



## Using the ADAM in a Linux Environment

As an organization that has made Linux its OS of choice, Alti-tech Engineering is pleased to offer its knowledge and experience in assisting Linux users accomplish tasks related to electronic design and engineering in the Linux environment.

The following is a brief explanation on incorporating the Advantech Data Acquisition Module (ADAM) into a Linux network.

The ADAM is a unit which can be used to read and act on analog data and to control electronic systems. The ADAM is controlled over ethernet and commands that can be issued include reading and setting the range on the analog inputs, reading the value on the inputs, and reading and setting the digital outputs. See the ADAM manual for a complete list. The following information applies specifically to the ADAM 6017, although the same techniques will function with any unit from the ADAM 6000 series.

### Contents:

#### 1 Working with the ADAM

##### 1.1 Configuration

##### 1.2 ADAM Command Format

##### 1.3 Communicating with the ADAM

#### 2 Code and Reverse Engineering examples

##### 2.1 Example of custom Advantech command in translation

##### 2.2 Example of Perl code to communicate with the ADAM

##### 2. Example of analyzing Wireshark logs

## **1 Working with the ADAM**

### ***1.1 Configuration***

As Advantech currently does not provide support for Linux, initial configuration of the ADAM will unfortunately require use of a Windows OS. Once the ADAM has been configured it can be used in a Windows-free environment.

Begin by altering the netmask and IP address on the ADAM to match those on your network – you will have to use the Windows utility supplied by Advantech to do this. We connected the ADAM directly to the computer using a crossover ethernet cable during this step.

### ***1.2 ADAM Command Format***

The ADAM is capable of accepting standard Modbus commands as well as an array of customized Advantech commands. More information about Modbus commands and support can be found at the Modbus website (<http://www.modbus.org/specs.php>). The Modbus commands can be used in Linux directly and the ADAM Manual contains a list of specific Modbus commands that are used to control the ADAM. The custom Advantech commands use a Windows .DLL file to translate the text commands into a format that can be sent to the ADAM. If you wish to use these custom Advantech commands in a Linux environment, you must first "translate" them yourself. Use the Windows utility supplied to obtain the hexadecimal equivalent for each command (see section 2.1 below).

Users should be aware that some of the Advantech commands provided actually consist of two separate Modbus commands rolled into one, and that the utility provided only provides translation for the first part of the command. Use of a network protocol analyzer such as Wireshark (<http://www.wireshark.org/>) allows you to monitor communications between the unit and the utility, and to pick out the second half of the command (see section 2.3 below).

In the ADAM manual:

- Section 6-3 (pg. 106): Modbus Commands
- Section 6-4 (pg. 112): Advantech Commands

### ***1.3 Communicating with the ADAM***

Although there are a great many methods to communicate over TCP/IP, we opted to use Perl for its cross-platform capabilities and that therefore all of our examples are in Perl. We find that using the Perl IO::Socket module to connect to the ADAM and a modified version of the "Simple Perl MODBUS TCP/IP Client" software provided here (<http://www.modbus.pl/downloads.htm>) is an effective way to perform communications (see section 2.2 below).

Note that in some Linux distributions, a network setting can interfere with communicating with the ADAM. In our case, upon initial viewing the Wireshark logs, it appeared the checksum was being calculated incorrectly. We found that by changing the `tcp_timestamps` variable to 1 from 0, the checksum value was corrected and communications proceeded successfully. If you encounter a similar problem, follow the instructions provided on the following webpage: <http://www.faqs.org/docs/securing/chap6sec75.html>

## 2 Code and Reverse Engineering Examples

### 2.1 Example of custom Advantech command in translation

We'll use the command to set the range on an analog input. In this case, channel 0 will be set to an input range of 4 ~ 20 mA. The text command is `$01A0007` (See the ADAM manual for the format of this command). Using the "Terminal" tool in the Windows utility produces a translated command of `000000000110110270F00050A24303141303030370D00`. This is what you would actually submit to the ADAM. The translated command consists of three parts: a header, the command, and a footer. In this case:

Header: `000000000110110270F00050A`

Command: `2430314130303037`

Footer: `0D00`

Luckily, the command portion is just an ASCII translation of the text command so you don't have to use the Windows utility if you want to set the input range of a different channel or change the input range. In this case:

- Text command: `$01A0007`
- Decimal ASCII translation of the text command: `3648496548484855`
- Hex ASCII translation: `2430314130303037`

Text	\$	0	1	A	0	0	0	7
Decimal ASCII	36	48	49	65	48	48	48	55
Hex ASCII	24	30	31	41	30	30	30	37

### 2.2 Example of Perl code to communicate with the ADAM

For the following Perl code to work correctly, make an entry for the ADAM in the `hosts` file of the computer that will be controlling it (in this case we've given it the name "adam").

The subroutine that sends the data to the ADAM uses the Perl `pack` command to convert the data into hexadecimal format:

```
my $string = pack "H*", $data;
```

where \$string is the converted data actually sent to the ADAM and \$data is the original text command.

When the response comes back from the ADAM, the inverse operation is performed –use the Perl "unpack" command:

```
my $string = unpack "H*", $data;
```

The string can then be read as text and parsed using Perl's text manipulation commands.

```
use IO::Socket;
```

```
my $sock_name = "TCP SOCK";
```

```
#this opens the socket for communication
```

```
sub init_socket {
```

```
    my $remote_port = shift || 502;
```

```
    my $remote = 'adam';
```

```
    my $remote_host = gethostbyname( $remote );
```

```
    my $trans_serv = getprotobyname( 'tcp' );
```

```
    my $destination = sockaddr_in( $remote_port, $remote_host );
```

```
    socket( $sock_name, PF_INET, SOCK_STREAM, $trans_serv )
```

```
        or die "Socket creation failed: $!\n";
```

```
    my $con_ok = connect( $sock_name, $destination )
```

```
        # stop the program if the ADAM isn't connected
```

```
        or return 0;
```

```
    next unless $con_ok;
```

```
}
```

```
#this is used to send a command to the ADAM and retrieve the response
```

```
sub send_adam {
```

```
    my $data = shift(@_);
```

```
    my $string = pack "H*", $data;
```

```
    my $hex = "";
```

```
    send( $sock_name, $string, 0 )
```

```
        or warn "Problem with send: $!\n";
```

```
    my $from_who = recv( $sock_name, my $data_back, 65535, 0 );
```

```
    if ($data_back) {
```

```
        $hex = unpack "H*", $data_back;
```

```
    }
    else {
        warn "Problem with recv: $!\n";
    }

    return $hex;
}

#this is used to close the socket before the program exits
sub close_socket {
    close $sock_name
    or warn "Close failed: $!\n";
}
```

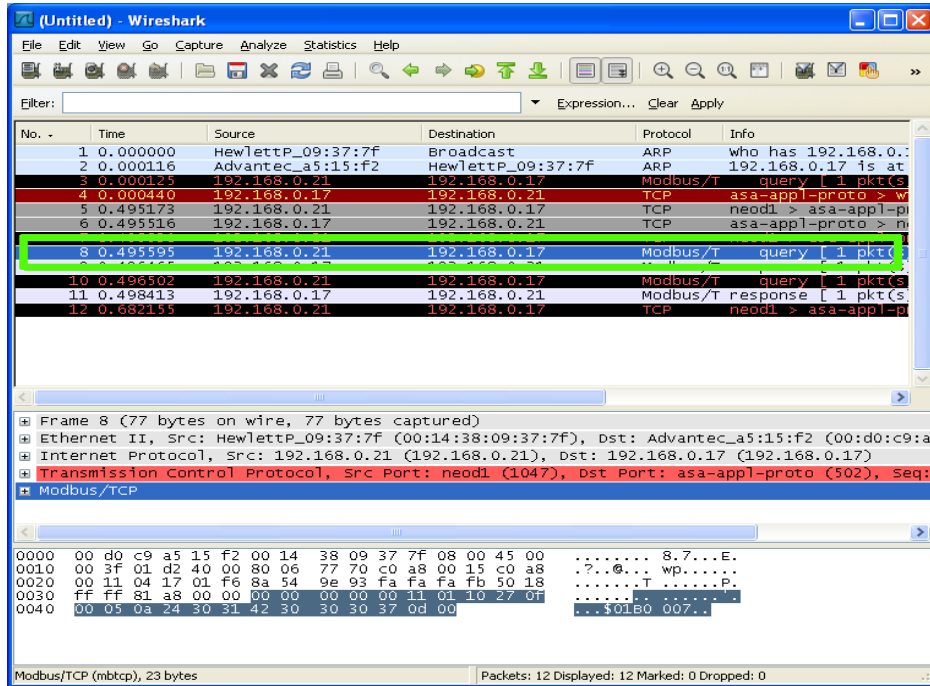
### ***2.3 Example of analyzing Wireshark logs***

