Model JI-4516
High Voltage, High Current Digital I/O Module - USB

Programmer's Interface Document

Version 1.9

Jupiter Instruments

8/27/2013 Edition

www.jupiteri.com

TABLE OF CONTENTS

1. INTRODUCTION

This document presents the software interface requirements between the JI-4516 I/O module and a host computer. It describes the concept of operation for the interface, defines message structure and protocol, and describes the flow of data between I/O module and host computer. Specific details on the USB interface, JI-4040 command set, and code examples are presented.

2. HOST COMMUNICATION

2.0 USB Interface

Communication with the JI-4516 is by way of a USB connection. The JI-4516 incorporates a USB IC (FT245R from FTDI) that handles both the physical interface and USB protocol. Drivers and DLLs for this device support operation with several programming language types (C++, C#, LabVIEW, etc.) and operating systems (Wndows 7, 8, Vista, XP, Linux, etc.) Additionally, a Virtual Com Port (VCP) driver is available that allows communication via a terminal emulator program such as Microsoft HyperTerminal. In this mode, commands can be rapidly tested using either keystroke entry or script file. Detailed information on the operating systems supported and programming examples can be found at the FTDI website (www.ftdichip.com). Specifically, driver downloads are available at http://www.ftdichip.com/FTDrivers.htm and the API for the FTD2XX.dll is available at http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf

2.1 Hardware

- FT245R USB Interface IC from FTDI
  http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT245R.pdf
- Data Transfer Rate:
  - 1Mbit/sec – D2XX Direct Driver
  - 1Mbit/sec - VCP

2.2 JI-4516 Commands

No USB-specific programming is required to control the JI-4516. Communication with the USB port is accomplished via a VCP or manufacture supplied DLL: FTD2xx.dll or FTD2xx_NET.dll. In either case, the command and response messages are comprised of simple ASCII strings. A straightforward command/response type protocol ensures that all transmission are acknowledged. A "!" response indicates a valid command message, and "?" indicates invalid.

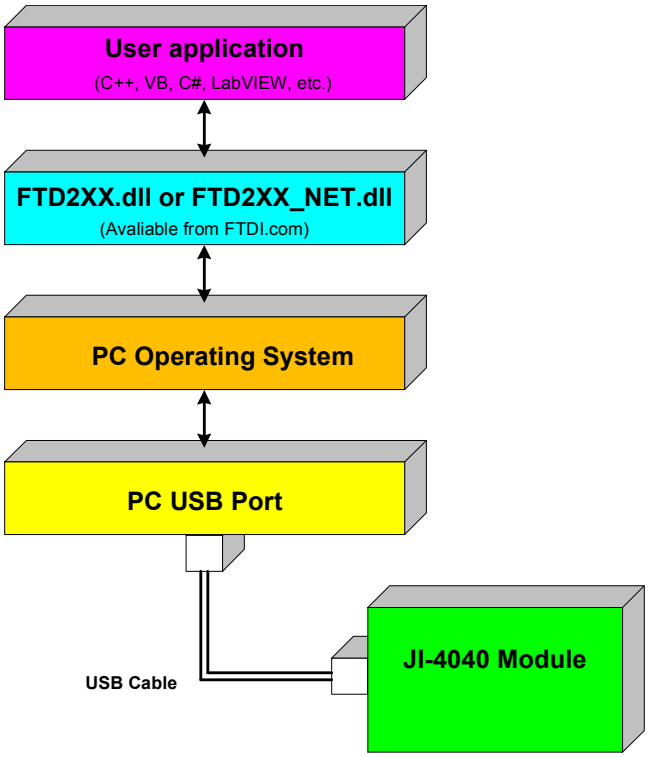

Figure 1. Communication Flow Diagram

2.2.1 **Syntax**

Command structure:

[$][Command Characters(2)][Data Argument][Carriage Return]

Each command begins with a start delimiter '$' followed by a two character command. Character commands use upper-case alpha characters. Following is the Data Argument that can range in length from 0 to 2 bytes, depending on the command type. Note that not all commands have arguments. Data arguments use lower-case alpha characters. A carriage return terminates all commands.

Response structure:

[Argument string][!]

A response is generated for all valid or invalid commands sent to the JI-4516 module. A '!' character is returned for all valid commands, and a '?' response indicates an invalid command has been received. Depending on the command type, a single character or character string may precede the '!'.

