

TechNote #18: QuickBuilder String Handling Example

This document shows you an example program that sends and receives a string over an Ethernet Raw Socket connection. This program can be easily modified for serial communications with a simple port set-up change. Simply omit the 22000 Raw Socket Commands (i.e. \$REGISTERS[22000] = 7; through \$REGISTERS[22007] = 1;) and set \$REGISTERS[12000] to the desired comm. port.

For more specific information on general communications and using the message.ini file, refer to the 5300 Enhancements Overview Manual:

http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530001.pdf

For more information on specific communication register settings, refer to the 5300 Quick Reference Register Guide:

http://www.ctc-control.com/customer/techinfo/docs/5300_951/951-530006.pdf

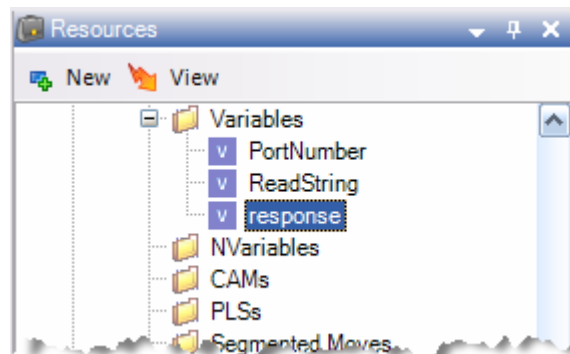
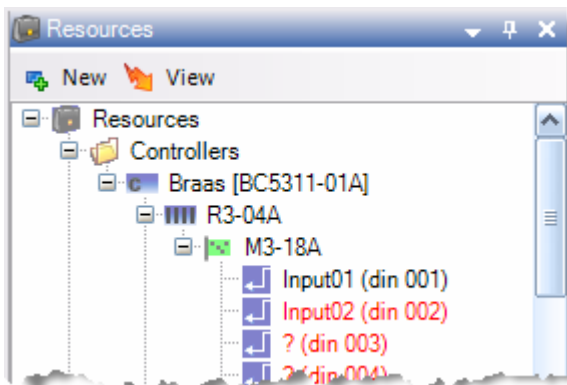
This program example uses one input to trigger the transmission of an ASCII string and then waits for a response. The string being sent is stored in a file called message.ini and copied to the Messages directory within the _system directory of the controller. This message.ini file can be created using notepad. Each line in the message.ini file represents a different string that you would like to send. Make sure that the last character of this file is a carriage return.

This program does the following:

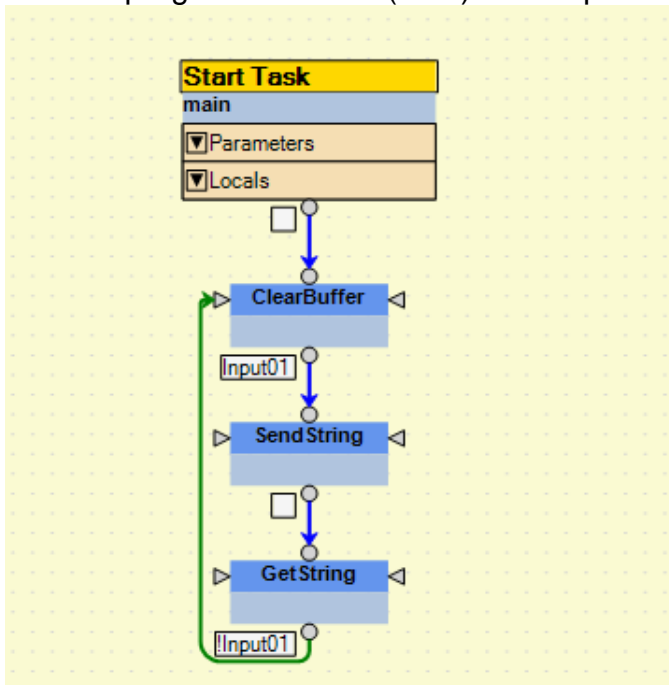
- 1) Set-up the Ethernet port settings and initialized the connection.
- 2) Clears the buffer and waits for input 1.
- 3) Sends the 1st line of the message.ini file in ASCII format over Ethernet Raw Socket.
- 4) Gets the response back from the device that the message was sent to.

