# MCS Protocol

# 1 Revision History

| Revision Date | Notes |
|---|---|
| **01/08/2022** | Revision 3.21<br>**Add new DoubleLine display ^W at Rotate**<br>Add new special command “VR” and “VS”. |
| 21/03/2021 | Revision 3.2<br>Modify the command “CF” to “WF”. |
| 25/8/2017 | Reversion3.14<br>● Add new stopwatch format |
| 28/4/2015 | Reversion3.03<br>● General counter with thousand separator |
| 27/1/2015 | Reversion3.02<br>● Add new special command “BF” read the speed of the box’s fan<br>● Add new special command “BT” read the temperature of the box<br>● Add new special command “BV” read the voltage of the box |
| 7/6/2011 | Revision 3.01<br>● Add new command “F” to opens the file.<br>● Add new command “G” to closes the file.<br>● Add new command “H” to reads the file.<br>● Add new command “I” to writes the file. |
| 23/12/2010 | Revision 3.0<br>● Add support for new file format: script files. Refer to “4.5 Script Files” and “6.5 Write to Script File – Code E” and “Appendix B: Script File Format” for details. |
| 25/11/2010 | Revision 2.64<br>● Add new command “D” to writes to text file without restart. |
| 20/7/2010 | Revision 2.63<br>● User can specify to display any of the two temperature sensors on the wzp controller now. Refer to control code ^R for details. |
| 6/4/2010 | Revision 2.62<br>● Add new display effect “SCL0” and “SCL1”. “SCL0” is exactly the same as “SCL”. “SCL1” just like “SCL”, but will only scroll as many pixels as the content width. |
| 21/1/2010 | Revision 2.61<br>● Add new command “CCF” to write configurations the sign. User can use this command to reconfigure the sign such as adjust the display brightness. |
| 30/12/2009 | Revision 2.6<br>● All control codes such as effect control can be used in variable file now. The only different now is that updating variable is faster than text.<br>● Add new command “CQR” to quick restart running. When variables are updated, they will not update immediately, using this command if you need to quickly restart running. |

| | |
|---|---|
| | ● Add support for absolute positioned TAB control.<br>● Add new command "COU" to control the external general purpose output ports, which can be use to control other devices power for example.<br>● Add new display effect "SCU0" and "SCU1". "SCU0" is exactly the same as "SCU". "SCU1" just like "SCU", but will only scroll as many lines as the content height.<br>● Pause time control code "^JFF" means stop permanently now. This is useful when the display need to keep until new messages received. |
| 27/10/2009 | Revision 2.5<br>● Add system predefined variables: 10 stopwatches in four display formats. Refer to "4.2 Variable Files" for all these predefined variables. All these variable files are read only. Stopwatch variables are very suitable for time counting which need to stop and resume, like in sport game.<br>● Add special command to control the 10 stopwatches which can be display as 10 system predefined variables. Refer to "6.3 Write/Read to Special Function" for new special commands "CWI", "CWS" and "CWT", they tell how to initialize a stopwatch, and how to start and stop it. |
| 30/6/2009 | Revision 2.4<br>● Add support for RGB color control, text color can be specified to any RGB value now, both foreground color and background color can be set. The actual display color is decided by the sign's color type which is currently configured. Refer to control code ^O.<br>● Add support to display humidity and dew point. Refer to control code ^R.<br>● Add new command to retrieve the sensors values. Refer to command "C" with new sub command "RS". |
| 12/5/2009 | Revision 2.3<br>● Add support for general counter which can count from an origin value, relative to a given date time, increase or decrease by a given step size, at a given interval. Refer to "^R" control for general counter. |
| 4/3/2009 | Revision 2.2<br>● Add support for Tri-Color display with new colors: White, Blue, Cyan, Purple, Rainbow6. |
| 30/8/2008 | Revision 2.1<br>● Add support for new decounter & incounter format: hh:nn:ss, hh:nn, and nn:ss. |
| 21/8/2007 | Revision 2.0<br>● Add support to read and write all formats file<br>● Add support for full colors in E2000<br>● Add support for full fonts in E2000<br>● Add support for full display methods in E2000<br>● Add support to display bmp, gif and png image<br>● Add support to display gif animation<br>● Add support to display temperature<br>● Add support to display decounter/incounter<br>● Add support for Tab control |

|  |  |
|---|---|
| 28/12/2006 | Revision 1.1<br>● Add support to large memory size (exceed 64K bytes)<br>● Add support to long file label up to 8 characters.<br>● Add general response for communication testing.<br>● Add support to checksum for high reliable data transferring. |
| 5/6/2006 | Initial Revision |

# 2 Introduction

This document has been developed to allow the users to understand the communication protocol.

The protocol can be used to display text messages, update date and time, and other useful functions.

# 3 Document Conventions

The following conventions are used throughout this document:

| Convention/Symbol | Definition |
|---|---|
| 11 | Number in decimal. |
| $0C | Number in hexadecimal format. |
| %00110011 | Number in binary format. |
| <STX> or ^B | ASCII control character – in this case it is a Ctrl-B. |
| “A” | ASCII character (in this case, the letter A – code $41 or 65. |

# 4 Protocol Overview

This protocol can control a wide range LED display from 8x8 pixels to 2048x256 pixels in single-color or bi-color.
The sign itself can contain many different types of "files". Each file is downloaded to the sign as needed. The maximum file count and a single file size are unlimited, but the total size can not exceed 256K bytes.

## *4.1 File Label*

Each file is named by a file label. Different type of files can use the same file label. File label can be short: a single character, or long: 1-8 characters enclosed by two "$". For example, "A" is a short label and "$TEXT0001$" is a long label, they are both good file labels, and short label "A" can be write in long label "$A$". Text file "A" is initially allocated with size of 256 bytes. Other files must be allocated before using (see special function command code section).
Only these characters can be used for file label:

- Uppercase Letters: "A" to "Z"
- Lowercase Letters: "a" to "z"
- Numbers: "0" to "9"
- Symbols: ~!@#%&_

## *4.2 Text Files*

A text file contains ASCII message data and display control codes to display text. Text files may also include **variable** files as well.

## *4.3 Variable Files*

Variable files are files that contain frequently changing information such as number values. You can easily change these variable files without affecting the text files that contain these. When the sign has received a variable file, the sign will not restart (however the text file will), but keep running as if nothing has happen, and the display will partially updated a few seconds later (the value of the variable changed).
If you need the display restart quickly, you can issue a quick restart command "CQR".
There are some system predefined variables, they are read only and should not use the command code "B" to write to them. Predefined variable's name is prefixed with two underline characters "__". Using ^N$XXXX$ control code in your text file to display them on the screen .

**Predefined Variables : 10 Stop Watches**

| **Name** | **Description** | **Control Code** | **Display** | **Hint** |
|---|---|---|---|---|
| __t0a | Stopwatch number 0, display format hh:nn:ss | ^N$__t0a$ | 23:59:59 | Stopwatch variables are very suitable for time counting which need to stop and resume, like in sport game. Use variable control code |
| __t0b | Stopwatch number 0, display format hh:nn | ^N$__t0b$ | 23:59 | |
| __t0c | Stopwatch number 0, display format nn:ss | ^N$__t0c$ | 59:59 | |
| __t0d | Stopwatch number 0, display | ^N$__t0d$ | 59:59.99 | |

| | | | | |
|---|---|---|---|---|
| | format nn:ss.cc | | | ^N to embedded these variables in your text file.<br><br>Use command codes “CWI” “CWS” and “CWT” to initialize, start and stop the stopwatch. |
| __t0e | Stopwatch number 0, display format nn:ss | ^N$__t0e$ | 99:59<br>0:59 | |
| __t0f | Stopwatch number 0, display format nnn:ss | ^N$__t0f$ | 999:59<br>0:59 | |
| __t1a | Stopwatch number 1, display format hh:nn:ss | ^N$__t1a$ | 23:59:59 | |
| __t1b | Stopwatch number 1, display format hh:nn | ^N$__t1b$ | 23:59 | |
| __t1c | Stopwatch number 1, display format nn:ss | ^N$__t1c$ | 59:59 | |
| __t1d | Stopwatch number 1, display format nn:ss.cc | ^N$__t1d$ | 59:59.99 | |
| __t1e | Stopwatch number 1, display format nn:ss | ^N$__t1e$ | 99:59<br>0:59 | |
| __t1f | Stopwatch number 1, display format nnn:ss | ^N$__t1f$ | 999:59<br>0:59 | |
| __t2a | Stopwatch number 2, display format hh:nn:ss | ^N$__t2a$ | 23:59:59 | |
| __t2b | Stopwatch number 2, display format hh:nn | ^N$__t2b$ | 23:59 | |
| __t2c | Stopwatch number 2, display format nn:ss | ^N$__t2c$ | 59:59 | |
| __t2d | Stopwatch number 2, display format nn:ss.cc | ^N$__t2d$ | 59:59.99 | |
| __t2e | Stopwatch number 2, display format nn:ss | ^N$__t2e$ | 99:59<br>0:59 | |
| __t2f | Stopwatch number 2, display format nnn:ss | ^N$__t2f$ | 999:59<br>0:59 | |
| __t3a | Stopwatch number 3, display format hh:nn:ss | ^N$__t3a$ | 23:59:59 | |
| __t3b | Stopwatch number 3, display format hh:nn | ^N$__t3b$ | 23:59 | |
| __t3c | Stopwatch number 3, display format nn:ss | ^N$__t3c$ | 59:59 | |
| __t3d | Stopwatch number 3, display format nn:ss.cc | ^N$__t3d$ | 59:59.99 | |
| __t3e | Stopwatch number 3, display format nn:ss | ^N$__t3e$ | 99:59<br>0:59 | |
| __t3f | Stopwatch number 3, display format nnn:ss | ^N$__t3f$ | 999:59<br>0:59 | |
| __t4a | Stopwatch number 4, display format hh:nn:ss | ^N$__t4a$ | 23:59:59 | |
| __t4b | Stopwatch number 4, display | ^N$__t4b$ | 23:59 | |

| | | | | |
|---|---|---|---|---|
| | format hh:nn | | | |
| __t4c | Stopwatch number 4, display format nn:ss | ^N$__t4c$ | 59:59 | |
| __t4d | Stopwatch number 4, display format nn:ss.cc | ^N$__t4d$ | 59:59.99 | |
| __t4e | Stopwatch number 4, display format nn:ss | ^N$__t4e$ | 99:59<br>0:59 | |
| __t4f | Stopwatch number 4, display format nnn:ss | ^N$__t4f$ | 999:59<br>0:59 | |
| __t5a | Stopwatch number 5, display format hh:nn:ss | ^N$__t5a$ | 23:59:59 | |
| __t5b | Stopwatch number 5, display format hh:nn | ^N$__t5b$ | 23:59 | |
| __t5c | Stopwatch number 5, display format nn:ss | ^N$__t5c$ | 59:59 | |
| __t5d | Stopwatch number 5, display format nn:ss.cc | ^N$__t5d$ | 59:59.99 | |
| __t5e | Stopwatch number 5, display format nn:ss | ^N$__t5e$ | 99:59<br>0:59 | |
| __t5f | Stopwatch number 5, display format nnn:ss | ^N$__t5f$ | 999:59<br>0:59 | |
| __t6a | Stopwatch number 6, display format hh:nn:ss | ^N$__t6a$ | 23:59:59 | |
| __t6b | Stopwatch number 6, display format hh:nn | ^N$__t6b$ | 23:59 | |
| __t6c | Stopwatch number 6, display format nn:ss | ^N$__t6c$ | 59:59 | |
| __t6d | Stopwatch number 6, display format nn:ss.cc | ^N$__t6d$ | 59:59.99 | |
| __t6e | Stopwatch number 6, display format nn:ss | ^N$__t6e$ | 99:59<br>0:59 | |
| __t6f | Stopwatch number 6, display format nnn:ss | ^N$__t6f$ | 999:59<br>0:59 | |
| __t7a | Stopwatch number 7, display format hh:nn:ss | ^N$__t7a$ | 23:59:59 | |
| __t7b | Stopwatch number 7, display format hh:nn | ^N$__t7b$ | 23:59 | |
| __t7c | Stopwatch number 7, display format nn:ss | ^N$__t7c$ | 59:59 | |
| __t7d | Stopwatch number 7, display format nn:ss.cc | ^N$__t7d$ | 59:59.99 | |
| __t7e | Stopwatch number 7, display format nn:ss | ^N$__t7e$ | 99:59<br>0:59 | |
| __t7f | Stopwatch number 7, display | ^N$__t7f$ | 999:59 | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | format nnn:ss |  | 0:59 |  |
| __t8a | Stopwatch number 8, display format hh:nn:ss | ^N$__t8a$ | 23:59:59 |  |
| __t8b | Stopwatch number 8, display format hh:nn | ^N$__t8b$ | 23:59 |  |
| __t8c | Stopwatch number 8, display format nn:ss | ^N$__t8c$ | 59:59 |  |
| __t8d | Stopwatch number 8, display format nn:ss.cc | ^N$__t8d$ | 59:59.99 |  |
| __t8e | Stopwatch number 8, display format nn:ss | ^N$__t8e$ | 99:59<br>0:59 |  |
| __t8f | Stopwatch number 8, display format nnn:ss | ^N$__t8f$ | 999:59<br>0:59 |  |
| __t9a | Stopwatch number 9, display format hh:nn:ss | ^N$__t9a$ | 23:59:59 |  |
| __t9b | Stopwatch number 9, display format hh:nn | ^N$__t9b$ | 23:59 |  |
| __t9c | Stopwatch number 9, display format nn:ss | ^N$__t9c$ | 59:59 |  |
| __t9d | Stopwatch number 9, display format nn:ss.cc | ^N$__t9d$ | 59:59.99 |  |
| __t9e | Stopwatch number 9, display format nn:ss | ^N$__t9e$ | 99:59<br>0:59 |  |
| __t9f | Stopwatch number 9, display format nnn:ss | ^N$__t9f$ | 999:59<br>0:59 |  |

## *4.4 Special Functions*

These are not really files, but a set of codes that setup the sign itself, like setting the time and date.

