServerOnValidatingConnectionAsync;
    _server.ClientConnectedAsync
        += ServeOnClientConnectedAsync;
    _server.ClientDisconnectedAsync
        += ServerOnClientDisconnectedAsync;
    _server.InterceptingPublishAsync +=
        ServerOnInterceptingPublishAsync;
    await _server.StartAsync();
}
```

3 参考

3.1 ESL Gen 3.0 型号列表

种类	尺寸（英尺）	屏幕	颜色	像素 (H*W)	是否冷冻
ET0154-33	1.54	Eink	B/W/R	200*200	-
ET0213-36	2.13	Eink	B/W/R	250*122	-
ET0213-39	2.13	Eink	B/W	250*122	Y

ET0266-3A	2.66	Eink	B/W/R	296*152	-
ET0266-5B	2.66	Eink	B/W	296*152	Y
ET0290-3D	2.9	Eink	B/W/R	296*128	-
ET0290-3F	2.9	Eink	B/W	296*128	-
ET0290-54	2.9	Eink	B/W	296*128	Y
ET0420-40	4.2	Eink	B/W/R	400*300	-
ET0420-42	4.2	Eink	B/W	400*300	-
ET0750-44	7.5	Eink	B/W/R	800*480	-
ET0750-46	7.5	Eink	B/W	800*480	-
ET0430-4C	4.3	Eink	B/W/R	522*152	-
ET0580-4F	5.8	Eink	B/W/R	648*480	-
ET0350-55	3.5	Eink	B/W/R	384*184	-
ET1250-58	12.5	Eink	B/W/R	1304*984	-
ET0420-5D	4.2	TFT	B/W/R	400*300	-
ET1020-64	10.2	Eink	B/W/R	960*640	-
ET1020-67	10.20	Eink	B/W	960*640	-
ET1330-68	13.3	Eink	B/W/R	960*680	-
ET0579-6F	5.8	Eink	B/W/R	792*272	-
ET0154-80	1.54	Eink	B/W/R/Y	200*200	-
ET0213-81	2.13	Eink	B/W/R/Y	250*122	-
ET0266-82	2.66	Eink	B/W/R/Y	296*152	-
ET0266-83	2.66	Eink	B/W/R/Y	360*184	-
ET0290-84	2.9	Eink	B/W/R/Y	296*128	-
ET0290-85	2.9	Eink	B/W/R/Y	384*168	-
ET0350-86	3.5	Eink	B/W/R/Y	384*184	-
ET0420-87	4.2	Eink	B/W/R/Y	400*300	-
ET0580-88	5.8	Eink	B/W/R/Y	648*480	-
ET0750-89	7.5	Eink	B/W/R/Y	800*480	-
ET0750-8A	7.5	Eink	B/W/R/Y	880*528	-
ET1020-8B	10.2	Eink	B/W/R/Y	960*640	-
ET0097H-69	0.97	Eink	B/W/R	184*88	-
ET0130H-73	1.3	Eink	B/W/R	200*144	-
ET0154H-71	1.54	Eink	B/W/R	200*200	-
ET0097H-8C	0.97	Eink	B/W/R/Y	184*88	-
ET0130H-8D	1.3	Eink	B/W/R/Y	200*144	-

注意: 在颜色栏中，B 表示黑色，W 表示白色，R 表示红色，Y 表示黄色。

3.2 Pattern

Pattern	描述
0- UpdateDisplay	更新并且显示
1- Update	更新但不显示
2- Display	显示
3- Query	查询 ESL 信息
4- Check	检查 ESL 是否存在
5- LED	闪烁 LED 灯
6- Key	更新密钥（参见 2.12.1 价签密钥）

3.3 PageIndex

页面 ID	描述
0- P0	第 1 页
1- P1	第 2 页
2- P2	第 3 页
3- P3	第 4 页
4- P4	第 5 页，图片压缩后大小不能超过 32KB
5- P5	第 6 页，图片压缩后大小不能超过 32KB
6- P6	第 7 页，图片压缩后大小不能超过 32KB
7- P7	第 8 页，图片压缩后大小不能超过 32KB

注意: 目前所有 ESL 只有 8 页的数据缓存，受存储空间限制，第 5-8 页下发的图片压缩后不能超过 32KB（使用 LZSS 压缩算法，基站内部实现，服务端依然传递原始图片）。

3.4 RGB 颜色定义

价签 LED 灯使用 RGB 三色灯，可视觉上达到 7 色效果。

颜色	R	G	B
灭	×	×	×
蓝色	×	×	√
绿色	×	√	×
青色	×	√	√
红色	√	×	×
紫色	√	×	√
黄色	√	√	×
白色	√	√	√

3.5 数据定义（C#版本）

eStationInfor：用于定义 eStation 的配置、状态、参数等信息。其中 ID、MAC、

AppVersion、DummyVersion、TotalCount、SendCount 是只读的。它的定义如下：

