

# FBx Script Developer User Manual

Line	Time	Command/Argument
71		CONNECT FB3000_IP2
72		ALARM_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\AlarmEvent\Newest_Oldest_Alam\$D\$T CSV NEWTOOLD ALL
73		ALARM_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\AlarmEvent\Oldest_Newest_Alam\$D\$T CSV OLDTONNEW ALL
74		
75		EVENT_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\AlarmEvent\Newest_Event\$D\$T PDF NEWTOOLD ALL LEGAL
76		EVENT_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\AlarmEvent\Oldest_Event\$D\$T PDF NEWTOOLD ALL LEGAL
77		
78		TRANSHISTORY_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\TransHistory\TransHistory\$D\$T CSV NEWTOOLD ALL ALL BATC...
79		TRANSHISTORY_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\TransHistory\OldestTransHistory\$D\$T PDF OLDTONNEW ALL ALL...
80		
81		HISTORY_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\History\History_Newest\$D\$T PDF NEWTOOLD ALL ALL HOURLY,DAIL...
82		HISTORY_REPORT C:\Users\Public\Documents\Emerson\FieldTools\FBx\Reports\History\History_Oldest\$D\$T PDF OLDTONNEW ALL ALL HOURLY,DAILY,...
83		
84		DIAG_REPORT DebugLog C:\Users\Public\Documents\Emerson\Logs
85		

Variable Name	Value

## Application & Device Safety Considerations

### ▪ Reading these Instructions

Before operating a device or application, read these instructions carefully and understand their safety implications. In some situations, improper use may result in damage or injury. Keep this manual in a convenient location for future reference. Note that these instructions may not cover all details or variations in equipment or cover every possible situation regarding installation, operation, or maintenance. Should problems arise that are not covered sufficiently in the text, immediately contact Energy and Transportation Solutions (ETS) Customer Support for further information.

### ▪ Protecting Operating Processes

The failure of a device or application – for whatever reason – may leave an operating process without appropriate protection and could result in possible damage to property or injury to persons. To protect against this, review the need for additional backup equipment or provide alternate means of protection (such as alarm devices, output limiting, fail-safe valves, relief valves, emergency shutoffs, emergency switches, etc.). Contact ETS for additional information.

### ▪ Using Qualified Personnel

Installation, configuration, and any subsequent modifications to a device or application should only be performed by qualified, suitably trained personnel.

### ▪ System Training

A well-trained workforce is critical to the success of your operation. Knowing how to correctly install, configure, program, calibrate, and troubleshoot your Emerson equipment provides your engineers and technicians with the skills and confidence to optimize your investment. ETS offers a variety of ways for your personnel to acquire essential system expertise. Our full-time professional instructors can conduct classroom training at several of our corporate offices, at your site, or even at your regional Emerson office. You can also receive the same quality training via our live, interactive Emerson Virtual Classroom and save on travel costs. For our complete schedule and further information, contact the ETS Training Department at 800-338-8158 or email us at [education@emerson.com](mailto:education@emerson.com).

### ▪ Grounding Equipment

Ground metal enclosures and exposed metal parts of electrical instruments in accordance with relevant safety standards. For the USA, refer to OSHA rules and regulations as specified in *Design Safety Standards for Electrical Systems*, 29 CFR, Part 1910, Subpart S, dated: May 16, 1981 (OSHA rulings are in agreement with the National Electrical Code). For international locations, refer to IEC 60364-4-41: PROTECTION AGAINST ELECTRIC SHOCK. You must also ground mechanical or pneumatic instruments that include electrically operated devices such as lights, switches, relays, alarms, or chart drives. The chassis (or earth ground) lug provides a safe connection point to a customer-designated ground location for ESD and transient voltage suppression. Do not use the chassis ground lug for signal, common, or return connections. **Do not connect the chassis ground lug directly to a lightning arrester/lightning rod.** Do not run signal wiring in conduit or open trays with power wiring or near heavy electrical equipment. If shielded wiring is used, ground the shield of the signal wiring at any one point of the signal loop.

**Important:** Complying with the codes and regulations of authorities having jurisdiction is essential to ensuring personnel safety. The guidelines and recommendations in this manual are intended to meet or exceed applicable codes and regulations. If differences occur between this manual and the codes and regulations of authorities having jurisdiction, those codes and regulations must take precedence.

### ▪ Protecting from Electrostatic Discharge (ESD)

Any device contains sensitive electronic components which can be damaged by exposure to an ESD voltage. Depending on the magnitude and duration of the ESD, it can result in erratic operation or complete failure of the equipment. Ensure that you correctly care for and handle ESD-sensitive components.

### ▪ Ethernet Connectivity

This automation device is intended to be used in an Ethernet network which **does not** have public access. The inclusion of this device in a publicly accessible Ethernet-based network is **not recommended**.

### ▪ Returning Equipment

If you need to return any equipment to ETS, it is your responsibility to ensure that the equipment has been cleaned to safe levels, as defined and/or determined by applicable federal, state and/or local law regulations or codes. You also agree to indemnify ETS and hold ETS harmless from any liability or damage which ETS may incur or suffer due to your failure to ensure device cleanliness.

# Contents

**Chapter 1. Introduction – What is FBx Script Developer? 1**

---

- 1.1 How Are Scripts Used? ..... 1
- 1.2 Commands ..... 2
- 1.3 Local Variables ..... 2

**Chapter 2 – Starting FBx Script Developer and Creating a Script 3**

---

- 2.1 Before You Begin ..... 3
- 2.2 Starting the FBx Script Developer ..... 3
- 2.3 Creating a Simple Script ..... 3
- 2.4 Saving the Script ..... 6
- 2.5 Modifying Lines of a Script ..... 6
- 2.6 Making a Copy of the Current Script File with a Different Name ..... 7
- 2.7 Entering Arguments in the Command Arguments dialog box ..... 7
- 2.8 Using the Set Arguments dialog box ..... 8
- 2.9 Creating an all New Script ..... 9
- 2.10 Opening an Existing Script ..... 9
- 2.11 Printing the Script ..... 9
- 2.12 Packaging a Script and its Associated Files ..... 9

**Chapter 3. Running Scripts 11**

---

- 3.1 Launching a Script from Within Field Tools ..... 11
  - 3.1.1 Generating a Support Bundle for a Device ..... 12
- 3.2 FBxScriptRunner Command Line Arguments ..... 13
  - 3.2.1 Example – Collecting Alarms and Events ..... 14
    - 3.2.1.1 Including a Date/Time Stamp in the Output Filename ..... 15

**Chapter 4. Debugging Scripts 17**

---

- 4.1 Error Reporting ..... 17
- 4.2 Working with Breakpoints ..... 18
  - 4.2.1 Setting a Breakpoint ..... 18
  - 4.2.2 Stepping through the Script using Step Mode ..... 19
  - 4.2.3 Clearing a Single Breakpoint ..... 19
  - 4.2.4 Clearing All Breakpoints ..... 19

4.3	Manually Breaking Execution of the Currently Executing Script.....	19
4.4	Stopping Execution of the Currently Executing Script.....	19

---

## Appendix A. – Script Commands 21

---

A.1	Output File Commands .....	21
A.1.1	OUTPUTFILE <Function> <Target file>.....	21
A.1.2	WRITE <Text> [<Text> ....].....	22
A.1.3	WRITE_WITH_STAMP <Text> [<Text> ....].....	23
A.1.4	IMPORT <File path> .....	23
A.1.1	DIR <Function> <Folder>.....	24
A.1.2	ZIP <Folder to Zip> <Zip file name> .....	25
A.2	Script Control Commands.....	25
A.2.1	RUN <File path> [<arg1> <arg2> ...].....	25
A.2.2	GOTO <Label> .....	26
A.2.3	PAUSE <Number of seconds to wait> .....	26
A.2.4	ABORT [<Text> <Text> ... ].....	27
A.2.5	EXIT [<Text> <Text> ... ].....	27
A.2.6	PROMPT <Message><Show entry text box><top button text><bottom button text> 28	
A.2.7	ECHO <Text>[ <Text> ...].....	30
A.3	Local Variable Commands .....	30
A.3.1	VAR_SET <Destination Variable> <Value or Variable>.....	31
A.3.2	VAR_ADD <Destination Variable> <Value or Variable> .....	31
A.3.3	VAR_SUB <Destination Variable> <Value or Variable>.....	32
A.3.4	VAR_MULT <Destination Variable> <Value or Variable>.....	33
A.3.5	VAR_DIV <Destination Variable> <Value or Variable>.....	34
A.3.6	VAR_DECR <Variable> .....	35
A.3.7	VAR_INCR <Variable>.....	35
A.3.8	IF <Left Operand Variable> <Condition> <Right Operand Variable or Value> <Label to jump when condition is true>.....	36
A.4	Device Connection Command .....	37
A.4.1	CONNECT <connection name>.....	37
A.5	Device System Commands.....	38
A.5.1	WARM_START <Time to wait for device to reboot>.....	39
A.5.2	COLD_START <start type> <timeout> .....	39
A.5.3	TIME_SYNCH .....	40
A.6	Device Parameter Commands.....	40
A.6.1	READ_PARAM <Parameter name> <Variable name> .....	41
A.6.2	WRITE_PARAM <Parameter name> <Data type> <Value or Variable> .....	41
A.6.3	RECIPE <Function> <Recipe file> .....	43
A.7	Device File Commands .....	44
A.7.1	FILE_DELETE <Device file path> <File Not Found flag>.....	45
A.7.2	FILE_DOWNLOAD <PC file path> <Device file path> .....	45
A.7.3	FILE_UPLOAD <Device file path> <PC file path> [<File Not Found flag>].....	46
A.8	Device Application Commands.....	47
A.8.1	GET_APPS_INFO [<slot>].....	47

---

- A.8.2 APPLICATION <Function> <Slot> .....48
- A.9 Device Firmware Commands .....49
  - A.9.1 UPDATE\_FIRMWARE <Firmware file> [<Time to wait for device to reboot>] ..49
- A.10 Solution Commands .....49
  - A.10.1 SOLUTION\_UPLOAD <File path> .....49
  - A.10.2 SOLUTION\_DOWNLOAD <File path> .....50
  - A.10.3 FLASH\_DOWNLOAD <File path>.....50
  - A.10.4 PARTIAL\_DOWNLOAD <File path> .....51
  - A.10.5 PARTIAL\_UPLOAD <Objects & Instances file> <Partial Configuration File> ..51
  - A.10.6 FLASH\_SAVE.....53
- A.11 Device Report Generation Commands.....53
  - A.11.1 DIAG\_REPORT <ClearDump or ReportType> <File Name> .....53
  - A.11.2 ALARM\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From Time> <End\_Time>] .....54
    - A.11.1 EVENT\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From time> <End\_time>] [<Sub-type>].....56
    - A.11.1 HISTORY\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From time> <End\_time>] <History Group(s)> <Interval(s)> .....57
    - A.11.1 TRANSHISTORY\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From time> <End\_time>] <History Group(s)> <Sub-type(s)> .....59

# FBx Script Developer User Manual

D301953X012

November 2024

---

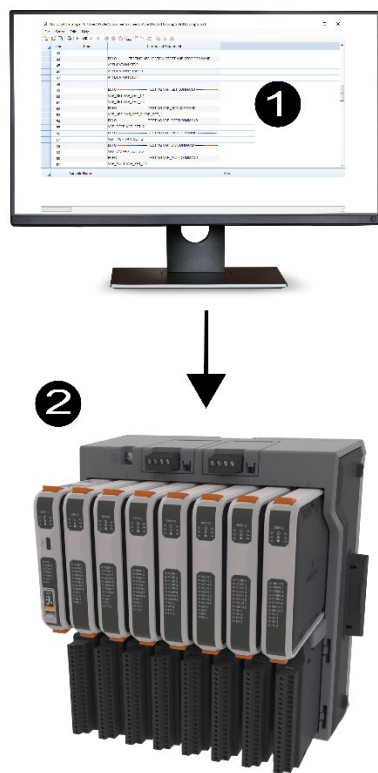
# Chapter 1. Introduction – What is FBx Script Developer?

FBx Script Developer allows you to create pre-defined sets of instructions – called **scripts** – which you can use to access an FB3000 RTU or an FB1100/FB1200/FB2100/FB2200 flow computer.

You edit scripts directly in FBx Script Developer.

Scripts have the file extension \*.SCR. The FBx Script Developer stores the scripts on the PC.

**Figure 1-1. FBx Script Developer Overview**



- 1 Script executes on the PC/laptop
- 2 Commands from the script sent to the device (RTU or flow computer)

## 1.1 How Are Scripts Used?

Scripts are typically used for testing and to verify that the RTU/flow computer is

operating as expected.

Scripts can also serve as a method for automating certain day-to-day tasks you might need to perform. For example, you might want to create a script that collects periodic history records from your flow computers. You can test it out in the FBx Script Developer, and then save the script (.SCR) file.

You can launch a script manually through Field Tools, or you can run it from the command line using FBxScriptRunner.

## 1.2 Commands

*Appendix A* includes a full list of the available commands you can use in your scripts.

When you create your scripts, you can type the command parameters in directly, or optionally, you can click the Command icon and enter the text there, or depending on the command, make selections that enter the text for you.

## 1.3 Local Variables

Since scripts are essentially programs that FBx Script Developer can run, scripts can include local variables to store values. Based on these values you can make logical decisions for executing portions of the script or use the local variables as arguments for particular script commands.



# Chapter 2 – Starting FBx Script Developer and Creating a Script

## 2.1 Before You Begin

When you install Field Tools, you automatically install FBx Script Developer.

## 2.2 Starting the FBx Script Developer

To start the FBx Script Developer, click: **Start > Emerson Field Tools > FBxScripting**.

**Note**

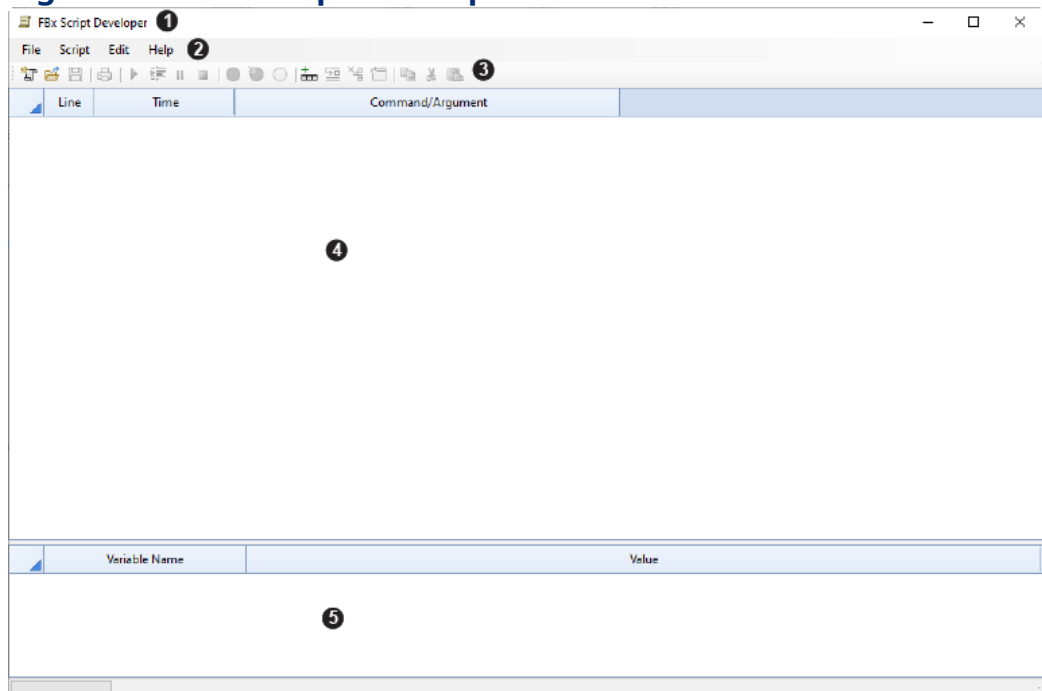
You can also start the tool from the command line. Typically, you would use this technique to automate execution of scripts you already created. See *Chapter 3* for more information.

## 2.3 Creating a Simple Script


These steps outline how to make a very simple script.