## *4.5 Script Files*

Script files contain many command lines to tell the sign how to display messages, such as create window, close window, and so on.

Some script files with special name have special meanings. The sign will search for script file $autorun$ to execute automatically on startup. When writing to script file $cmd$, the script file will be executed immediately without saving to disk.

Please refer to Appendix B for the script file format.

## *4.6 Send or Read File*

The host can send or read file from the sign use advance open file command( ‘F’ )、 advance close file command( ‘G’ )、advance read file command( ‘H’ ) and advance write file command( ‘I’ ).If you want to read the file from WZP sign, you must get the handle from the sign use advance open file command( ‘F’ ) first, then you can use this handle and advance read file command( ‘H’ ) to read the file from WZP sign, finally you must close the file use the handle and the advance close file command( ‘G’ ). If you want to send the file to the WZP Sign, you must get the handle from the WZP Sign use advance open file command ( ‘F’ ) first, then you can use this handle and advance write file command ( ‘I’ ) to write file to WZP sign, finally you must close the file use the handle and the advance close file command ( ‘G’ ).

The host can read the Sign Information and read the Sign file list from the WZP sign as read file. The host can configure the WZP sign、turn power on、turn power off and Change baud rate as write file.

How to send or read file from the sign, please 6.6、6.7、6.8、6.9 for detail.

# 5 Base Protocol Format

The sign responds to two different types of protocol streams. One that uses the full ASCII set (Binary) and one that “escapes” the non-printable ASCII codes. The last one is extremely useful for PLC’s, as only printable ASCII codes are used.

For the binary format, each code shown (i.e. ^B) is the actual ASCII code that is to be transmitted. So ^B would send a code of $02. **Due to the nature of the printable format below, if you wish to have a “^” in your message, you MUST send “^^”.**

For the printable format, each code shown (i.e. ^B) is the ACTUAL series of codes to send. So ^B would send out two ASCII characters “^” and “B”. **If you need to actually display the “^” character in your message, use “^^”.**

**The protocol is flexible enough that you can mix and match codes as desired.**

**For serial communications, the protocol specifics are ALWAYS 8 data bits, 1 stop bit, and no parity. 9600 baud is the factory default value.**

## *5.1 Standard Transmission Packet*

This is the base transmission packet that is needed for all communications:

**Standard Transmission Packet**

| <STX> **^B** | Sign Address | <SOH> **^A** | Command Code | Data Area | <EOT> **^D** | Checksum | <SOH> **^A** | Command Code | … | <ETX> **^C** |
|---|---|---|---|---|---|---|---|---|---|---|

| **Item** | **Description** |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | List of sign address, in hexadecimal format, separated by commas. Each address is 2 ASCII hex digits long. I.E. “01, 0A, 64” is address 1, 10 and 100. The sign will only respond when its address is in this list. Address “00” will cause every sign that is receives this to respond. |
| <SOH> | Start of command. ^A<br>More than one command can be transferred in one transmission packet by using <SOH> instead of <ETX>. You can restart a new command with <SOH> and need not to match the sign address again. Otherwise if <ETX> is found, the next command must begin with <STX> and sign address should rematch. |
| Command Code | Command code is a single uppercase letter “A” to “Z”, represents the command to use. Each command is documented in its own section.<br>Some kind of command will cause the sign restart after <ETX>, such as “A” (Write to text file).<br>**Command Codes** |

| **Command Code** | **Description** |
|---|---|
| “A” | Write to text file. |
| “B” | Write to variable file. |
| “C” | Write/read to/from special function. |

| | |
|---|---|
| | "D" — Write to text file without restart. |
| Data Area | Data area as required for each command. See the appropriate section for each command |
| <EOT> | End of text. ^D<br>Use <EOT> at the end of each command to append checksum. This is optional for high reliable data transmission. |
| Checksum | Checksum is optional and should be used with <EOT> together. They are appended at the end of each command to provide high reliable data transferring. The checksum is a 4 hexadecimal digits string representing a hex number "0000" to "FFFF", which is a word value sum up from <SOH> to <EOT> (inclusive, byte by byte). If <EOT> and Checksum exist, If the checksum mode is Sum, the sign will compare the value with the SUM of bytes actually received, else if the checksum mode is Crc16, the sign will compare the value with the crc16 calculate of bytes actually received. If a bad checksum is checked, the sign will ignore the command to protect the sign from accident damage. |
| <ETX> | End of transmission. ^C<br>Use <ETX> to end all transmission or use <SOH> to start a new command. |

## *5.2 General Response*

When the sign receive a packet ended with <ETX>, it responses some messages to tell communication succeeded or an error occurred.

The sign do not response on these circumstances:

- The packet is broadcasting to all signs with zero address.
- The packet is sent to a group of signs with more than one address in the address list.

The following information may be responded:

- **ok**
  Communication is good and all commands are handled successfully.
- **unknown command code**
  An unknown command code is found. Maybe you should upgrade the software on the sign.
- **bad checksum**
  A checksum is provided but the checksum is not equal to the calculated value.
- **invalid file label**
  The file label includes invalid characters.
- **invalid file size**
  You should provide a 1-8 hex digits file size when you allocate memory for a file.
- **not enough memory**
  There is not enough memory and the memory size allocated for this file does not change.
- **file does not exist**
  You are trying to write to a file which has no memory allocated.

- **file out of allocated size**

  More file data than the allocated memory size are received. The file data is truncated.
- **invalid hexadecimal number**

  A valid hex number should make up of hex digits “0” to “9”, “A” to “F” or “a” to “f”.
- **invalid decimal number**

  A valid dec number should make up of dec digits “0” to “9”.
- **invalid time format**

  A good time format is HHMMSS where HH is hour “00” to “23”, and MM is minute “00” to “59”, and SS is second “00” to “59”.
- **invalid date format**

  A good date format is MMDDYYYYX where MM is month “01” to “12”, and DD is day “01” to “31”, and YYYY is year “2000” to “2099”, and X is day of the week “0” to “6”.
- **unknown beep method**

  The beep method is not currently supported. Maybe you should upgrade the software on the sign.
- **invalid address**

  A valid address should be hex digits “01” to “FF”.
- **bad command parameters**

  The command parameters is in bad value.
- **unknown error**

  An error occurred but the reason is unknown. The software on the sign needs to modify to avoid this information.

# 6 Command Code Sections

This area of the document describes each command code that is used and what the data area must consist of.

### *6.1 Write to Text File – Code "A"*

ASCII messages along with the codes to display them are stored in text files. Text files MUST be allocated (using the special function command) before they can be used. When the sign is first used, a single text file is automatically allocated – it is labeled "A" and has a size of 64K bytes.
There are a few items to note when transmitting text files:

- The display will continue running without disturbance during communication. Once the sign receives a valid text file, it will reallocate memory for the file according to the last "Set Memory" command, clear the file content first, and then copy the new file content.
- This command requires the sign restart after <ETX>. To keep the sign running without restart, use variable file please.
- In addition to containing text, text files can contain other files, specifically variable files. See write to variable file section for further details.
- The message in the file is a set of pairs of mode fields and data to display. Further details below.

**Write To Text File – Command Code "A" – Data Area**

| | Repeat as needed for each message | | | |
|---|---|---|---|---|
| File Label | Mode Field (Optional) | | | ASCII Message |
| 1..10 ASCII Character | <BEL> ^G | Display Position 1 ASCII Character | Mode Code 1 ASCII Character | 1..N ASCII Characters |

| **Item** | **Description** |
|---|---|
| File Label | A file label (i.e. "A" or "$TEXT0001$") |
| Mode Field (Optional) | Set of 3 characters (optional) to define the position and effect to use for the display of message following it. |

**Mode Field**

| **Code** | **Description** |
|---|---|
| <BEL> | Start of mode field. ^G.<br>You can use ^G to set the display position and display effect at one time, or use ^P to set the display position and use ^E to set the display effect separately. ^P and ^E give more advanced features for expert usage. |
| Display Position | Single ASCII character defining the line position on a multi-line sign. If a single-line sign is used, this character is ignored but must be present.<br>**Position Codes** |

| Code | Description |
|---|---|
| “M”<br>$4D | Middle line – text centered vertically. |
| “T”<br>$54 | Top Line - Text begins on the top line of the sign and the sign will use all its lines minus 1 in order to display the text. For example, a 6-line sign will allow a maximum of 5 lines (6 minus 1) for the Top Position. The Top/Bottom Line break will remain fixed until the next Middle or Fill position is specified. |
| “B”<br>$42 | Bottom Line - The starting position of the Bottom Line(s) immediately follows the last line of the Top Line. For example, a 6-line sign with 3 lines of text associated with the Top Line would start the Bottom Line text on the 4th line of the sign. |
| “F”<br>$46 | Fill – The sign will fill all available lines, centering them vertically. |
| “L”<br>$4C | Left - Text begins on the left side of the sign and the sign will use all its lines minus 1 in order to display the text |
| “R”<br>$52 | Right - Text begins on the right side of the sign and the sign will use all its lines minus 1 in order to display the text |

Mode Code

**Mode Codes:**

| Code | Name | Description |
|---|---|---|
| “S”<br>$53 | Scroll | Message scrolls right to left. |
| “H”<br>$48 | Hold | Message displays stationary. |
| “F”<br>$46 | Flash | Message displays stationary and flashes. |
| “A”<br>$41 | Slide Up | Previous message is slide up by new message. |
| “B”<br>$42 | Slide Down | Previous message is slide down by new message. |
| “C”<br>$43 | Slide Left | Previous message is slide left by new message. |
| “D”<br>$44 | Slide Right | Previous message is slide right by new message. |
| “a”<br>$61 | Roll Up | Previous message is rolled up by new message. |
| “b”<br>$62 | Roll Down | Previous message is rolled down by new message. |
| “c”<br>$63 | Roll Left | Previous message is rolled left by new message. |
| “d” | Roll Right | Previous message is rolled right by new |

| | |
|---|---|
| | $64 \| \| message. |
| ASCII Message | Message to display. Can contain various codes (listed below) to affect the color, font, speed, pause, embed dates and times, and display variable files. It also can contain codes to set new lines and new pages to display.<br><br>NOTE: Most of the codes have a default – the default is used at the start of the message when displaying. If changed, subsequently, the message will use the new changes until the end of the message is reached. Once the message cycles and starts over, the defaults are reset. |

### Message Codes

| **Code** | **Description** |
|---|---|
| ^E<br>$05 | Set display effect. Other than ^G, ^E can set both single character and multi-characters mode codes.<br><br>**a. Single character mode codes:** |

| **Code** | **Name** | **Description** |
|---|---|---|
| "S"<br>$53 | Scroll | Message scrolls right to left. |
| "H"<br>$48 | Hold | Message displays stationary. |
| "F"<br>$46 | Flash | Message displays stationary and flashes. |
| "A"<br>$41 | Slide Up | Previous message is slide up by new message. |
| "B"<br>$42 | Slide Down | Previous message is slide down by new message. |
| "C"<br>$43 | Slide Left | Previous message is slide left by new message. |
| "D"<br>$44 | Slide Right | Previous message is slide right by new message. |
| "a"<br>$61 | Roll Up | Previous message is rolled up by new message. |
| "b"<br>$62 | Roll Down | Previous message is rolled down by new message. |
| "c"<br>$63 | Roll Left | Previous message is rolled left by new message. |
| "d"<br>$64 | Roll Right | Previous message is rolled right by new message. |

