GENERACIÓN DE DRIVERS PARA COMPACT TOUCH Y TMS

MANUAL DE SINTAXIS

Introducción

Un driver de la COMPACT TOUCH o del TMS se basa en un fichero de texto plano, que no precisa de ningún tipo de compilado. La sintaxis del nombre de archivo es fija, expresándose de la siguiente forma:

<fabricante>.<dispositivo>.<versión>

De este modo, la versión 5 del driver del RTX 600V de Eliwell daría lugar al nombre de fichero:

Eliwell.RTX600V.5

El formato de un driver es un CSV que emplea el punto y coma (;) como separador de columnas. Asimismo, el driver cuenta con diccionarios que traducen los nombres de las variables a distintos idiomas. Dichos diccionarios son ficheros JSON. Hay dos diccionarios a emplear: uno general para todos los drivers con *<código de idioma>.json* como nombre de fichero y otro específico del dispositivo con el nombre de fichero *<dispositivo>_<código de idioma>.json*. De este modo, el diccionario genérico en castellano se llamaría *es-ES.json* y en inglés (británico y estadounidense, respectivamente) *en-GB.json* o *en-US.json*; mientras que los diccionarios específicos del RTX 600V en los mismos idiomas serían *RTX600V_es-ES.json*, *RTX600V_en-GB.json* y *RTX600V_en-US.json*.

Partes de un driver

Un driver se compone de tres partes: comentarios, la instrucción ACK y los recursos y parámetros.

Excepto los comentarios, el resto de partes son instrucciones que siguen una sintaxis parecida:

Variable;Read;513;1;Int16_ML;FF_FF;1;°C;AI27(1;1;continuous;-1;;;Off;-1;;;;;<-670; >3200;@33018>0

Debido a que estas instrucciones se presentan en formato CSV, tomando los puntos y comas como delimitadores de columnas e ignorando aquellas que no contienen nada (dado que su espacio está reservado), es fácil expresar una instrucción en formato tabla.

Variable	Read	513	1	Int16_ML	FF_FF	1	°C	AI27(1	1	continuous	-1	Off	-1	<-670	>3200	@33018>0

Comentarios

Los comentarios vienen precedidos por el símbolo almohadilla (#) y son líneas explicativas que no tienen repercusión en el funcionamiento del driver. Su función es aclarar partes del documento o invalidar instrucciones que ya no deben ser usadas.

#Esto es un comentario
Variable;Read;513;1;Int16_ML;FF_FF;1;°C;AI27(1;1;continuous;-1;;;Off;-1;;;;;<-670;>3200;@33018>0 #Un comentario puesto tras una instrucción

ACK

El ACK es la instrucción que sirve para el reconocimiento del dispositivo. El propósito de esta instrucción, que **siempre debe ser la primera** presente en el fichero del driver, es describir el método por el que la Compact Touch o el TMS distingan el instrumento de acuerdo a una serie de características identificativas. Este reconocimiento se puede realizar de dos maneras: o bien mediante la función 43 de ModBus, o mediante la función 3, leyendo un registro concreto cuyo valor sea fijo y permita identificar adecuadamente al dispositivo.

Con el comando 43:

Recurso	Comando	Dirección	Word	Conversión	Máscara			ACK	Valor	MAXW	Máx. Peticiones
Variable	43_04_02	16	11	4	FF_FF			ACK43	01FD	MAXW	100

El campo “Comando” describe los parámetros de la función 43 que se emplea. Su sintaxis es:

43_<Código de identificación>_<objeto>

El código de identificación se encuentra especificado en la documentación de cada instrumento. Cada código establece una serie de información identificativa que puede obtenerse y que a su vez contiene distintos objetos. El código 03 contiene tres objetos, siendo el 00 el código de vendedor, el 01 el POLI y el 02 la máscara y el firmware.