1. Start FBx Script Developer by clicking: **Start > Emerson Field Tools > FBxScripting**. FBx Script Developer opens:

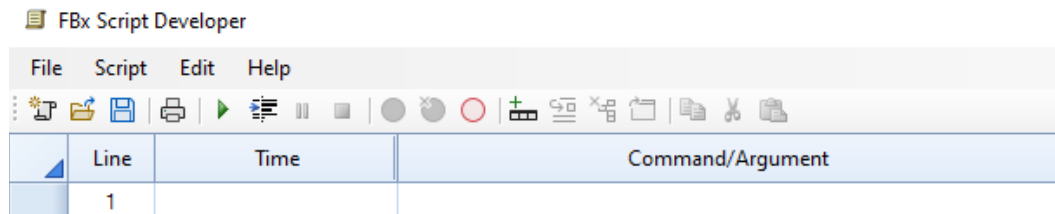
**Figure 2-1. FBx Script Developer Screen**



- 1 Title bar
- 2 Menu bar
- 3 Toolbar
- 4 Script pane
- 5 Variable status pane

2. Either click **Script > Commands > Add Command** or click the Add command  toolbar icon to add a line for the script.

## Figure 2-2. Inserting a Line into the Script



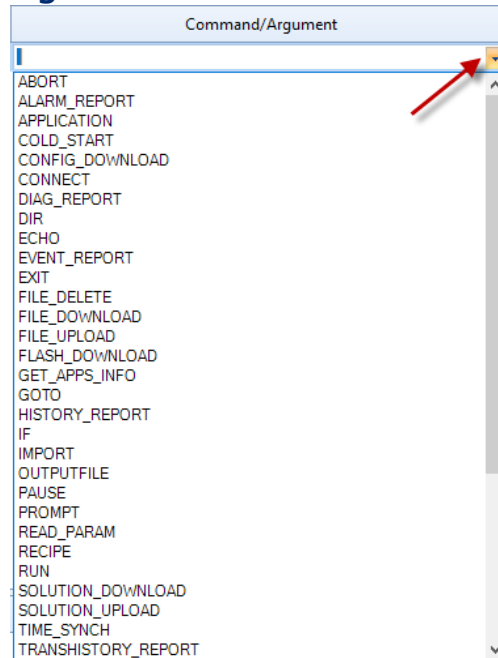
Line The line number of the script

Time The timestamp when the command is executed

Command/Argument The Command name and any arguments.

3. Click in the center of the Command/Argument column, then click on the down arrow to open the command selection menu.

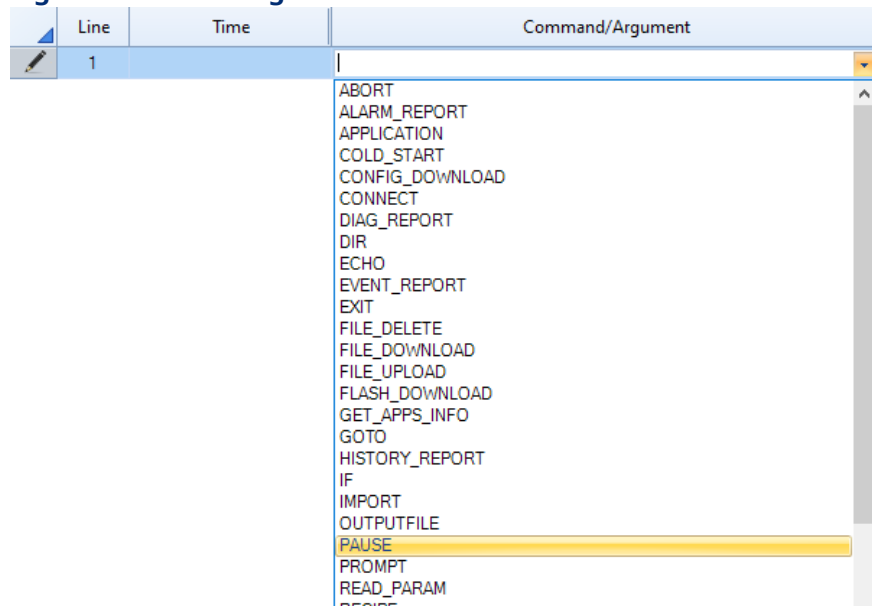
## Figure 2-3. Command Selection Menu




4. Scroll through the command list and select the first command. In this case, we

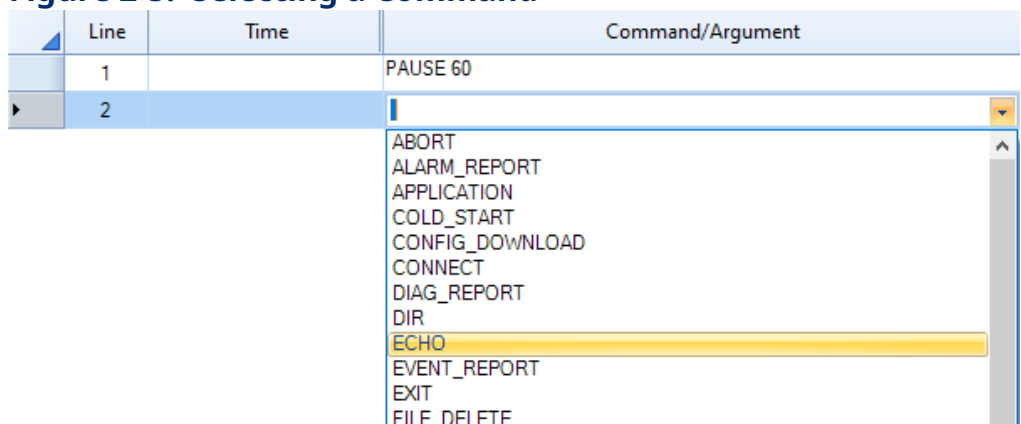
choose the PAUSE command, which pauses execution of a script for a specified number of seconds.

**Figure 2-4. Selecting a Command**



- Now, enter "60" which means that execution should pause for 60 seconds and press the [Enter] key.
- Click the Add command  icon to select another command, in this case we choose the ECHO command, which displays whatever text is specified after it.

**Figure 2-5. Selecting a Command**




Line	Time	Command/Argument
1		PAUSE 60
2		ECHO

- Now enter the text "Hello everybody" after the ECHO command and press [Enter].

**Figure 2-6. Entering an argument for a command**

Line	Time	Command/Argument
1		PAUSE 60
2		ECHO Hello everybody

8. We can now test this simple 2-line script. Click the “Run”  toolbar icon or click **Script > Run > Run**. The script starts and you can watch the timing of the PAUSE command count down in the Status field.

**Figure 2-7. Script Executes a PAUSE command**

Line	Time	Command/Argument	Status
1	04/25/2024 14:06:58 PM	PAUSE 60	Waiting 00:00:57.667
2		ECHO Hello everybody	

9. When the 60 seconds have expired, the Status column shows the echoed text “Hello everybody:”

**Figure 2-8. Script Executes an ECHO command**

Line	Time	Command/Argument	Status
1	04/25/2024 14:07:56 PM	PAUSE 60	Pause Complete
2	04/25/2024 14:07:56 PM	ECHO Hello everybody	Hello everybody

10. When you finish adding commands to the script, save the script.

## 2.4 Saving the Script

Click **File > Save** or click the Save  icon.

Enter a name for the script and click **Save**.

FBx Script Developer saves the script with an extension of .SCR.



By default, scripts are saved in the folder:



\\Users\Public\Public Documents\Emerson\FieldTools\FBxScripts\

## 2.5 Modifying Lines of a Script

When you edit a script, you can use standard editing techniques to cut, paste, copy, etc. entire lines in a script. The table, below, shows the supported editing commands:

**Table 2-1. Editing Functions**

Editing Function	Toolbar Icon	Menu Bar Sequence	Description
Cut		Edit > Cut	Cuts the selected line of the script and stores it in the paste buffer.
Copy		Edit > Copy	Copies the selected line of the script and

Editing Function	Toolbar Icon	Menu Bar Sequence	Description
			puts it in the paste buffer.
Paste		Edit > Paste	Pastes the contents of the paste buffer to the current script line.
Insert Command		Script >Commands > Insert Command	Inserts a blank line in the script between two other lines.


## 2.6 Making a Copy of the Current Script File with a Different Name

Click **File > Save As**.

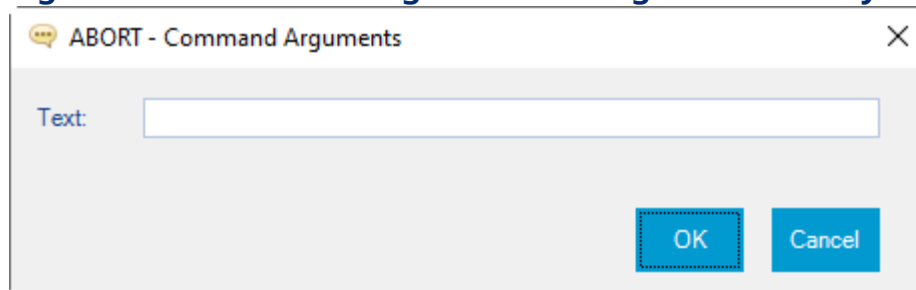
Enter a name for the script and click **Save**.

## 2.7 Entering Arguments in the Command Arguments dialog box

While you can enter some arguments directly after the command, as we showed in *Section 2.3*, for anything more than the simplest commands, you should use the Command Arguments dialog box.

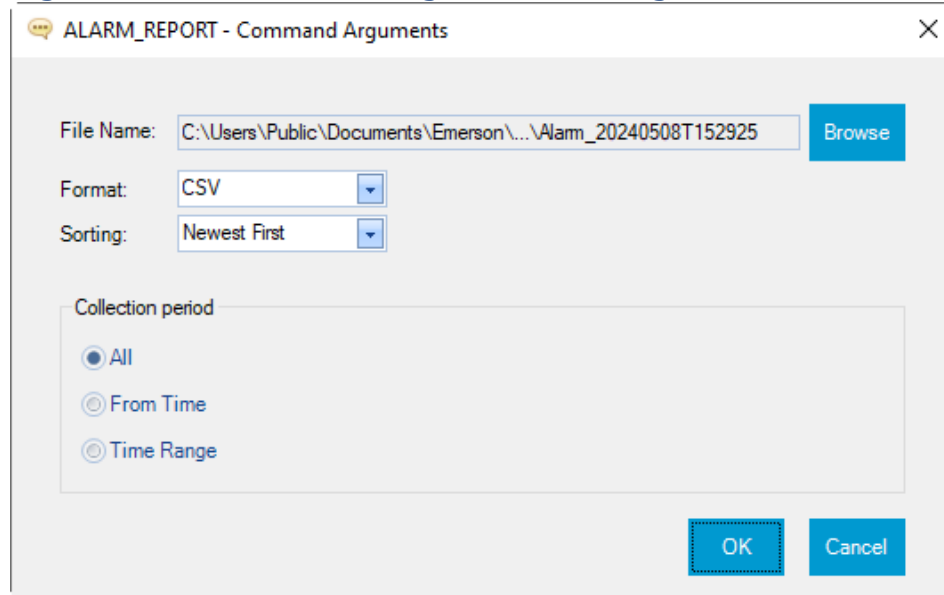
1. Enter the command on the desired line of the script and press the **[Enter]** key.
2. Position the cursor in the line after the command and click the Command  icon.
3. This launches a dialog box that allows you to enter arguments. The appearance of the dialog box varies depending on the command. In some cases, you enter arguments in a text field:


**Figure 2-9. Command Arguments dialog box -text only**



In other cases, you make selections to generate the text for you:

**Figure 2-10. Command Arguments dialog box – make selections**



4. When you finish editing the script, you can run the script as we did, directly in the FBx Script Developer using the Run  icon. Alternatively, you can run the script by invoking FBxScriptRunner from the command line. See *Chapter 3* for information on command line options.

---

### Note

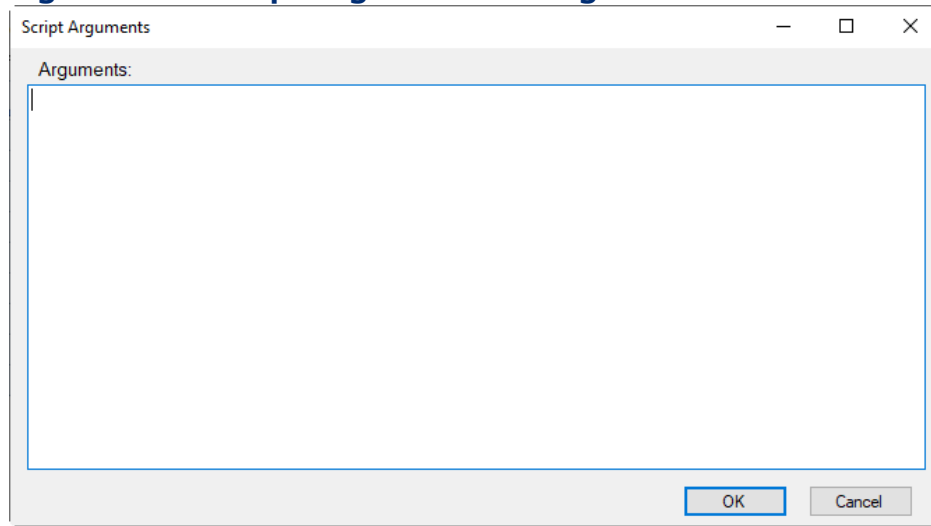
Because there are dozens of different commands you can include in a script, it is fairly easy for you to create very complex scripts that might be hundreds of lines long. If you want to create lengthy scripts, you should review the debugging techniques in *Chapter 4*.

---

## 2.8 Using the Set Arguments dialog box

You can set arguments for the entire script using the Set Arguments dialog box.

1. Click **Script > Set Arguments**


**Figure 2-11. Script Arguments dialog box**

2. Enter the arguments for the script, and click **OK**.

## 2.9 Creating an all New Script

Click the New Script  icon or click **File > New**.

## 2.10 Opening an Existing Script


Click the Open Script  icon or click **File > Open** and select the script you want to open. By default, scripts are stored in the folder:

**\Users\Public\Public Documents\Emerson\FieldTools\FBxScripts**

If the script you want to open was open recently, you can click **File > Recent** and select the desired script.

You can also open a script file by double-clicking on the SCR file.

## 2.11 Printing the Script

Click the Print Script  icon, or click **File > Print** to open the Windows print dialog box and make your selections to print the current script.

## 2.12 Packaging a Script and its Associated Files

If desired, you can combine a script, together with the files it uses (.ZSL solution files, recipe files) into a single zipped file (\*.SCZ), place the script and files in a folder, then click **File > Package** and browse to the folder containing the files, then select the folder to create the package file.

# FBx Script Developer User Manual

D301953X02

November 2024

---



# Chapter 3. Running Scripts

While you can run scripts manually in the FBx Script Developer, that’s really only intended for trying out the script and debugging it.

The main ways you might run a script are:

- Run the script manually from within the Field Tools tree.
- Run the script from the command line or possibly using a third-party scheduling program.

Whether invoked from the FBx Script Developer, from the command line, or from the Field Tools tree, the scripts are executed by the FBxScriptRunner program.

## 3.1 Launching a Script from Within Field Tools

You can launch a script you created from within the Field Tools tree:

1. Right-click on the device on which you want to run the script, and select **Run Script**. (Alternatively, you can single click on the device to highlight it and then click the **Run Script** button.)

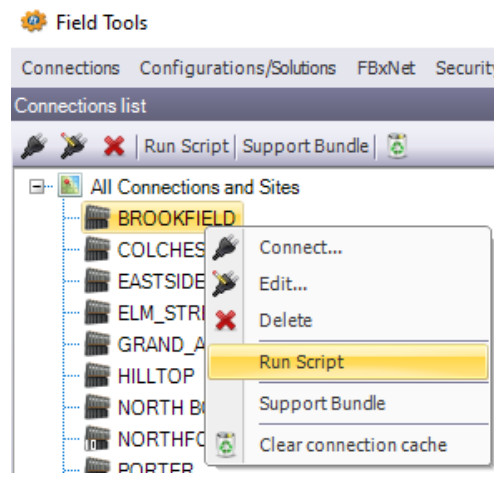
---

### Note

If your script includes a “CONNECT {\$A1}” command, the connection name is automatically passed to the script to fill in the A1 parameter.

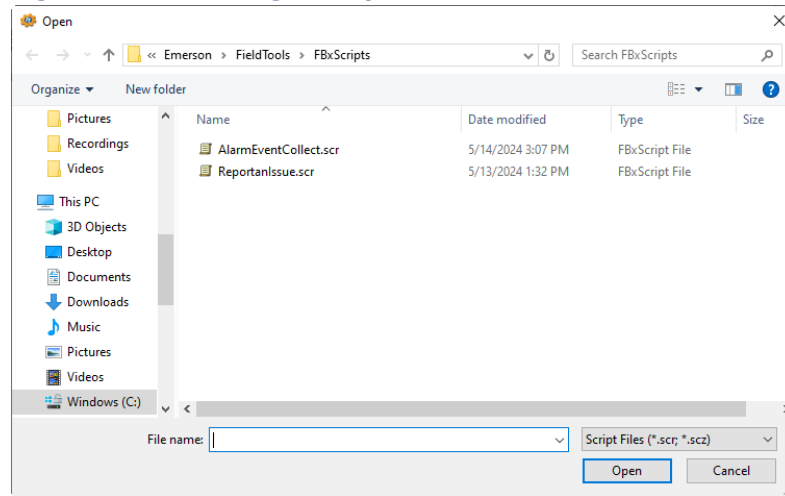
---

**Figure 3-1. Launching a Script within Field Tools**



2. Select the script file you want to run.

**Figure 3-2. Selecting a Script File**



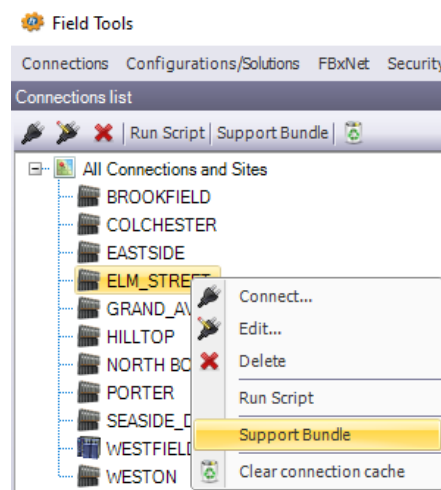
3. Click **Open** and FBxScriptRunner launches the script.

