UNIVERSAL SERIAL INTERFACE

Coastal Environmental Systems
Application Note
ZENO_MANUAL_USI.DOC 4/21

UNIVERSAL SERIAL INTERFACE

Overview

The Universal Serial Interface lets you program your ZENO to communicate with serial sensors using ASCII communications – including multiple poll-response commands, unsolicited data from the sensors and built-in checksum verification.

The Universal Serial Interface largely retains the ZENO's existing menu-driven environment. Poll and response string formats are described using the C language **`printf`** formatting approach (asynchronous sensors, with no polling command, are also permitted). String, floating-point and long integer values can be read, logged, and transmitted; floating-point values can also be processed, with very limited processing is permitted for the long integer values.

SENSOR MENU

To set up a Universal Serial Sensor, you start with the standard ZENO Sensor Menu, and choose the Sensor Type Code to be 16 (Universal Serial). The following Sensor Menu will then appear.

```
Item  1: Sensor Type Code              16 (Universal Serial)
Item  2: Sensor Name                   TEST
Item  3: Sensor Input Channel          COM2
Item  6: Switched Power Code           0 (NO SWITCHED POWER)
Item  9: Switched Power Warmup Time    0
Item 11: Maximum Sensor Readings       0
Item 12: Sensor Timing Loop            2 (1.0 seconds)
Item 16: Retry Count                   0
Item 17: Sensor Address                0
Item 18: Sensor Port Type              RS232
Item 19: Sensor Baud Rate              9600
Item 20: Sensor Bits Per Character     8
Item 21: Sensor Parity                 N
Item 22: Sensor Start Bits             1
Item 23: Sensor Stop Bits              1
Item 24: Sensor GSI Script Number      1
```

The serial sensor address can be any alphanumeric character.

The port type options are dependent on the sensor port. The port settings can be: **`N/O/E`** parity; **`5/6/7/8`** data bits; **`1/2`** start bits; and **`1/2`** stop bits.

The Sensor Definition refers to the Serial Script Menu, defined below. Should the script fail, it is retried up to the number of times defined in Line Item 16.

UNIVERSAL SERIAL SCRIPT MENU

The Universal Serial Script Menu is accessed from the ZENO Program Menu. Up to six scripts can be defined. Line Item 24 in the Sensor Menu refers to the script number.

Individual lines of script will be entered in the Universal Serial Script Menu. The menu will initially only have one line. Additional lines can be created (up to 20 lines total) by using the “I” menu option. In the menu, each line must be changed without errors. If an error is detected, the change will be rejected and the original contents of the line will be kept. Script lines can be edited via a text editor by using a text editor on a configuration file.

The ZENO compiles the script while the user is in the Universal Serial Script Menu. The ZENO gives a meaningful error message when an error in a script line is detected.

A typical script might be as follows. (Note that the language is case-sensitive). Additional checks are carried out when the user quits the ZENO Program Menu.

```
UNIVERSAL SERIAL SCRIPT MENU
(Cn/m) Change Line n to m             (D)  Delete this Script
(In)   Insert Line n                  (Jn) Jump to Script n
(Rn)   Remove Line n                  (N)  Go to Next Script
(En)   Erase Line n                   (P)  Go to Previous Script
(L)    Delete All Lines               (X)  Delete All Scripts
(A)    Insert After this Script       (Z)  Zeno Program Menu
(B)    Insert Before this Script      (H)  Help

Script 1 of 1
Line 1: FLOAT 3
Line 2: LONG 1
Line 3: STRING 2 ( 6, 1 )
Line 4: SET_CHECKSUM SIMPLE, 256
Line 5: SEND “\r\n\CS%1s,DATA NOW:\CS%2x\x03”, sID, sCS
Line 6: WAIT 1000
Line 7: RECEIVE 1000 “\r\n\CS%*1s,%f,%f,%s,%ld,%f,%1s,\CS%2x”,F1,
F2, S1, L1, F3, S2, sCS
```

Placing an * in a format field tells the ZENO that the field is to be ignored.