A bajo nivel, el comando que sería enviado por ModBus, según cómo se especifica aquí, sería el siguiente (expresado en hexadecimal):

1	2	3	4	5
01	2B	0E	04	02

Donde:

- 01 (byte 1) – Dirección ModBus
- 2B (byte 2) – Comando ModBus (43 en decimal)
- 0E (byte 3) – Tipo MEI (13 = CanOpen, 14 = Identificación)
- 04 (byte 4) – Código de identificación del dispositivo (01, 02, 03 o 04)
- 02 (byte 5) – Objeto

La respuesta ModBus sería la siguiente:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
01	2B	0E	04	81	00	00	01	02	09	30	31	46	44	5F	30	30	31	45	90	98

Donde:

- 01 (byte 1) – Dirección ModBus
- 2B (byte 2) – Comando ModBus
- 0E (byte 3) – Tipo MEI
- 04 (byte 4) – Código de identificación del dispositivo
- 81 (byte 5) – Nivel de conformidad
- 00 (byte 6) – Si quedan más paquetes por llegar
- 00 (byte 7) – ID del siguiente objeto
- 01 (byte 8) – Número de objetos
- 02 (byte 9) – ID de objeto
- 09 (byte 10) – Longitud del objeto (número de bytes que lo componen)
- 30 31 46 44 5F 30 30 31 45 (bytes 11 a 19) – Objeto solicitado, que es una cadena de códigos ASCII que denotan la máscara y el firmware.

- 90 98 (bytes 20 y 21) – CRC

Analizando el objeto solicitado y convirtiendo a ascii (se puede hacer en la siguiente web: https://www.rapidtables.com/convert/number/hex-to-ascii.html):

Hexadecimal	30	31	46	44	5F	30	30	31	45
ASCII	0	1	F	D	_	0	0	1	E

El resultado es una cadena: “01FD_001E”, que nos da la máscara y firmware, respectivamente, del dispositivo y que en este caso corresponden a los del RTX 600V.

Llegados a este punto se puede observar que la primera parte, “01FD”, es el valor a leer por el comando ACK del driver, como se puede ver en el campo “Valor”, que comienza en el byte 11, como se puede ver en el campo “Word” y que se compone de 4 bytes, como también queda expresado en el campo “Conversión”.

Y con el comando 3:

Recurso	**Comando**	**Dirección**	**Nº Words**	**Conversión**	**Máscara**			**ACK**	**Valor**	**MAXW**	**Máx. Peticiones**
Variable	Read	10423	1	Int16_ML	FF_FF	0	num	ACK	9002	MAXW	100

En este caso, el comando es más sencillo: se lee el registro especificado en el campo “Dirección” (en este caso, el registro 10423), que se compone de 1 word, y se espera leer el valor 9002 para identificar al dispositivo como tal.

Mención especial merece el campo “Máx. Peticiones”: siempre que sea posible, los recursos se leerán en bloque mientras las direcciones de memoria sean consecutivas; de este modo se conseguiría reducir el número de peticiones hechas. El campo “Máx. Peticiones” define el número máximo de registros que pueden ser solicitados en bloque. No se recomienda exceder los 100.

Recursos y parámetros

El cuerpo básico de un driver se compone de las peticiones de recursos y parámetros y, en menor medida, de comandos. Pero, ¿cuál es la diferencia entre ellos?

- Recursos: registros del dispositivo que son representativos del estado actual del mismo. Son de sólo lectura y no se puede escribir sobre ellos. Entradas/salidas digitales, entradas/salidas analógicas, estados y alarmas.
- Parámetros: registros del dispositivo que son tanto de lectura y escritura y que permiten la configuración del dispositivo. Por ejemplo, la consigna de temperatura o el número de compresores.
- Comandos: registros que, al escribir sobre ellos, activan de manera manual una función del propio dispositivo. Por ejemplo, encendido/apagado de luces, activación manual del desescarche o el stand-by del dispositivo.

La estructura de los recursos, parámetros y comandos es exactamente la misma y se describe en el siguiente ejemplo:

Variable;Read;513;1;Int16_ML;FF_FF;1;°C;AI27(1;1;continuous;-1;;;Off;-1;;;;;<-670;>3200;@33018>0

Como se ha comentado anteriormente, hay determinadas columnas que están vacías y están reservadas. Obviándolas, la estructura sería la siguiente:

Tipo	Cmd	Dir.	Núm. Words	Conversión	Máscara	PD	UM	Varname	Formato	Modo lectura	Periodo lectura	Modo escritura	Periodo escritura	Mín.	Máx.	Condición existencia
Variable	Read	513	1	Int16_ML	FF_FF	1	°C	AI27(1	1	continuous	-1	Off	-1	<-670	>3200	@33018>0

En la página siguiente se describe cada campo.