## 3.1.1 Generating a Support Bundle for a Device

If you are experiencing problems with a device, a predefined script exists that you can use to gather diagnostic and system information in a ZIP file called a **support bundle** that you can forward to our technical support group for analysis.

1. Right-click on the device for which you want to gather support information and select **Support Bundle**. Alternatively, you can single click on the device to highlight it and click the **Support Bundle** button.

**Figure 3-3. Generating a Support Bundle**



2. The script launches to collect diagnostic information, memory dumps, alarm and event information and more.
3. The collected information is stored in a zip file. By default, the support bundle zip file is stored in the \\Emerson\Logs folder:

\Users\Public\Public Documents\Emerson\Logs\connection\_name\_Issue\_timestamp

## 3.2 FBxScriptRunner Command Line Arguments

You might want to use a third-party scheduling program to launch FBxScriptRunner from the command line to handle certain day-to-day tasks.

The syntax for launching FBxScriptRunner is as follows:

C:\>**FBxScriptRunner** *script\_name* [*arg1 arg2 arg3 ... argn*]

*script\_name* the name of the script file including the extension (.SCR or .SCZ).  
*arg1... argn* arguments used by commands in the script file, where “*n*” is the number of the last argument for this script. Using arguments, allows you to substitute parameters as needed, so you can make the scripts more versatile. The same script you use, for example, to communicate with RTU1, can be re-used with RTU2 and RTU3, by passing in the RTU name as an argument.

FBxScriptRunner stores the arguments in local string variables so they can be referenced by commands in your script file. Local string variable names must be preceded by a dollar sign “\$” to identify them as string variables.

Local String Variable Name:	Argument:
\$A1	<i>arg1</i>
\$A2	<i>arg2</i>
\$A3	<i>arg3</i>
:	
\$An	<i>argn</i>

### Notes

- FBxScriptRunner updates a numeric local variable with a count of the number of arguments used called ARG\_CNT. ARG\_CNT does not include the script name as one of the arguments in the count.
- Local variables, such as these arguments, may be used in script commands by enclosing them in curly brackets. For example, to use arg1 with the PAUSE command, you enter **PAUSE** {\$A1}
- If an argument includes spaces, you must enclose it in double quotes “ ”.

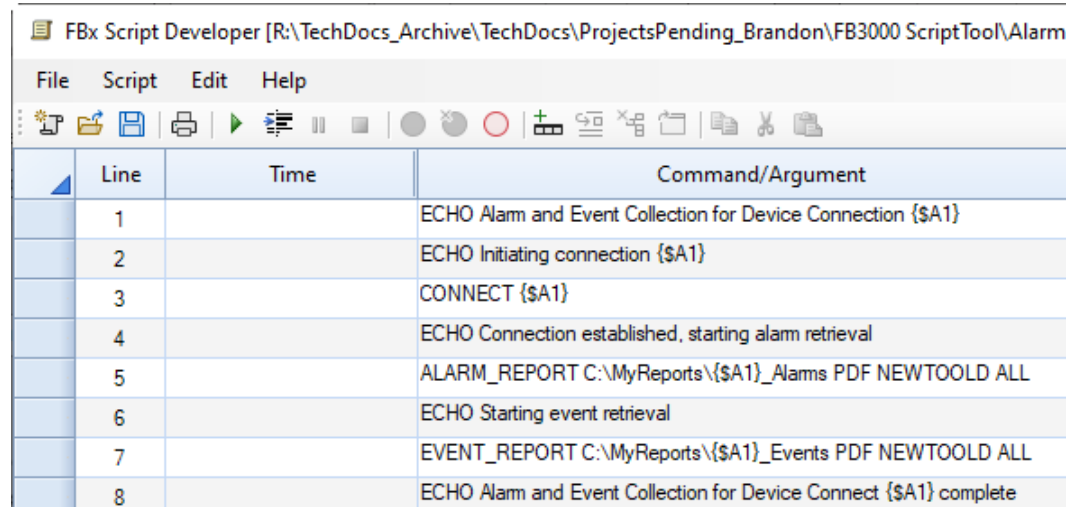
## 3.2.1 Example – Collecting Alarms and Events

Let's say we want to create a script that lets us connect to an RTU or flow computer and retrieve its alarm history and event history while also sending messages to the screen reporting what it's doing.

We're going to need to use the CONNECT command to connect to the device, the ECHO command to report what the script is doing, and the ALARM\_REPORT and EVENT\_REPORT commands to collect the alarms and events and store them in files.

1. Make sure Field Tools communications are active.
2. Start the FBx Script Developer and enter the script as follows. Note that we are using the argument \$A1 to hold the device connection name for the RTU or flow computer.

**Figure 3-4. Example Script to Collect Alarms and Events**



The screenshot shows the FBx Script Developer window with the following script content:

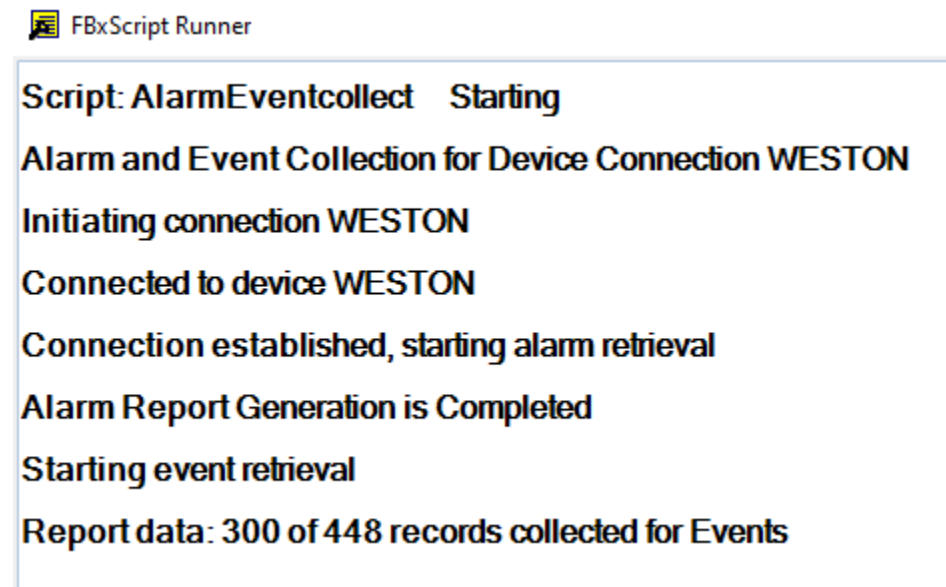
Line	Time	Command/Argument
1		ECHO Alarm and Event Collection for Device Connection {A1}
2		ECHO Initiating connection {A1}
3		CONNECT {A1}
4		ECHO Connection established, starting alarm retrieval
5		ALARM_REPORT C:\MyReports\{A1}_Alarms PDF NEWTOOLD ALL
6		ECHO Starting event retrieval
7		EVENT_REPORT C:\MyReports\{A1}_Events PDF NEWTOOLD ALL
8		ECHO Alarm and Event Collection for Device Connect {A1} complete

3. Save the script with the name AlarmEventCollect.scr.
4. Open a command prompt and move to the folder where you saved the script.
5. Invoke the FBxScriptRunner with the script name and the connection name for the device. In this case, we connect to a device connection named "WESTON." We give the script name followed by the argument "WESTON" to tell which connection to activate.

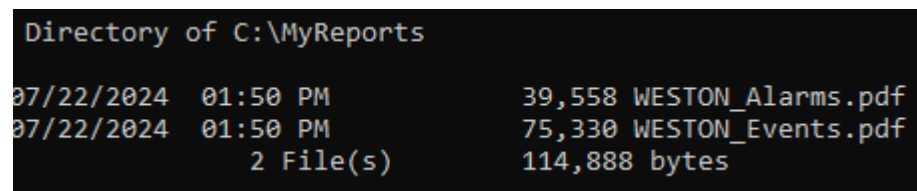
```
C:\Myscripts>FBxScriptRunner AlarmEventCollect.scr WESTON
```

The FBxScriptRunner window opens and reports the progress of the script:

**Figure 3-5. FBxScript Runner Output**

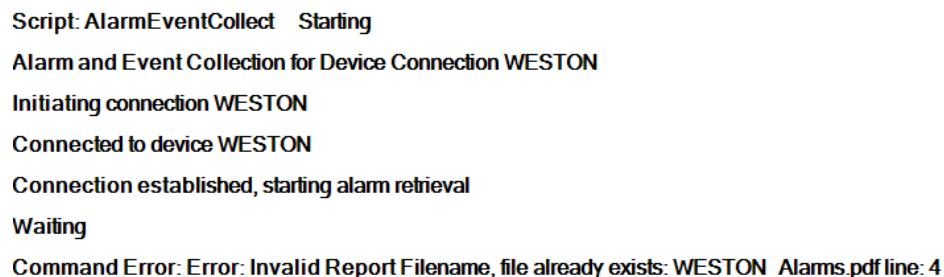


When the script completes execution, two PDF files (1 for alarms, 1 for events) are created in the specified output folder, in this case C:\MyReports:



### 3.2.1.1 Including a Date/Time Stamp in the Output Filename

If you were to run the exact same script, twice in a row with the same device, in this case, "WESTON" there will be a problem and an error is generated:



Why does this happen? Because the report you are trying to generate already exists. You can avoid this, however, by making some small changes to your script. If you change the lines:

## FBx Script Developer User Manual

D301953X012

November 2024

---

ALARM\_REPORT C:\MyReports\{\$A1}\_Alarms PDF NEWTOOLD ALL

EVENT\_REPORT C:\MyReports\{\$A1}\_Events PDF NEWTOOLD ALL

Include the text: “**\$D\$T**” to include a timestamp in the filename:

ALARM\_REPORT C:\MyReports\{\$A1}\_**\$D\$T**\_Alarms PDF NEWTOOLD ALL

EVENT\_REPORT C:\MyReports\{\$A1}\_**\$D\$T**\_Events PDF NEWTOOLD ALL

Now the report filenames will include a timestamp and there will not be any name conflicts:

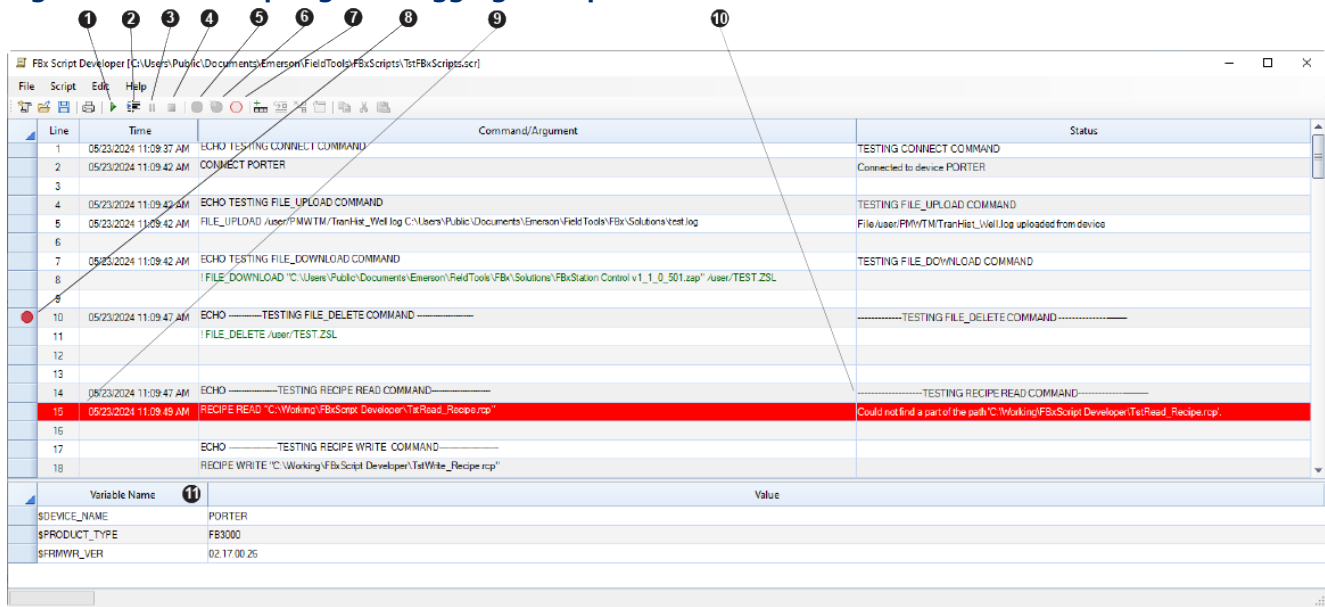
```
Directory of C:\MyReports
07/22/2024  03:57 PM    <DIR>          .
07/22/2024  03:57 PM    <DIR>          ..
07/22/2024  03:54 PM           39,579 WESTON_20240722T155410_Alarms.pdf
07/22/2024  03:54 PM           80,754 WESTON_20240722T155412_Events.pdf
07/22/2024  03:54 PM           39,577 WESTON_20240722T155457_Alarms.pdf
07/22/2024  03:55 PM           81,302 WESTON_20240722T155459_Events.pdf
07/22/2024  03:55 PM           39,573 WESTON_20240722T155552_Alarms.pdf
07/22/2024  03:55 PM           81,888 WESTON_20240722T155554_Events.pdf
07/22/2024  03:57 PM           39,582 WESTON_20240722T155745_Alarms.pdf
07/22/2024  03:57 PM           82,462 WESTON_20240722T155747_Events.pdf
                8 File(s)         484,717 bytes
                2 Dir(s)  13,204,393,984 bytes free
```

# Chapter 4. Debugging Scripts


## 4.1 Error Reporting

If FBx Script Developer encounters a problem when it executes a line of your script, it highlights that line in red, and displays an error message in the **Status** column. This helps you identify common syntactical errors.

**Figure 4-1. FBx Scripting – Debugging a Script**




- 1 Run script (or resume after a break)
- 2 Step to next line of script but do not execute
- 3 Break (pause) script execution
- 4 Stop script execution entirely
- 5 Set breakpoint
- 6 Remove a single breakpoint
- 7 Remove all breakpoints
- 8 Breakpoint active
- 9 Line containing an error highlighted in red
- 10 Error message
- 11 Variables pane – shows current values of local variables

You can resume execution after the line containing the error by clicking on the Run  icon.

Although the FBxScriptRunner detects syntactical errors during execution, sometimes script errors relate more to the logic of the script. To solve those kinds of problems often requires you to look at more than just the current script line. You might need to pause execution at a particular point, and then proceed line by line, while examining the state of different script variables. To do this, you need to use the breakpoint feature and step mode.


## 4.2 Working with Breakpoints

A **breakpoint** is a flag that you place on a line at which you would like execution to pause. As the script executes, it pauses on the line containing the breakpoint, but does not execute the command on the line. This allows you to look at the Variables pane to view the state of script variables immediately before the breakpoint. When the execution pauses, the script enters **step mode**. Step mode allows you to execute one script line at a time, by clicking on the Step  icon.


By breaking execution at certain points, and then looking at script variables, it is possible to better understand how the script works and identify errors in the script's logic. As you advance line by line, you may be able to identify the source of a problem so you can correct it.

### 4.2.1 Setting a Breakpoint

There are three different ways to set a breakpoint:

- Click anywhere in the line of the script where you want the breakpoint, then click on the Breakpoint  icon, *-or-*
- Click anywhere in the line of the script where you want the breakpoint, then click **Script > Breakpoint > Set Breakpoint**.
- Double-click in the leftmost column of the line where you want the breakpoint.


Line	Time	Command/Argument
1		ECHO Alarm and Event Collection for Device Connection WESTON
2		ECHO Initiating connection WESTON
3		CONNECT WESTON
4		ECHO Connection established, starting alarm retrieval
5		ALARM_REPORT C:\MyReports\WESTON_Alarms PDF NEWTOOLD ALL
6		ECHO Starting event retrieval

All of these methods place a stop  flag in the leftmost column. This means that line has a breakpoint.

The next time you execute the script, execution pauses on the line containing the flag, prior to executing the command on that line.




## 4.2.2 Stepping through the Script using Step Mode

Once execution pauses on a particular line of the script, you can proceed one line at a time by clicking on the Step icon  or click **Script > Run > Step**.

You can now look in the Variables pane to see how values of local variables in the script change.

## 4.2.3 Clearing a Single Breakpoint


Click in the **Line** containing the breakpoint, then click the Remove Breakpoint  icon or click **Script > Break point > Remove Breakpoint**.


Either of these methods remove the  flag for that line and clear the breakpoint.

## 4.2.4 Clearing All Breakpoints


To clear all breakpoints in the script, click the Remove All Breakpoints  icon, or click **Script > Break point > Remove all Breakpoints**.

## 4.3 Manually Breaking Execution of the Currently Executing Script


To break (pause) the currently executing script, click the Break  icon or click **Script>Run>Break**.

You can then proceed to make edits to the script. To resume the script, from the point where it stopped, click the Run  icon.

## 4.4 Stopping Execution of the Currently Executing Script

To stop the currently executing script, click the Stop  icon or click **Script>Run>Stop**.

You can then proceed to make edits to the script or run a new script.

If you then click the Run  icon, the script restarts from line 1.

# FBx Script Developer User Manual

D301953X012

November 2024

---


# Appendix A.– Script Commands

The following conventions apply to these scripts:

< >            Replace with required text argument.