```
/// <summary>
/// AP information
/// </summary>
[MessagePackObject]
public class eStationInfor
{
    /// <summary>
    /// ID: Code4
    /// </summary>
    [Key(0)]
    public string ID { get; set; } = "0000";
    /// <summary>
    /// Alias
    /// </summary>
    [Key(1)]
    public string Alias { get; set; } = "";
    /// <summary>
    /// Local IP
    /// </summary>
    [Key(2)]
    public string IP { get; set; } = "127.0.0.1";
    /// <summary>
    /// MAC, refer ID
    /// </summary>
    [Key(3)]
    public string MAC { get; set; } = "90A9F7300000";
    /// <summary>
    /// AP type
    /// </summary>
    [Key(4)]
    public int ApType { get; set; } = 4; // AP05 = 3;
    /// <summary>
    /// Firmware version
    /// </summary>
    [Key(5)]
    public string ApVersion { get; set; } = "1.0.1";
    /// <summary>
    /// MOD ModVersion
    /// </summary>
    [Key(6)]
    public string ModVersion { get; set; } = "";
    /// <summary>
    /// Disk size (MB)
    /// </summary>
    [Key(7)]
    public int DiskSize { get; set; } = 0;
    /// <summary>
    /// Disk free space (MB)
    /// </summary>
    [Key(8)]
    public int FreeSpace { get; set; } = 0;
    /// <summary>
    /// Server address
    /// </summary>
    [Key(9)]
    public string Server { get; set; } = "192.168.4.92";
    /// <summary>
    /// Connection parameters
    /// </summary>
    [Key(10)]
    public string[] ConnParam { get; set; } = Array.Empty<string>();
```

```
    /// <summary>
    /// Auto network
    /// </summary>
    [Key(11)]
    public bool AutoIP { get; set; } = true;
    /// <summary>
    /// Local IP, empty if auto network
    /// </summary>
    [Key(12)]
    public string LocalIP { get; set; } = string.Empty;
    /// <summary>
    /// Subnet mask, empty if auto network
    /// </summary>
    [Key(13)]
    public string Subnet { get; set; } = string.Empty;
    /// <summary>
    /// Gateway, empty if auto network
    /// </summary>
    [Key(14)]
    public string Gateway { get; set; } = string.Empty;
    /// <summary>
    /// Heartbeat speed
    /// </summary>
    [Key(15)]
    public int Heartbeat { get; set; } = 15;
}
```

eStationConfig：用于定义基站的配置信息。它的定义如下：

```
[MessagePackObject]
internal class eStationConfig
{
    /// <summary>
    /// Alias
    /// </summary>
    [Key(0)]
    public string Alias { get; set; } = "";
    /// <summary>
    /// Server address
    /// </summary>
    [Key(1)]
    public string Server { get; set; } = "192.168.4.92";
    /// <summary>
    /// Connection parameters
    /// </summary>
    [Key(2)]
    public string[] ConnParam { get; set; } = Array.Empty<string>();
    /// <summary>
    /// Auto network
    /// </summary>
    [Key(3)]
    public bool AutoIP { get; set; } = true;
    /// <summary>
    /// Local IP, empty if auto network
    /// </summary>
    [Key(4)]
    public string LocalIP { get; set; } = string.Empty;
    /// <summary>
    /// Subnet mask, empty if auto network
    /// </summary>
    [Key(5)]
    public string Subnet { get; set; } = string.Empty;
    /// <summary>
    /// Gateway, empty if auto network
```