**b. Multi-characters mode codes:**

| Code | Name | Description |
|---|---|---|
| “$AUT$” | Auto | Randomly choose a display method. |
| “$HLD$” | Hold | Message display stationary. |
| “$SLU$” | Slide Up | Previous message is slide up by new message. |
| “$SLD$” | Slide Down | Previous message is slide down by new message. |
| “$SLL$” | Slide Left | Previous message is slide left by new message. |
| “$SLR$” | Slide Right | Previous message is slide right by new message. |
| “$SFC$” | Slide From Center | Previous message is slide from center by new message. |
| “$STC$” | Slide To Center | Previous message is slide to center by new message. |
| “$CVU$” | Cover Up | Previous message is covered from bottom by new message. |
| “$CVD$” | Cover Down | Previous message is covered from top by new message. |
| “$CVL$” | Cover Left | Previous message is covered from right by new message. |
| “$CVR$” | Cover Right | Previous message is covered from left by new message. |
| “$CFC$” | Cover From Center | Previous message is covered from center by new message. |
| “$CTC$” | Cover To Center | Previous message is covered to center by new message. |
| “$ROU$” | Roll Up | Previous message is rolled up by new message. |
| “$ROD$” | Roll Down | Previous message is rolled down by new message. |
| “$ROL$” | Roll Left | Previous message is rolled left by new message. |
| “$ROR$” | Roll Right | Previous message is rolled right by new message. |
| “$RFC$” | Roll From Center | Previous message is rolled from center by new message. |
| “$RTC$” | Roll To Center | Previous message is rolled to center by new message. |
| “$INS1$” | Inter-Slide 1 | Message slide in with interlaced mode 1. |
| “$INS2$” | Inter-Slide 2 | Message slide in with interlaced mode 2. |
| “$INS3$” | Inter-Slide 3 | Message slide in with interlaced mode 3. |

| | | |
|---|---|---|
| “$INS4$” | Inter-Slide 4 | Message slide in with interlaced mode 4. |
| “$INR1$” | Inter-Roll 1 | Message roll in with interlaced mode 1. |
| “$INR2$” | Inter-Roll 2 | Message roll in with interlaced mode 2. |
| “$INR3$” | Inter-Roll 3 | Message roll in with interlaced mode 3. |
| “$INR4$” | Inter-Roll 4 | Message roll in with interlaced mode 4. |
| “$INR5$” | Inter-Roll 5 | Message roll in with interlaced mode 5. |
| “$INR6$” | Inter-Roll 6 | Message roll in with interlaced mode 6. |
| “$SHU1$” | Shutter 1 | Message display like shutter using method 1. |
| “$SHU2$” | Shutter 2 | Message display like shutter using method 2. |
| “$SHU3$” | Shutter 3 | Message display like shutter using method 3. |
| “$SHU4$” | Shutter 4 | Message display like shutter using method 4. |
| “$JMP$” | Jump | Message blocks jump to the screen. |
| “$SNO$” | Snow | Message displays like snow fall. |
| “$RAN$” | Random | Message pixels displays on the screen in random order. |
| “$SHO$” | Shoot | Messages shoot on the screen. |
| “$EXP$” | Explode | Message blocks explode on the screen. |
| “$FLS$” | Flash | Message display stationary and flash. |
| “$TWK$” | Twinkle | Message display stationary and twinkle. |
| “$PAC$” | Pac Man | Previous message is eat by a big mouse and new message is drop down. |
| “$SCL$” | Scroll Left | Message continually scrolls from right to left. |
| “$SCL0$” | Scroll Left 0 | Same as “$SCL$”. |
| “$SCL1$” | Scroll Left 1 | Just like “$SCL$”, but only as many pixels as the content width will be scroll, no blank pixels appended. |
| “$SCU$” | Scroll Up | Message continually scrolls from bottom to top. Blank lines will be appended after all content scrolled up. |
| “$SCU0$” | Scroll Up 0 | Same as “$SCU$”. |
| “$SCU1$” | Scroll Up 1 | Just like “$SCU$”, but only as many lines as the content height will be scroll, no blank lines appended. |

| | |
|---|---|
| ^F<br>$06 | Set font.<br>**a. Set font by index:** followed by one of the following codes to change the font:<br>● “0” – SS7 (default)<br>● “1” – SF7 |

| | | |
|---|---|---|
| | | ● “2” – SF10<br>● “3” – SS16<br>● “4” – SF16<br>**b. Set font by name:** $FONTNAME$<br>● “$SS5$” – SS5<br>● “$SS7$” – SS7<br>● “$SF7$” – SF7<br>● “$SF10$” – SF10<br>● “$SS16$” – SS16<br>● “$SF16$” – SF16<br>● “$TM16$” – TM16<br>● “$AR16$” – AR16<br>● “$SMA$” – SMA<br>NOTE: the default font SS7 is used when the font does not exist.<br>Warning:The index of the fonts must be follow the fonts' order set in downloading. |
| | ^H<br>$08 | Set character attribute (flash, wide, bold). Followed by one of the codes:<br>**Character attribute code:**<br>● “0” – Set flashing off. (default)<br>● “1” – Set flashing on.<br>● “2” – Set wide off. (default)<br>● “3” – Set wide on.<br>● “4” – Set bold off. (default)<br>● “5” – Set bold on. |
| | ^I<br>$09 | Set speed. Followed by one speed ASCII character “B” ,“A”,“0” to “8” for twelve different speeds. (default=”3”)<br>“B” is fastest,”8”is slowest |
| | ^J<br>$0A | Set pause. Followed by 2 hexadecimal ASCII digits representing the amount of pause time, in seconds. When used for scrolling right to left messages – it will immediately pause and wait for X seconds. When used for any other mode – the pause is used for the page, before going to the next page or message.<br><br>Setting this to “00” will set no pauses for each of the pages. Once this is set for non-scrolled pages/message, it is subsequently used for every page during message display. (default=”02”)<br><br>Setting this to “FF” will stop running permanently, until new messages have received and the sign restart running. |
| | ^K<br>$0B | Display time/date. This will embed the current time and date. Followed by 2 ASCII characters.<br><br>The first ASCII character describes how to display.<br>**Time format code:** |

|  |  |  |  |
|---|---|---|---|
|  |  |  | ● “0” – Do NOT show leading zeroes.<br>● “1” – Show leading zeroes.<br>● “2” – Show leading zeroes as spaces.<br>● “5” – Show as ALL CAPS.<br>● “6” – Show as lowercase.<br>● “7” – Show as First-Letter Caps.<br><br>The second ASCII character describes what to display.<br>**Time element code:**<br>● “0” = Numeric day<br>● “1” = Numeric month<br>● “2” = Numeric year (last 2 digits only)<br>● “3” = Numeric year (all four digits)<br>● “4” = Month Abbreviation name.<br>● “5” = Month full name.<br>● “6” = Day of the week abbreviation.<br>● “7” = Day of the week full name.<br>● “8” = Hour in 12-hour mode.<br>● “9” = Hour in 24-hour mode.<br>● “A” = Minute<br>● “B” = Seconds<br>● “C” = AM/PM as a single character A/P<br>● “D” = AM/PM as two characters AM/PM<br>● “E” = Suffix of the day, like ‘st’, ‘nd’, ‘rd’, or ‘th’ |
|  |  | ^L<br>$0C | New page – starts a new display page (based on mode, etc). |
|  |  | ^M<br>$0D | New line – starts a new line for multi-line displays. |
|  |  | ^N<br>$0E | Embed variable file. Followed by a file label (i.e. “A” or “$VAR01$”) representing the variable file. |
|  |  | ^O<br>$0F | Change color.<br>**a. Change color by index:** followed by a single ASCII character representing the color to change to:<br>● “0” = Red (default)<br>● “1” = Green<br>● “2” = Yellow/Amber<br>● “3” = Rainbow 1<br>**b. Change color by name:** $COLORNAME$<br>● “$ACL$” = Auto Color<br>● “$WHT$” = White<br>● “$RED$” = Red<br>● “$GRN$“ = Green<br>● “$BLU$” = Blue<br>● “$YEL$” = Yellow |

| | | | |
|---|---|---|---|
| | | | ● “$CYN$” = Cyan<br>● “$PUR$” = Purple<br>● “$RB1$” = Rainbow 1<br>● “$RB2$” = Rainbow 2<br>● “$RB3$” = Rainbow 3<br>● “$RB4$” = Rainbow 4<br>● “$RB5$” = Rainbow 5<br>● “$RB6$” = Rainbow 6<br>● “$MIX1$” = Mixture 1<br>● “$MIX2$” = Mixture 2<br>● “$MIX3$” = Mixture 3<br>● “$MIX4$” = Mixture 4<br>● “$INV1$” = Invert 1<br>● “$INV2$” = Invert 2<br>● “$INV3$” = Invert 3<br>● “$INV4$” = Invert 4<br>● “$INV5$” = Invert 5<br>● “$INV6$” = Invert 6<br>● “$INV7$” = Invert 7<br>● “$INV8$” = Invert 8<br>● “$INV9$” = Invert 9<br><br>● $[F: COLOR] [,] [ B: COLOR]$<br>Where “F:” indicate the text foreground color, and “B:” indicate the text background color. “COLOR” can be tree dec value like rgb(255, 128, 0), or a hex value like #FF8000, or a lowercase string of standard color name like red. Please refer to Appendix A to see what standard color names are available now.<br><br>Example 1: to set the text foreground color to Blue, and background color to Red, usage ^O$F:blue, B:red$, or ^O$F:rgb(0,0,255), B:rgb(255,0,0)$, or ^O$F:#0000FF, B:#FF0000$.<br><br>Example 2: to set the text foreground color to Blue, and background color to transparent, usage ^O$F:blue$<br><br>Example 3: to set the text foreground color to transparent, and background color to red, usage ^O$B:red$<br><br>NOTE: the default color RED is used when the color does not exist. |
| | | ^P<br>$10 | Set display position. Other than ^G, ^P can set both single character and multi-characters position codes.<br><br>**a. Single character position codes:** |

| Code | Description |
|---|---|
| "M" $4D | Middle line – text centered vertically. |
| "T" $54 | Top Line - Text begins on the top line of the sign and the sign will use all its lines minus 1 in order to display the text. For example, a 6-line sign will allow a maximum of 5 lines (6 minus 1) for the Top Position. The Top/Bottom Line break will remain fixed until the next Middle or Fill position is specified. |
| "B" $42 | Bottom Line - The starting position of the Bottom Line(s) immediately follows the last line of the Top Line. For example, a 6-line sign with 3 lines of text associated with the Top Line would start the Bottom Line text on the 4th line of the sign. |
| "F" $46 | Fill – The sign will fill all available lines, centering them vertically. |
| "L" $4C | Left - Text begins on the left side of the sign and the sign will use all its lines minus 1 in order to display the text |
| "R" $52 | Right - Text begins on the right side of the sign and the sign will use all its lines minus 1 in order to display the text |

**b. Multi-characters position codes:**

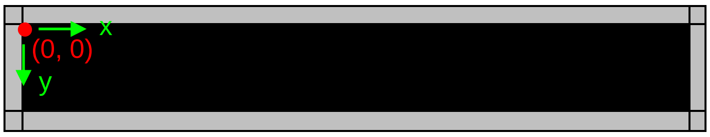
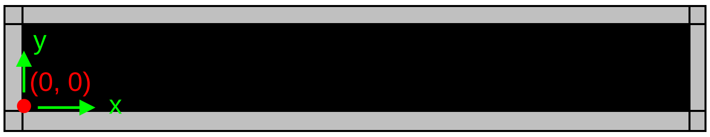
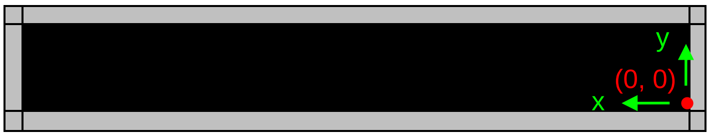
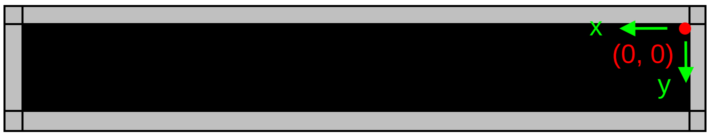
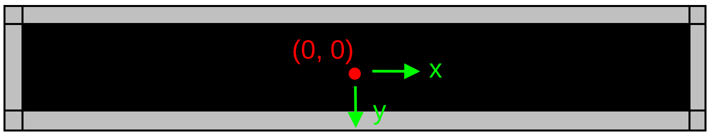
Format: $XXXX,YYYY,WWWW,HHHH,A$ where

- XXXX = x coordinate, 1..4 decimal digits
- YYYY = y coordinate, 1..4 decimal digits
- WWWW = width, 1..4 decimal digits
- HHHH = height, 1..4 decimal digits
- A = text alignment, 1 ASCII character
  - "0" = left, top
  - "1" = center, top
  - "2" = right, top
  - "3" = left, middle
  - "4" = center, middle (default)
  - "5" = right, middle
  - "6" = left, bottom
  - "7" = center, bottom
  - "8" = right, bottom

NOTE 1: the ^Q control will affect the x and y coordinates of display position.

NOTE 2: using the expression with symbol # can specify a relational value to the full screen size. Symbol # represent the value of screen width when setting the window's left and width, also represent the value of screen height when setting the window's top and height.