Command/Response Examples:

Command	Response	Description
$IR<CR>	5c!	Read Inputs (5ch)
$SW21<CR>	!	Write to 21h to output switches Switches 6,1 = Closed Switches 8,7,5,4,3,2 = Open
$WP05<CR>	!	Set watchdog to 0.5 second interval
$R5<CR>	?	Invalid command

2.2.2 **Initialization**

No special initialization or setup is required prior to use.

2.2.3 Command Set

This section list the command set for the JI-4516. Code examples make use of the JI4516A_NET DLL. A copy of this file can be found in Apendix A.

Legend	
aa or aa	8-bits ASCII HEX value – lower case
aaaa or aaaa	16-bit ASCII HEX value – lower case
<CR>	Carriage Return – control character "\r"
tt or tt	Text character(s)
. . . .	Hidden Code

2.2.3.1 Read Input – **IR**

Syntax: $IR<CR>

Direction: Read

Argument: None

Reset Value: N/A

Response: *aa*! = Input Data
? = Syntax error

Description: This command reads input data.

Command Example:

Command	Response	Description
$IR<CR>	5c!	Input bits 6, 4, 3, & 2 = High. Input bits 7, 5, 1, & 0 = Low.
$IR<CR>	63!	Input bits 6, 5, 1, & 0 = High. Input bits 7, 4, 3, & 2 = Low.
$IR<CR>	d7!	Input bits 7, 6, 4, 2, 1 & 0 = High. Input bits 5 & 3 = Low.

Code:

```
/*********************************************************/
// Read Input Method
//
//*********************/
private void readInputMethod()
{
            string str1 = "";
            if (DeviceA.Read("IR", out str1) == 0)
            {
```

```
                readInputTextBox.Text = str1;
            }
            else
            {
                readInputTextBox.Text = "Error: Read";
            }
        }
        /*********************************************************/
```

2.2.3.2 Output Switchs, write – **SW*aa***

Syntax: $SW*aa*<CR>

Direction: Write

Argument: ***aa*** Switch States

Reset Value: All switches open

Response: ! = success
? = syntax error

Description: This command sets the state of all 8 output switches. Switch logic is as follows:
Closed (on) = 1
Open (off) = 0

Command Example:

Command	Response	Description
$SW21<CR>	!	Close switches 6 & 2. Open switches 8, 7, 5, 4, 3 & 1.
$SW3b<CR>	!	Close switches 6, 5, 4, 2, & 1. Open switches 8, 7, 5, & 3.
$SWff<CR>	!	All Switches Closed

Code:

```
/*********************************************************/
// Write All Switches Method
//
/*********************/
private void writeSwitchesMethod()
{
    if (DeviceA.Write("SW" + writeSwitchesDataTextBox.Text) == 0)
    {
        writeSwitchStatusTextBox.Text = "Ok: Write";
    }
    else
    {
        writeSwitchStatusTextBox.Text = "Error: Write";
    }
}
/*********************************************************/
```

2.2.3.3 Output Switch, write individual – **SI*aa***

Syntax: $SI*aa*<CR>

Direction: Write

Argument: *aa* *a* = selected switch 1 - 8, *a* = switch state: 0 = Open, 1 = Closed

Reset Value: All switches open

Response: ! = success
? = syntax error

Description: This command provides individual switch control.

Command Example:

Command	Response	Description
$SI11<CR>	!	Switch 1 = Closed
$SI10<CR>	!	Switch 1 = Open
$SI51<CR>	!	Switch 5 = Closed

Code:

```
/*********************************************************/
// Write Switches - Individual
//
/**********************/
private void writeSwitchesIndMethod()
{
    string str1 = (writeSwitchNumber.SelectedIndex + 1).ToString()
                + writeSwitchState.SelectedIndex.ToString();

    if (DeviceA.Write("SI" + str1) == 0)
    {
         writeSwitchIndStatusTextBox.Text = "Ok: Write";
    }
    else
    {
         writeSwitchIndStatusTextBox.Text = "Error: Write";
    }
}
/*********************************************************/
```

2.2.3.4 Output Switch, read – **SR**

Syntax: $SR<CR>

Direction: Read

Argument: None

Reset Value: N/A

Response: ***aa***! = Switch States
? = Syntax error

Description: This command reads the state of the 8 output switches. Bit logic is as follows:
Switch closed (on) = 1
Switch open (off) = 0

