First, there is a simple hardware issue that may prevent communications on the RS-485 bus: There needs to be a 100 OHM Termination resistor between D+ and D- at one of the cable endpoints. I did not see that in your cable setup. This is mentioned on the DKN oven manuals, as below:

Note 1) Configuration of the non-standard accessory, "external communication adaptor (RS485-232C) ODK18" is as follows.

① Com cable 1 : PC side connector (for connecting IBM9 pin device)RS-232C cable 1m, KS-485 side connector (Dsub25 pin, male) System Sacom CBL16

② Com cable 2 : KS-485 side connector (Dsub9 pin, male) UL2464TASB 2-core AWG20 cable 3m, with a Y-terminal on the device side (with terminal resistance of 100Ω)

③ RS-232C⇔KS-485 converter unit : System Sacom KS-485, with an AC adaptor

The communication protocol documentation, although extensive, can lead to some confusion on the basics (like binary vs. ASCII, etc.). There is nothing better to explain it than to show a capture of the RS485 bus traffic during communications between a PC host and an RS485 Target Device. For this, I use two RS485 adapters connected to the same PC: one adapter is a RS-232 to RS-485 converter (with external AC/DC adapter) connected to the PC via an USB to RS-232 dongle. The other adapter is a simple USB to RS-485, connected to the same PC.

Software setup:

The communication protocol used by the Yamato furnaces is the Toho protocol. Toho makes available a simple, easy to use Windows application that sends and receives commands using their protocol. I recommend installing it on the test PC and assigning it to one of the RS-485 adapters. The English version of the Windows application (with English manual) can be downloaded from
here: http://r7.s901v.smilestart.ne.jp/toho/english/ComSamp3Ver0102Setup.zip

I used also another serial communication application (assigned to the secondary RS-485 adapter), to simulate the furnace behavior and to generate traffic back to the "master" RS-485 adapter being used by the Toho application. It is called Real term and the version I'm using, although a bit old, can be downloaded from here:
http://www.i2cchip.com/realterm/old_versions/Realterm_2.0.0.70_SignedWrapper_setup.exe

HyperTerminal will not work properly as you need to send/receive binary numbers over the RS-485 interface that are non-printable ASCII characters. Better suited communication applications should be used.

So, Toho's ComSamp3 app sends user typed commands to the "furnace" after appending it with a calculated BCC byte (XOR of previous bytes), and validates the "furnace" replies, printing everything on the main screen. To make it easier to input a command string, the Application converts the user typed "(" character to the non-printable 0x02 character. Likewise, it converts the ")" character into the non-printable 0x03 character. When the application receives these same binary characters, it converts them to "(" and ")" so that they can be printed on the screen.

Also, if you are creating the command strings manually instead of using Toho's application, you need to calculate and append the BCC byte yourself. I wrote an MS Excel spreadsheet with a formula to calculate the BCC byte from a data set but I can't find it right now...

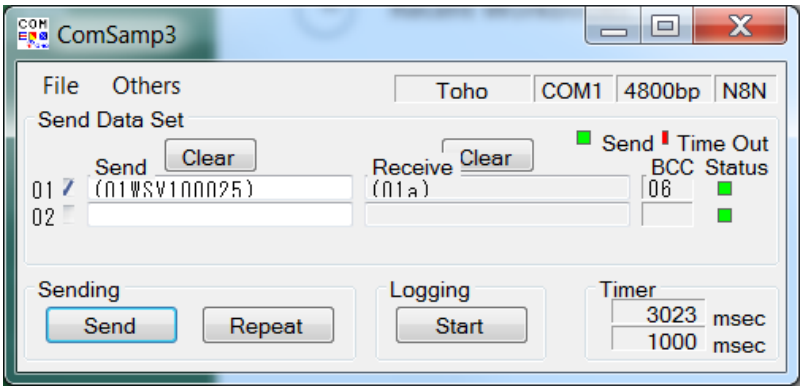
Screen shots:

Toho application: Command sent is a Write request to device 01, register SV1, value = "000025". No BCC byte in this setup

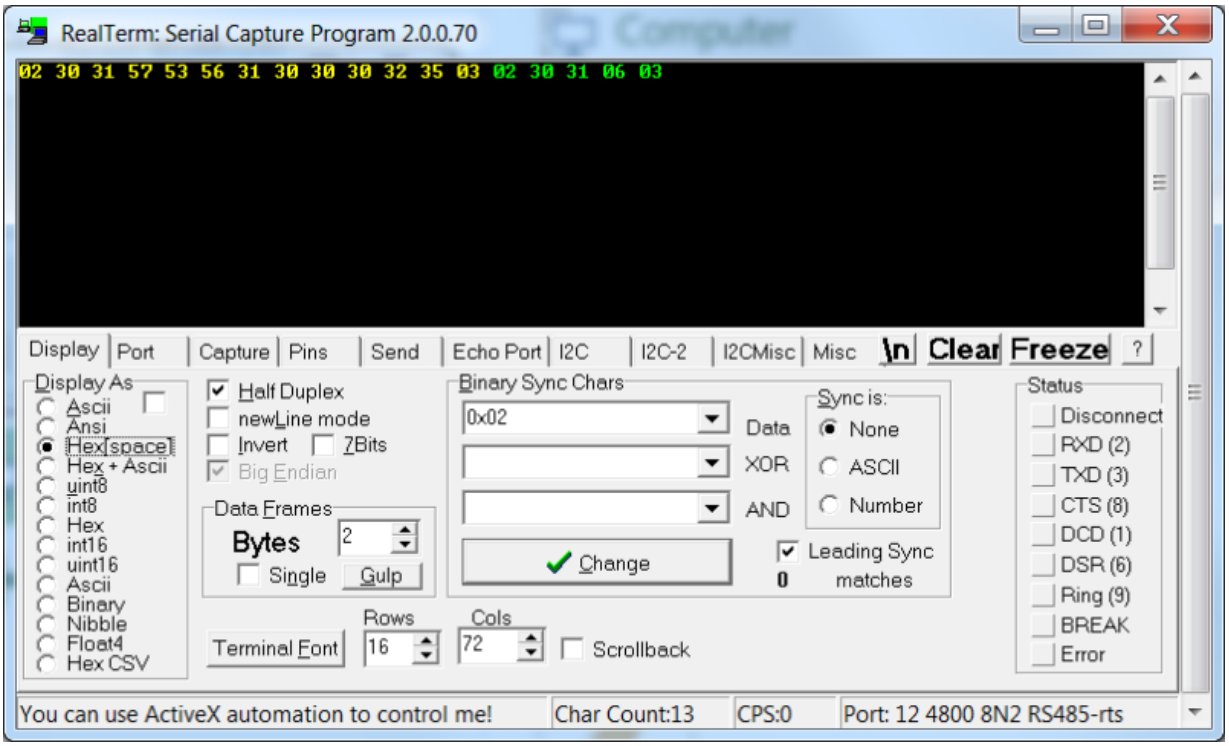
"Furnace" reply (received data) is: Device 01, ACK.

RealTerm Application emulating Furnace:

First screen shot shows furnace received bytes in Yellow and manually sent bytes in Green.
(Hex representation with added spaces between received and transmitted bytes.)



Second screen shot shows Binary representation of received and transmitted characters, as well as serial port setup (simulating the Furnace behavior). This clarifies the Binary vs ASCII question, hopefully.



Third screen shot shows the manually crafted 6 byte reply from the "furnace", represented in Hex notation 0xnn.