This script corresponds to the following actions, for a sensor with address Q.

- The ZENO sends the command **`<CR><LF>Q,DATA now:xx^C`**, where **`xx`** is a hexadecimal checksum evaluated by summing the ASCII values of all characters from the **`Q`** to the final colon.
- The ZENO waits 1000 ms before flushing the buffer.
- The ZENO waits for up to 1000 ms further, and attempts to receive a string of the type:

  `<CR><LF>Q,15.2,55.999,YELLOW,256000,-15.3,F,xx^C`

(For space-delimited, the **`RECEIVE`** command include spaces rather than commas in the format specifier: `“\r\n\CS%*1s %f %f.....”`).

- The ZENO tests that the checksum is correct.

The ZENO will return the floating-point values 15.2, 55.999, and -15.3; the string values YELLOW and F; and the long integer 256000.

Declarations

All commands must be followed by a white space to be valid. Commands can be typed in either uppercase or lowercase. The commands STRING, FLOAT, LONG and SET_CHECKSUM can only be used once in any script.

`STRING` *n* (*l1*[,*l2*[,...]])

Declare *n* string variables, and their maximum lengths. These will be referred to as **`S1`**, **`S2`**, etc., within the remainder of the script. String lengths of up to 132 characters are permitted; such long strings will typically be used only to log and transmit entire strings received from sensors which cannot be parsed adequately using the tools available in the current interface definition. The lengths of the strings must be contained within open and close parentheses and the parentheses must be space delimited from everything.

`FLOAT` *n*

`LONG` *n*

Declare *n* floating-point variables and *n* long (32-bit integer) variables. These will be referred to as **`F1`**, **`F2`** and as **`L1`**, **`L2`**, etc., within the remainder of the script. These values can be read from the Process, Data Output, and Test menus. Short integer values are better declared as floating-point, rather than long integer, since this allows more extensive data processing.

Up to 15 variables of each type can be defined within the standard version of ZenoSoft. The ZENO will buffer up to 132 bytes in the standard version of the software.

Other special variables available within the script, which need not be declared, are the following. The values of these variables are not available

`SET_CHECKSUM` *type*, *modulus* [, *offset*]

Define subsequent checksums to be calculated as defined. The permitted checksum types are currently as follows; contact Coastal Environmental Systems for other checksum types. The checksum types can be entered in either uppercase or lowercase.

- **`CRC_16`** (standard CRC-16).
- **`CRC_CCITT`** (standard CRC-CCIT)
- **`XOR`** (Logical XOR between all characters within the delimiters; used in NMEA messages).
- **`SIMPLE`** (ASCII sum of all characters within the delimiters). The **`SIMPLE`** checksum type requires two additional arguments; the modulus used of the ASCII sum and the offset applied to the ASCII sum before the modulus is taken.

The checksum **`sCS`** may not be referenced in a script before the **`SET_CHECKSUM`** command is used.

`sID`

The sensor address, from the Sensor Menu. This is read-only. It is treated as a string value, so the format field in the script format string must be a string format field; otherwise, a syntax error will be declared.

The sensor address can be placed within a **`RECEIVE`** command. In this case, the ZENO checks that the value read from the received string equals the sensor address. If the values are different, the ZENO flags an error, and quits the script. So, in the above script, the command:

```
RECEIVE 1000 “\r\n\CS%1s,%f,%f,%s,%ld,%f,%1s,\CS%2x”, sID, F1, F2,
S1, L1, F3, S2, sCS
```

instructs the ZENO to check that the first comma-delimited field matched the sensor ID.

`sCS`

The checksum calculated according to the **`SET_CHECKSUM`** command. The actual calculation of the checksum is described below. If no SET_CHECKSUM command has been used, it is not valid to use the “\CS” codes within a script format string. The sCS identifier can only be used with long integer format fields (either decimal or hexadecimal).

In **`SEND`** commands, the checksum is calculated as a long integer, formatted as specifiedreted as binary (the next 4 bytes are interpreted as a 4-byte float) rather than as ASCII (the next 4 bytes are interpreted as an ASCII representation of a float) values.