Command Example:

Command	Response	Description
$SR<CR>	5c!	Switches 7,5,4,3 = Closed Switches 8,6,2,1 = Open
$SR<CR>	6F!	Switches 7,6,4,3,2,1 = Closed Switches 8,5 = Open
$SR<CR>	88!	Switches 8,4 = Closed Switches 7,6,5,3,2,1 = Open

Code:

```
/*********************************************************/
// Read Output Switches Method
//
//**********************/
private void readSwitchesMethod()
{
        string str1 = "";
        if (DeviceA.Read("SR", out str1) == 0)
        {
                readSwitchesTextBox.Text = str1;
        }
        else
        {
                readSwitchesTextBox.Text = "Error: Read";
        }
}
/*********************************************************/
```

2.2.3.5 Configuration Register, write – **CW*aa***

Syntax: $CW<CR>

Direction: Write

Argument: ***aa*** Configuration Data

Reset: 00h All functions disabled

Response: ! = success
? = syntax error

Description: This command sets the behavior of the COS function and Input Filter.

Configuration Byte

Bit	Description
7	
6	
5	
4	Input Filter 1 = Enabled
3	B1 COS Configuration
2	B0 COS Configuration
1	COS Enabled 1 = Enabled
0	COS Mask 1 = Enabled

Input Filter

When enabled, the Input Filter function provides Low Pass (LP) filtering on each input signal. The LP filter time constant is 20mS.

COS Configuration

B1 B0

0 0

Nominal Mode: When a COS event occurs , the COS Event Occurrence bit in the Status Register is set and the COS Enable bit in the Configuration Register is cleared. When the status register is read, the COS Event Occurrence bit is cleared.

0 1

Single Event Mode: Following the occurrence of a COS event, a response is sent to the host and the COS Enable bit in the Configuration Register is cleared. The response is Input Data subsequent to the COS event. Its structured consist of a start delimiter *, followed by two characters (Input Data), and terminated by a ! *aa!

1 1

Multiple Event Mode: A COS event response is sent to the host on each COS occurrence. The response is Input Data subsequent to the COS event. Its structured consist of a start delimiter *, followed by two characters (Input Data), and terminated by a ! *aa!

COS Enabled

When enabled, the COS function selected by bits 3 (B1) and 2 (B0) are implemented.

COS Mask

When enabled, the contents of the COS Mask register are applied to the input signals.

Command Example:

Command	Response	Description
$CW10<CR>	!	Input filter enabled

$CW03<CR>	!	1. COS Normal Mode enabled 2. COS Mask enabled
$CW0e<CR>	!	1. COS Multiple Event Mode enabled

Code:

```
/*********************************************************/
// Write Configuration Register Method
//
/*********************/
private void writeConfigRegMethod()
{
        if (DeviceA.Write("CW" + writeConfigDataTextBox.Text) == 0)
        {
                writeConfigStatusTextBox.Text = "Ok: Write";
        }
        else
        {
                writeConfigStatusTextBox.Text = "Error: Write";
        }
}
/*********************************************************/
```

2.2.3.6 Configuration Register, read – **CR**

Syntax: $CR<CR>

Direction: Read

Argument: N/A

Reset: All functions disabled

Response: *aa*! = Configuration Data
? = Syntax error

Description: This command reads the contents of Configuration Register. See Configuration Register Write for detailed bit descriptions.

Configuration Byte

Bit	Description
7	
6	
5	
4	Input Filter 1 = Enabled
3	B1 COS Configuration
2	B0 COS Configuration
1	COS Enabled 1 = Enabled
0	COS Mask 1 = Enabled

Command Example:

Command	Response	Description

$CR<CR>	1f!	1. Input filter enabled 2. COS Multiple Event Mode enabled 3. COS Mask enabled
$CR<CR>	10!	1. Input Filter enabled
$CR<CR>	00!	1. No functions enabled

Code:

```
/*********************************************************/
// Read Configuration Register Method
//
//*********************/
private void readConfigRegMethod()
{
        string str1 = "";
        if (DeviceA.Read("CR", out str1) == 0)
        {
                readConfigRegTextBox.Text = str1;
        }
        else
        {
                readConfigRegTextBox.Text = "Error: Read";
        }
}
/*********************************************************/
```

2.2.3.7 Status Register, read – **HR**

Syntax: $HR<CR>

Direction: Read

Argument: None

Reset Value: N/A

Response: *aa*! = Status Register Data
? = Syntax error

Description: This command reads the contents of Status Register. This register indicates the status of the Watchdog Timer and the occurrence of a COS event. Reading this register will clear the COS Event Occurrence bit (0).

