Version	1.0
Date	2022-06-05

D19 ESL Gen 3.0 Middleware Developer Manual

Instance ID _____

Project Code _____

This document is for SERTAG ESL Gen 3.0 system integration only.

Version History

Ver.	Date	Description	Author	Review
1.0	2022-06-05	Document initialize	Huang Haipeng	Huang Haipeng

Contents

1 Introduce

1.1 Background

This middleware, ESL Gen 3.0 Middleware (Cronus) which designed for system integrator to develop quickly for their business project, using .NET 6.0.

The middleware is an opensource project with MIT license.

You can obtain relevant software and documents from our engineers.

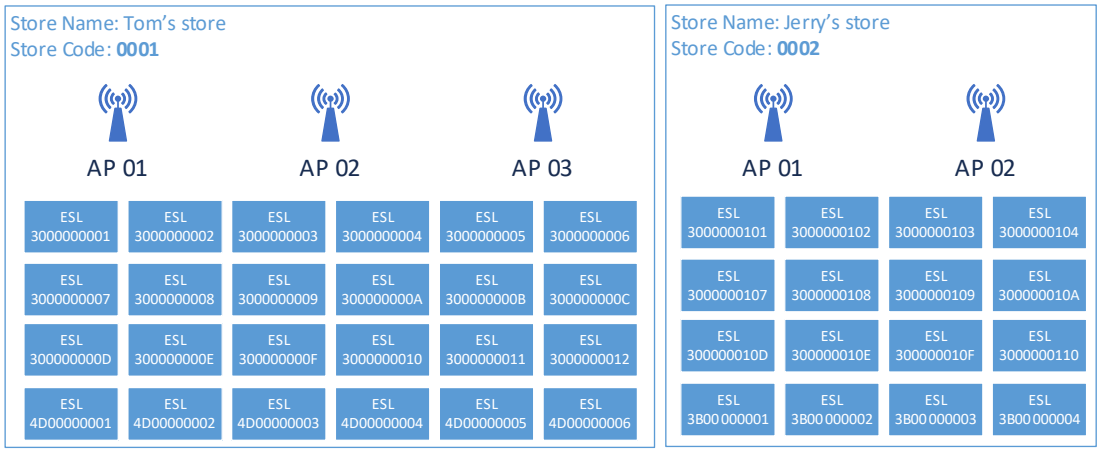
This project has three folders, include Cronus, Cronus.API and Cronus.Demo:

1. **Cronus**: the middleware project, you can directly add into your project solution in Visual Studio.
2. **Cronus.API**: a Restful API package project, which encapsulate the Cronus project. If your project is using other languages like Java, Python and NodeJS etc., you can use HTTP POST to push data to the Cronus.API. Cross platform support.
3. **Cronus.Demo**: a WPF desktop demo application. Show how to send data to the middleware.

Before you start to using the middleware, you must understand bellowing keywords:

1. **Store Code**: The ID of one store in the system.
2. **AP**: The radio frequency (RF) access point which connect to the labels;
3. **Tag**: The electronical shelf label (ESL).

For example, your project has two stores, looks like:



The data structure should be:

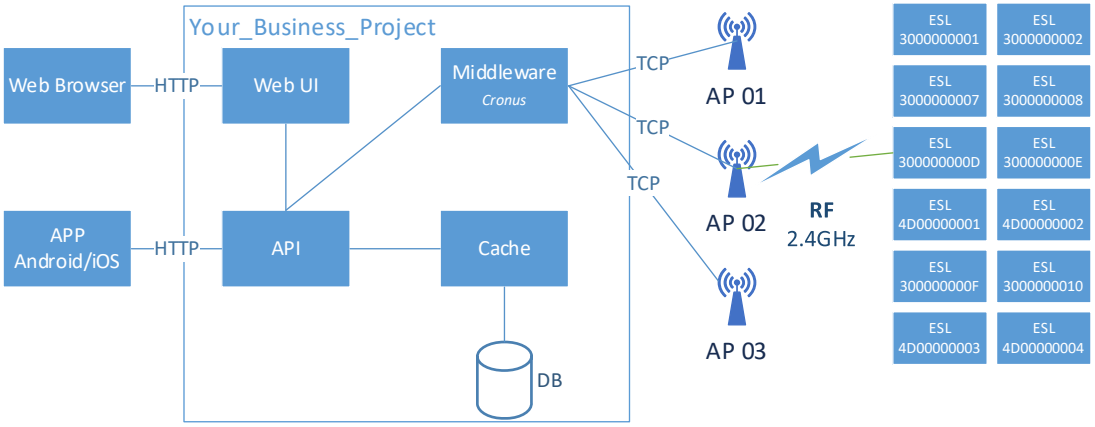
Store Code	AP ID	Tag ID
0001	01	30000001, 30000002, 30000007, 30000008, 3000000D …
	02	30000003, 30000004, 30000009, 3000000A, 3000000F …
	03	30000005, 30000006, 3000000B, 3000000C, 30000011 …
0002	01	30000101, 30000102, 30000107, 30000108, 3000010D …
	02	30000103, 30000104, 30000109, 3000010A, 3000010F …

Note: Both tags and APs are belonging to a store, but tags are not static belonging to any AP. The tag just tries to remember the last default AP ID which has successfully send data to it.