The text command to read the range of an input is \$01B0007 (to read Vin0). The Windows utility reports that the translation of this command is

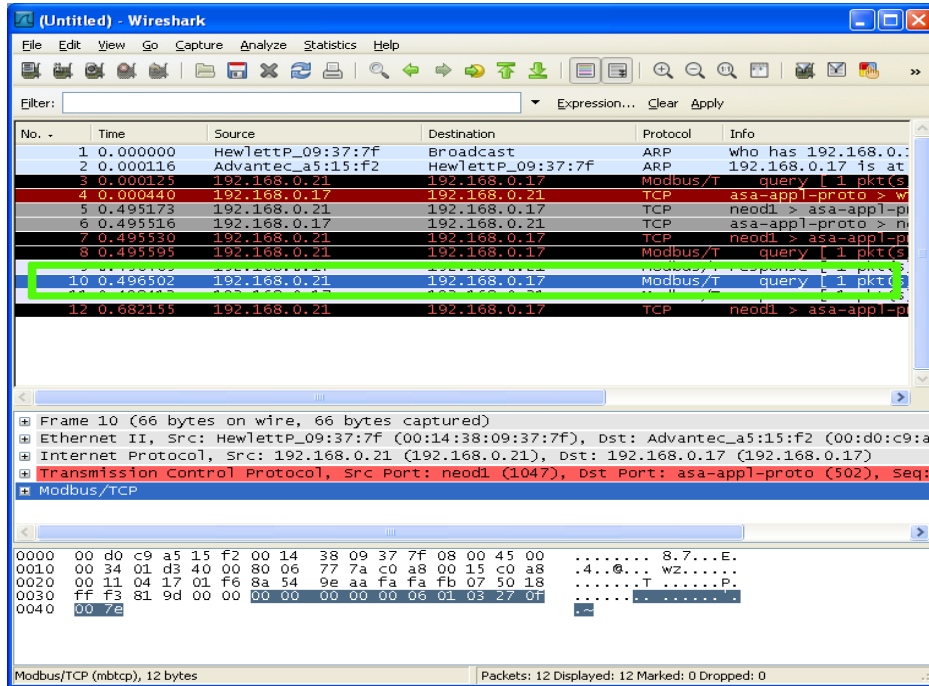
0000000000F0110270F00040824303142303030370D00

This, however, is only part of the command. Submitting this string by itself will not return the range of the input. To determine the rest of the command, use Wireshark to eavesdrop on the translation process. Start Wireshark capturing data on the ethernet device you are using to communicate with the ADAM. Then use the Windows utility to translate \$01B0007. In the Wireshark window you will see many lines of communication between the ADAM and your computer. The ones we are looking for have the source as the computer and the destination as the ADAM, the protocol as "Modbus/T" and the info as "query".

The highlighted line in the screenshot is the first part of the request from the computer. Clicking on the "Modbus/TCP" line in the middle screen highlights the Modbus part of the message in the bottom window. You can see that it is the same as the translation above (and, in fact, the text command \$01B0007 can be seen in the ASCII version of the message).



The next query (highlighted in the second screenshot) is the second part of the request from the computer, the part that the Windows utility does not provide. In this case the Modbus part of the message is **000000000060103270f007e** (which happens to be an unspecified Modbus command).



To send this command to the ADAM, you would send the first part of the command (the translation) and then the second part (the Modbus command). The answer to the query (what range a specific input is set to) would be contained in the reply to the second part.

If you would like more information, or have any specific questions, please feel free to contact Hanna at Alti-tech.