[ ]             Encloses optional arguments.

**CMD**            Command name shown **CAPITALIZED AND BOLD**.

For the simplest commands, you can select the commands and then enter arguments directly in the **Command/Argument** column of FBx Script Developer. For more complex commands, select the command and then click the command  icon to launch the Command Arguments dialog box, and make your selections.

## A.1 Output File Commands

Scripts can write to a local output file on the PC. The file can be used to log the progress/status of the script, or to write other information to the file.

---

### Note

A script may only have one output file open at any time.

---

### A.1.1 OUTPUTFILE <Function> <Target file>

Writes status or other information to a local file on the PC.

<Function>            is one of the following:

WRITE                Open a new file for writing.

APPEND              Append to an existing file.

CLOSE                Close the file.

DELETE               Delete the file. Can delete any file on the PC.

<Target file>        specifies the path and filename of the output file. **The drive and folder must already exist.**



Examples:

The following command creates a new file on the PC called Scriptstatus.log in the C:\Myscripts folder:

```
OUTPUTFILE WRITE C:\Myscripts\Scriptstatus.log
```

The following command opens an existing file called Script27.txt in the folder C:\Myscripts for writing:

```
OUTPUTFILE APPEND C:\Myscripts\Script27.txt
```

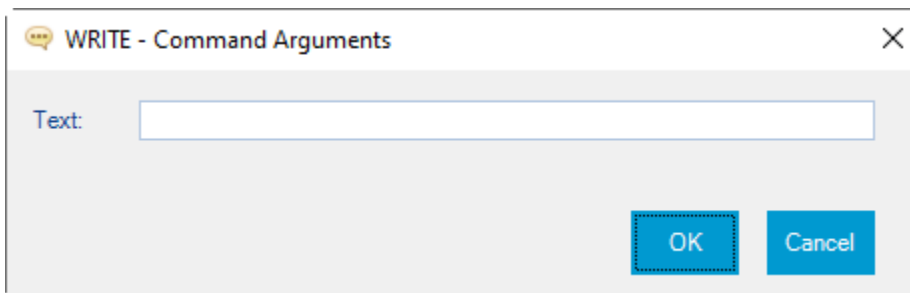
The following command closes the file Script25.txt in the folder C:\Myscripts:

```
OUTPUTFILE CLOSE C:\Myscripts\Script25.txt
```

## A.1.2 WRITE <Text> [<Text> ....]

Writes each text argument to the currently open file. The text arguments are combined and separated with single spaces.

<Text>           Contiguous characters of text written to the output file. If there are multiple <text> arguments, each argument is written to the output file, with a space separating each argument from the next argument.



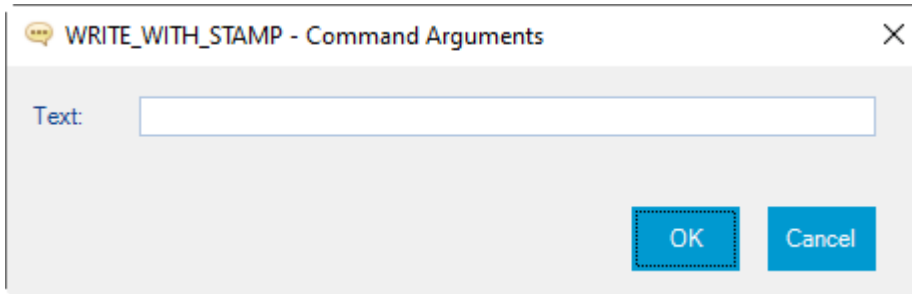
Example:

```
WRITE Write this text to the file.
```

### A.1.3 WRITE\_WITH\_STAMP <Text> [<Text> ....]

Writes the current time stamp to the currently open file, The text arguments are combined and separated with single spaces.

<text> Contiguous characters of text written to the output file. If there are multiple <text> arguments, each argument is written to the output file, with a space separating each argument from the next argument.



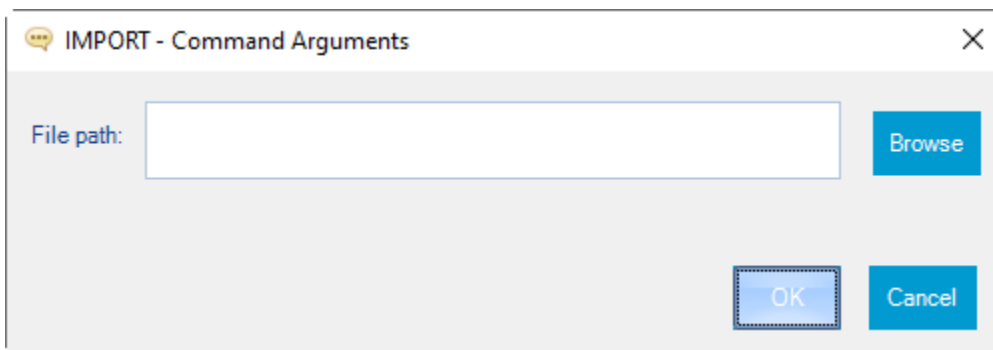
Example:

**WRITE\_WITH\_STAMP** Write this text to the file.

### A.1.4 IMPORT <File path>

Copies the contents of the specified text file and appends it to the end of the currently open output file.

<File path> specifies the path and filename on the PC of the file to be appended to the currently open output file. **The path and filename must exist.**



Example:

To import the contents of mydatafile.txt into the current output file:

**IMPORT** C:\>mydatafile.txt

## A.1.1 DIR <Function> <Folder>

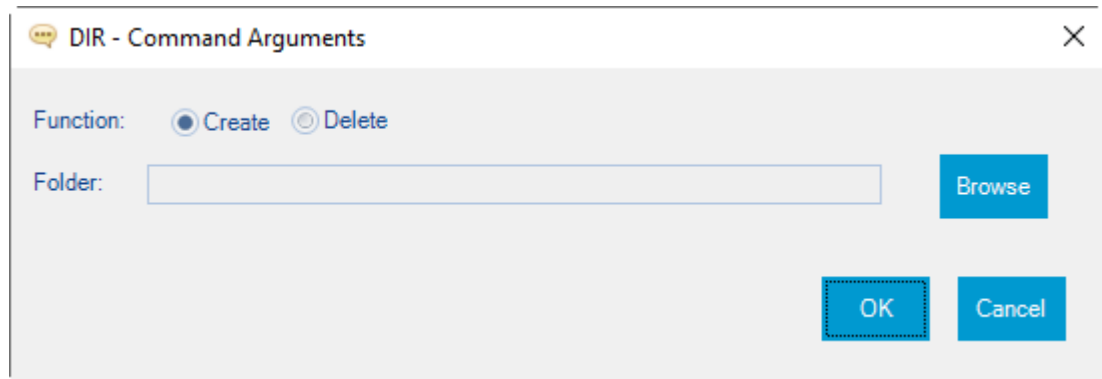
Creates or deletes a folder on the PC.

<Function> is one of the following:

CREATE Creates a new folder. The name of the newly created folder is stored in the \$DIR local variable>

DELETE Deletes an existing folder.

<Folder> is the folder name on the PC.



### Examples:

This creates a folder called logs in an existing folder called myfiles:

```
DIR CREATE C:\myfiles\logs
```

This deletes a folder called history from the C drive:

```
DIR DELETE C:\history
```

This creates a folder called "DailyReports", and once created, lets you use the folder name (using the \$DIR local variable) in another command, in this case, the ALARM\_REPORT command:

```
DIR CREATE C:\DailyReports
```

```
ALARM_REPORT {$DIR}\Alarm CSV
```

This would be equivalent to entering the command:

```
ALARM_REPORT C:\DailyReports\Alarm CSV
```

## A.1.2 ZIP <Folder to Zip> <Zip file name>

Makes a compressed copy of a folder and its contents and stores it in a single file with the extension \*.ZIP.

<Folder to Zip>      The name of the folder to be compressed, including any sub-folders.

<Zip file name>      The name to assign to the zipped file.



### Example:

To store a copy of the folder C:\myfiles\logs in a file called mylogs.zip:

**ZIP** C:\myfiles\logs mylogs

## A.2 Script Control Commands

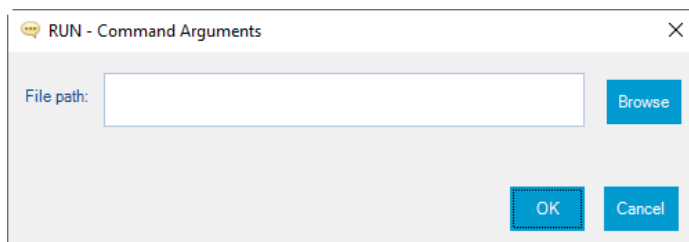
These commands control operation of the script or external programs.

### A.2.1 RUN <File path> [<arg1> <arg2> ...]

Runs an external process with or without arguments.

<File path>                      specifies the path and filename of an executable (\*.exe) program you want to start on the PC.

[<arg1> <arg2> ...]              one or more optional arguments for the executable program.



### Example:

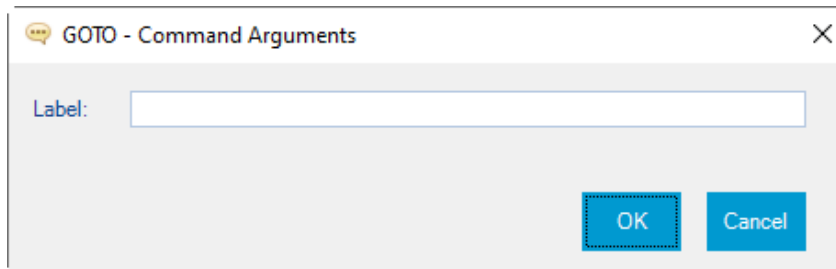
This command starts the Notepad application.

**RUN** c:\windows\notepad.exe

## A.2.2 GOTO <Label>

The order of script line execution jumps to the <label>.

<Label>                                      Name of a location in the script to jump to. The actual label location in the script must be preceded by a colon “:”

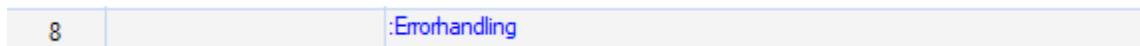


Example:

This jumps to the location of the script with the name “:Errorhandling”

**GOTO** Errorhandling

In the actual script the label line would look like this:



## A.2.3 PAUSE <Number of seconds to wait>

Pauses execution of the script for the specified number of seconds.

<Number of seconds to wait>            Number of seconds to pause the script.



Example:

This pauses the script for 60 seconds.

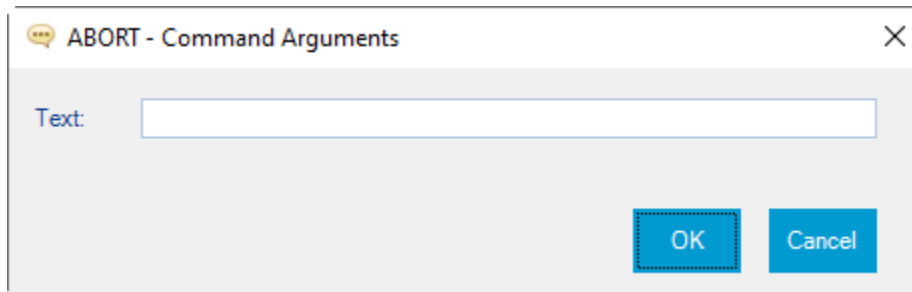
**PAUSE** 60



## A.2.4 ABORT [<Text> <Text> ... ]

Stops execution of the script, and optionally writes text arguments to PC screen running the script. The text arguments are combined and separated with single spaces.

<Text> Contiguous characters of text written to the output file. If there are multiple <text> arguments, each argument is written to the screen, with a space separating each argument from the next argument.



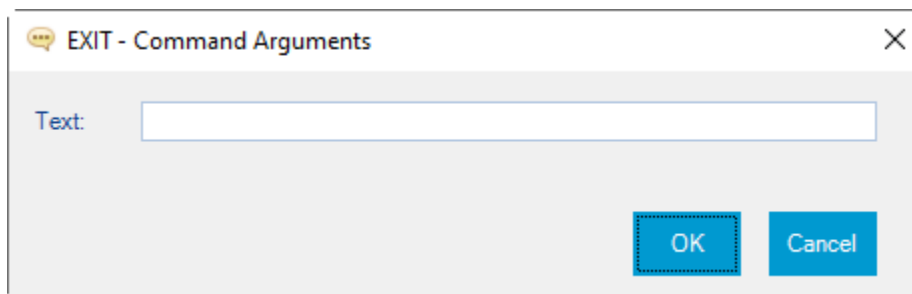
Example:

**ABORT** Script terminating now.

## A.2.5 EXIT [<Text> <Text> ... ]

Stops execution of the script, and optionally writes text arguments to the PC screen running the script. The text arguments are combined and separated with single spaces.

<Text> Contiguous characters of text written to the output file. If there are multiple <text> arguments, each argument is written to the screen, with a space separating each argument from the next argument.



Example:

**EXIT** Script completed successfully.

## A.2.6 PROMPT <Message><Show entry text box><top button text><bottom button text>

Opens a dialog box for the user with informational text provided by the <Message> argument.

<Message> Text message to show the user in the dialog box. Must be enclosed in double quotes.

<show entry text box> Show text entry field in the dialog box (yes/no).

<top button text> Text to display in the top button.

<bottom button text> Text to display in the bottom button.

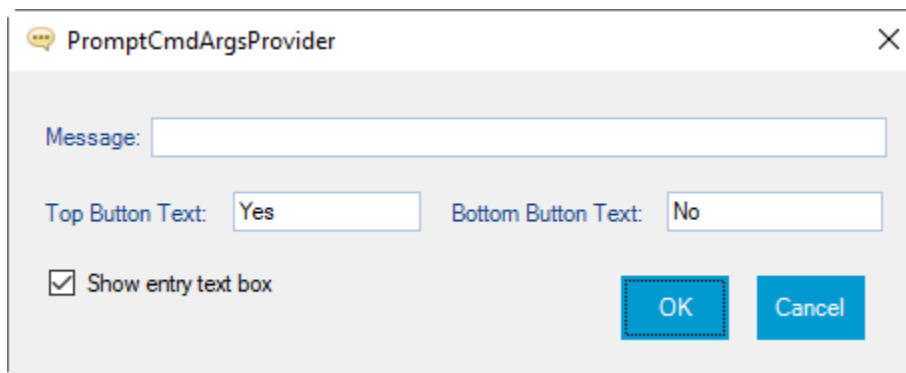
The following local variables are set based on user responses through the prompting dialog box:

PROMPT 0 = User clicked the "No" button. (Bottom button)

1 = User clicked the "Yes" button. (Top button).

P\_VAL Value entered by the user in numeric format.

\$P\_VAL Value entered by the user in string format.

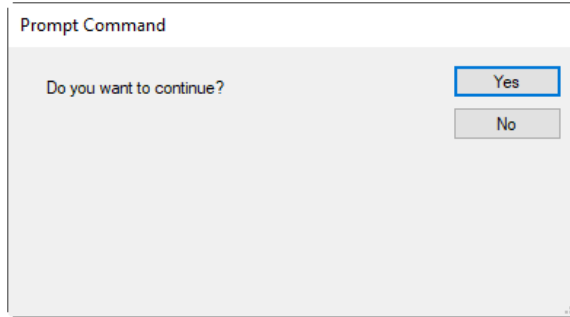


### Examples:

Asking a "Yes" or "No" question:

**PROMPT** "Do you want to continue?" no Yes No

Results in the following prompt:



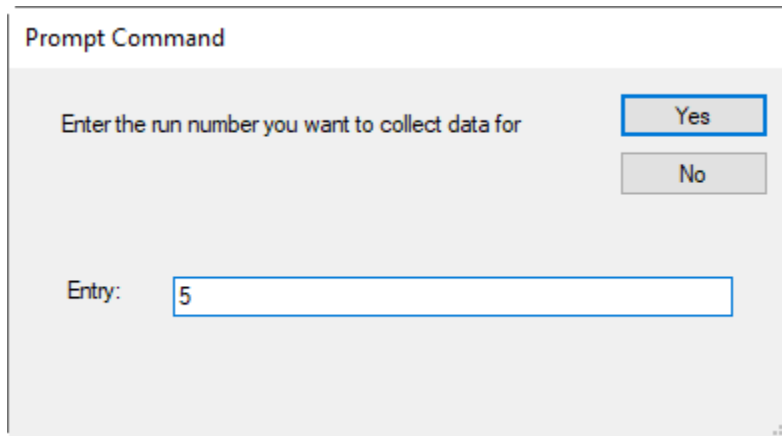
And stores the user’s Yes/No choice as a 1 (Yes) or 0 (No) in the PROMPT local variable:

PROMPT	1
--------	---

Asking a question requiring a numerical entry:

**PROMPT** “Enter the run number you want to collect data for:” YES

Results in the following prompt:



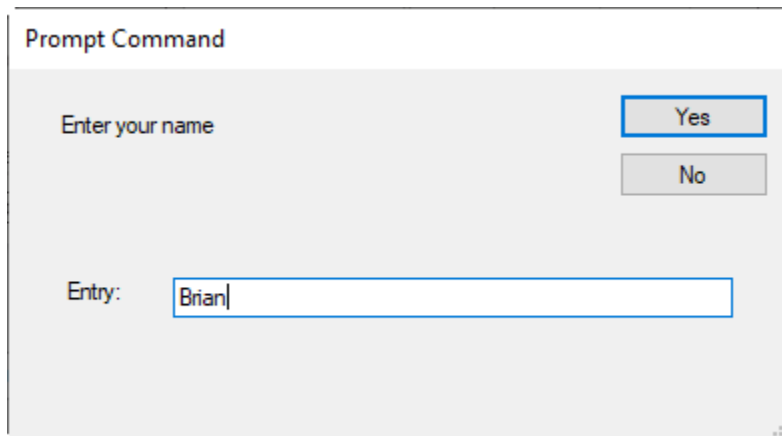
And the user’s choice is stored in the P\_VAL local variable:

P_VAL	5
-------	---

Asking a question requiring a string entry:

**PROMPT** “Enter your name:” YES

Results in the following prompt:



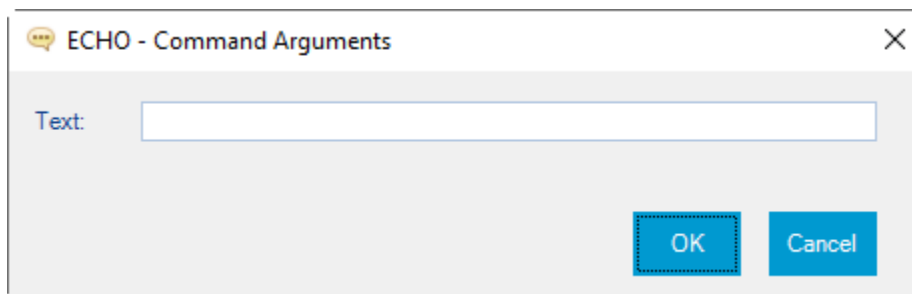
And the user's choice is stored in the \$P\_VAL local variable:

\$P_VAL	Brian
---------	-------

## A.2.7 ECHO <Text>[ <Text> ...]

Writes text arguments to the PC screen running the script. The text arguments are combined and separated with single spaces.