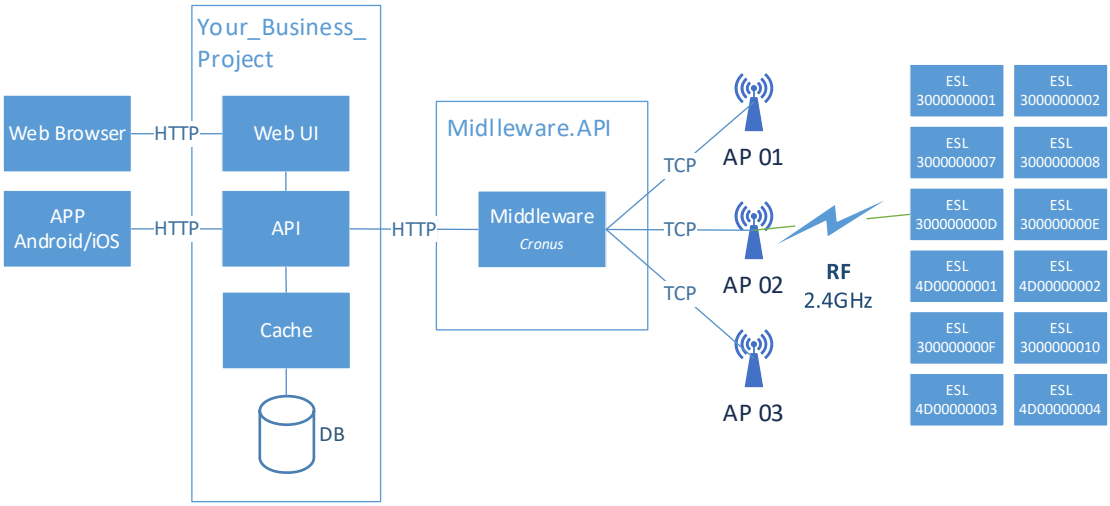
1.2 System Architecture

Basically, your project sends image data to the middleware, and the middleware find an AP and send the image data to a label, also the middleware returns the result to your project. In another words, the middleware is between your application and the labels.

For .NET source code project, you can directly add it into your project. After you build your project source and publish into a server, the middleware is DLL file in your software. It looks like:



For other language project, you can using HTTP POST to Cronus.API. The API middleware can be deployed in Linux, macOS, and Windows.

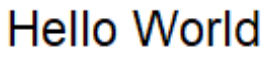


Remember, if your project and the middleware are not in a same private network, you should add HTTPS support, identity authentication and session token.

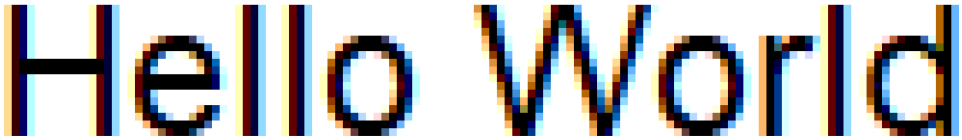
1.3 About the Image

As some public issues, in .NET 6.0 the image library System.Drawing.Common only supported on Windows, so the middleware using SkiarSharp instead of GdiPlus.

The ESL screen is a Black-White or Black-White-Red color screen, no gray color. For example, your image looks like:



Zoom in 800% looks like:



And after remove gray scale parts, it looks like:

The larger the font size, the clearer the jagged shape.

So, its not a good idea to using labels to show images, if you want to do that, you may need dithering the image to looks like smoothly.

An algorithm for dithering images using C# is: Even more algorithms for dithering images using C# - Articles and information on C# and .NET development topics • Cyotek

For example, an image looks like:

With dithering and without dithering the image effect looks like:

2 Start with Cronus

The middleware namespace is Cronus, and the main class is SendServer. It's a singleton class, so you can use **SendServer.Instance** to send/receive data to/from the middleware.

2.1 Register Events Handler

Because of the labels communication asynchronously, the middleware using events handler to feedback results.

There are two events handlers: task result event and AP status event. If you want to know the result of these tasks which you have pushed to the middleware, or if you care about the AP status, you need register these two events handlers.