```
    /// </summary>
    [Key(6)]
    public string Gateway { get; set; } = string.Empty;
    /// <summary>
    /// Heartbeat speed
    /// </summary>
    [Key(7)]
    public int Heartbeat { get; set; } = 15;
}
```

eStationMessage：用于定义基站的配置信息。它的定义如下：

```
[MessagePackObject]
public class eStationMessage
{
    /// <summary>
    /// Message code
    /// </summary>
    [Key(0)]
    public MessageCode Code { get; set; } = MessageCode.OK;
}
```

ESLEntity：用于定义 ESL 的任务数据信息。它的定义如下：

```
/// <summary>
/// ESL entity
/// </summary>
[MessagePackObject]
public class ESLEntity
{
    [Key(0)]
    public string TagID { get; set; }
    [Key(1)]
    public Pattern Pattern { get; set; }
    [Key(2)]
    public PageIndex PageIndex { get; set; }
    [Key(3)]
    public bool R { get; set; }
    [Key(4)]
    public bool G { get; set; }
    [Key(5)]
    public bool B { get; set; }
    [Key(6)]
    public int Times { get; set; }
    [Key(7)]
    public int Token { get; set; }
    [Key(8)]
    public string OldKey { get; set; }
    [Key(9)]
    public string NewKey { get; set; }
    [Key(10)]
    public string Base64String { get; set; } = "";
}
```

TaskResult：用于定义一组 ESL/PTL 通信结果的数据信息。它的定义如下：

```
/// <summary>
/// Task result entity
/// </summary>
[MessagePackObject]
public class TaskResult
{
    /// <summary>
    /// COM Port
    /// </summary>
    [Key(0)]
    public int Port { get; set; } = 0;
```

```
    /// <summary>
    /// Total count in cache
    /// </summary>
    [Key(1)]
    public int TotalCount { get; set; }

    /// <summary>
    /// Current sending count
    /// </summary>
    [Key(2)]
    public int SendCount { get; set; }

    /// <summary>
    /// Message
    /// </summary>
    [Key(3)]
    public MessageCode Message { get; set; } = 0;

    /// <summary>
    /// Tag results list
    /// </summary>
    [Key(4)]
    public List<TagResult> Tags { get; set; } = new List<TagResult>();
}
```

TagResult：用于定义单个 ESL/PTL 通信结果的数据信息。它的定义如下：

```
[MessagePackObject]
/// <summary>
/// Tag result
/// </summary>
public class TagResult
{
    /// <summary>
    /// Tag ID
    /// </summary>
    [Key(0)]
    public string TagID { get; set; } = string.Empty;
    /// <summary>
    /// RF power
    /// </summary>
    [Key(1)]
    public int RfPower { get; set; } = -256;
    /// <summary>
    /// Battery
    /// </summary>
    [Key(2)]
    public byte Battery { get; set; } = 0;
    /// <summary>
    /// Screen
    /// </summary>
    [Key(3)]
    public byte Version { get; set; } = 0;
    /// <summary>
    /// Version
    /// </summary>
    [Key(4)]
    public byte Status { get; set; } = 0;
    /// <summary>
    /// Token
    /// </summary>
    [Key(5)]
    public int Token { get; set; } = 0;
```

```
    /// <summary>
    /// Temperature
    /// </summary>
    [Key(6)]
    public int Temperature { get; set; } = 0;
    /// <summary>
    /// Channel
    /// </summary>
    [Key(7)]
    public int Channel { get; set; } = 0;
    /// <summary>
    /// [DSL]UTC time,
    /// </summary>
    [Key(8)]
    public byte[] UtcTime { get; set; } = Array.Empty<byte>();
    /// <summary>
    /// [DSL]UTC time,
    /// </summary>
    [Key(9)]
    public byte[] TimePercent { get; set; } = Array.Empty<byte>();
    /// <summary>
    /// [DSL]Count
    /// </summary>
    [Key(10)]
    public byte Count { get; set; } = 0;
}
```

■