<Text> Contiguous characters of text written to the screen. If there are multiple <text> arguments, each argument is written to the screen, with a space separating each argument from the next argument.



Example:

**ECHO** Data collection in progress...

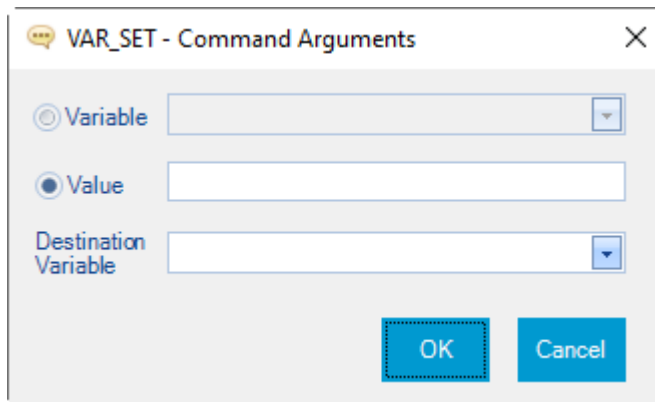
## A.3 Local Variable Commands

Scripts support local variables which you can reference inside the script. Script variable names are referenced by <var> in the following commands. If using a string variable, start the name with a dollar sign "\$".

### A.3.1 VAR\_SET <Destination Variable> <Value or Variable>

Creates a local variable and sets its initial value.

- <Destination Variable> Name of the local variable, either a numeric variable or a string variable.
- <Value> Initial value to be assigned to the destination variable. This could be a numeric value or a string value.
- <Variable> Variable containing the initial value to be assigned to the destination variable. Could be a numeric value or a string value. String values must be surrounded by double quotation marks “ ”.



Examples:

This creates a destination variable named “count” and sets its initial value to 0.

```
VAR_SET count 0
```

This creates a string destination variable named “StationName” and sets it to the value “Ocean Drive”.

```
VAR_SET $StationName “Ocean Drive”
```

This creates a destination variable named “count” and sets its initial value to the value of another variable named “total.”

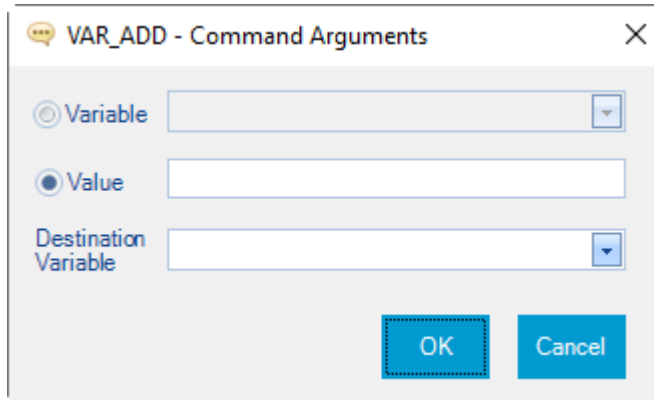
```
VAR_SET count total
```

### A.3.2 VAR\_ADD <Destination Variable> <Value or Variable>

Adds a number or the value of a local numeric variable to the value of the destination local variable and stores the result in the destination variable. The variable(s) must exist, and must be numeric.

- <Destination Variable> Name of the destination numeric local variable.

- <Value> An integer or floating point number to be assigned to the destination variable.
- <Variable> Variable containing an integer or floating point number to be assigned to the destination variable.



### Examples:

This example adds 7 to the current value of the destination variable named "count."

```
VAR_ADD count 7
```

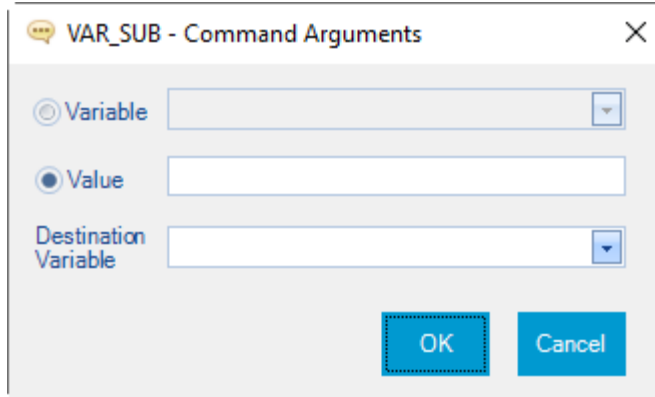
This example adds the value of the "new" variable to the value of the destination variable "count."

```
VAR_ADD count new
```

## A.3.3 VAR\_SUB <Destination Variable> <Value or Variable>

Subtracts a number or the numeric value of a local variable from the value of the destination variable and stores the result in the destination variable. The variable(s) must exist, and must be numeric.

- <Destination Variable> Name of the destination numeric local variable.
- <Value> An integer or floating point number to be subtracted from the destination variable.
- <Variable> Variable containing an integer or floating point number to be subtracted from the destination variable.



Examples:

This example subtracts 5 from the current value of the variable named “count.”

**VAR\_SUB** count 5

This example subtracts the value of the variable “daily” from the value of the variable “count.”

**VAR\_ADD** count daily

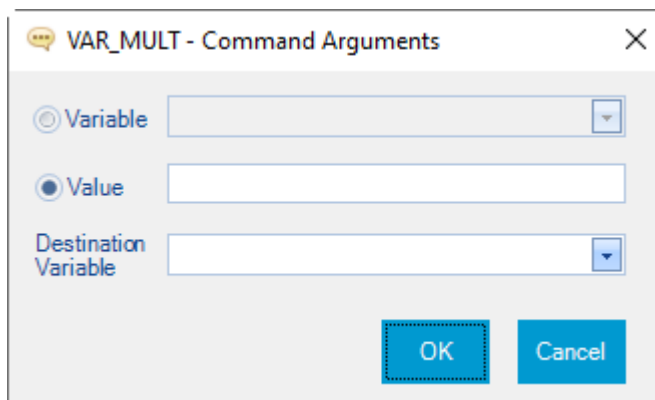
### A.3.4 VAR\_MULT <Destination Variable> <Value or Variable>

Multiplies a number or the numeric value of a local variable by the value of the numeric destination variable and stores the result in the destination variable. The variable(s) must exist and must be numeric.

<Destination Variable> Name of the destination numeric local variable.

<Value> An integer or floating point number to be multiplied by the destination variable.

<Variable> Variable containing an integer or floating point number to be multiplied by the destination variable.



Examples:

This example multiplies the current value of the variable named “total” by 5 and stores the result in “total.”

**VAR\_MULT** total 5

This example subtracts the value of the variable “daily” from the value of the variable “total” and stores the result in “total.”

**VAR\_MULT** total daily

## A.3.5 VAR\_DIV <Destination Variable> <Value or Variable>

Divides the value of the destination numeric local variable by a number or the value of another numeric local variable, then stores the result in the destination local variable. The variable(s) must exist and must be numeric. Attempts to divide by zero generate an “Attempted divide by 0” message in the status column.

- <Destination Variable>      Name of the destination numeric local variable.
- <Value>                      An integer or floating point number by which the destination variable’s value is divided.
- <Variable>                  Variable containing an integer or floating point number by which the destination variable’s value is divided.



Examples:

This example divides the value of the destination variable “total” by 4.

**VAR\_DIV** total 4

This example divides the value of the destination variable “daily” by the value of the variable “mf.”

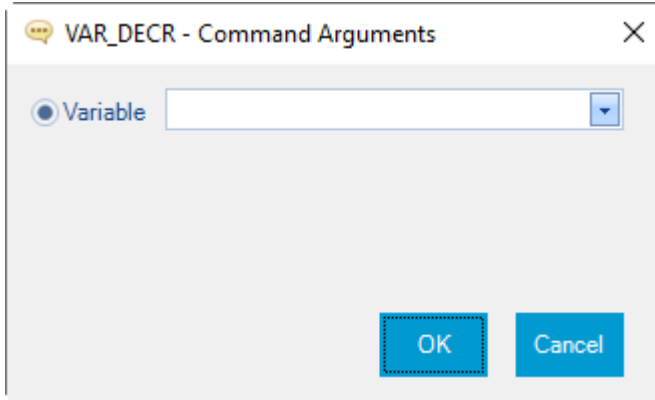
**VAR\_DIV** daily mf



### A.3.6 VAR\_DECR <Variable>

Decrements a local variable by 1. The variable must exist and be numeric.

<Variable>                      Name of the numeric variable.



Example:

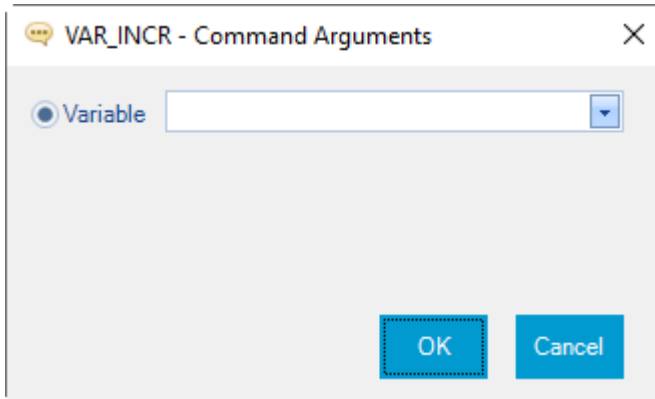
This example decrements the value of the variable “total” by 1.

**VAR\_DECR** total

### A.3.7 VAR\_INCR <Variable>

Increments a local variable by 1. The variable must exist and be numeric.

<Variable>                      Name of the numeric variable.



Example:

This example increments the value of the variable “total” by 1.

**VAR\_INCR** total

## A.3.8 IF <Left Operand Variable> <Condition> <Right Operand Variable or Value> <Label to jump when condition is true>

Checks the condition of a local variable and jumps to the label, if the condition is TRUE.

<Left Operand Variable> Name of the numeric or string variable.

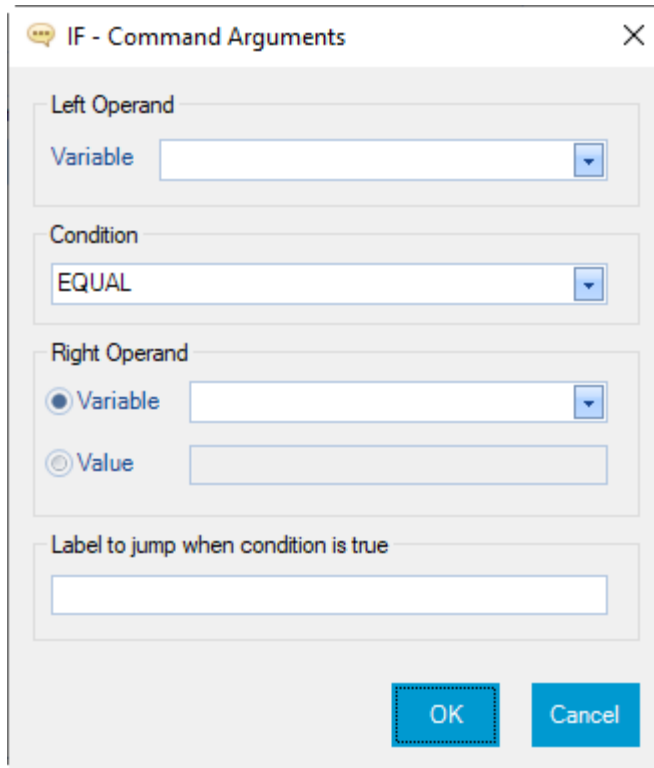
<Condition> Must be a logical operator appropriate for whether “Variable” is numeric or string.

**Table A-1. Local Variable Conditional Operators**

Operator	Description	Variable Data Types Supported
EQ	<b>EQUAL</b>	Numeric, String
NE	<b>NOT EQUAL</b>	Numeric, String
GE	<b>GREATER THAN OR EQUAL</b>	Numeric
GT	<b>GREATER THAN</b>	Numeric
LE	<b>LESS THAN OR EQUAL</b>	Numeric
LT	<b>LESS THAN</b>	Numeric

<Right operand Variable or Value> Numerical or string variable or value.

<Label to jump when condition is true> Name of a location in the script to jump to. The actual location in the file must be preceded by a colon “:”



Examples:

If "Total" is greater than "MyCount" then jump to the label named "DailyLog":

**IF** MyCount GT Total DailyLog

If "StationName" equals "Ocean Drive" then jump to the label named "Finish":

**IF** StationName EQ "Ocean Drive" Finish

## A.4 Device Connection Command

### A.4.1 CONNECT <connection name>

Establishes communication with an FB3000 RTU or an FB1000/FB2000-series flow computer through Field Tools. If the connection is successful, the software returns the product type and firmware version in a pair of local variables.

---

**Notes**

- Only **one** device connection can be active at any one time. If, after establishing a connection, you execute another CONNECT command, the existing connection is shut down, and the new CONNECT command then executes.

- If, instead of entering the connection name directly, you enter {\$A1} in the script, it allows you to reuse the same script with different devices by specifying whatever device name you want when you invoke the script.

**Table A-2. Local Variables Updated If the Connection is Successful**

Local Variable Name	Description	Local Variable Data Type
\$PRODUCT_TYPE	Device type	String
\$FRMWR_VERSION	Firmware version	String

<connection name> The name of the connection, as it exists in Field Tools software. Field Tools must be running in order to establish communications.



Examples:

Establish communications between the script and the RTU using the connection name WESTMOOR.

**CONNECT WESTMOOR**

Use CONNECT in a script called MyCollectionScript using a parameter for the connection name instead of entering it directly:

**CONNECT {\$A1}**

When you invoke the script you can pass in the connection name, in this case "WESTFARMS":

```
C:\Myscripts>FBxScriptRunner MyCollectionScript.SCR WESTFARMS
```

## A.5 Device System Commands

These commands may only be used after a connection has been established through the CONNECT command.

## A.5.1 WARM\_START <Time to wait for device to reboot>

Force the device to warm start.

<Time to wait for device to reboot> The number of minutes to wait for the device to reboot.



Example:

To force the device to perform a warm start, and then wait 5 minutes for the warm start to complete.

**WARM\_START 5**

## A.5.2 COLD\_START <start type> <timeout>

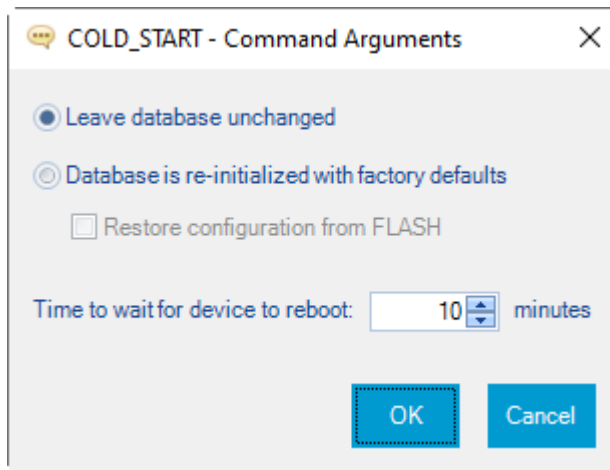
Force the device to cold start.

<start type> Identifies an action to perform on the RTU/flow computer's database.