COS Configuration Byte

Bit	Description
7	WDT Enabled 1 = Enabled
6	
5	
4	WDT Time-Out 1 = Time-out
3	
2	
1	
0	COS Event Occurrence 1 = true

Command

Example:

Command	Response	Description
$HR<CR>	00!	1. WDT is not enabled 2. No COS Event
$CR<CR>	80!	1. WDT is enabled and a time-out has not occurred. 2. No COS Event
$CR<CR>	91!	1. WDT is enabled and a time-out has occurred. 2. A COS event has occurred.

Code:

```
/*********************************************************/
// Read Status Register Method
//
//**********************/
private void readStatusRegMethod()
{
        string str1 = "";
        if (DeviceA.Read("HR", out str1) == 0)
        {
                readStatusRegTextBox.Text = str1;
        }
        else
        {
                readStatusRegTextBox.Text = "Error: Read";
        }
}
/*********************************************************/
```

2.2.3.8 COS Enable, write – **KE**

Syntax: $KE<CR>

Direction: Write

Argument: None

Reset: N/A

Response: None

Description: This command enables the COS function by setting bit 1 in the Configuration Register. It is normally used as the final command to arm either a single or multiple event COS function since it does not generate a response.

Command Example:

Command	Response	Description
$KE<CR>	None	COS function enabled

Code:

```
/*********************************************************/

// Enable COS Method
//
/*********************/
private void enableCOSMethod()
{
	if (DeviceA.WriteWO("KE") == 0)
	{
		enableCOS_TextBox.Text = "Ok: Enable";
	}
	else
	{
		enableCOS_TextBox.Text = "Error: Enable";
	}
}
/*********************************************************/
```

2.2.3.9 COS Disable, write – **KD**

Syntax: $KD<CR>

Direction: Write

Argument: None

Reset: N/A

Response: None

Description: This command disables the COS function by clearing bit 1 in the Configuration Register. The command does not generate a response.

Command Example:

Command	Response	Description
$KD<CR>	None	COS function disabled

Code:

```
/*********************************************************/
// Disable COS Method
//
/*********************/
private void disableCOSMethod()
{
	if (DeviceA.WriteWO("KD") == 0)
	{
		disableCOS_TextBox.Text = "Ok: Disable";
	}
	else
	{
		disableCOS_TextBox.Text = "Error: Disable";
	}
}
/*********************************************************/
```

2.2.3.10 COS Mask, write – **MW*aa***

Syntax: $MW*aa*<CR>

Direction: Write

Argument: ***aa*** 8-bits (ASCII HEX value)

Reset Value: 00h All inputs are masked out.

Response: ! = success
? = syntax error

Description: This command writes to the COS Mask register. When enabled, the mask is applied to the 8 input signals. A '1' enables a COS event, a '0 disables.

Command Example:

Command	Response	Description
$MW32<CR>	!	Inputs 6,5,2 are enabled for COS
$MWff<CR>	!	All Inputs are enabled for COS
$MW00<CR>	!	No Inputs are enabled for COS

Code:

```
/*********************************************************/
// Write COS Mask Method
//
/*********************/
private void writeCOSMaskMethod()
{
	if (DeviceA.Write("MW" + writeCOSMaskDataTextBox.Text) == 0)
	{
		writeCOSMaskStatusTextBox.Text = "Ok: Write";
	}
	else
	{
		writeCOSMaskStatusTextBox.Text = "Error: Write";
	}
}
/*********************************************************/
```

2.2.3.11 WDT Enable (temporary), write – **WE**

Syntax: $WE<CR>

Direction: Write

Argument: None

Reset: WDT is reset

Response: ! = success
? = syntax error

Description: This command enables the WDT function. The value is temporary and becomes active only after it is saved to EEPROM (WDT Configuration Save Sequence)

Command Example:

Command	Response	Description
$WE<CR>	!	WDT function enabled (activated after save to EEPROM)