2.1.1. Task Result Handler

The task result handler is:

```
event EventHandler<TaskResultEventArgs> TaskEventHandler
```

The TaskResultEventArgs property is:

Property	Type	Remark
TaskResults	List	List of TaskResult

And the TaskResult properties are:

Property	Type	Remark
TaskID	Guid	The task ID, refer to your project task record's PK, refer to the 2.3.2 Task ID
TagID	String	The AP ID
RouteRecord	List	List of history AP ID, route record
Status	Status	Refer to 4.4 Task Status
SendCount	Int	Send times
FirstSendTime	DateTime?	First send time, null means no data
LastSendTime	DateTime?	Last send time, null means no data
LastRecvTime	DateTime?	Last receive time, null means no data
Token	Int	Token, internal data token
Battery	Float?	Battery, Voltage, null/0 mean no data
Temperature	Int?	Temperature, ℃, null means no data
RfPower	Int?	RF power, dBm, null/-256 means no data
Version	String	Keep in future
Screen	String	Keep in future

The task result event store in a concurrent queue, so it is not a real time event, and your project should use less time cost (like DB update).

2.1.2. AP Status Handler

The AP status event handler is:

```
event EventHandler<APStatusEventArgs> APEventHandler
```

The APStatusEventArgs properties are:

Property	Type	Remark
StoreCode	String	The store code
APID	String	The AP ID
Status	APStatus	Refer to 4.2 AP Status

The AP status event store in a concurrent queue, so it is not a real time event, and your project should use less time cost (like DB update).

2.2 Run Middleware

You can inject your application logger into the middleware, but the logger must implements from `Microsoft.Extensions.Logging.ILogger`.

The middle ware using a configure class `CronusConfig` to set the middleware parameters like AP port, single store mode etc.

Method	`Start(CronusConfig config, ILogger log = null)`	
Parameters	**Type**	**Remark**
`config`	`CronusConfig`	
`log`	`ILogger`	Will create if null (using Serilog)
Return	`Result`(See 4.1 Result)	

The middleware configure class looks like:

Property	**Type**	**Remark**
`APPort`	Int	Default is 1234
`DefaultStoreCode`	String	Default is 0001
`OneStoreModel`	Bool	Default is true

Note: If your project instance only has one store, you can using the CronusConfig.DefaultStoreCode and CronusConfig.OneStoreModel, and below 2.3, 2.4 and 2.5 you can simply without store code parameter.

2.3 Push Image Data

2.3.1. Push an image to a label

Push an image to a label:

Method	`Result Push(string storeCode, string tagID, SKImage image)`	
Parameters	**Type**	**Remark**
`storeCode`	`String`	Store code
`tagID`	`String`	Tag ID
`Image`	`SKImage`	Skiasharp Image object
Return	`Result`(See 4.1 Result)	

Note: The label will remember its last successfully AP ID.

2.3.2. Push a task data list

Push a tasks list to the middleware:

Method	`Result Push(string storeCode, List<TaskData> tasks)`	
Parameters	**Type**	**Remark**
`storeCode`	`String`	Store code

Tasks	List	List of tag tasks
Return	Result(See 4.1 Result)	

The task data object properties are:

Parameters	Type	Remark
TaskID	Guid	The task ID, refer to your project task record's PK
TagID	String	Tag ID
APID	String	Keep empty then will use the default ID
Bitmap	SKBitmap	Image data
R	Bool	Flashing red color LED light
G	Bool	Flashing green color LED light
B	Bool	Flashing blue color LED light
Times	Int	LED light flashing time, 1 second 1 time*

* The LED lights flashing speed depends on the hardware, cannot changed.

2.4 Push LED Flashing Data

2.4.1. Flashing label's LED light

Push an image to a label:

Method	Result LED(bool r, bool g, bool b, int times, List<string> idList = null)	
Parameters	**Type**	**Remark**
storeCode	String	Store code
R	Bool	Flashing red color LED light
G	Bool	Flashing green color LED light
B	Bool	Flashing blue color LED light
Times	Int	LED light flashing time, 1 second 1 time*
idList	List	List of tag ID, will try to send all tags if null
Return	Result(See 4.1 Result)	

2.5 Broadcast

Note:

1. Broadcast will affect all labels in the radio communication coverage of the AP (about 8-15M).

2. Broadcast will not add into task queue, it is a directly working order. So, it requires AP is in idle status.

2.5.1. Switch Page Cache

The page index will be added in next version.

Switch page cache:

Method	SwitchPage(string storeCode, int page)	
Parameters	**Type**	**Remark**
storeCode	String	Store code
Page	Int	Page index, from 0 to 7.
Return	Result(See 4.1 Result)	

2.5.2. Display Barcode

Display the tag's ID barcode on the screen.