**Table A-3. Start type's Action on the Device database**

Start Type Value	Description
0	Leave database unchanged (default)
1	Database is re-initialized with factory defaults. This restores User Data descriptions to factory defaults and clears alarm/event/history, user protocol maps, total, and averages. I/O modules also restart.
2	Restore configuration from FLASH. Restore read/write database parameter values from flash memory. This restores User Data descriptions to factory defaults and clears alarm/event/history, user protocol maps, total, and averages. I/O modules also restart.

<timeout> The number of minutes to wait for the device to reboot.



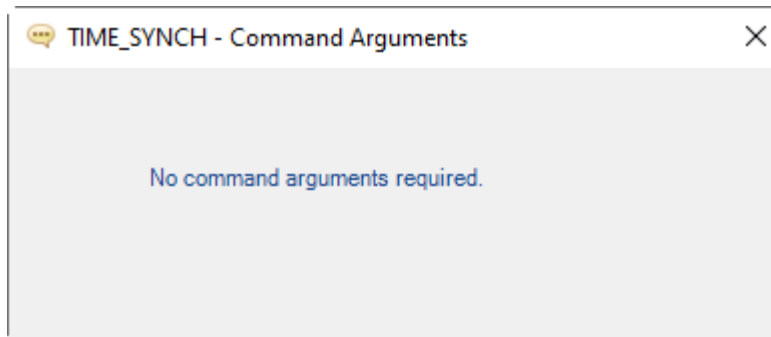
Example:

To force the device to perform a cold start, and then wait 7 minutes for the cold start to complete:

**COLD\_START 7**

## A.5.3 TIME\_SYNC

Synchronizes the time in the device with the PC's local time.



Examples:

To synchronize the device's time with the time on the PC:

**TIME\_SYNC**

## A.6 Device Parameter Commands

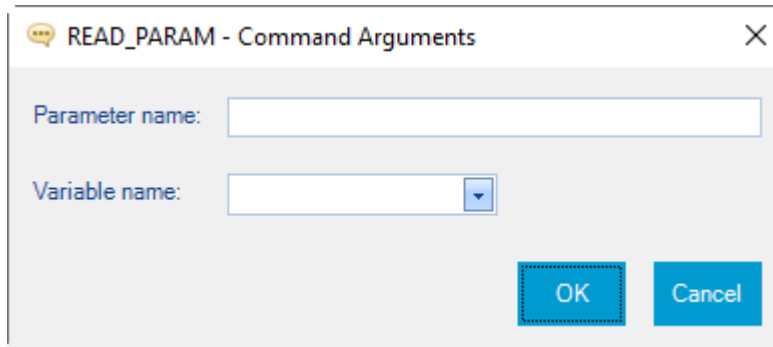
These commands allow you to read and write to parameters in the device.

## A.6.1 READ\_PARAM <Parameter name> <Variable name>

Reads the value of a parameter on the device and stores it in a local variable for use in the script.

<Parameter name> The full parameter name. If it contains spaces, it must be enclosed in double quotes.

<Variable name> The local variable name. If it does not exist, it is created. If the parameter read contains a string, you must create a string local variable using the VAR\_SET command.



Example:

Read analog input IoConfig\_2.CHAN1\_1\_SELECT and store the value in a local variable called "Flow."

**READ\_PARAM** IoConfig\_2.CHAN1\_1\_SELECT Flow

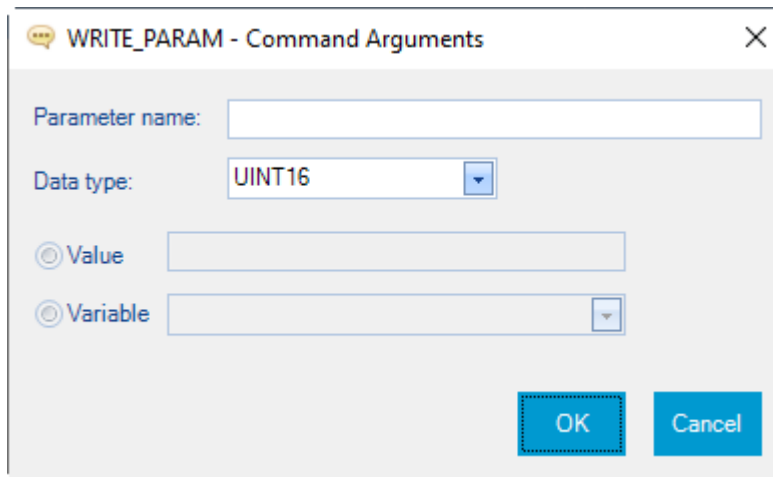
## A.6.2 WRITE\_PARAM <Parameter name> <Data type> <Value or Variable>

Writes a value to a parameter in the device.

<**Parameter name**> The full parameter name. If it contains spaces, it must be enclosed in double quotes " ".

<**Data type**> Device parameter data type. See [Table A-4](#) for a list of valid parameter data types.

<**Value or Variable**> Constant value or local variable.



Examples:

Write the value 25.37 to the parameter User Data\_1.FLOAT\_1.

**WRITE\_PARAM** "User Data\_1.FLOAT\_1" FLOAT 25.37

Write the string "Flow Totals" to the Hist\_7.HIST\_GROUP\_OBJ.DESC parameter which is a 20 character string type:

**WRITE\_PARM** Hist\_7.HIST\_GROUP\_OBJ.DESC UC20 "Flow Totals"

**Table A-4. WRITE\_PARAM Data Types**

FBx Data Type	Description
UINT8	1-byte signed integer
INT8	1-bytes unsigned integer
UINT16	2-bytes signed integer
INT16	2-bytes unsigned integer
UINT32	4-bytes signed integer
INT32	4-bytes unsigned integer
UINT64	8-bytes unsigned integer
INT64	8-bytes signed integer
FLOAT	Single-precision floating point number
DOUBLE	Double-precision floating point number
UC10	10-byte character string
UC20	20-byte character string
UC30	30-byte character string
UC40	40-byte character string
BYTEARRY4	4 bytes



FBx Data Type	Description
BYTEARRAY6	6 bytes
BYTEARRAY32	32 bytes
TIME	8 bytes representing Julian time
ENUM16	2 bytes enumeration code
BIN8	8 bits
BIN16	16 bits
BIN32	32 bits
PRMREF	Parameter reference – 40 bytes maximum
OBJREF	Object reference – 20 bytes maximum

### A.6.3 RECIPE <Function> <Recipe file>

The RECIPE command can perform two different operations, depending upon how the mode is set.

**<Function>**

In READ mode, it reads a recipe file and then collects the values of those parameters from the device and updates the recipe file with those values.

In WRITE mode, it reads a recipe file of parameters and values on the PC and then writes those values down to their corresponding parameters in the device.

**<Recipe file>**

Specifies the path and filename of the recipe file. Recipe files have an extension of (\*.RCP). Each line of the recipe file consists of Parameter name, followed by a space, followed by the value of the parameter.

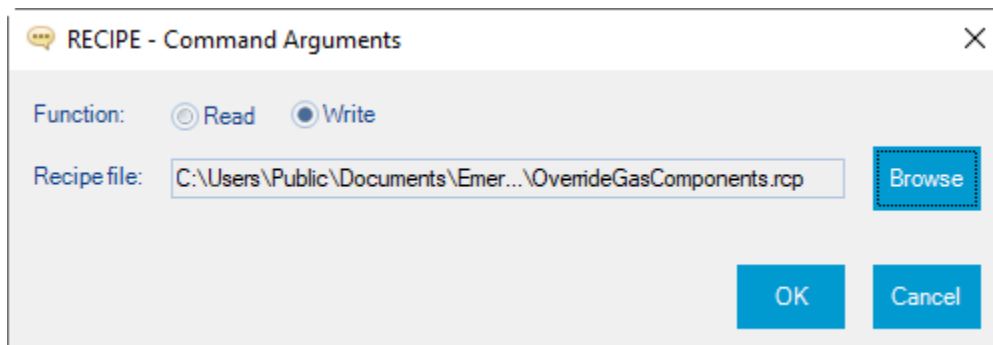
Each line in the recipe file follows the format:

*Parameter\_Name Data\_Type Value*

For example, suppose you want to specify gas component parameters for an application, you could create a recipe called gascomponents.rcp. Each line of the recipe would specify the parameter, and, in this case, a double-precision floating point value for the given percentage of that component. The last line of the recipe automatically applies the components:

```
Components_1.C1_OVRD, Double, 94.6
Components_1.N2_OVRD, Double, 0.4
Components_1.CO2_OVRD, Double, 0.3
Components_1.C2_OVRD, Double, 4.4
```

```
Components_1.C3_OVRD, Double, 0.2  
Components_1.H2O_OVRD, Double, 0  
Components_1.H2S_OVRD, Double, 0  
Components_1.H2_OVRD, Double, 0.01  
Components_1.CO_OVRD, Double, 0  
Components_1.O2_OVRD, Double, 0.005  
Components_1.IC4_OVRD, Double, 0.01  
Components_1.NC4_OVRD, Double, 0.01  
Components_1.IC5_OVRD, Double, 0.002  
Components_1.NEOC5_OVRD, Double, 0  
Components_1.C6_OVRD, Double, 0.002  
Components_1.C7_OVRD, Double, 0  
Components_1.C8_OVRD, Double, 0  
Components_1.C9_OVRD, Double, 0  
Components_1.C10_OVRD, Double, 0  
Components_1.HE_OVRD, Double, 0  
Components_1.AR_OVRD, Double, 0  
Components_1.BENZENE_OVRD, Double, 0  
Components_1.TOLUENE_OVRD, Double, 0  
Components_1.APPLY_COMP, Enum16, 1
```



### Examples:

To write recipe values from the file `OverrideGasComponents.rcp` to parameters in the device:

```
RECIPE WRITE C:\OverrideGasComponents.rcp
```

To read values from the device and update the values in the recipe with those values:

```
RECIPE READ C:\MyOtherRecipe.rcp
```

## A.7 Device File Commands

These commands are used with files stored on the RTU or flow computer.

**Note**

In FBxConnect, you can go to **Services > File Transfer** to see a tree of the folders and files on the device.

## A.7.1 FILE\_DELETE <Device file path> <File Not Found flag>

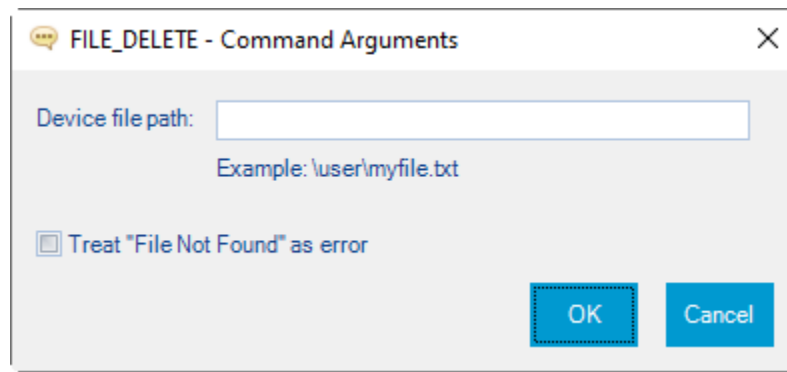
Deletes a flash file from the device.

**<Device file path>** Specifies the path and filename of the flash file on the device to be deleted.

**<File Not Found flag>** This argument is optional. It indicates whether a file not found is reported as an error.

0 File not found will be reported as success (default).

1 File not found will be reported as an error.



Example:

To delete an FBxDesigner source file called Myproject.zwt from the device:

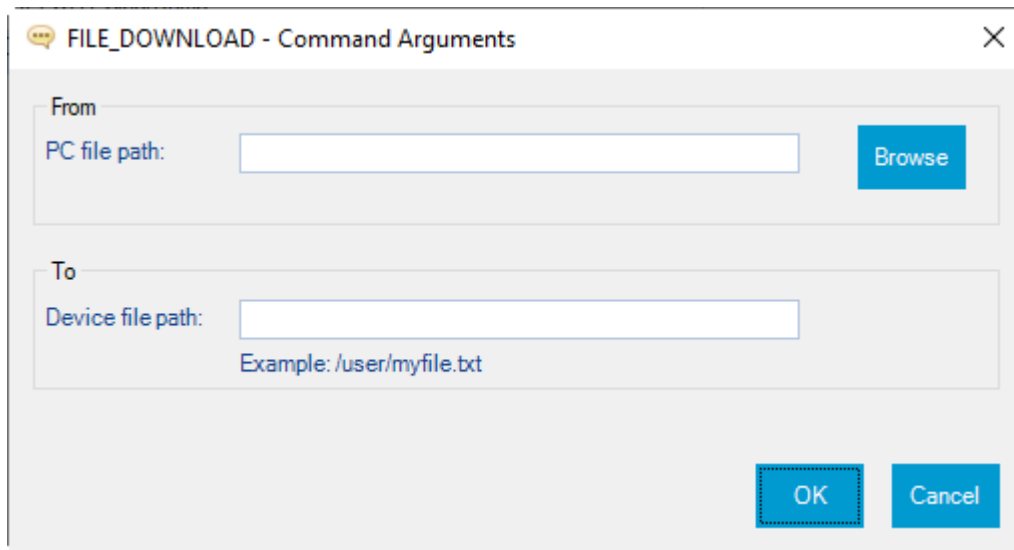
**FILE\_DELETE** Myproject.zwt

## A.7.2 FILE\_DOWNLOAD <PC file path> <Device file path>

Downloads a file from the PC to the device flash area.

**<PC file path>** Specifies the path and filename of the file to be downloaded.

**<Device file path>** Specifies the destination path in the flash area where the file will be copied.



Example:

To download an FBxDesigner source file called Myproject.zwt to the device:

**FILE\_DOWNLOAD** C:\myfiles\MyProject.zwt

## A.7.3 FILE\_UPLOAD <Device file path> <PC file path> [<File Not Found flag>]

Upload a flash file on the device to the PC.

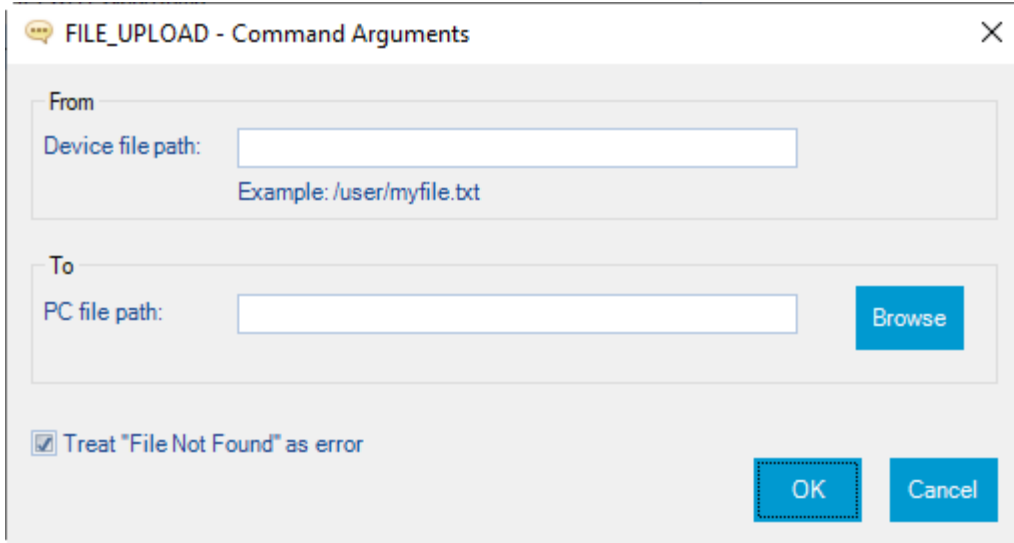
**<device path>** Specifies the path and filename on the device you want to upload.

**<PC file>** Specifies destination path and filename for the uploaded file.

**<flag>** This optional argument indicates whether a file not found will be reported as an error.

0 = File not found reported as success

1=File not found reported as an error (default)



Example:

To upload an FBxDesigner source file called Myproject.zwt from the device:

**FILE\_UPLOAD** Myproject.zwt C:\ProjectSource\Myproject.zwt 1

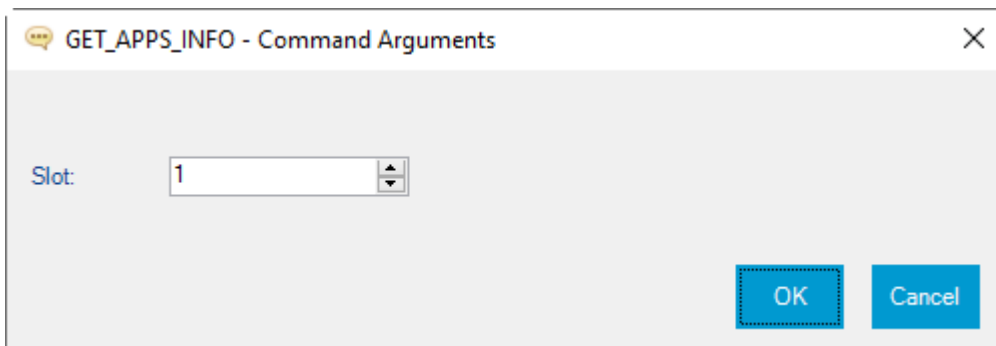
## A.8 Device Application Commands

### A.8.1 GET\_APPS\_INFO [<slot>]

Retrieves information about applications running in the FB3000 such as the application name and version. This information returned is stored in local variables.