Format fields within format strings must match their identifier type following the format string; i.e., “%f” must be used with F1, “%lx” must be used with L1 and “%s” must be used with S2. If the type of format field does not match the identifier, the ZENO-3200 will generate an error.

To ignore a particular field in a message, use a * immediately following the % sign, such as “%*2s” or “%*lX”.

Commands

`SEND` *format_specifier* [`,`*value1* [`,`*value 2. . . .*]]

Send the specified values, with the format defined according to the C-type specifier, to the sensor. If one of the specified values is the checksum, calculate the checksum according to the **`SET_CHECKSUM`** command.

`WAIT` *time*

Wait for the specified length of time in milliseconds, and then flush the buffer. Wait times from 1 millisecond to 5 minutes are permitted. Note that the serial port is allocated to the serial sensor throughout this time – another sensor cannot use the port while the script is waiting.

`RECEIVE` *time*`,` *format_specifier* [`,`*value1* [`,`*value 2. . . .*]]

Wait for up to the specified length of time in milliseconds, in an attempt to receive and parse the data line from the sensor. (For asynchronous sensors, a **`RECEIVE`** statement is specified with no prior **`SEND`** statement). Wait times from 0 milliseconds to 5 minutes are permitted. Note that the serial port is allocated to the serial sensor throughout this time – another sensor cannot use the port while the script is waiting.

If either a checksum **`sCS`** or the sensor address **`sID`** are referenced in the value list, check that these values are received correctly.

When entering RECEIVE and SEND commands, the field formats used in the format string must match the format specifiers that follow the string, otherwise an error will be generated and the command will not be accepted into the ZENO-3200.

If the string is not received in the specified time, or if the ID or checksum are incorrect, then the ZENO flags an error and quits the script. It may retry the script, if this was specified within the Sensor Menu. If the string is received and parsed successfully, the ZENO places the floating-point, long integer and string values into the corresponding sensor outputs.

REFERENCING SENSOR READINGS

Standard sensor readings – for example, from analog or frequency sensors – are referred to within the ZENO Program and Data Output menus using the format **`S3.2`** or **`P6.1`**. Floating-point values obtained from the Universal Serial Interface are also referred to in this way.

Long integer and string outputs are referred to using the extended formats **`S6.L1`**, **`S6.S1`**, and **`P2.L1`**.

To store string outputs in the Data Output Menu, use the Character String class (code 9). The Field Width specifier (Item 6) is used to specify the length of the string for both storage and data output. When a Character String storage class is defined, the Input Record (Item 4) must be a string record of type S1.S1. If either a floating-point or long integer input record is used, then a Character String storage class cannot be used. If either of these situations is violated, the ZENO-3200 will declare an error and not all the user to leave the ZENO Program Menu.

Null values are:

- -1,000,000 for floating-point outputs;
- 0xFFFFFFFF for long integer outputs;
- a null (zero-length) field for the string.

String outputs are transmitted and logged in fixed width formats, using the field width defined in the Data Output menu. Strings which are too long are truncated; strings which are too short are padded with spaces. String outputs can only be transmitted and logged using Data Output types 7, 8 and 9 (Transmit Only, Log Only, and Transmit and Log). The data storage type will be ignored when a string output is defined. An error will be generated if a user attempts to use a string output as a Block Start, a GOES Binary Format, or an ARGOS Binary Format field.

The user must use literal fields to begin and end a string output with quotation marks (or some other character) to specify the beginning and ending of the string. Since strings can contain commas or spaces, confusion can result with type of delimiter being used with data outputs.

- When using quotation marks to designate strings, it is recommended that a single quotation mark be used at the beginning of the string and a combination of a quotation mark and a delimiter (comma or space) be used at the end of the string.
- The long and string readings will be output in the Test Menu, after the floating-point readings.

Troubleshooting

It's always worth starting by setting up a single string as input, to check whether you are getting any communications at all, before going into more details.