The system components (Resources) used in the program are shown below

NOTE: Both ReadString and response variables will need to be set as a string types:



The overall program structure (SFC) is set-up as shown below:

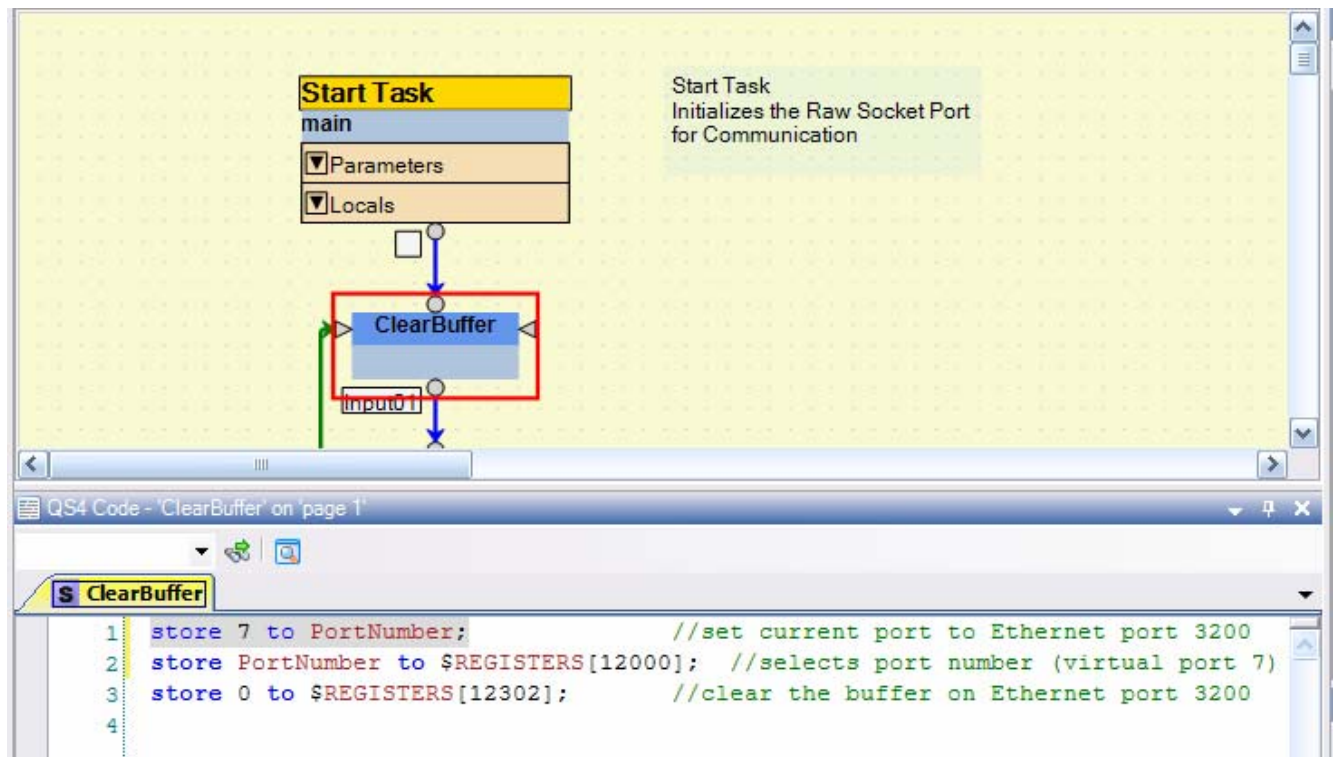


The initial Start Task sets up the Ethernet parameters as described in the code below.

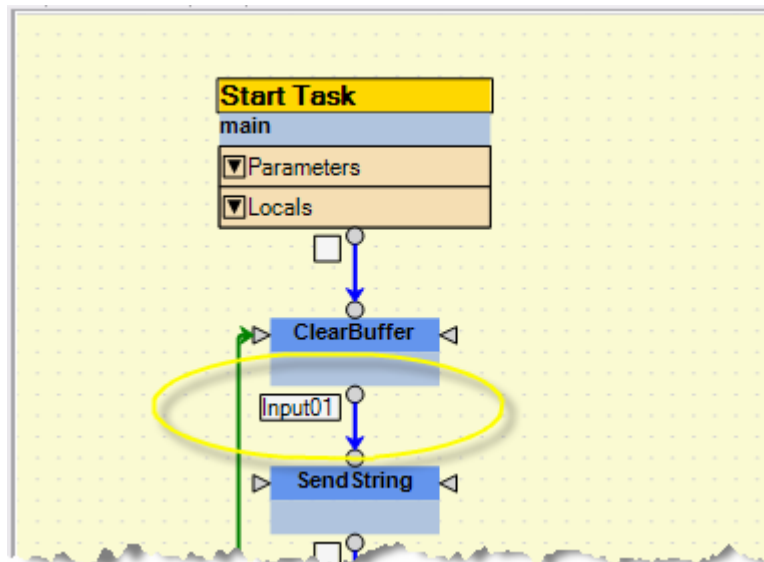
The screenshot shows a software development environment. The top part displays the SFC diagram from the previous image, with a red box around the 'Start Task' and a text box to its right that reads: 'Start Task
Initializes the Raw Socket Port
for Communication'. Below the diagram is a code editor window titled 'QS4 Code - 'main' on 'page 1''. The code is as follows:

```
1  $REGISTERS[22000] = 7; // set up this Ethernet Port as virtual com port 7
2  $REGISTERS[22001] = 0; // set up the 5300 as the client, initiating connection
3  $REGISTERS[22002] = 192; // most significant octet of IP address 192.168.254.10
4  $REGISTERS[22003] = 168; //
5  $REGISTERS[22004] = 254; //
6  $REGISTERS[22005] = 10; // least significant octet of IP address 192.168.254.3
7  $REGISTERS[22006] = 3200; // TCP port to attempt connection to 3200
8  $REGISTERS[22007] = 1; // attempt to connect to Ethernet port 3200
9
10 store 7 to $REGISTERS[12000]; // set current port the Ethernet port 3200 (virtual port7)
11 store 0 to $REGISTERS[12303]; //Inhibit Port Parsing. This means that the buffer will not
12 //parse the characters until a 0 is stored to register
13 //12302
14
```