NOTE3:The with and the height can not Over the Screen With and Screen Height.

| | | |
|---|---|---|
| | | Examples:<br>**^P$0, 0, 128, 16, 4$** set the display position to the rectangle area (0, 0, 128, 16), and set the text alignment to center-middle.<br><br>For a 128x32 sign, **^P$32, 16, (#-32)/2, #, 3$** set the display position to the rectangle (32, 16, 48, 16), and set the text alignment to left-middle. The width (#-32)/2 = (128-32)/2 = 48, and height # using the maximum spacing, that is 32-16 = 16.<br><br>For a 128x32 sign, **^P$(#/4)*3, 0, #/4, #, 0$** set the display position to the rectangle (96, 0, 32, 32), and set the text alignment to left-top. The x coordinate is (#/4)*3 = (128/4)*3 = 96, width is #/4 = 128/4 = 32, and height # using the maximum spacing, that is 32. |
| | ^Q<br>$11 | Set coordinate reference. Followed by a single ASCII character.<br><br>**Coordinate reference code:**<br>• “0” (default)<br> <br>•<br><br>• “1”<br> <br>•<br><br>• “2”<br> <br>•<br><br>• “3”<br> <br>•<br><br>• “4”<br> <br>• |

|  |  |  |
|---|---|---|
|  |  | Examples:<br>For a 128x32 sign, **^Q1^P$0, 0, 128, 16, 4$** set the display position to the rectangle (0, 16, 128, 16). |
|  | ^R<br>$12 | Display embedded object such as temperature, decounter, incounter etc. Followed by two ASCII characters representing the object to embed:<br><br>**Temperature**<br>Format: followed by a single ASCII character.<br>● "0" = Centigrade temperature, such as 20°C<br>● "1" = Fahrenheit temperature, such as 68°F<br>● "2" = Kelvin temperature, such as 293K<br><br>The temperature will display "??°C" or "??°F" or "???K" when the temperature sensor is not installed.<br><br>**Decounter**<br>Format: $XX, MM-DD-YYYY[ HH:NN:SS]$ where<br>XX is:<br>● "10" = decounter, count in days<br>● "11" = decounter, count in hours<br>● "12" = decounter, count in minutes<br>● "13" = decounter, count in seconds<br>● "14" = decounter, count in hh:nn:ss<br>● "15" = decounter, count in hh:nn<br>● "16" = decounter, count in nn:ss<br><br>MM is month "1" to "12".<br>DD is day "1" to "31".<br>YYYY is year such as "2000".<br>HH:NN:SS is optional, treat as "00:00:00" if absent.<br>HH is hour "00" to "23".<br>NN is minute "00" to "59".<br>SS is second "00" to "59".<br><br>The decounter can display number within 99999999, a larger number will be displayed as '--------'. 0 will be displayed when the time passed.<br><br>**Incounter**<br>Format: $XX, MM-DD-YYYY[ HH:NN:SS]$ where<br>XX is:<br>● "20" = incounter, count in days<br>● "21" = incounter, count in hours<br>● "22" = incounter, count in minutes<br>● "23" = incounter, count in seconds |

- “24” = incounter, count in hh:nn:ss
- “25” = incounter, count in hh:nn
- “26” = incounter, count in nn:ss

MM is month “1” to “12”.
DD is day “1” to “31”.
YYYY is year such as “2000”.
HH:NN:SS is optional, treat as “00:00:00” if absent.
HH is hour “00” to “23”.
NN is minute “00” to “59”.
SS is second “00” to “59”.

The incounter can display number within 99999999, a larger number will be displayed as ‘--------’. 0 will be displayed when the time is not reached.

**General Counter**
Format: $XX, MM-DD-YYYY[ HH:NN:SS], ORIGIN, STEP, INTERVAL$ where
XX is:

- “30” = General down counter
- “31” = General up counter
- “32” = General down counter with thousand separator
- “33”= General up counter with thousand separator

MM is month “1” to “12”.
DD is day “1” to “31”.
YYYY is year such as “2000”.
HH:NN:SS is optional, treat as “00:00:00” if absent.
HH is hour “00” to “23”.
NN is minute “00” to “59”.
SS is second “00” to “59”.
ORIGIN is the origin counter value.
STEP is the step sizes the counter value should increase or decrease.
INTERVAL is the interval time between increasing or decreasing counter value, in seconds.

**General Sensors**
Format: $TYPE, UNIT, WIDTH, PRECISION$ where
TYPE is:

- “40” = Temperature
- “41” = Humidity
- “42” = Dew Point
- “43” = Temperature A
- “44” = Temperature B

| | | | |
|---|---|---|---|
| | | | UNIT is:<br>● “C” = the temperature value should display in Celsius and should append “°C” after the temperature value.<br>● “F” = the temperature value should display in Fahrenheit and should append “°F” after the temperature value.<br>● “K” = the temperature value should display in Kelvin and should append “K” after the temperature value.<br>● “P” = the humidity value should display in Percent and should append “%” after the humidity value.<br>● “c” = the temperature value should display in Celsius but should not append “°C” after the temperature value.<br>● “f” = the temperature value should display in Fahrenheit but should not append “°F” after the temperature value.<br>● “k” = the temperature value should display in Kelvin but should not append “K” after the temperature value.<br>● “p” = the humidity value should display in Percent but should not append “%” after the humidity value.<br><br>WIDTH is one character “2” to “3” to indicate how many digits the integer portion should display.<br><br>PRECISION is one character “0” to “2” to indicate how many digits the fraction portion should display.<br><br><br>**Examples:**<br>^R0 — Centigrade temperature, such as 20°C<br>^R$10, 1/1/2008$ — display days count to 1/1/2008 00:00:00<br>^R$20, 1/1/2000 00:00:00$ — display days count from 1/1/2000 00:00:00<br>$R$41,P,2,1$ — display the humidity, such as 57.6% |
| | | ^S<br>$13 | Display embedded object from file. The file can be a text file, image file or animation file.<br><br>**Format 1:** Followed by a file label (i.e. “A” or “$0001$”) representing the *.bmp image file.<br><br>**Format 2:** $NAME.EXT$ where NAME is 1-8 characters file name, and EXT is 1-3 characters file extension. Both NAME and EXT are case sensitive. The EXT must be the following values:<br><br>● “txt” = Include another text file<br>● “bmp” = embedded a *.bmp image<br>● “gif” = embedded a *.gif image or animation<br>● “png” = embedded a *.png image |

| | | |
|---|---|---|
| | | **Include another text file**<br>Unlike the variable file, the current text stop running and text attributes (such as font, color) are saved, and the included text will run. At last, the text attributes restore and the previous text continue.<br><br>The inclusion depth is limit to 20, cross inclusion is inhibited.<br><br>NOTE: The default file type is *.bmp image, so “^SA” is equal to “^S$A.bmp$”, and “^S$0001$” is equal to “^S$0001.bmp$”. |
| | ^T<br>$14 | Tab control which makes the subsequence text align to “grids”.<br><br>**Relative positioned TAB control**<br><br>Followed by a single ASCII character:<br>● “0” = left aligned Tab control<br>● “1” = right aligned Tab control<br>● “2” = center aligned Tab control<br>● “3” = decimal point aligned Tab control<br><br>NOTE1: the default TAB step size is 32 pixels.<br>NOTE2: using multiple “^T0” before other TAB control to specify different TAB size, or using “^U$1,n$” before any TAB control.<br>NOTE 3: using “^U0” before to set the horizontal text alignment to left aligned, or the TAB position may not aligned to the same position in different lines.<br><br>**Absolute positioned TAB control**<br>Format: $TYPE, POSITION$<br>Where TYPE is one character:<br>● “0” = left aligned TAB control<br>● “1” = right aligned TAB control<br>● “2” = center aligned TAB control<br>● “3” = decimal point aligned TAB control<br><br>Where POSITION is 1-4 decimal characters, which is the absolute coordinate in horizontal. |
| | ^U | Other settings such as text alignment.<br><br>**Set text alignment**<br>Format: followed by a single ASCII character<br>● “0” = left, top<br>● “1” = center, top |

| | | |
|---|---|---|
| | | ● “2” = right, top<br>● “3” = left, middle<br>● “4” = center, middle (default)<br>● “5” = right, middle<br>● “6” = left, bottom<br>● “7” = center, bottom<br>● “8” = right, bottom<br><br>**Set Tab step size**<br>Format: $1, TABSTEP$<br>Where TABSTEP is 1-4 decimal characters. Default value is “32”.<br><br>**Set horizontal spacing between characters**<br>Format: $2, HSPACE$<br>Where HSPACE is one character “0”-“9”. Default value is “0”.<br><br>**Set vertical spacing between lines**<br>Format: $3, VSPACE$<br>Where VSPACE is one character “0”-“9”. Default value is “0”.<br><br>**Enable/disable Word-Wrap**<br>Format: $4, WORDWRAP$<br>Where WORDWRAP is one character: “0” = Disable; “1” = Enable.<br>When disabled, a word may be divided and display in two lines.<br>Default value is “1”.<br><br>**Enable/disable word space compressing**<br>Format: $5, WORDCMPR$<br>Where WORDCMPR is one character: “0” = Disable; “1” = Enable.<br>When enabled, the word spacing may be compressed to fit one more word in the line.<br>Default value is “1”.<br><br>**Enable/disable word space expanding**<br>Format: $6, WORDEXPD$<br>Where WORDEXPD is one character: “0” = Disable; “1” = Enable.<br>When enabled, the word spacing will expand to fill the whole line.<br>Default value is “1”. |
| | ^V | **Beep**<br>Format: followed by a single ASCII character.<br>● “1” = [BEEP1]<br>● “2” = [BEEP2]<br>● “3” = [BEEP3] |

| | | |
|---|---|---|
| | | "4" = [BEEP4] |
| | ^W | **Double Line Mode**<br>Format: followed by a single ASCII character.<br>● "0" = "L" = Left<br>● "1" = "C" = Center<br>● "2" = "R" = Right<br>Use ^M to second line<br>Use ^L to end Double Line Mode<br>**Examples:** ^W0Abcdefg^MHij^L<br>Abcdefg<br>Hij<br>**Examples:** ^WLMan Utd^MLiverpool^WR 2^M 1^L<br>**Man Utd 2**<br>**Liverpool 1** |
| | | |
| | | |

## *6.2 Write to Variable File – Code "B"*

Variables files are used to store frequently changing information, such as measurements, short pieces of text, and other ASCII text/numeric values.

When writing to a variable file, the sign need NOT to restart. Once the sign receives a variable file, it will reallocate memory for the file according to the last "Set Memory" command, clear the file content first, and then copy the new file content.

Before writing to a variable file, the file must be setup using the special function command to allocate memory for the file. The maximum size of a variable file is unlimited.

Variable files can only be displayed by embedding codes for them in a text file. Anytime the text file goes to show the variable file, it will pull out the last data sent to that variable file. That way, you can continuously update the variable file without affecting the running of the current text file.

Variable files do NOT have any mode options and simply contain the ASCII message to display. They are allowed some of the simple embedded codes to change the fonts/colors, etc.

Steps for using variable files:

1. Allocate memory in the sign for the variable file and the text file that embeds it. [Use the Set Memory Special function to do this.
2. Write the text file that has the embedded variable file code in it.
3. Update the variable file as much as needed to change the data on the display.

**Write To Variable File – Command Code "B" – Data Area**

| File<br>Label<br>1..10 ASCII | ASCII<br>Message<br>1..N ASCII |
|---|---|

| Character | characters |
|---|---|

| Item | Description |
|---|---|
| File Label | A file label (i.e. “A” or “$VAR0001$”) |
| ASCII Message | Message to display. Can contain all of the codes that are used for text files. Please refer to the text file section for details on what each of the codes will do. |

## *6.3 Write/Read to Special Function – Code “C”*

This protocol packet will allow you to set certain functions, including setting up memory, setting the date and time, etc.