Code:

```
/*********************************************************/
// Enable WDT Method
//
/*********************/
private void enableWDTMethod()
{
	if (DeviceA.Write("WE") == 0)
	{
		enableWDT_TextBox.Text = "Ok: Enable";
	}
	else
	{
		enableWDT_TextBox.Text = "Error: Enable";
	}
}
/*********************************************************/
```

2.2.3.12 WDT Disable (temporary), write – **WD**

Syntax: $WD<CR>

Direction: Write

Argument: None

Reset: WDT is reset

Response: ! = success
? = syntax error

Description: This command disables the WDT function. The value is temporary and becomes valid only after it is saved to EEPROM (WDT Configuration Save Sequence)

Command Example:

Command	Response	Description
$WD<CR>	!	WDT function disabled (activated after save to EEPROM)

Code:

```
/*********************************************************/
// Disable WDT Method
//
/*********************/
private void disableWDTMethod()
{
	if (DeviceA.Write("WD") == 0)
	{
		disableWDT_TextBox.Text = "Ok: Enable";
	}
	else
	{
		disableWDT_TextBox.Text = "Error: Enable";
	}
}
/*********************************************************/
```

2.2.3.13 WDT Switch Safe State (temporary), write – **WF*aa***

Syntax: $WF*aa*<CR>

Direction: Write

Argument: ***aa*** Switch Safe State Data.

Reset Value: 00h All Switches Open.

Response: ! = success
? = syntax error

Description: This command sets the WDT switch safe state. In the event of a WDT time-out, the output switches will be set to this state. The value is temporary and becomes valid only after it is saved to EEPROM (WDT Configuration Save Sequence)

Command Example:

Command	Response	Description
$SW3b<CR>	!	WDT switch safe state set as follows: Close switches 5, 4, 2, & 1. Open switches 8, 7, 6, 5, & 3.
$SW63<CR>	!	WDT switch safe state set as follows: Close switches 7, 6, 2, & 1 Open switches 8, 5, 4, & 3

Code:

```
/*********************************************************/
// Write WDT Safe Switch State Method
//
/*********************/
private void writeWDTSSSMethod()
{
	if (DeviceA.Write("WF" + writeWDT_SSSDataTextBox.Text) == 0)
```

```
        {
                writeWDT_SSS_StatusTextBox.Text = "Ok: Write";
        }
        else
        {
                writeWDT_SSS_StatusTextBox.Text = "Error: Write";
        }
}
/*********************************************************/
```

2.2.3.14 WDT Switch Safe State (read from EEPROM), read – **WG**

Syntax: $WG<CR>

Direction: Read

Argument: N/A

Reset Value: 00h All Switches Open.

Response: aa! Switch Safe State Data.
? = syntax error

Description: This command reads (from EEPROM) the active WDT switch safe state. In the event of a WDT time-out, the output switches will be set to this state.

Command Example:

Command	Response	Description
$WG<CR>	5c!	Switches 7,5,4,3 = Closed Switches 8,6,2,1 = Open
$WG<CR>	6F!	Switches 7,6,4,3,2,1 = Closed Switches 8,5 = Open
$WG<CR>	88!	Switches 8,4 = Closed Switches 7,6,5,3,2,1 = Open

Code:

```
/*********************************************************/
// Read WDT Safe Switch State Method
//
//**********************/
private void readWDTSSSMethod()
{
        string str1 = "";
        if (DeviceA.Read("WG", out str1) == 0)
        {
                readWDT_SSS_DataTextBox.Text = str1;
        }
        else
        {
                readWDT_SSS_DataTextBox.Text = "Error: Read";
        }
}
```

```
/*********************************************************/
```

2.2.3.15 WDT Time-Out Period (temporary), write – **WP*aa***

Syntax: $WP*aa*<CR>

Direction: Write

Argument: ***aa*** WDT Time-out period.

Reset Value: ffh, maximum time-out period.

Response: ! = success
? = syntax error

Description: This commands sets the WDT time-out period. Each bit represents a 100 mS increment. Range is from 100 mS (01h) to 25.5 seconds (ffh). The value is temporary and becomes valid only after it is saved to EEPROM.