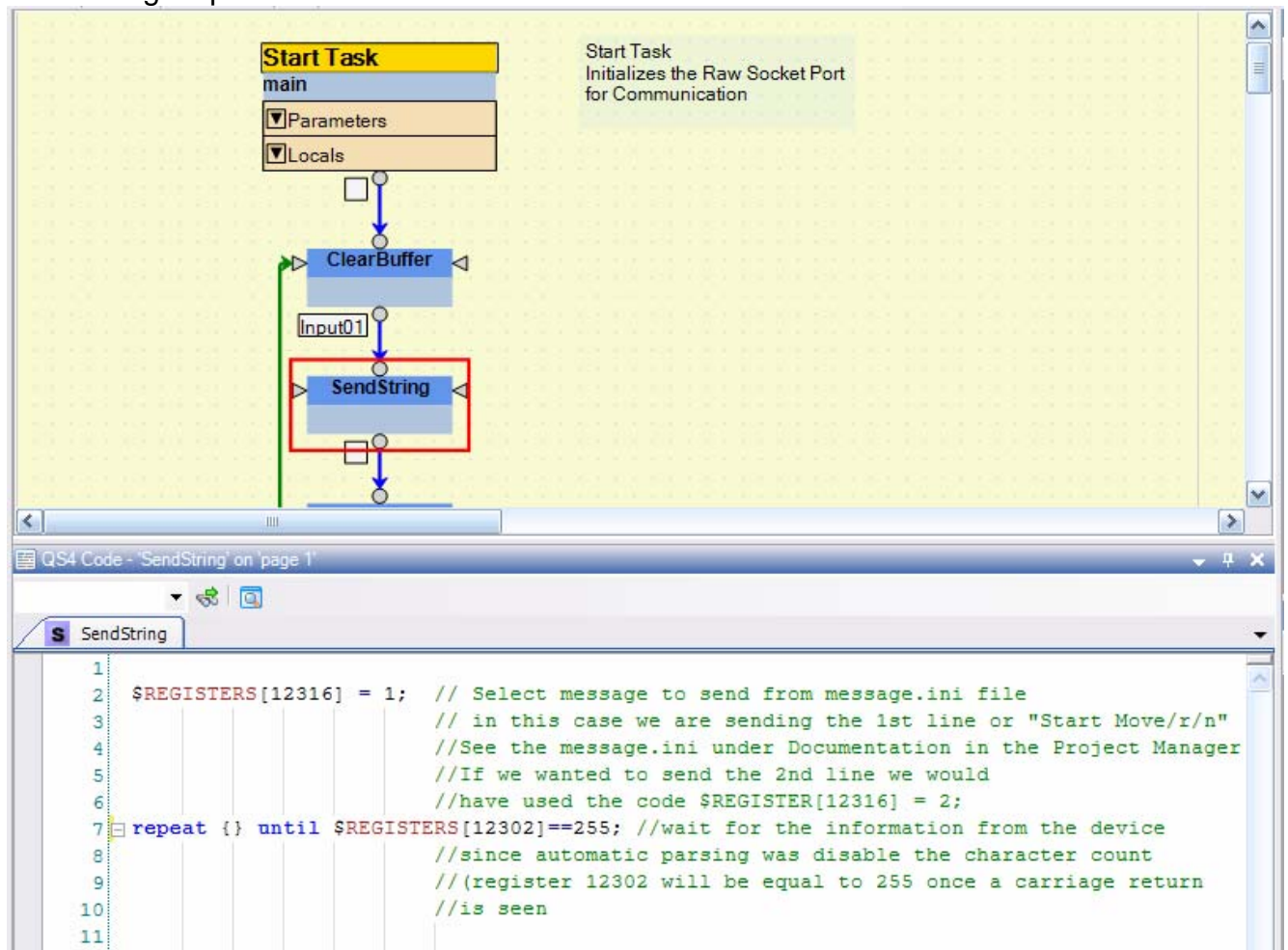
The buffer is then cleared in the next step ClearBuffer as shown below:



Wait for Input 1 to be on.