In addition, some of the commands respond with data as needed. This response is sent before “General Response” and is always in the ASCII printable protocol stream as is as follows:

**Standard Response Packet**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| <STX> **^B** | Sign Address | <SOH> **^A** | Command Code “C” | Special Function Code | Special Function Response Data | <EOT> **^D** | Checksum | <ETX> **^C** |

| Item | Description |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | Sign address of the sign that is responding. 2 ASCII hexadecimal digits. |
| <SOH> | Start of command. ^A |
| Command Code | A single ASCII character representing the command response code. In this case, “C” |
| Special Function Code | Original 2-ASCII character special function request code. |
| Special Function Data | 0 to N characters of special function response data. Depends on the original request. |
| <EOT> | End of text. ^D |
| Checksum | 4 hex digits represent a hex word value from “0000” to “FFFF”, if the checksum mode is Sum, which is the SUM of bytes from <SOH> to <EOT> (inclusive, byte by byte), else if the checksum mode is Crc16,which is the crc16 calculate of bytes from <SOH> to <EOT>(inclusive, byte by byte). |
| <ETX> | End of transmission. ^C |

**Write To Special Function – Command Code “C” – Data Area**

| | |
|---|---|
| Special Function Code | Special Function Data |

| 2 ASCII Characters | 0..N ASCII Characters |
|---|---|

| Item | Description |
|---|---|
| | |
| Special Function Code and Data | Code consisting of 2 ASCII characters plus an additional 0 to N characters for the data. Each code is described below and the data that is required. |

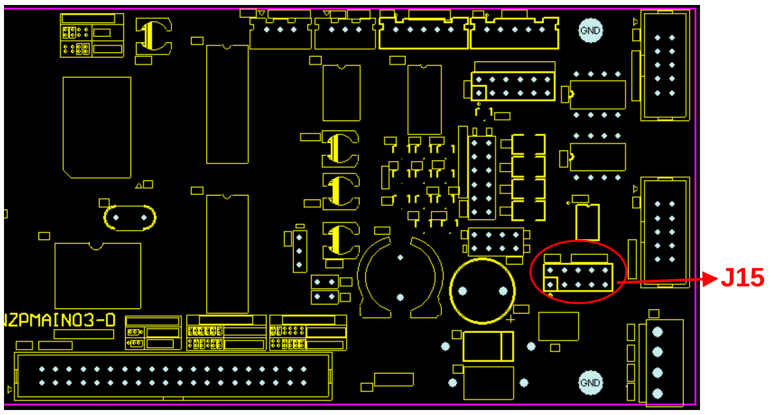
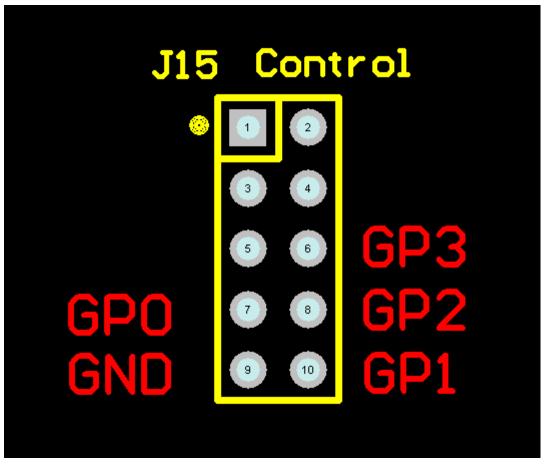
**Special Function Codes – Write Only, No response**

| Code | Description |
|---|---|
| “ST” | Set time.<br><br>**Set time format:** HHMMSS where:<br>● HH = Hour (decimal), “00” to “23”<br>● MM = Minute (decimal), “00” to “59”<br>● SS = Second (decimal), “00” to “59”.<br><br>A separator “:” is optional between HH, MM and SS like HH: MM: SS.<br><br>The sign’s time will not change until <ETX> is received. You can use <SOH> to continue a “Set Date” command. |
| “CM” | Clear entire memory. This will clear the entire memory and reset to factory default.<br>This command will set the memory to one text file (“A”) and no variable files.<br>This command will NOT reset the address.<br>This command requires the sign to restart after <ETX>. |
| “SM” | Set memory. This is used to set the file memory size. Followed by sets of ASCII characters as follows:<br><br>**Set memory format:** FTSSSS where:<br>● F = File Label (i.e. “A” or “$TEXT0001$”)<br>● T = File type, currently:<br>○ “T” = Text file.<br>○ “V” = Variable file.<br>○ “N” = Text file without restart.<br>○ “S” = Script file.<br>● SSSS = Size, 1-8 hexadecimal ASCII digits “0” to “7FFFFFFF”, representing the size, in bytes, of the file to allocate. Set to “0” to remove the file from the list.<br><br>NOTE1: This command only affects the next “Write to Text/Variable/Script File” command. The file’s content will not be |

| | | |
|---|---|---|
| | | changed until the “Write to Text/Variable/Script File” command has been received.<br><br>NOTE2: When file type is “T”, the sign will restart running after <ETX>. If you don’t want restart, use file type “N” instead.<br><br>To continue configuring the memory, using <SOH> (^A or $01) and repeat this command. |
| | “SD” | Set date.<br><br>**Set date format:** MMDDYYYYX where:<br>● MM = Month (decimal), “01” to “12”<br>● DD = Day (decimal), “01” to “31”<br>● YYYY = Year (decimal) – 4 digits, “2000” to “2099”.<br>● X = Day of the week, where: “0”=Sunday to “6”=Saturday.<br><br>A separator “/” or “,” is optional between MM, DD, YYYY and X like MM/DD/YYYY, X.<br><br>The sign’s date will not change until <ETX> is received. You can use <SOH> to continue a “Set Time” command. |
| | “SR” | Set a run sequence. Followed by 1 to N characters representing text file labels that are to be displayed in order. This code immediately starts the order with the first text file label given. Once the last file is displayed, the order starts over again. Transmitting any text file to the sign will stop the run sequence (variable files do not do this). Only text file labels can be used.<br>A separator “ “ or “,” is optional between file labels.<br>A good sequence example is “ABC$TEXT0001$$TEXT0002$”.<br>This command requires the sign to restart after <ETX>. |
| | “SB” | Set beep. Beeps the internal speaker. Followed by 1 ASCII character as follows:<br>**Beep code:**<br>● “0” = Beep continuously for 1 second<br>● “1” = Beep continuously for 2 seconds.<br>● “2” = Beep on/off quickly for 2 seconds.<br>● “3” = A short beep<br>● “4” = Three short beep<br>● “5” = Eight short beep and one long beep<br>● “6” = Light beep<br>● “7”= Beep on<br>● “8”= Beep off |
| | “SA” | Set sign address. Followed by the new sign address that take effect immediately: |

| | | |
|---|---|---|
| | | **Set address format:** AA where AA is two ASCII hexadecimal digits representing the new address (“01” to “FF”).<br><br>The default address from the factory is “01”. |
| | “PF” | Turn power off |
| | “PO” | Turn power on |
| | “PR” | Explicitly require the sign to restart after <ETX>. Followed by a single ASCII character:<br>● “0” = Restart and show startup screen.<br>● “1” = Restart but do not show startup screen. |
| | “FM” | Allocate memory for any format file.<br><br>Format: NAME.EXT=SIZE where:<br>NAME is 1-8 characters file name, and EXT is 1-3 characters file extension. Both NAME and EXT are case sensitive. The EXT can be the following values:<br><br>● “txt” = Text file<br>● “var” = Variable file<br>● “sh” = Script file<br>● “bmp” = *.bmp image<br>● “gif” = *.gif image or animation<br>● “png” = *.png image<br><br>SIZE is 1-8 hex digits “0” to “7FFFFFFF” representing the size, in bytes, of the file to allocate. Set to “0” to remove the file from the list.<br><br>The “FM” command is similar to “SM” command, but can allocate memory for all format files.<br><br>It Should use with FW commnad or Write Text or Write Variable with One Frame. |
| | “FW” | Write to any format file include text file, variable file, image file, animation file etc.<br><br>**Write file format:** NAME.EXT=CONTENT where:<br>NAME is 1-8 characters file name, and EXT is 1-3 characters file extension. Both NAME and EXT are case sensitive. The EXT can be the following values:<br><br>● “txt” = Text file |

| | | |
|---|---|---|
| | | ● "var" = Variable file<br>● "sh" = Script file<br>● "bmp" = *.bmp image<br>● "gif" = *.gif image or animation<br>● "png" = *.png image<br><br>CONTENT is 1-N characters file content, must be encoded in Base64.<br><br>User should use this command instead of "A" and "B" to write a text or variable file which contains 8-bits characters in 7-bits system. |
| | "WI" | There are 10 stopwatches can be use to counting time like in sport game. You should use the variable control code ^N to display a system predefined stopwatch variable. Refer to "4.2 Variable Files" for detail.<br><br>To display a stopwatch, you should use this command to initialize it first, and then issue a start command.<br><br>Format: INDEX [ ENABLE DOWN VALUE ] where:<br>● INDEX '0' to '9', indicate which of the 10 stopwatches to be initialized.<br>● ENABLE '0' or '1', indicate that the stopwatch will be disabled or enabled. Default is '1'.<br>● DOWN '0' or '1', indicate that the stopwatch should count up or count down. Default is '0'.<br>● VALUE set the stopwatch initial time value, in 0.01 seconds. Default is zero.<br><br>Examples:<br>^B01^ACWI0^C initialize stopwatch 0, use all default values.<br>^B01^ACWI1 1 0 270000^Cinitialize stopwatch 1, enabled, count up, starting from time 00:45:00.00 |
| | "WS" | Start a stopwatch. This will start to count the time from the current time value.<br><br>The stopwatch should have been initialized and enabled.<br><br>Format: INDEX where:<br>● INDEX '0' to '9', indicate which of the 10 stopwatches to be initialized.<br><br>Examples:<br>^B01^ACWS0^C start stopwatch 0 |
| | "WT" | Stop a stopwatch. This will stop the time counting, the current time value |

| | | |
|---|---|---|
| | | will not be changed until an initialize command or start command is issued. This is useful to control other devices power for example.<br><br>The stopwatch should have been initialized and enabled.<br><br>Format: INDEX where:<br>● INDEX ‘0’ to ‘9’, indicate which of the 10 stopwatches to be initialized.<br><br>Examples:<br>^B01^ACWT0^C stop stopwatch 0 |
| | “QR” | Quick restart. Using this command to quickly restart running when variables are updated and display need to restart. |
| | “OU” | External Port Output Control. Using this command to control the external port in “J15” on the WZP controller. This is<br><br> <br><br>J15 contain four general purpose output ports: GP0-GP3. But currently only GP0 can be used on WZPMAIN03-D control board.<br><br>J15 External Output Ports Definition:<br><br> J15 Control<br>1<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>10<br>GP3<br>GP0<br>GP2<br>GND<br>GP1 |

| | |
|---|---|
| | GP0 default output level 5.0V DC. When it output low level, it can sink 20mA current. And when it output high level, it can source 5mA current.<br><br>Format: PV[PV…] where:<br>● P one digit ‘0’ to ‘3’, indicate which port will change output.<br>● V one digit ‘0’ or ‘1’, indicate low or high level.<br><br>Example:<br>^B01^ACOU0110^C output high level to external port GP0, and output low level to GP1. |
| “WF” | Write configurations to the sign.<br><br>Format: KEY=VALUE<CR><LF> [ KEY=VALUE<CR><LF> KEY=VALUE<CR><LF> ] where:<br><br><CR> is ASCII code $0D.<br><LF> is ASCII code $0A. |

| KEY | VALUE |
|---|---|
| brightness | From 0 to 255 represent the lowest and highest brightness. |
| brightmode | 0 represent the brightness control by the value of brightness,<br>1 represent the brightness control by auto |
| temperaturechar | The value represent the temperature char. |

Examples:
^B01^ACWFbrightness=200<CR><LF>^C set brightness to 200.
Hex codes are:
5E 42 30 31 5E 41 43 57 46 62 72 69 67 68 74 6E 65 73 73 3D 32 30 30 0D 0A 5E 43
^B01^ACWFbrightmode=1<CR><LF>^C set brightness control by auto
Hex codes are
5E 42 30 31 5E 41 43 43 46 62 72 69 67 68 74 6D 6F 64 65 3D 31 0D 0A 5E 43

### Special Function Codes – Read request with response.

| **Code** | **Description** |
|---|---|
| “RT” | Read time of day.<br>**Read time response format:**<br>HHMMSS where HH is the hours (in 24-hour mode, decimal) and MM is |

| | |
|---|---|
| | the minutes (decimal), and SS is the seconds (decimal). |
| “RM” | Read memory status.<br>**Read memory response format:**<br>UUUU-FFFF:FTSSSS[,FTSSSS,…] where:<br>● UUUU is the amount of used memory overall in bytes. 1-8 ASCII hexadecimal digits “0” to “7FFFFFFF”.<br>● FFFF is the amount of free space left in sign, in bytes. 1-8 ASCII hexadecimal digits “0” to “7FFFFFFF”.<br>● The following is repeated for each file that is allocated:<br>○ F = File label (as described in set memory)<br>○ T = File type (as described in set memory)<br>○ SSSS = Allocation size. 1-8 ASCII hexadecimal digits. “1” to “7FFFFFFF”.<br>○ Followed by a “,” except the last one. |
| “RV” | Read sign size and versions.<br>**Read version response format:**<br>WWWWHHHHCVV where:<br>● WWWW = Width of sign, in dots, 4 hexadecimal ASCII digits.<br>● HHHH = Height of sign, in dots, 4 hexadecimal ASCII digits.<br>● C = Color type of sign as follows:<br>○ “1” = Single color sign.<br>○ “2” = Bi-color (Red,Green,Amber) sign.<br>○ “3” = RGB sign.<br>● VV = Protocol version, 2 decimal ASCII digits. Currently responds “02” for this protocol. |
| “RD” | Read date.<br>**Read date response format:**<br>MMDDYYYYX where:<br>● MM = Month (decimal) , “01” to “12”<br>● DD = Day (decimal) , “01” to “31”<br>● YYYY = Year (decimal) – 4 digits., “2000” to “2099”<br>● X = Day of the week, where: “0”=Sunday to “6”=Saturday. |
| “RA” | Read sign address. The response format is “AA” where AA is 2 ASCII hexadecimal digits representing the address (“01” to “FF”).<br>Note: This command is useful for “pinging” the display or when used with the address of “00” to find out the address of a sign. |
| “FR” | Read file content.<br><br>Format: NAME.EXT where:<br>NAME is 1-8 characters file name, and EXT is 1-3 characters file extension. Both NAME and EXT are case sensitive. The EXT can be the following values:<br><br>● “txt” = Text file |