Command Example:

Command	Response	Description
$WP05<CR>	!	Time-out Period set to 500 mS
$WP32<CR>	!	Time-out Period set to 5.0 S
$WP0a<CR>	!	Time-out Period set to 1.0 S

Code:

```
/*********************************************************/
// Write WDT Time-Out Period Method
//
/*********************/
private void writeWDT_TimeoutMethod()
{
	if (DeviceA.Write("WP" + writeWDT_TimeOut_DataTextBox.Text) == 0)
	{
		writeWDT_TimeOut_StatusTextBox.Text = "Ok: Write";
	}
	else
	{
		writeWDT_TimeOut_StatusTextBox.Text = "Error: Write";
	}
}
/*********************************************************/
```

2.2.3.16 WDT Time-Out Period (read from EEPROM), read – **WR**

Syntax: $WR<CR>

Direction: Write

Argument: N/A

Reset Value: ffh, maximum time-out period.

Response: ***aa***! WDT Time-out period

? = syntax error

Description: This command reads (from EEPROM) the active WDT time-out period. Each bit represents a 100 mS increment. Range is from 100 mS (01h) to 25.5 seconds (ffh).

Command Example:

Command	Response	Description
$WR<CR>	14!	Time-out Period set to 2.0 S
$WR<CR>	64!	Time-out Period set to 10.0 S
$WR<CR>	02!	Time-out Period set to 200 mS

Code:

```
/*******************************************************/
// Read WDT Time-Out Period Method
//
//*********************/
private void readWDT_TimeoutMethod()
{
        string str1 = "";
        if (DeviceA.Read("WR", out str1) == 0)
        {
                readWDT_TimeOut_DataTextBox.Text = str1;
        }
        else
        {
                readWDT_TimeOut_DataTextBox.Text = "Error: Read";
        }
}
/*******************************************************/
```

2.2.3.17 WDT Configuration Save Sequence. write – **WL*aa***

Syntax: $WL***aa***<CR>

Direction: Write

Argument: ***aa*** Save Code

Reset: N/A

Response: ! = success
? = syntax error

Description: This command is used to save temporary WDT parameters to non-volatile memory (EEPROM). It is issued in a three step sequence with the following arguments: 81h, 16h, and 79h. Following the sequence,

temporary WDT parameters become valid after execution of a Reset JI-4516 command ($XX<CR>) or after JI-4516 power cycling. The save sequence is as follows:

1. Send an 81h, $WL81<CR>
2. Wait for a valid response, !
3. Send a 16h, $WL16<CR>
4. Wait for a valid response, !
5. Send a 79h, $WL79<CR>
6. Wait for a valid response, !
7. New WDT parameters are now saved to EEPROM and become valid after a reset or power cycling.

Code:

```
/*********************************************************/
// WDT Save Sequence Method
//
/*********************/
private void WDT_SaveSequenceMethod()
{
        // Begin Sequence //
        if (DeviceA.Write("WL" + "81") != 0)
        {
                WDT_SaveSequenceStatusTextBox.Text = "Error: Save";
                return;
        }
        if (DeviceA.Write("WL" + "16") != 0)
        {
                WDT_SaveSequenceStatusTextBox.Text = "Error: Save";
                return;
        }
        if (DeviceA.Write("WL" + "79") != 0)
        {
                WDT_SaveSequenceStatusTextBox.Text = "Error: Save";
                return;
        }

        // Delay 15 mS to allow for completion of EEPROM write cycle //
        Thread.Sleep(15);

        // Send Reset //
        if (DeviceA.Write("XX") != 0)
{
                WDT_SaveSequenceStatusTextBox.Text = "Error: Save";
        }
        else
        {
                WDT_SaveSequenceStatusTextBox.Text = "OK: Save";
        }
}
/*********************************************************/
```

2.2.3.18 WDT Start Sequence, write – **WS*aa***

Syntax: $WS*aa*<CR>

Direction: Write

Argument: *aa* Start Code

Reset: WDT is reset

Response: ! = success
? = syntax error

Description: This command is used to start the WDT function. It is issued in a three step sequence. The WDT must be enabled prior to the sequence start. The start sequence is as follows:

1. Send an 53h, $WS53<CR>
2. Wait for a valid response, !
3. Send a 96h, $WS96<CR>
4. Wait for a valid response, !
5. Send a 12h, $WS12<CR>
6. Wait for a valid response, !
7. WDT is now running.