The 1st line of the message.ini is now sent out the Ethernet port as described in the SendString step code below:



The screenshot displays a software development environment. The top portion shows a flowchart for a 'Start Task' named 'main'. The flowchart includes a 'ClearBuffer' step, followed by an 'Input01' step, and then a 'SendString' step which is highlighted with a red box. A green arrow indicates the flow from 'Input01' to 'SendString'. Below the flowchart, a code editor window titled 'SendString' shows the following code:

```
1
2 $REGISTERS[12316] = 1; // Select message to send from message.ini file
3 // in this case we are sending the 1st line or "Start Move/r/n"
4 //See the message.ini under Documentation in the Project Manager
5 //If we wanted to send the 2nd line we would
6 //have used the code $REGISTER[12316] = 2;
7 repeat {} until $REGISTERS[12302]==255; //wait for the information from the device
8 //since automatic parsing was disable the character count
9 //(register 12302 will be equal to 255 once a carriage return
10 //is seen
11
```

After we see the response from our device is complete we can retrieve the data as described in the GetString step code on the next page:

QS4 Code - 'GetString' on 'page 1'

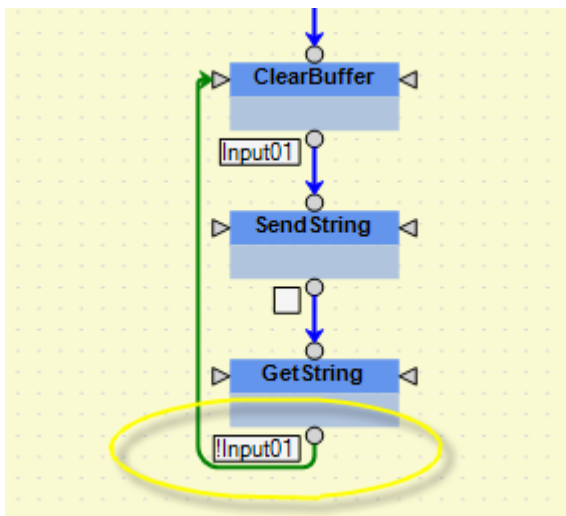
```

1 ReadString=string( \"%cr12001,200\" ); //get the first 200 characters from the buffer
2
3 response = ReadString; //make response = ReadString and you can now extract data
4 //from response as needed using string functions such as
5 //left, right and mid.
6
7 delay 250ms;
8

```

Note: If you wanted to omit the first character received and just read the next 20 characters after that you could use the statement `ReadString=string(\"%cr12002,19\");`, which would get the 2nd through 20th characters of the received string and assign those characters to `ReadString`.

We now wait for Input1 to turn off before we go back to the ClearBuffer step to repeat the whole process.



APPENDIX – MSB CODE

This appendix has the code shown in the steps above but will allow you to cut and past into your documents for your convenience.

StartTask: Main Code:

```
$REGISTERS[22000] = 7;           // set up this Ethernet Port as virtual com port 7
$REGISTERS[22001] = 0;           // set up the 5300 as the client, initiating
connection
$REGISTERS[22002] = 192;         // most significant octet of IP address
192.168.254.10
$REGISTERS[22003] = 168;         //
$REGISTERS[22004] = 254;         //
$REGISTERS[22005] = 10;         // least significant octet of IP address 192.168.254.3
$REGISTERS[22006] = 3200;       // TCP port to attempt connection to 3200
$REGISTERS[22007] = 1;         // attempt to connect to Ethernet port 3200

store 7 to $REGISTERS[12000]; // set current port the Ethernet port 3200
store 0 to $REGISTERS[12303]; //Inhibit Port Parsing. This means that the buffer
// will not parse the characters until a 0 is
//stored to register 12302
```

Step: ClearBuffer Code:

```
store 7 to PortNumber;           //set current port to Ethernet port 3200
store PortNumber to $REGISTERS[12000]; //selects port number
store 0 to $REGISTERS[12302];     //clear the buffer on Ethernet port 3200
```

Step: SendString Code:

```
$REGISTERS[12316] = 1;           // Select message to send from message.ini file
// in this case we are sending the 1st line.
//If we wanted to send the 2nd line we would
//have used the code $REGISTER[12316] = 2;
repeat {} until $REGISTERS[12302]==255; //wait for the information from the
// device since automatic parsing was disable
// the character count (register 12302 will
// be equal to 255 once a carriage return
//is seen
```

Step: GetString Code:

```
ReadString=string("%cr12001,200"); //get the first 200 characters from the buffer

response = ReadString;           //make response = ReadString and you can now
// extract data from response as needed using
//string functions such as left, right and
//mid.

delay 250ms;
```