| | | |
|---|---|---|
| | | ● “var” = Variable file<br>● “sh” = Script file<br>● “bmp” = *.bmp image<br>● “gif” = *.gif image or animation<br>● “png” = *.png image<br><br>The response data is 1-N characters file content, encoded in Base64. |
| | “FL” | List files.<br>Format: NAME.EXT where NAME is 1-8 characters file name and EXT is 1-3 characters file extension. Both NAME and EXT are case sensitive.<br><br>Character “*” and “?” can be used for wide range matching. For example, “FL*.*” will list all files, and “FLA.txt” will tell you some information about the text file A.<br><br>The response format is =FILE; FILE; FILE; …; FILE<br><br>Files are separated by semicolon “;”.<br><br>Each file is in format NAME.EXT, SIZE, TIME, ATTR where:<br><br>NAME is 1-8 characters file name.<br><br>EXT is 1-3 characters file extension.<br><br>SIZE is 1-8 ASCII hexadecimal digits “1” to “7FFFFFFF” representing the file size.<br><br>TIME is in format MM-DD-YYYY HH:MM:SS, representing the last modified time.<br><br>ATTR is 1-N characters string representing the file attributes. This can contain any combinations of the following characters:<br>● “S” = System file<br>● “R” = Read only<br>● “H” = Hidden |
| | “RS” | Read Sensors Values.<br>**Read sensors response format:**<br>KEY:VALUE, [ KEY:VALUE, … ] where:<br>KEY may be:<br>● “temperature”<br>● “humidity” |

| | | |
|---|---|---|
| | | ● “dewpoint”<br>VALUE is the sensor’s currently measured value. For temperature and dewpoint, the temperatures are in Celsius only. Humidity is a relative value between 0.1 percent and 100.0 percent. |
| | BF | Read Fan speed<br>**Read sensors response format:**<br>VALUE,VALUE<br>VALUE is the fan speed’s currently measured value.It need to convert to Hex Value.The Fan Speed=VALUE*100 |
| | BT | Read Box Temperature<br>**Read sensors response format:**<br>VALUE<br>VALUE is the Box Temperature’s currently measured value.It need to convert to Hex Value.If the Hightest Bit is 1,the Temperature is negative.If the value is 0x80.The Temperature .The Temperature value is lowest 7 bits. |
| | BV | Read Box Voltage<br>VALUE,VALUE<br>VALUE is the Box Voltage’s currently measured value.It need to covert to the Hex Value.The Box Voltage=VALUE/10 |
| | VR | Read .bin File From Scan Board.<br>If file name is scbdef.bin. it will be save as system file.<br>System file will not be del.<br>Examples:<br>^B01^ACVRscb1.bin^C<br>It will save scan board setting to scb1.bin<br>And return by Com as base64 code. |
| | VS | Write .bin File to Scan Board and save<br>Bin File must had save in WZP.<br>Examples:<br>^B01^ACVSscb1.bin^C<br>Write scb1.bin to Scan Board.<br>5E 42 30 31 5E 41 43 56 53 73 63 62 31 2e 62 69 6e 5E 43 |
| | VV | Read FPGA version<br>**FPGA Version response**<br>Examples:<br>^B01^ACVV^C<br>^B01^ACVV=PLD31PRO23^D0507^C<br>Version is PLD31PRO23 |
| | | |

## *6.4 Write to Text File without Restart – Code “D”*

If you want to update a text file but do not restart running, you can use command “D” instead of command “A” to write to text file.

When the sign has received a text file using the command “D”, the sign will not restart (however the command “A” will), but keep running as if nothing has happen. If you need to restart running after all, you can issue a quick restart command “CQR”.

Before writing to a text file, the file must be setup using the special function command to allocate memory for the file.

## *6.5 Write to Script File – Code “E”*

All are same with command “D”, except that command “E” is use to write to script file.

Please refer to Appendix B for script file format.

## *6.6 Advance Open file-Code “F”*

If you want to send the file to sign or read the file from sign, you must use this command first; you can get the handle of the file (you want to read or write) from the response data. The handle can be identified in future operations, it has 8 hexadecimal characters.

If you want to open the file use advance open file command (‘F’), you must specify the communication id (it should be a random integer)、filename and open mode. If the open mode is Write (W), you must specify the file size and the file crc32 value, if you want to set the time of file, you need specify the file time at this communication.

How to specify the configure information at this communication? You need comply with the format of the following:

Format:  KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This must specify.<br>This is a communication ID, the value is 1-8 hexadecimal characters.<br>This should be a random integer.<br>Each command has unique ID, resend the same command use the same ID. |
| name | This must specify.<br>The value is 1-12 lengths characters. It can not contain space character.<br>The VALUE is the name of the file you want to read or write.<br>If you want to read the information of the WZP sign, the VALUE is $INF$. |

| | |
|---|---|
| | If you want to read the WZP sign file list, the VALUE is $ROOT$.<br>If you want to configure the WZP sign, the VALUE is $CFG$.<br>If you want to change the baud rate of the WZP sign, the VALUE is $CMD$. |
| mode | This must specify.<br>The value is 1 character; it is 'R' or 'W'.<br>If you want to read the file from the sign, the value is 'R'.<br>If you want to write the file to the sign, the value is 'W'. |
| size | This must specify if you want to write the file to the sign, otherwise ignore.<br>The value is 1-8 lengths hexadecimal characters.<br>This is the size of the file content you want to send. |
| crc | This must specify if you want to write the file to the WZP sign, otherwise ignore.<br>The value is 1-8 lengths hexadecimal characters.<br>The value is crc32 calculate of the file content. |
| time | This need specify if you want to write the file to WZP and set the time of the file, otherwise ignore.<br>The value must contain this format: MM-dd-yyyy,HH:mm:ss where<br>● MM = Month (decimal), "01" to "12"<br>● dd = Day (decimal), "01" to "31"<br>● yyyy= Year (decimal) – 4 digits, "2000" to "2099".<br>● HH = Hour (decimal), "00" to "23"<br>● mm = Minute (decimal), "00" to "59"<br>● ss = Second (decimal), "00" to "59". |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

（1）^B01^AFid=17C59ED3 mode=R name=B.txt ^D354B^C

(2) ^B01^AFid=606525EC mode=R name=A.txt ^D2F84^C

2、Write the file to the WZP sign(here the checksum mode is Crc16)

(1) ^B01^AFid=7FF7047C mode=W name=A.txt size=00000005 crc=F7D18982 time=06-14-2011,10:30:10 ^D72C6^ C

(2) ^B01^AFid=72CE2CB5 mode=W name=B.txt size=00000005 crc=F7D18982 time=06-14-2011, 10:47:36 ^D3E5D^C

The commands respond with data as needed. This response is sent before "General Response" and is as follows:

| **Standard Response Packet** |
|---|

| <STX> **^B** | Sign Address | <SOH> **^A** | Command Code "F" | Response Data | <EOT> **^D** | Checksum | <ETX> **^C** |
|---|---|---|---|---|---|---|---|

| **Item** | **Description** |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | Sign address of the sign that is responding. 2 ASCII hexadecimal digits. |
| <SOH> | Start of command. ^A |
| Command Code | A single ASCII character representing the command response code. In this case, "F" |
| Response Data | 0 to N characters of special function response data. The data contain the communication id, error, handle. How to get the communication id、error、 handle from the data, please refer "Responses Data Format" for detail. |
| <EOT> | End of text. ^D |
| Checksum | 4 hex digits represent a hex word value from "0000" to "FFFF", if the checksum mode is Sum, which is the SUM of bytes from <SOH> to <EOT> (inclusive, byte by byte), else if the checksum mode is Crc16,which is the crc16 calculate of bytes from <SOH> to <EOT>(inclusive, byte by byte). |
| <ETX> | End of transmission. ^C |

**Response Data Format**

Format: KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This is Communication ID.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to the communication id you just send, If not the communication need resend the same command. |
| error | This is Communication error.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to zero, If not it means that the sign open the file is Fail, you should end of read file or write file. |
| handle | This is file handle.<br>The value exists if the error equal to zero, it is 8 hexadecimal characters.<br>The file handle is very important. If you want to read file from the sign、write the file to the sign and close the file, you must use this handle. |
| size | This is the size of the file you want to read.<br>The value exists if the error equal to error and the openmode is 'W', it is 8 hexadecimal characters. |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

(1) ^B01^AFid=17c59ed3 error=00000000 handle=1945882b size=00000005^DD30A^C.

(2) ^B01^AFid=606525ec error=00000000 handle=188d408f size=00000005^DF689^C.

2、Write the file to the WZP sign(here the checksum mode is Crc16)

(1) ^B01^AFid=7ff7047c error=00000000 handle=25a250f 6 ^D962A^C

(2) ^B01^AFid=72ce2cb5 error=00000000 handle=34641dba ^DDDE2^C

## *6.7 Advance Close file-Code "G"*

When you read the file from the sign or write the file to the sign is completed, you must close the file; otherwise the file content is not saved and the memory is not released.

If you want to close the file, the communication id (it should be a random integer) must be specified, also the handle of the file must be required; the handle is the one get from the open file operation.

How to specify the configure information at this communication? You need comply with the format of the following:

Format: KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This must specify.<br>This is a communication ID, the value is 1-8 hexadecimal characters.<br>This should be a random integer.<br>Each command has unique ID, resend the same command use the same ID. |
| handle | This must specify<br>This is the handle of the file you want to read or write, The value is 1-8 hexadecimal characters.<br>This must be the handle that gets from the open file operation. How to get the handle? Please refer "6.6 advance open file-Code "F" for detail. |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

(1) ^B01^AGid=2597E4CE handle=1945882B ^DDCC4^C

(2) ^B01^AGid=07333C2D handle=188D408F ^DA37B^C

2、Write the file to the WZP sign(here the checksum mode is Crc16)

(1) **^**B01^AGid=78D7C804 handle=25A250F6 ^D10F1^C

(2) ^B01^AGid=2E8DC569 handle=34641DBA ^D210A^C

The commands respond with data as needed. This response is sent before “General Response” and is as follows:

**Standard Response Packet**

| <STX> **^B** | Sign Address | <SOH> **^A** | Command Code “G” | Response Data | <EOT> **^D** | Checksum | <ETX> **^C** |
|---|---|---|---|---|---|---|---|

| **Item** | **Description** |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | Sign address of the sign that is responding. 2 ASCII hexadecimal digits. |
| <SOH> | Start of command. ^A |
| Command Code | A single ASCII character representing the command response code. In this case, “G” |
| Response Data | 0 to N characters of special function response data. The data contain the communication id, error, handle. How to get the communication id、error from the data, please refer “Responses Data Format” for detail. |
| <EOT> | End of text. ^D |
| Checksum | 4 hex digits represent a hex word value from “0000” to “FFFF”, if the checksum mode is Sum, which is the SUM of bytes from <SOH> to <EOT> (inclusive, byte by byte), else if the checksum mode is Crc16,which is the crc16 calculate of bytes from <SOH> to <EOT>(inclusive, byte by byte). |
| <ETX> | End of transmission. ^C |

**Response Data Format**

Format: KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This is Communication ID.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to the communication id you just send, If not the communication need resend the same command. |
| error | This is Communication error.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to zero, If not it means that the sign close the file is Fail, you should end of read file or write file. |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

(1) ^B01^AGid=2597e4ce error=00000000 ^D8940^C

(2) ^B01^AGid=07333c2d error=00000000 ^D9189^C

2、Write the file to the WZP sign(here the checksum mode is Crc16)

(1) ^B01^AGid=78d7c804 error=00000000 ^D53C1^C

(2) ^B01^AGid=2e8dc569 error=00000000 ^D5F95^C

## *6.8 Advance Read file-Code “H”*

If you want to read the file from the sign, you should open file first. How to open file from the sign, please refer “6.6 advance open file-Code “F” “for detail. Then you could read the file content from the sign use this command.

It should not read too many bytes at one communication, so if the content of the file is too long, it should be read many times. At one communication, the length of the file content should not be over 512 before encode. So it will communication many times use advance read file command (‘H’) until the end of the file.

If the content of the file has control characters (0x00 to 0x1f); the content of the file should be encode to base64 characters at the communication.

If you want to read the file content use advance read file command (‘H’), the communication id must be specified, the handle of the file must be required; also the size (before encode) of block you want to read at this communication and the position of the file must be specified; If you want to encode the content at the response data, the encode format should be specified.