Following this start sequence, the WDT Timer Service command must be issued cyclically at an interval less than the WDT Time-Out Period to avoid a WDT time-out event. In the event of a time-out, the above steps must be repeated to restart.

Code:

```
/*********************************************************/
// WDT Start Sequence Method
//
/**********************/
private void WDT_StartSequenceMethod()
{
// Begin Sequence //
       if (DeviceA.Write("WS" + "53") != 0)
       {
              WDT_StartSequenceStatusTextBox.Text = "Error: Start";
              return;
       }
       if (DeviceA.Write("WS" + "96") != 0)
       {
              WDT_StartSequenceStatusTextBox.Text = "Error: Start";
              return;
       }
       if (DeviceA.Write("WS" + "12") != 0)
       {
              WDT_StartSequenceStatusTextBox.Text = "Error: Start";
              return;
       }
       else
       {
              WDT_StartSequenceStatusTextBox.Text = "OK: Start";
       }
}
/*********************************************************/
```

2.2.3.19 WDT Timer Service, write – **WT**

Syntax: $WT<CR>

Direction: Write

Argument: None

Reset: WDT is reset

Response: ! = success
? = syntax error

Description: This command services the WDT. When the WDT is enable and running, issuing this command on a regular interval prevents a WDT time-out event. The service interval time must be less than the WDT time-out period.

Code:

```
/*************/
// WDT Timer Service
//
/*********************************************************/
private void startWDT_ServiceButton_Click(object sender, EventArgs e)
{
	WDT_ServiceTimer.Interval =
		Convert.ToUInt16(WDT_ServicePeriodTextBox.Text, 10);
	WDT_ServiceTimer.Enabled = true;
	WDT_ServiceTimer.Start();
	WDT_ServiceTextBox.Text = "Running";
}

private void stopWDT_ServiceButton_Click(object sender, EventArgs e)
{
	WDT_ServiceTimer.Stop();
	WDT_ServiceTimer.Enabled = false;
	WDT_ServiceTextBox.Text = "Stopped";
}

private void WDT_ServiceTimer_Tick(object sender, EventArgs e)
{
	if (DeviceA.Write("WT") != 0)
	{
		//Error!!//
		WDT_ServiceTextBox.Text = "Error: WDT";
	}
}
/*********************************************************/
```

2.2.3.20 Reset JI-4516, write – **XX**

Syntax: $XX<CR>

Direction: Read

Argument: None.

Response: ! = success

? = syntax error

Description: This command resets the JI-4516 hardware. All internal registers are set to their power-up reset state.

Command Example:

Command	Response	Description
$WR<CR>	14!	Time-out Period set to 2.0 S

Code:

```
/*********************************************************/
// Reset JI-4516 Method
//
/*********************/
private void resetJI4516Method()
{
	if (DeviceA.Write("XX") == 0)
	{
		resetJI4516TextBox.Text = "Ok: Reset";
	}
	else
	{
		resetJI4516TextBox.Text = "Error: Reset";
	}
}
/*********************************************************/
```

2.2.3.21 Version Register, read – **VV**

Syntax: $VV<CR>

Direction: Read

Argument: None.

Response: ***aa***! = success
? = syntax error

a (MSB) = Hardware version
a (LSB) = Firmware version

Description: This command returns the hardware and firmware versions of the JI-4516 module.

Command Example:

Command	Response	Description
$VV<CR>	B2!	Translation: HW Ver B, Firmware Ver 2
$VV<CR>	C5!	Translation: HW Ver C, Firmware Ver 5

Code:

```
/*********************************************************/
// Read Version Method
//
```

```
//*********************/
private void readVersionMethod()
{
	string str1 = "";
	if (DeviceA.Read("VV", out str1) == 0)
	{
		readVersionRegTextBox.Text = str1;
	}
	else
	{
		readVersionRegTextBox.Text = "Error: Read";
	}
}
/*********************************************************/
```

APPENDIX A

1. JI-4516A_NET DLL

The latest version of this file can be found on the Jupiter Instruments website (http://jupiteri.com/JI-4516_Files/USB_High-Voltage_High-Current_Digital_IO_top.html).

APPENDIX B

1. JI-4516_Test_Application2 C# Project File

The latest version of this file can be found on the Jupiter Instruments website (http://jupiteri.com/JI-4516_Files/USB_High-Voltage_High-Current_Digital_IO_top.html).