Method	Result DisplayBarcode(string storeCode)	
Parameters	**Type**	**Remark**
storeCode	String	Store code
Return	Result(See 4.1 Result)	

Note: this method always using for the project deploy section, for some reason the workers not easy scan the barcode on the label's top side (for example, the label is on the topmost layer of the shelf).

2.6 Query Status

2.6.1. Query Tags

Method	List<Tag> GetTags(string storeCode)	
Parameters	**Type**	**Remark**
storeCode	String	Store code
Return	Tag list	

The tag class properties are:

Parameters	Type	Remark
TagID	String	Tag ID
DefaultAPID	String	Default AP ID
TaskID	Guid	Last Guid, Guid.Empty means no task
StoreCode	String	Store code
TagType	TagType	Refer to 4.5 ESL Gen 3.0 Type List
TagStatus	TagStatus	Refer to 4.3 Tag Status
Battery	Float?	Battery, Voltage, null/0 mean no data
Temperature	Int?	Temperature, ℃, null means no data

RfPower	Int?	RF power, dBm, null/-256 means no data
LastSendTime	DateTime?	Last send time, null means no data
LastRecvTime	DateTime?	Last receive time, null means no data
TotalSend	Int	Total send count
ErrorCount	Int	Error count from last boot time
ErrorCountTemp	Int	Error count from last successfully time

2.6.2. Query Aps

Method	`List<Tag> GetAPList(string storeCode)`	
Parameters	**Type**	**Remark**
storeCode	String	Store code
Return	AP list	

The AP class properties are:

Parameters	**Type**	**Remark**
APID	String	AP ID
StoreCode	String	Store code
APStatus	APStatus	Refer to 4.2 AP Status
CurrentTasks	Int	Count of current tasks
LastOnlineTime	DateTime?	Last online time, null means no data
LastOfflineTime	DateTime?	Last offline time, null means no data
LastHeartbeatTime	DateTime?	Last heartbeat time, null means no data
LastSendTime	DateTime?	Last send time, null means no data
LastRecieveTime	DateTime?	Last receive time, null means no data

3 Start with Cronus.API

(TODO)

4 Reference

4.1 Result

OK	OK
InvalidTagID	Invalid tag ID, should using the barcode print on the label side
InvalidStoreCode	Invalid store code, should be the same as the AP's configure
InvalidApId	Invalid AP ID, should be the same as the AP's configure
InlivadTaskData	Invalid task data
InvalidImage	Invalid image
NullData	Null data error

Error	General error message
NoTaskCreate	No task has been created
NotAllTaskCreate	Not all tasks have been created
NoApOnline	No AP online currently
NoApIdle	No AP idle currently
APBusying	One or more AP is busying now, cannot execute order

4.2 AP Status

- Init = 0,
- Online = 1,
- Working = 2,
- Offline = 3,
- Error = 4,
- Heartbeat = 5

4.3 Tag Status

- Init = 0,
- Idle = 1,
- Working = 2,
- LowPower = 3,
- Error = 4,
- Lost = 5,

4.4 Task Status

- Init = 0,
- Sending = 1,
- Success = 2,
- Failed = 3,
- Lost = 4,
- Drop = 5,

4.5 ESL Gen 3.0 Type List

Type	Size (Inch)	Screen	Color	Pixels(H*W)	Temperature	Appearance
ET0154-33	1.54	Eink	Black/White/Red	200*200	Normal	Normal
ET0213-36	2.13	Eink	Black/White/Red	250*122	Normal	Normal
ET0213-39	2.13	Eink	Black/White	250*122	Freezing	Normal

ET0266-3A	2.66	Eink	Black/White/Red	296*152	Normal	Normal
ET0266-5B	2.66	Eink	Black/White	296*152	Freezing	Normal
ET0290-3D	2.90	Eink	Black/White/Red	296*128	Normal	Normal
ET0290-3F	2.90	Eink	Black/White	296*128	Normal	Normal
ET0290-54	2.90	Eink	Black/White	296*128	Freezing	Normal
ET0420-40	4.20	Eink	Black/White/Red	400*300	Normal	Normal
ET0420-43	4.20	Eink	Black/White/Red	400*300	Normal	Waterproof
ET0750-44	7.50	Eink	Black/White/Red	800*480	Normal	Normal
ET0430-4C	4.30	Eink	Black/White/Red	522*122	Normal	Normal
ET0580-4F	5.80	Eink	Black/White/Red	648*480	Normal	Normal
ET0350-55	3.50	Eink	Black/White/Red	384*184	Normal	Normal
ET1250-58	12.50	Eink	Black/White/Red	1304*984	Normal	Normal
ET1010-5C	10.10	TFT	24 Colors	1280*800	Normal	POP

■