Tipo	Tipo del objeto. Puede tomar los siguientes valores: • *Variable*: recurso. Se registra periódicamente. Recomendado para entradas/salidas analógicas y estados. • *Status_Dig*: recurso. Se registra periódicamente y en el momento en el que cambia de estado. Recomendado para entradas/salidas digitales. • *Alarm*: recurso. Denota una alarma que se activará si en el registro leído hay un número distinto de 0, y permanecerá inactiva en caso contrario. • *Action*: comando. • *Parameter*: parámetro.	**Máscara**	Máscara de bits que se aplicará sobre los datos, expresada como números hexadecimales separados por guiones bajos (_). Por ejemplo: FF_FF, 00_01, 03_00, FF_FF_FF_FF. Según los bits que deseen obtenerse del registro. Si se leyera el valor hexadecimal 04A5 (1189 en decimal) y se le aplicara la máscara 00_0F, el resultado sería 0005 (5 en decimal). Si se le añade el prefijo "B_", el valor obtenido tras aplicar la máscara se transforma en booleano, de modo que en el ejemplo anterior, con la máscara B_00_0F, el valor devuelto sería 1 (true). Si el contenido del registro hubiera sido 04A0, entonces el valor devuelto sería 0 (false).
Cmd	Tipo de comando. Cómo se va a leer el registro: • *Read*: función 3 de ModBus. El más usado. • *Read4*: función 4 de ModBus. • *Write*: función 6 de ModBus. • *Write16*: función 16 de Modbus. El más usado.	**PD**	Número de decimales que contiene el registro. Por ejemplo, si se leyera un 301 y PD = 1, en realidad se interpretaría como un 30.1, y si PD = 2, sería un 3.01.
Dir	Dirección del registro a leer.	**UM**	Unidad de medida.
Núm. Words	Número de words de 16 bits a leer.	**Varname**	Nombre del recurso. Ver el siguiente apartado.
Conversión	La conversión de los datos. Puede tomar estos valores: • *Int8*: entero de 8 bits. • *Int16_ML*: entero de 16 bits. El más usado. • *Int16_LM*: entero de 16 bits intercambiado. • *Int32_MwLw_MbLb*: entero de 32 bits. • *Int32_MbLb_MwLw*: entero de 32 bits intercambiado. • *Float32_BE*: Coma flotante de 32 bits. • *Float32_LE*: Coma flotante de 32 bits intercambiado. • *Int64*: Entero de 64 bits.	**Formato**	Suele tener el mismo valor que el punto decimal. Muestra cómo se representa un número decimal, el número de decimales que mostrará.

<table>
<tr><td>Modo lectura</td><td>De qué modo se leerá el recurso.
continuous: se lee de manera continua.
periodic: se lee cada cierto tiempo. Se utiliza con los parámetros.</td><td>Mín.</td><td>Valor mínimo que puede tomar el recurso. No afecta a los parámetros, comandos ni alarmas.</td></tr>
<tr><td>Periodo lectura</td><td>Periodo en el que se realiza la lectura. Los valores por defecto son “-1” en el caso de que el modo de lectura sea “continuous” y “5” en caso de que sea “periodic”.</td><td>Máx,</td><td>Valor máximo que puede tomar el recurso. No afecta a los parámetros, comandos ni alarmas.</td></tr>
<tr><td>Modo escritura</td><td>De qué modo se realizará la escritura del recurso:
Off: registro de solo lectura. Se utiliza con los recursos.
change: se escribe cuando se solicita. Se utiliza con los parámetros y los comandos.</td><td rowspan="2">Condición existencia</td><td rowspan="2">Es la condición que debe cumplirse para que se considere que el registro en cuestión existe y debe ser leído. Si la condición no se cumple en el momento de hacer el reconocimiento de red, la condición es desechada. La condición más básica es “Y” (el recurso existe) o “N” (el recurso no existe). Los parámetros sólo pueden tomar esas condiciones.
Además de “Y” o “N”, una condición puede estar en función del valor de uno o más parámetros, a los que se denotará como @<dirección del parámetro>.
De este modo, si la condición de existencia de un recurso es que el parámetro “HDP”, que está en la dirección 3562, sea igual a 2, se expresaría del siguiente modo:
@3562==2
Si además precisara que el parámetro FCq en la dirección 8795 fuera mayor que 0:
@3562==2 && @8795>0
Es posible crear condiciones más complejas y más largas empleando paréntesis, números negativos y operadores como and (&&) y or (||) y comparadores como igual que (==), distinto de (!=), mayor que (>), menor que (<), mayor o igual que (>=) o menor o igual que (<=).
Es importante notar que sólo se puede emplear parámetros en una condición de existencia.</td></tr>
<tr><td>Periodo escritura</td><td>Cuándo se escribe. El valor por defecto es “-1” siempre, salvo en el caso de los comandos, en los que el valor corresponde al valor entero decimal que deba escribirse en el registro para activar el comando.</td></tr>
</table>