How to specify the configure information at this communication? You need comply with the format of the following:

Format: KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This must specify.<br>This is a communication ID, the value is 1-8 hexadecimal characters.<br>This should be a random integer.<br>Each command has unique ID, resend the same command use the same ID. |
| handle | This must specify<br>This is the handle of the file you want to read, the value is 1-8 hexadecimal characters.<br>This must be the handle that gets from the open file operation. How to get the handle? Please refer “6.6 advance open file-Code”F” for detail. |
| pos | This must specify.<br>This is the position of the file, 0 is begin of the file. The value is 1-8 hexadecimal characters.<br>It is a hexadecimal number. |

| | |
|---|---|
| size | This must specify<br>This is the size of the block you want to read, the value is 1-8 hexadecimal characters.<br>The value should be 1 to 512, it is the size of block before encode. It is a hexadecimal number. |
| format | If you want to encode file content that read at this communication to base64 characters, this must be specified.<br>The value must be “base64”, if it is not “base64”, the communication error will not equal to zero. |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

(1) ^B01^AHid=45B2F18B handle=1945882B pos=00000000 size=00000005 ^DD1EA^C

(2) ^B01^AHid=3C3FCB5D handle=188D408F pos=00000000 size=00000005 format=base64 ^DD80F^C

The commands respond with data as needed. This response is sent before “General Response” and is as follows:

**Standard Response Packet**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| <STX><br>**^B** | Sign Address | <SOH><br>**^A** | Command Code<br>“H” | Response Data | <EOT><br>**^D** | Checksum | <ETX><br>**^C** |

| **Item** | **Description** |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | Sign address of the sign that is responding. 2 ASCII hexadecimal digits. |
| <SOH> | Start of command. ^A |
| Command Code | A single ASCII character representing the command response code. In this case, “H” |
| Response Data | 0 to N characters of special function response data. The data contain the communication id, error, handle. How to get the communication id、error、size、content、base64content from the data, please refer “Responses Data Format” for detail. |
| <EOT> | End of text. ^D |
| Checksum | 4 hex digits represent a hex word value from “0000” to “FFFF”, if the checksum mode is Sum, which is the SUM of bytes from <SOH> to <EOT> (inclusive, byte by byte), else if the checksum mode is Crc16,which is the crc16 calculate of bytes from <SOH> to <EOT>(inclusive, byte by byte). |
| <ETX> | End of transmission. ^C |

**Response Data Format**

Format: KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20), where if the KEY is "content" or "base64content", the <CR> doesn't exist.

| KEY | VALUE |
|---|---|
| id | This is Communication ID.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to the communication id you just send, If not the communication need resend the same command. |
| error | This is Communication error.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to zero, If not it means that the sign read the file is Fail, you should end of read file<br>If the encodeformat is not "base64" (if you set the encodeformat); also if the pos less and the size less zero or is not specified, also if the handle is not get from the open file or not specified, the error will not equal to zero. |
| size | This is the really size of the content that read.<br>The value must exist if the error equal to zero, it is a hexadecimal characters.<br>It is a hexadecimal number.<br>It is the size of block before encode, it equal to or less than the size you want to read. |
| content | This must exist if you don't set the encodeformat and the error equal to zero.<br>This is the block that you want to read, the value is a lot of bytes, and it may be contain the control characters (0x00 to 0x1f).<br>This must be the end of the response data, if the KEY is "content", After the '=', the rest of the data is the block that you want to read and don't contain the <CR> (space characters). |
| base64content | This must exist if you set the encodeformat and the error equal to zero.<br>The value is a lot of characters, and it is base64 characters.<br>This is the block that has been encoded. If you want to get the content, you must decode it from base64 characters.<br>This must be the end of the response data, if the KEY IS "base64content", After the '=', the rest of the data is the block that you want to read and don't contain the <CR> (space characters). |

Example:

1、Read the file from the WZP sign (here the checksum mode is Crc16)

（1）^B01^AHid=45b2f18b error=00000000 size=00000005 content=Hello^DE1F0^C

(2)^B01^AHid=3c3fcb5d error=00000000 size=00000005 base64content=SGVsbG8=^DD961^C

You can read the sign information or the file list from the sign; the operation is the same as read file from the sign, you just need set the file name as $INF$ or $ROOT$, you can refer "6.6 advance write file-code" for detail. You can get the sign information from the content directly; However get the file list from the content, it need to handle. How to deal with this content; please comply with the format of the following.

Format:  KEY=VALUE<CR>KEY=VALUE<CR> where:

<CR>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| filecount | The value is 8 hexadecimal characters, it is a hexadecimal number.<br>It is the count of the file list. |
| diskspace | The value is 8 hexadecimal characters, it is a hexadecimal number.<br>It is the size of the sign disk |
| diskfree | The value is 8 hexadecimal characters, it is a hexadecimal number.<br>It is the size of the free of the sign disk. |
| fileinfo | The value is a lot of bytes.<br>The count of fileinfo is filecount, if the value of filecount is 5, and then there are 5 fileinfo.<br>It contains the name of the file, the size of the file and the filetime by orders. The format of the time is "MM-DD-YYYY,hh:mm:ss". |

### *6.9 Advance Write file-Code "I"*

If you want to write the file to the sign, you should open file and get the handle first. How to open file from the sign, please refer "6.6 advance open file-Code "F" " for detail, then you could send the file content to the sign use this command.

It should not send too many bytes at one communication, so if the content of the file is too long, it should be sent many times. At one communication, the length of the file content should not be over 512 before encode. So it will communication many times use advance write file command ('I') until the end of the file.

If the content of the file has control characters (0x00 to 0x04 or "^A"、"^B" 、"^C" 、"^D"); the content of the file must be encode to base64 characters at the communication.

If you want to send the file content use advance write file command ('I"), the communication id must be specified, the handle of the file must be required, and the position of the file must be specified. If you want to encode the content that you want to send, you need encode it and the KEY is "base64content", else keep the content unchanged and the KEY is "content". If you specify the "content" or "base64content", it must be the end of the block.

How to specify the configure information at this communication? You need comply with the format of the following:

Format:  KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20), where if the KEY is "content" or "base64content", the <CR> don't exist.

| KEY | VALUE |
|---|---|
| id | This must specify. |

| | |
|---|---|
| | This is a communication ID, the value is 1-8 hexadecimal characters.<br>This should be a random integer.<br>Each command has unique ID, resend the same command use the same ID. |
| handle | This must specify<br>This is the handle of the file you want to write, the value is 1-8 hexadecimal characters.<br>This must be the handle that gets from the open file operation. How to get the handle? Please refer “6.6 advance open file-Code”F” for detail. |
| pos | This must specify.<br>This is the position of the file, 0 is begin of the file. The value is 1-8 hexadecimal characters.<br>It is a hexadecimal number. |
| content | This must specify if you don’t encode the content to base64 characters at the communication.<br>The value is a lot of bytes; it is the content of the file.<br>The content must be the end of the data block, after the value the <CR> will not exist |
| base64content | This must specify if you encode the content to base characters at the communication.<br>The value is a lot of base64characters, the content of the file must encode to base64 character at the communication, and it can not have non base64 characters.<br>It must be the end of the data block, after the value the <CR> will not exist. |

Example:

1、Write the file to the WZP sign(here the checksum mode is Crc16)

(1) ^B01^AIid=18F2D4C2 handle=25A250F6 pos=00000000 content=Hello^DD05C^C

(2)^B01^AIid=4EA8D226 handle=34641DBA pos=00000000 base64content=SGVsbG8=^DA235^C

The commands respond with data as needed. This response is sent before “General Response” and is as follows:

**Standard Response Packet**

| <STX><br>**^B** | Sign Address | <SOH><br>**^A** | Command Code<br>“I” | Response Data | <EOT><br>**^D** | Checksum | <ETX><br>**^C** |
|---|---|---|---|---|---|---|---|

| **Item** | **Description** |
|---|---|
| <STX> | Start of transmission. ^B |
| Sign Address | Sign address of the sign that is responding. 2 ASCII hexadecimal digits. |

| | |
|---|---|
| <SOH> | Start of command. ^A |
| Command Code | A single ASCII character representing the command response code. In this case, “I” |
| Response Data | 0 to N characters of special function response data. The data contain the communication id, error, handle. How to get the communication id、error from the data, please refer “Responses Data Format” for detail. |
| <EOT> | End of text. ^D |
| Checksum | 4 hex digits represent a hex word value from “0000” to “FFFF”, if the checksum mode is Sum, which is the SUM of bytes from <SOH> to <EOT> (inclusive, byte by byte), else if the checksum mode is Crc16,which is the crc16 calculate of bytes from <SOH> to <EOT>(inclusive, byte by byte). |
| <ETX> | End of transmission. ^C |

**Response Data Format**

Format:　KEY=VALUE<Sp> [, KEY=VALUE<Sp>] […] where:

<Sp>is space character (ASCII code 0X20)

| KEY | VALUE |
|---|---|
| id | This is Communication ID.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to the communication id you just send, If not the communication need resend the same command. |
| error | This is Communication error.<br>The value must exist, it is 8 hexadecimal characters.<br>The value must equal to zero, If not it means that the sign write the file is Fail, you should end of write file.<br>If the handle of the file not get from advance from file; if the position of the file less than zero; if the size of content equal to zero; if the size of the content plus the position greater than the size of the file, the error will not equal to zero. |

Example:

1、 Write the file to the WZP sign(here the checksum mode isCrc16)

(1) ^B01^AIid=18f2d4c2 error=0000000 ^DE083^C

(2) ^B01^AIid=4ea8d226 error=00000000 ^D509F^C

You can change the baud rate of the sign; the operation is the same as write the file to the sign. How to change the baud rate of the sign? First it need set the baud rate to the new value, and then use the new baud rate to confirm it. How to set the baud rate to the new value, first you use advance open file command (‘F’) to open file, here you need set the filename as “CMD”, and then you send the content use advance write command (‘I’), here the content is “baudrate=VALUE”, the VALUE is the new baud rate you want to set, it is a decimal number, finally you must close the file use advance close file command. How to confirm the new baud rate, you should use the new baud rate to communication, first you use advance open file command (‘F’) to open file, here you need set the filename as $CMD$, and then you send the content use advance write command (‘I’), here the content is “confirm=”, finally you must close the file use advance close file command.

You can configure the sign as write the file to the sign. First you use advance open file command (‘F’) to open file, here you need set the filename as $CFG$, and then you send the content use advance write command (‘I’), here the content is the configure information, finally you must close the file use advance close file command. The configure information format please refer as follow:

Format: KEY=VALUE<CR><LF> [ KEY=VALUE<CR><LF>    KEY=VALUE<CR><LF> ] where:

<CR> is ASCII code $0D.
<LF> is ASCII code $0A.

| KEY | VALUE |
|---|---|
| brightness | From 0 to 255 represent the lowest and highest brightness. |
| poweroff | If the value is 0, it means power up the sign, else if the value is 1,it means power down the sign. |

# 7 Multiple Line Sign Behavior

This section identifies the behavior of the signs when using multiple line displays and the protocol.

Looking at the mode when writing text files, the positions are:

- Middle
- Top
- Bottom
- Fill
- Left
- Right

Normally a single line will behave as follows:

- All characters line up at the bottom of the sign and work their way up for as many dots as the font supports:
- If a sign receives a font that is larger than the sign can display, then the sign will "size down" or reduce the font size. For example, on a one-line sign, SS16 characters would be replaced by SS7 characters.
- If a character font is not specified, then SS7 will be used.
- If Top, Bottom, or Fill positions are received Middle is used.
- The centerline is never placed further left than 8 pixels from the leftmost pixel of the sign.
- The centerline is never placed further right than 8 pixels from the rightmost pixel of the sign.

A two-line sign behaves as follows:

- The Top position is defined as the top 7 dots of the sign. The Top position functions in the same manner as a one-line sign.
- The Bottom position is defined as the bottom 7 dots of the sign. The Bottom position functions in the same manner as a one-line sign.
- The Middle position is treated as though it was a 1 line sign 16 dots high. Each line of text presented on this line is pre-scanned to determine the largest piece of text to be displayed. For example, if a line of SS7 text has just a single SF10 character, the line is viewed as a 10-high line. This means that 10-high characters will be displayed with 3 dots above and below the characters (3+10+3 = 16).
- Fill position: On a two-line sign, the Fill position indicates that you wish to use no more than 7-high characters and that you wish to fit as much text on the screen as you can. When using the Fill position, the sign sees itself as having two lines of 7-high characters and no means of displaying characters larger than 7-high. Also, if the last piece of a message is just one line, then the sign will center this line on the screen. If the sign is operating on the top row, then the bottom of that row is assumed to be the 7th row of dots. All text is started from there and worked up: 7-high characters will use rows 1 to 7. If the sign is operating on the bottom row, then the sign works its way up from row 16: 7-high characters will use rows 10 to 16.

Three or more line signs behave as follows:

- The Top and Bottom positions work in tandem with each other. There is an imaginary line between the top and bottom half of the sign. This is called the "centerline". The centerline divides what is used for the Top from what is used for the Bottom positions. The location of the centerline is usually established by the first Top command the sign receives, and the rest of the space is used for the Bottom position. If a Bottom position command comes first, then the centerline is placed at its highest position — row 8, allowing for a single line of 7-high characters on the Top position. Once a centerline has been established, it remains fixed until a Fill or Middle position command is received. The centerline can not be changed with another Top or Bottom position command. However, if the first command specifies a Top, and not a Bottom, position, then the centerline's position is determined by the amount of text following the position command. For example:

    - If one 7-high line of text is received (following a Top position command), then the centerline will be fixed at row 8.
    - If one line of 10-high characters is received (following a Top position command), then the centerline will be fixed at row 11.
  - The centerline is never placed higher than 8 rows from the top of the sign.
  - The centerline is never placed lower than 8 rows from the bottom of the sign.
- The Left and Right positions work in tandem with each other, much like the Top and Bottom positions for multi-line signs. An imaginary line (called the "centerline") divides what is used for the Left from what is used for the Right positions. The location of the centerline is usually established by the first Left command the sign receives, and the rest of the space is used for the Right position. The placement of this centerline will be determined by a new line. If no new line is given, the text will continue up to the rightmost 8 pixels, which will be reserved for the Right position. If a Right position command comes first, then the centerline is placed at the leftmost position — column 8, allowing for a single character in the Left position. Once a centerline has been established, it remains fixed until a Fill or Middle position has been received.
  - The centerline is never placed further left than 8 pixels from the leftmost pixel of the sign.
  - The centerline is never placed further right than 8 pixels from the rightmost pixel of the sign.
- The Middle position is treated as though it were a one-line sign with as many rows as the sign is tall. Each line of text on the sign is checked to determine the largest piece of text to be displayed. The line of text is then vertically centered based on that largest piece of text. For example, if you have a line of text which has mostly 7-high characters, but has one 10-high character, then this line is considered a 10-high line. Assuming that this is a 24-row sign, this would leave 14 extra rows so there would be 7 blank rows on top and 7 on the bottom (7+10+7=24). All text is then lined up on this new virtual bottom (the 21st line) and treated the same as in a one-line sign.
- The Fill position indicates that you wish to fit as much text on the screen as you can. You can select characters larger than 7-high. The sign will start from top of the screen working down. If you select a 15-high character set, then the sign will fit as many 15 row lines of text on the screen as possible. As soon as the sign detects that the next line will not fit, the sign will stop creating the current page and display it. The next page will begin with the line that did not fit. If the text does not use up the entire display, then the sign will center the text vertically, splitting the blank space between the top and the bottom.