**<slot>** Optional argument indicating the slot number, which must be an integer from 1 to 8. If specified, only retrieves details for the application in that slot.

If not specified, details on all slots are retrieved.



**Table A-5. Local Variable Holding Returned Device Application Information**

Local Variable Name	Description	Local Variable Data Type
\$APP_x_NAME	The name of the application in slot x.	STRING
\$APP_x_VERSION	The version of the application in slot x.	STRING
\$APP_x_SLOT	The application slot.	DOUBLE

(x is replaced with the appropriate slot number)

Examples:

To retrieve detail on all installed applications:

**GET\_APPS\_INFO**

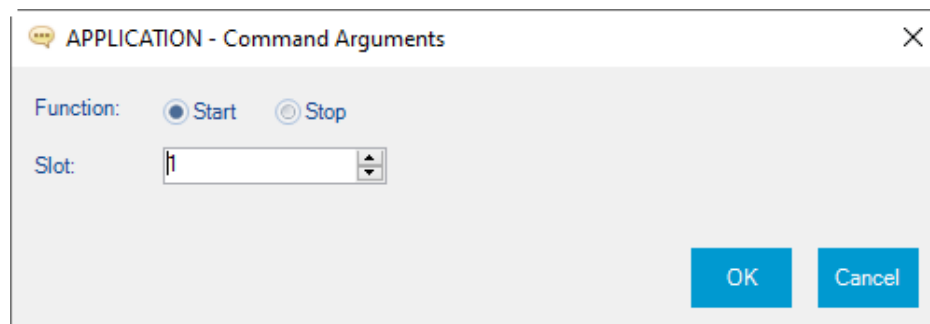
To retrieve detail on only the application in slot 4:

**GET\_APPS\_INFO 4**

## A.8.2 APPLICATION <Function> <Slot>

Starts or stops an IEC 61131 application in a particular slot on the FB3000 device.

- <Function>** Is one of the following:
  - START Starts the application in the specified slot.
  - STOP Stops the application in the specified slot.
- <Slot>** Specifies the application software slot on the FB3000. Can be an integer from 1 and 8.



Examples:

To start the application in slot 2:

**APPLICATION START 2**

To stop the application in slot 1:

**APPLICATION STOP 1**

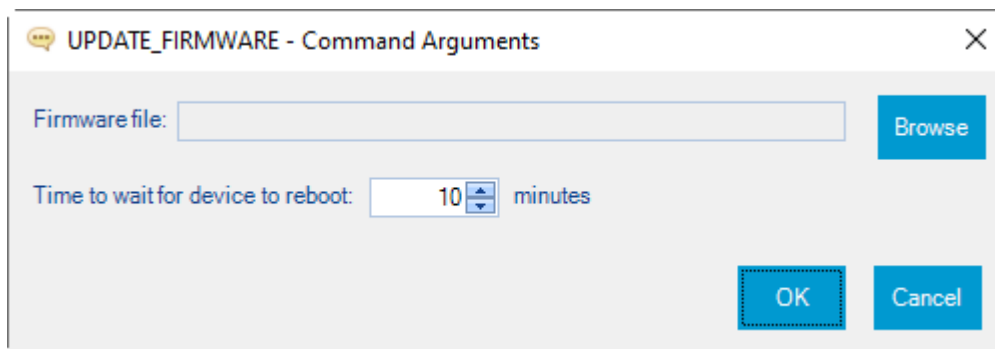
## A.9 Device Firmware Commands

These commands relate to the system firmware running in the device.

### A.9.1 UPDATE\_FIRMWARE <Firmware file> [<Time to wait for device to reboot>]

Downloads a firmware zip file to the device.

- <Firmware file>** The absolute path and filename of the firmware zip file.
- <Time to wait for device to reboot>** Specifies how long (in minutes) to wait for the device to reboot after the firmware update. This argument is optional. If not specified, it defaults to 10 minutes.



Example:

Upgrade firmware to version 02\_15\_014\_14 and wait 6 minutes for the reboot to occur before declaring an error.

```
UPDATE_FIRMWARE C:\firmware\FB3000_SystemPackage_02_15_02_14.zip 6
```

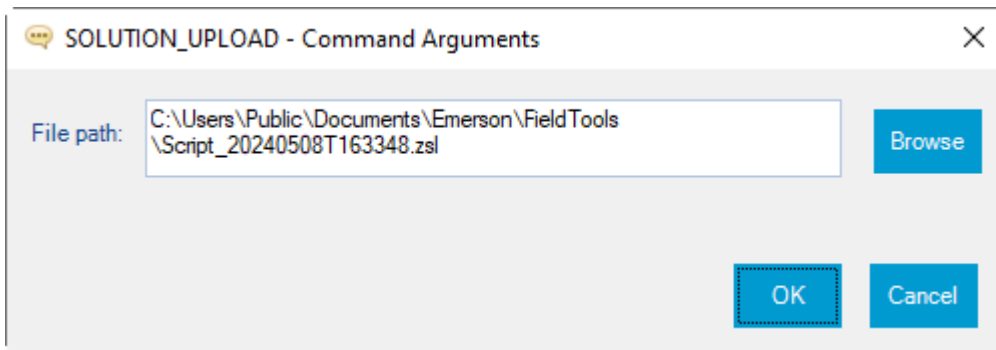
## A.10 Solution Commands

These commands can download or upload a solution file to or from the device.

### A.10.1 SOLUTION\_UPLOAD <File path>

Uploads a solution file from the device.

- <File path>** Absolute path and filename on the PC to which the script tool should copy the solution file.



Example:

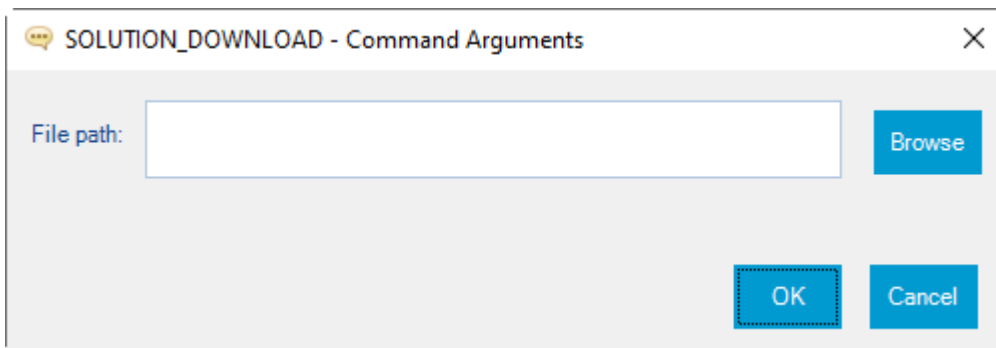
To upload the solution file (mysoln.zsl) from the device, and store it in the folder c:\work:

**SOLUTION\_UPLOAD** C:\work\mysoln.zsl

## A.10.2 SOLUTION\_DOWNLOAD <File path>

Downloads a solution file to the connected device.

**<File path>** Absolute path and filename on the PC where the solution file to be downloaded resides.



Example:

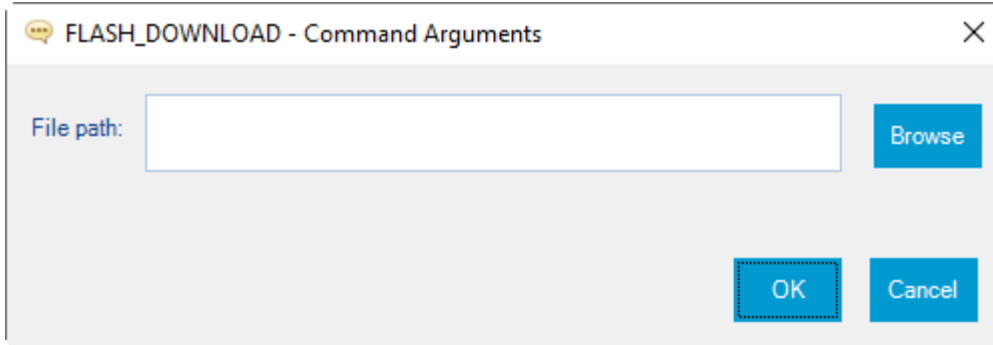
**SOLUTION\_DOWNLOAD** C:\work\mysoln.zsl

## A.10.3 FLASH\_DOWNLOAD <File path>

Downloads a solution's configuration to the device's flash memory.

**<File path>** Absolute path and name of the solution's .ZSL configuration file to be downloaded into the flash memory area.





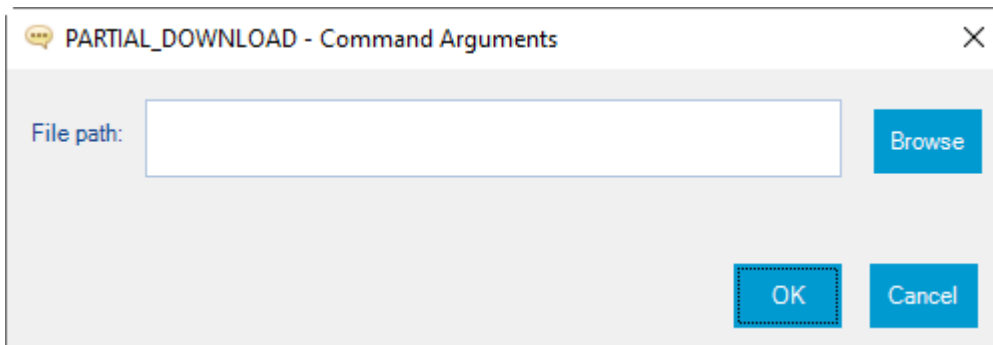
Example:

**FLASH\_DOWNLOAD** C:\work\mysoln.zsl

## A.10.4 PARTIAL\_DOWNLOAD <File path>

Downloads a partial configuration file to the device.

**<File path>** Absolute path and name of the partial .PTC configuration file to be downloaded to the device.



Example:

**PARTIAL\_DOWNLOAD** C:\work\myconfig.ptc

## A.10.5 PARTIAL\_UPLOAD <Objects & Instances file> <Partial Configuration File>

Reads an objects and instances CSV file on the PC and uses the entries in it to determine which objects and instances in the RTU should be included in a partial configuration file that the script will upload from the device. The ptc file does not contain read-only parameters.

**<Objects & Instances File>** Absolute path and name of a file on the PC that contains a list of objects and instances in the device to be uploaded into a PTC file. This file requires an

extension of OIF. Each line of an OIF file has the format:

*Object\_ [instance number(s)]* To specify a range of instances, separate the top and bottom values of the range with an underscore “\_”. If no instances are specified, all instances are uploaded.

Examples:

1. If a line in the OIF file looks like this:

PID\_, 1, 2, 3

Then PID objects 1, 2, and 3 are uploaded.

2. If a line in the OIF file looks like this:

Alarm\_, 1000, 1005\_1010

Then Alarm instances 1000, 1005, 1006, 1007, 1008, 1009, and 1010 will be uploaded.

3. If a line in the OIF file looks like this:

DP\_, 1-1, 2-1 ,3-1

Then DP objects 1-1, 2-1, and 3-1 are uploaded.

4. If a line in the OIF file looks like this:

HART\_, 2-1, 2-9\_2-12, 4-28

Then HART instances 2-1, 2-9, 2-10, 2-11, 2-12, and 4-28 are uploaded.

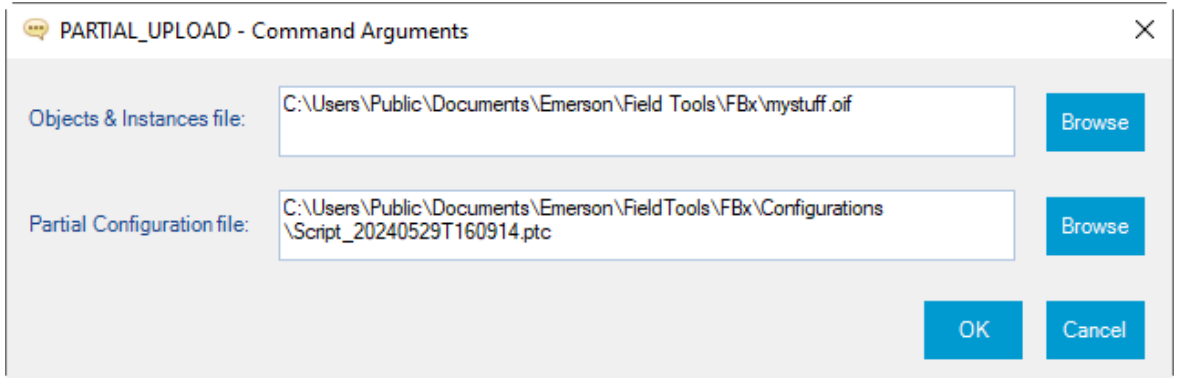
5. If a line in the OIF file looks like this:

User Data\_

Then because no instances are specified, **all** instances are uploaded.

## <Partial Configuration File>

Absolute path and name of the partial .PTC configuration file on the PC that holds the uploaded objects and instances from the device.



Example:

**PARTIAL\_UPLOAD** C:\work\mystuff.OIF c:\work\myconfig.ptc

## A.10.6 FLASH\_SAVE

Saves the device's configuration to flash. (Version 3.18 or newer)



Example:

**FLASH\_SAVE**

## A.11 Device Report Generation Commands

### A.11.1 DIAG\_REPORT <ClearDump or ReportType> <File Name>

Uploads diagnostic records from the device and stores them in a report file on the PC.

<ClearDump or ReportType> is one of the following:

ClearDump clear memory dumps on the device

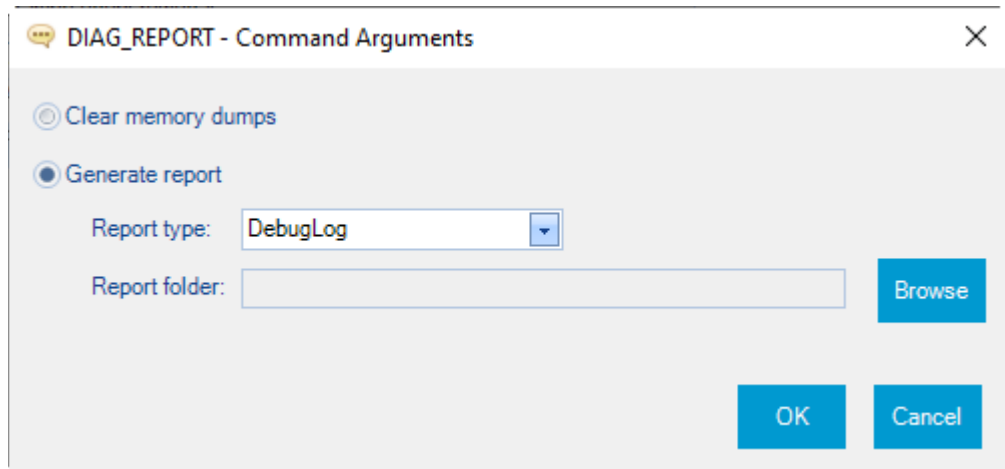
DebugLog collect device's debug log

MemoryDump collect memory dumps in the device if they exist

SecurityLog collect device's security log

**<File Name>**

Absolute path and name of the file which will hold the retrieved diagnostic records. Do not include the file extension.



Example:

**DIAG\_REPORT** SecurityLog C:\mydata\logons

## A.11.2 ALARM\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From Time> <End\_Time>]

Uploads alarm records from the device and stores them in a report file on the PC.

**<File Name>** Absolute path and name of the file which will hold the retrieved alarm records. Do not include the file extension.

**<Format>** Select the format of the report file:

PDF Portable document format

CSV Comma separated variable

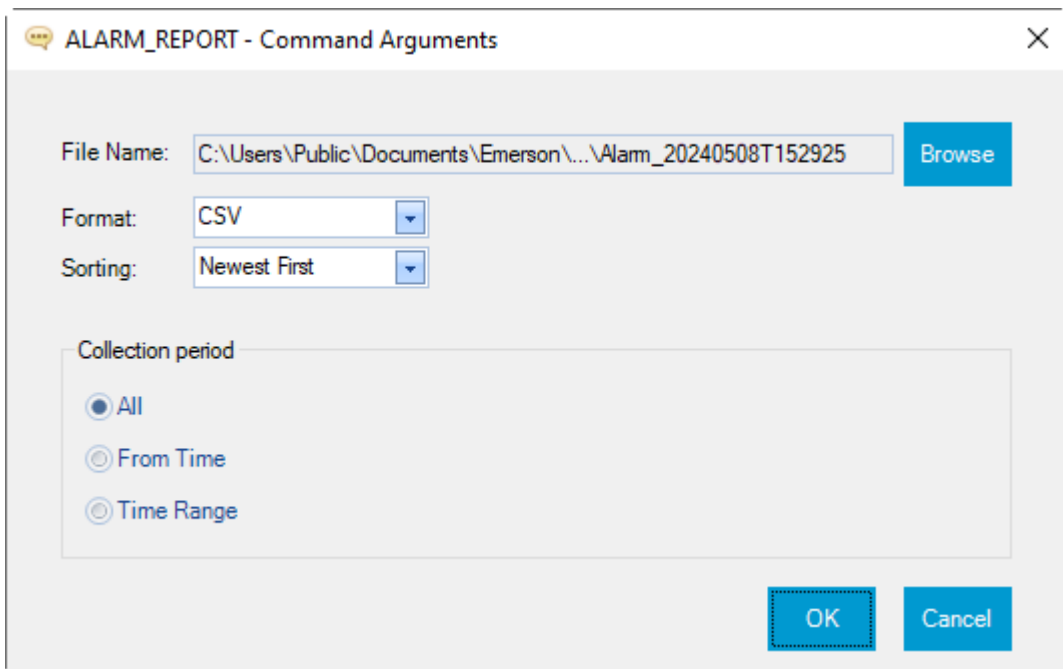
**<Sorting>** Specifies the order in which alarm records are stored in the alarm report file:

NEWTOLD Newest alarms appear at the top of the alarm report file.