Varname

El varname requiere un apartado propio. Debe ser un nombre que identifique al recurso, comando o parámetro en cuestión de manera unívoca y no debe incluir espacios, comas ni guión bajo (_), pero sí que puede incluir letras (mayúsculas y minúsculas), números, guión medio (-) y punto (.).

Los varnames luego son traducidos por los diccionarios. En el caso de los parámetros, estos varnames son sencillos, pero en el caso de los recursos y comandos, toman la forma de un código concreto seguido, si es necesario, de un paréntesis abierto y una serie de valores separados por comas y que se sustituirán en la traducción: *<código>(<valor 1>, <valor 2>, <valor 3>, etc…*

Los diccionarios, en el caso de los recursos, tienen fijados unos códigos establecidos, que se conforman de un prefijo y de un número identificativo:

- AIxx: entrada analógica
- AOxx: salida analógica
- DIxx: entrada digital
- DOxx: salida digital
- DSxx: estado digital
- ASxx: estado analógico
- ALxx: alarma
- CMxx-x: comando

De este modo, supongamos que tenemos un diccionario con estas dos definiciones:

```
{
        "AL00": "Alarma",
        "AI00": "Sonda $1",
        "AI01": "Sonda temperatura $1 compresor $2",
        "CM01-1" : "Encender luces",
        "CM01-2" : "Apagar luces"
}
```

Teniendo dicho diccionario, los siguientes varnames se traducirían así:

- “AL00”: Alarma
- “AL00(1”: Alarma
- “AI00(3”: Sonda 3
- “AI00”: Sonda
- “AI01(2,4”: Sonda temperatura 2 compresor 4
- “AI01(2”: Sonda temperatura 2 compresor
- “AI00(3,6”: Sonda 3
- “AI01(2,4,9”: Sonda temperatura 2 compresor 4
- “CM01-1”: Encender luces
- “CM01-2”: Apagar luces

Como se puede ver, $1 se sustituiría por el primer valor puesto tras el paréntesis, $2 por el segundo, y así sucesivamente, ignorándose el resto de valores. Esto permite, no obstante, poder cumplir la norma de que todos los varnames deben ser diferentes, lo que hace que, por ejemplo, pudieran crearse distintos “AI00(1” con distinta unidad de medida o distinto punto decimal, identificándolos como “AI00(1,1”, “AI00(1,2”, “AI00(1,3”, etc.

Ejemplos

Ejemplo de una entrada analógica:

Variable;Read;514;1;Int16_ML;FF_FF;1;°C;AI27(2;1;continuous;-1;;; Off;-1;;;;;<-670;>3200;@33020>0 && @33016>0

Ejemplo de una entrada digital:

Status_Dig;Read;5123;1;Int16_ML;B_FF_FF;0;bool;DI10(2;0;continuous;0;;;Off;-1;;;;;;;@33016>0

Ejemplo de una alarma:

Alarm;Read;1551;1;Int16_ML;00_01;0;num;AL8;0;continuous;-1;;;Off;-1;;;;;;;Y

Ejemplo de un comando:

Action;Write;2571;1;Int16_ML;FF_FF;0;num;CM12-1;0; ; ; ;change;1; ; ; ; ; ; ;Y

Ejemplo de un parámetro:

Parameter;Write16;32784;1;Int16_ML;FF_FF;0;num;L08;0;periodic;5; ; ;change;-1; ; ; ; ; ; ;Y