# 8 Protocol Examples

## *8.1 Send a message to all signs using the default text file “A”.*

The following example will display “HELLO” to all attached signs.

| <STX>00<SOH>AAHELLO<ETX> | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “00” | Sign address of 00 – to all signs. |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “A” | Write text command code |
| File Label | “A” | Use file “A” – which is the default and is already allocated. |
| Message | “HELLO” | Actual text to be displayed. |
| <ETX> | ^C or $03 | End of Transmission. |

## *8.2 Send a scrolling message to all signs.*

The following example will display “HELLO” on the bottom line of the sign scrolling from right to left.

| <STX>00<SOH>AA<BEL>BSHELLO<ETX> | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “00” | Sign address of 00 – to all signs. |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “A” | Write text command code |
| File Label | “A” | Use file “A” – which is the default and is already allocated. |
| <BEL> | ^G or $07 | Start of mode field |
| Position | “B” | Bottom of sign |
| Mode Code | “S” | Scrolling from right to left |
| Message | “HELLO” | Actual text to be displayed. |
| <ETX> | ^C or $03 | End of Transmission. |

## *8.3 Setup and send a text file containing a variable file.*

This example will show you the transmission sequences to setup and update a text file containing a variable file.
It will scroll from right to left the message “TEMP = nnnn” where nnnn is stored in a variable file that will be updated by itself to change the displayed number.

### 8.3.1 Step 1 – Setup variable memory

We don’t need to setup the text file area – we will use label “A” for the text file – which is automatically setup.

| <STX>00<SOH>CSMXV0010<ETX> | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “00” | Sign address of 00 – to all signs. |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “C” | Write special function command code |
| Special Function Code | “SM” | Set memory |
| Special Function File Label | “X” | File label to set memory for – “X” |
| Special Function File Type | “V” | File type is variable. |
| Special Function File Size | “0010” | Allocating 16 bytes (“0010” is in hexadecimal”) |
| <ETX> | ^C or $03 | End of Transmission. |

### 8.3.2 Step 2 – Setup text file to show message plus variable file

This will write to the text file “A” and show it. It includes embedding the variable file “X” that was just created. Since “X” is now empty, the display will scroll “TEMP =” and nothing else until we update the variable file.

| <STX>00<SOH>AA<BEL>BSHELLO<SO>X<ETX> | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “00” | Sign address of 00 – to all signs. |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “A” | Write text command code |
| File Label | “A” | Use file “A” – which is the default and is already allocated. |
| <BEL> | ^G or $07 | Start of mode field |
| Position | “B” | Bottom of sign |
| Mode Code | “S” | Scrolling from right to left |
| Message | “TEMP =” | Actual text to be displayed. |
| <SO> | ^N or $0E | Embed variable file code |
| File Label | “X” | Embedded file label – “X” |
| <ETX> | ^C or $03 | End of Transmission. |

### 8.3.3 Step 3 – Update the variable file with data

Now you will update only the variable file. This will then show the data when the message scrolls on again. Notice how the sign did not blank or hesitate. It will scroll “TEMP =1234”

| <STX>00<SOH>BX1234<ETX> | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “00” | Sign address of 00 – to all signs. |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “B” | Write variable command code |
| File Label | “X” | Use file “X” – which is the default and is already allocated. |
| Message | “1234” | Actual text to be displayed. |
| <ETX> | ^C or $03 | End of Transmission. |

## *8.4 Advanced usage about text alignment*

| ^B01^AAA^U3Apple^T0^T1100^T31.23^MMeat^T0^T110^T312.345^M^C | | |
|---|---|---|
| **Code Name** | **Value** | **Description** |
| <STX> | ^B or $02 | Start of Transmission |
| Sign Address | “01” | Sign address of 01 |
| <SOH> | ^A or $01 | Start of command. |
| Command Code | “A” | Write text command code |
| File Label | “A” | Use file “A” – which is the default and is already allocated. |
| Messages | “^U3” | Set text alignment to “left, middle” |
| | “Apple^T0^T1100^T31.23^M” | The first line display: Apple 100 1.23 |
| | “Meat^T0^T110^T312.345^M” | The next line display: Meat 10 12.345 |
| <ETX> | ^C or $03 | End of Transmission. |

# Appendix A: Standard Color Names

| | | | | | |
|---|---|---|---|---|---|
| aqua | 🟦 | hotpink | 🟥 | paleturquoise | 🟦 |
| aquamarine | 🟩 | honeydew | ⬜ | palegreen | 🟩 |
| aliceblue | ⬜ | indigo | 🟪 | palegoldenrod | 🟨 |
| azure | ⬜ | ivory | ⬜ | plum | 🟪 |
| antiquewhite | ⬜ | indianred | 🟥 | powderblue | 🟦 |
| black | ⬛ | khaki | 🟨 | papayawhip | ⬜ |
| blue | 🟦 | lime | 🟩 | peru | 🟫 |
| blueviolet | 🟪 | limegreen | 🟩 | peachpuff | 🟧 |
| beige | ⬜ | lightgray | ⬜ | red | 🟥 |
| blanchedalmond | ⬜ | lightpink | 🟥 | royalblue | 🟦 |
| burlywood | 🟫 | lightsteelblue | 🟦 | rosybrown | 🟫 |
| bisque | ⬜ | lightslategray | ⬛ | silver | ⬜ |
| brown | 🟫 | lightskyblue | 🟦 | skyblue | 🟦 |
| cyan | 🟦 | lightblue | 🟦 | slateblue | 🟪 |
| cream | ⬜ | lightcyan | ⬜ | slategray | ⬛ |
| crimson | 🟥 | lightseagreen | 🟦 | steelblue | 🟦 |
| cornflowerblue | 🟦 | lightgreen | 🟩 | springgreen | 🟩 |
| cornsilk | ⬜ | lightgoldenrodyellow | ⬜ | seagreen | 🟩 |
| cadetblue | 🟦 | lightyellow | ⬜ | seashell | ⬜ |
| chartreuse | 🟩 | lightsalmon | 🟧 | sandybrown | 🟧 |
| chocolate | 🟧 | lightcoral | 🟥 | saddlebrown | 🟫 |
| coral | 🟧 | lavender | ⬜ | sienna | 🟫 |
| darkgray | ⬜ | lavenderblush | ⬜ | salmon | 🟧 |
| darkmagenta | 🟪 | lawngreen | 🟩 | snow | ⬜ |
| darkviolet | 🟪 | lemonchiffon | ⬜ | teal | 🟦 |
| darkorchid | 🟪 | linen | ⬜ | thistle | 🟪 |
| darkslateblue | 🟪 | magenta | 🟪 | turquoise | 🟦 |
| darkslategray | ⬛ | maroon | 🟥 | tan | 🟫 |
| darkblue | 🟦 | moneygreen | 🟩 | tomato | 🟥 |
| darkturquoise | 🟦 | medgray | ⬜ | violet | 🟪 |
| darkcyan | 🟦 | mediumvioletred | 🟥 | white | ⬜ |
| darkseagreen | 🟩 | mediumorchild | 🟪 | whitesmoke | ⬜ |
| darkgreen | 🟩 | mediumpurple | 🟪 | wheat | ⬜ |
| darkolivegreen | 🟩 | mediumslateblue | 🟪 | yellow | 🟨 |
| darkkhaki | 🟨 | mediumblue | 🟦 | yellowgreen | 🟩 |
| darkgoldenrod | 🟨 | mediumturquoise | 🟦 | | |
| darkorange | 🟧 | mediumaquamarine | 🟩 | | |
| darksalmon | 🟧 | mediumspringgreen | 🟩 | | |
| darkred | 🟥 | mediumseagreen | 🟩 | | |
| dimgray | ⬛ | midnightblue | 🟦 | | |
| deeppink | 🟥 | mintcream | ⬜ | | |
| deepskyblue | 🟦 | moccasin | ⬜ | | |
| dodgerblue | 🟦 | mistyrose | ⬜ | | |
| fuchsia | 🟪 | navy | 🟦 | | |
| forestgreen | 🟩 | navajowhite | ⬜ | | |
| floralwhite | ⬜ | olive | 🟫 | | |
| firebrick | 🟥 | olivedrad | 🟩 | | |
| gray | ⬛ | orchid | 🟪 | | |
| green | 🟩 | oldlace | ⬜ | | |
| greenyellow | 🟩 | orange | 🟧 | | |
| gainsboro | ⬜ | orangered | 🟥 | | |
| ghostwhite | ⬜ | purple | 🟪 | | |
| gold | 🟨 | pink | 🟥 | | |
| goldenrod | 🟨 | palevioletred | 🟥 | | |

# Appendix B: Script File Format

Script file are text file with many command lines. Lines are separated by <CR><LF> ($0D $0A) in text file. One text line contains only one command and its parameters.

**Comment**

Lines begin with “//” or between “/*” and “*/” are comment lines, these lines contain no commands and will be skipped by the command shell.

Examples:

**//this is a comment line**

```
/*these
are
comment
lines*/
```

**Label**

Lines begin with a word ending with “:” are labels, they are used to indicate the destination for branches.

Examples:

```
…
label1:
…
goto label1
goto label2
…
label2:
…
```

**Create Window**

Command format:

window NAME {LEFT,TOP,WIDTH,HEIGHT} run {FILE[, FILE][…]} [once]

Please use “once” to specify the window just run once and close automatically.

Examples:

```
window A {0,0,32,32} run {A.txt, B.txt} once
window B {0,32,96,32} run {C.txt} once
window C {0,32,128,32} run {D.txt}
```

**Sleep**

Command format:

sleep TIME [UNIT]

Examples:

```
sleep 10          //Sleep 10 seconds
sleep 5 min       //Sleep 5 minutes
sleep 8h          //Sleep 8 hours
```

**Close Window**

Command format:

close NAME[,NAME][…]

Do not close windows just run once, use waitfor command instead.

Examples:

```
close A
close B,C
```

**Wait for Window Running Terminated**

Command format:

wait NAME[,NAME][…]

Do not wait for windows not run once, use close command instead.

Examples:

```
wait A
wait B,C
```

**Unconditional Branch**

Command format:

goto LABEL

Examples:

```
…
label1:

…
goto label1
goto label2
…
label2:

…
```

**Set Window Border**

This command just affects the next window command.

Command format:

set border thickness=VALUE [padding=VALUE] [color=COLOR]

"COLOR" can be tree dec value like rgb(255, 128, 0), or a hex value like #FF8000, or a lowercase string of standard color name like red. Please refer to Appendix A to see what standard color names are available now.

Examples:

```
set border thickness=1 color=red
set border thickness=1 padding=0 color=rgb(255,0,0)
set border thickness=1 color=#FF0000
```

**Set Window Background**

This command just affects the next window command.

Command format:

```
set background color=[COLOR] [image=FILE[,] [tile]]
set background transparent
```

Examples:

```
set background transparent
set background color=#000080
set background color=rgb(0,0,128)
set background color=navy image=b160x16.png
set background image=b16x16.png, tile
```

**Set Window Zorder**

Zorders indicate the windows cascading order. Windows with zorder 1 always display on the top of screen, and may cover the windows with zorder 2. Zorder 0 is default value, means the window can be coverd by any window with non-zero zorder.

This command just affects the next window command.

Command format:

```
set zorder VALUE
```

Examples:

```
set zorder 1
set zorder 2
```

**Play Other Script File**

Command format:

```
play {FILE[,FILE[,…]]} [from {TIME} to {TIME} [and from {TIME} to {TIME}] […] ] [weekdays XXXXXXX]
```

Examples:

```
play {A.sh}
play {B.sh} from {24/8/2010 8:00} to {24/8/2010 20:00}
play {C.sh, D.sh} from {8:00} to {20:00} weekdays 0111110
play {E.sh} from {24/8/2010} to {24/8/2011} and from {8:00} to {20:00} weekdays 0111110
```