OLDTONEW Oldest alarms appear at the bottom of the alarm report file.

<Collection period> Specifies which alarm records should be retrieved:

- ALL                      Retrieves all alarm records in the device.
- From Time              Collect all alarm records between the from time and the current device timestamp. Timestamps are in the format: MM/DD/YYYY HH:mm:ss
- Time Range              Retrieves all alarm records from the device between the From Time and end-time timestamps.
- End\_time                Specifies the end of the time range.



Example:

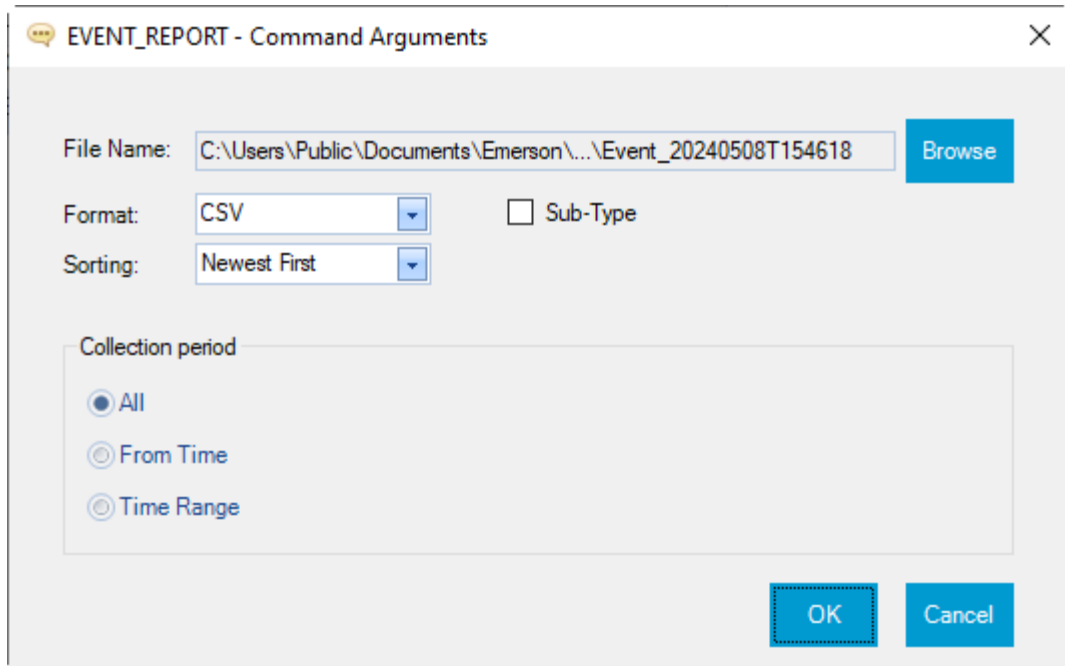
To collect all alarms between midnight and 8AM from May 18<sup>th</sup> 2024 and store them in a CSV file named alarmhistory in the folder C:\myalarms with the most recent alarms at the top of the file, enter:

**ALARM\_REPORT** c:\myalarms\alarmhistory CSV NEWTOOLD 05/18/24 00:00:00 5/18/24 08:00:00

## A.11.1 EVENT\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From time> <End\_time>] [<Sub- type>]

Uploads event records from the device and stores them in a report file on the PC.

<b>&lt;File Name&gt;</b>	Absolute path and name of the file which will hold the retrieved event records. Do not include the file extension.
<b>&lt;Format&gt;</b>	Select the format of the report file:  PDF Portable document format  CSV Comma separated variable
<b>&lt;Sorting&gt;</b>	Specifies the order in which event records are stored in the event report file:  NEWTOLD Newest events appear at the top of the event report file.  OLDTONEW Oldest events appear at the bottom of the event report file.
<b>&lt;Collection Period&gt;</b>	Specifies which event records should be retrieved:  ALL Retrieves all event records in the device.  From Time Collect all event records between the start time and the current device timestamp. Timestamps are in the format: MM/DD/YYYY HH:mm:ss  Time Range Retrieves all event records from the device between the start_time and end-time timestamps.  End_Time Specifies the end of the time range.
<b>&lt;Sub-type&gt;</b>	Optional argument:  LEGAL Only collect event records considered "legal" by an auditing authority.  NON-LEGAL Only collect event records considered "non-legal" by an auditing authority.



Example:

To collect all legally specified events from midnight to 8AM on May 18, 2024 into a file called eventhistory.csv in the C:\myevents folder:

```
EVENT_REPORT c:\myevents\eventhistory CSV NEWTOOLD 05/18/24 00:00:00 5/18/24
08:00:00 LEGAL
```

### A.11.1 HISTORY\_REPORT <File Name> <Format> <Sorting> <Collection Period> [<From time> <End\_time>] <History Group(s)> <Interval(s)>

Uploads history records from the device and stores them in a report file on the PC.

**<File Name>** Absolute path and name of the file which will hold the retrieved history records. Do not include the file extension.

**<Format>** Select the format of the report file:

PDF Portable document format

CSV Comma separated variable

**<Sorting>** Specifies the order in which history records are stored in the report file:

NEWTOOLD Newest records appear at the top of the history report file.

OLDTONEW Oldest records appear at the bottom of the history report file.

**<Collection Period>** Specifies which history records should be retrieved:

ALL Retrieves all history records in the device.

From Time Collect all history records between the start time and the current device timestamp. Timestamps are in the format: MM/DD/YYYY HH:mm:ss

Time Range Retrieves all history records from the device between the start\_time and end-time timestamps.

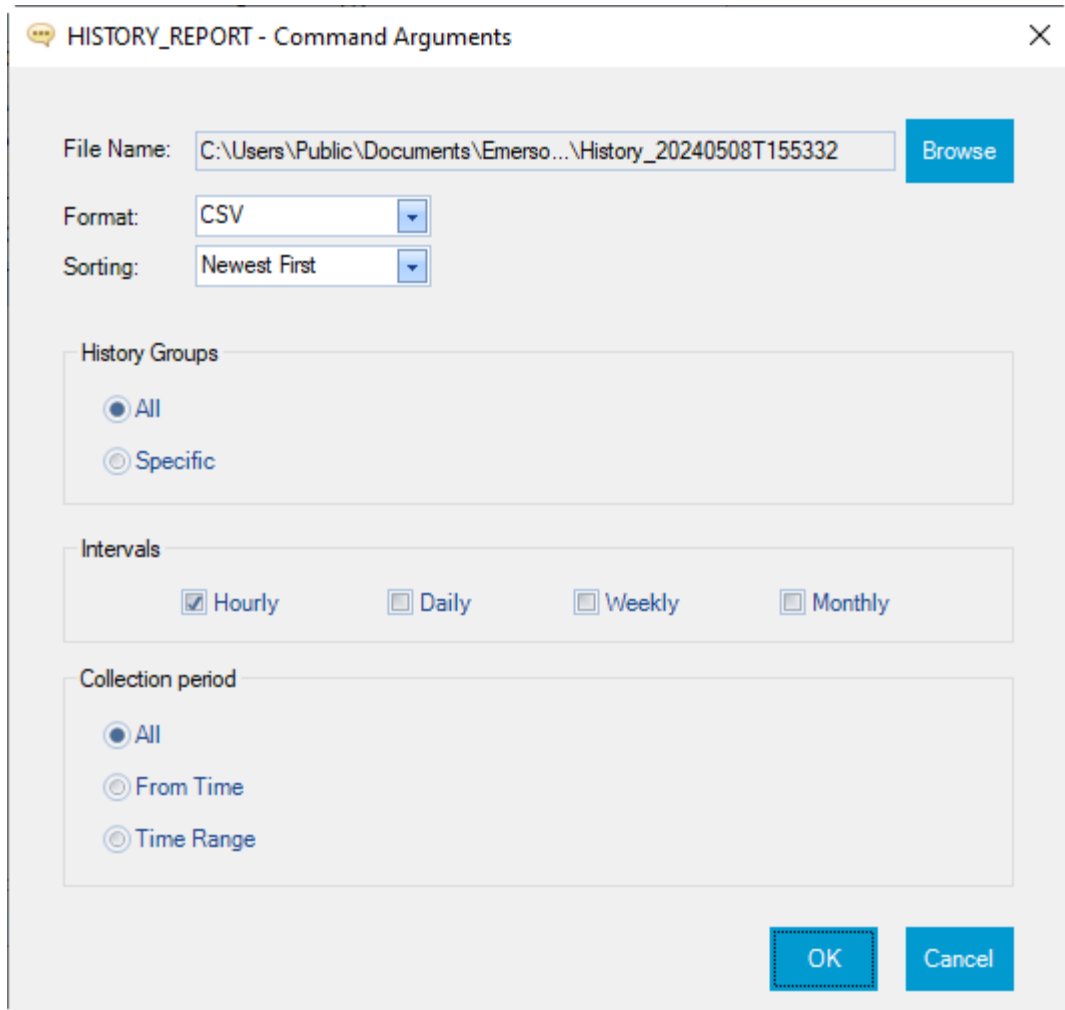
End\_Time Specifies the end of the time range.

**<History Group(s)>** ALL Collects records for all history groups.

Specific Single history group number as integer or multiple history group numbers separated by commas.

**<Intervals>** Specify Hourly, Daily, Weekly, or Monthly, or specify multiple interval types separated by commas.





Example:

To collect hourly and daily data from history groups 1, 2, and 3 and store it in the file historicalrecords.csv in the pre-existing folder "myhistory" with newest data shown first, enter:

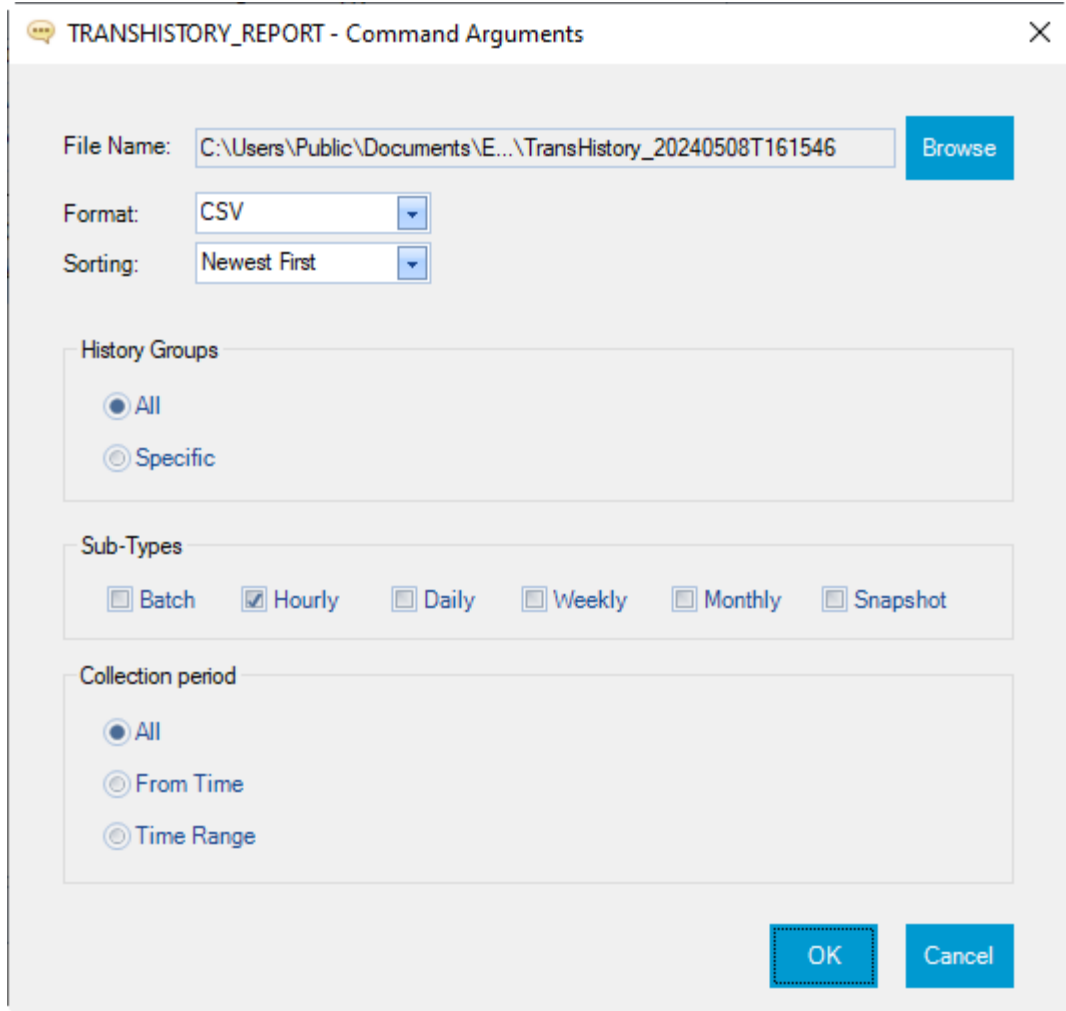
**HISTORY\_REPORT** c:\myhistory\historicalrecords CSV NEWTOOLD 1,2,3 Hourly, Daily

**A.11.1 TRANSHISTORY\_REPORT <File Name> <Format>  
<Sorting> <Collection Period> [<From time>  
<End\_time>] <History Group(s)> <Sub-type(s)>**

Uploads transaction history records from the device and stores them in a report file on the PC.

**<File Name>** Absolute path and name of the file which will hold the retrieved transaction history records. Do not include the file extension.

<b>&lt;Format&gt;</b>	Select the format of the report file:  PDF Portable document format  CSV Comma separated variable
<b>&lt;Sorting&gt;</b>	Specifies the order in which transaction history records are stored in the report file:  NEWTOLD Newest records appear at the top of the transaction history report file.  OLDTONEW Oldest records appear at the bottom of the transaction history report file.
<b>&lt;Collection Period&gt;</b>	Specifies which transaction history records should be retrieved:  ALL Retrieves all transaction history records in the device.  From Time Collect all transaction history records between the start time and the current device timestamp. Timestamps are in the format: MM/DD/YYYY HH:mm:ss  Time Range Retrieves all transaction history records from the device between the start_time and end-time timestamps.  End_Time Specifies the end of the time range.
<b>&lt;History Group(s)&gt;</b>	ALL Collects transaction records for all history groups.  Specific Single history group number as integer or multiple history group numbers separated by commas.
<b>&lt;Sub-Types&gt;</b>	Specify the categories of transactions to retrieve. Choices are: Batch, Hourly, Daily, Weekly, or Monthly, Snapshot or you can specify multiple categories separated by commas.



Example:

To collect all batch transaction history records, from newest to oldest, and store them in the pre-existing “myhistory” folder in the file transactionrecords.CSV, enter:

**TRANSHISTORY\_REPORT** c:\myhistory\transactionrecords CSV NEWTOOLD ALL Batch

# FBx Script Developer User Manual

D301953X012

November 2024

---

For customer service and technical support, visit [Emerson.com/Guardian](https://emerson.com/guardian).

## North America and Latin America:

Emerson Energy and Transportation Solutions  
6005 Rogerdale Road  
Houston, TX 77072 U.S.A.  
T +1 281 879 2699 | F +1 281 988 4445  
[Emerson.com/SCADAforEnergy](https://emerson.com/SCADAforEnergy)

## United Kingdom:

Emerson Process Management Limited  
Fosse House, 6 Smith Way  
Grove Park, Enderby  
Leicester LE19 1SX UK  
T +44 0 870 240 1978

## Europe:

Emerson S.R.L  
Regulatory Compliance Shared Services Department  
Company No. J12/88/2006  
Emerson 4 Street  
Parcul Industrial Tetarom 11  
Romania  
T +40 374 132 000

## Middle East/Africa:

Emerson Energy and Transportation Solutions  
Emerson FZE  
P.O. Box 17033  
Jebel Ali Free Zone – South 2  
Dubai U.A.E.  
T +971 4 8118100 | F +971 4 8865465

## Asia-Pacific:

Emerson Energy and Transportation Solutions  
1 Pandan Crescent  
Singapore 128461  
T +65 6777 8211 | F +65 6777 0947

© 2024 Bristol Inc. All rights reserved.

This publication is for informational purposes only. While every effort has been made to ensure accuracy, this publication shall not be read to include any warranty or guarantee, express or implied, including as regards the products or services described or their use or applicability. Bristol Inc. (hereinafter “Energy and Transportation Solutions” or ETS) reserves the right to modify or improve the designs or specifications of its products at any time without notice. All sales are governed by ETS terms and conditions which are available upon request. ETS accepts no responsibility for proper selection, use or maintenance of any product, which remains solely with the purchaser and/or end-user. Emerson and the Emerson logo are trademarks and service marks of Emerson Electric Co. All other marks are the property of their respective owners.