BNET Radar API

User's Manual for Model Numbers:

700-0005-201_DAA, 700-00050203_SSR

840-0001-461

Release Date: 2021-April

REVISION	REASON FOR CHANGE	DATE	BY
REV 1	Initial Release	170427	Womack
REV 2	Removed Void Pointers	170518	Womack
REV 3	Generic nomenclature for DAA and SSR	170802	Graves
REV 4	Include TFTP instructions	170830	Womack
REV 5	Content, explain block saving and reading	171024	Womack
REV 6	Update examples to use C++ source. Update MATLAB commands. Add static library.	171116	Womack
REV 7	Clarified wording, updated copyright & examples	180930	Straka
REV 8	Updated to include information about measurement related commands	180412	Womack
REV 9	Update to reflect new C++ interfaces and Linux support	190715	Womack
REV 10	Organized for better Windows and Linux API & Executable separation. Added POSIX time setting. Support for saving extended track packets. Updated manual # to -461 (from -221) for Doc Control consistency.	191201	Womack
REV 11	Improved build instructions and cmake generator	200305	Straka
REV 12	v11.0.25 MATLAB interface enhancements; fix for Linux saved_data_converter	200526	Straka
REV 13	v11.0.28 Documented set_posix time command. Documented ability to select arbitrary data ports for connection (radar data ports fixed). Removed beam related MATLAB API documentation.	200915	Womack
REV 14	v11.1.5 Cleaned up MATLAB interface documentation; removed connection to Command Port in monitor mode	210414	Womack

TABLE OF CONTENTS

1 Introduction and background

1.1 Introduction to BNET API and MESA radar control

Echodyne's BNET API software is a lightweight and flexible interface to Echodyne MESA radars. The structure of BNET is such that the interface can be accessed across various computing languages such as C, C++, Java, or MATLAB. The BNET API is also available as an executable Windows program that provides a shell command line interface to the user with scripting features. The simplicity and flexibility of BNET allows for robust and efficient communication with the radar.

Key benefits of the BNET include:

1) Access to data output from MESA radars
2) File logging ability to record all outputted radar data
3) Lightweight, flexible, and simple
4) Object oriented design has support for performing crucial mission critical operations on big data

1.2 Functional Description of the BNET Interface

BNET is an API that allows a Windows or Linux PC to interface with the MESA radar's gigabit Ethernet connection. The base API is provided through a C++ class called bnet_interface. A command line interface is provided in the BNET executable, which is implemented using the bnet_interface class. The executable is a straightforward way to allow any PC to interface with a radar unit. For more specific applications, the bnet_interface class can be used to interface with the radar using C++. For Windows, a MATLAB interface is also provided.

1.3 Version Number

The BNET version number contains 3 fields:

[MAJOR].[MINOR].[PATCH]

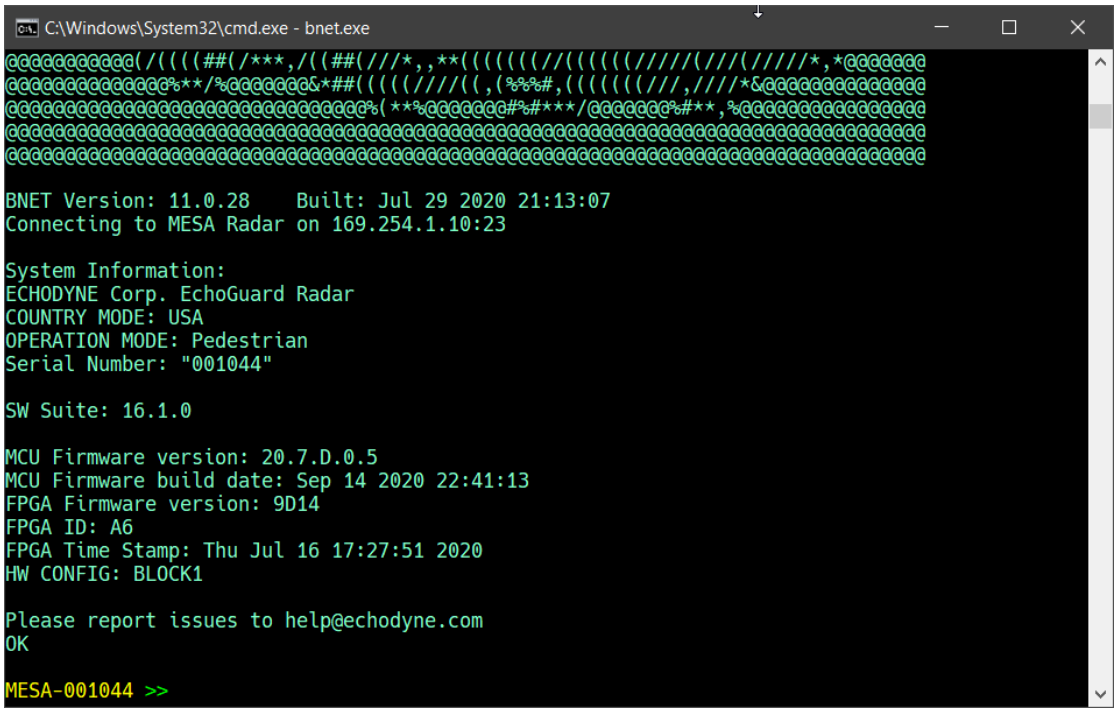
The fields of the version number serve as a quick way to understand what a given version of BNET is compatible with. When the major version is changed, incompatible API changes have been introduced. When the minor version is changed, backwards-compatible features are added. When the patch version is changed, backwards-compatible bug fixes are made.

2 BNET Standalone Interface

The simplest way to interface with a MESA radar unit is to open the standalone executable version of BNET. The executable offers a shell-like interface to the radar. In addition to sending commands and receiving responses, the executable version of BNET has a variety of additional features:

- Connecting to MESA radar data ports
- Saving collected radar data to hard disk
- Scripting command sequences (with timing)
- Command logging
- Firmware update utility

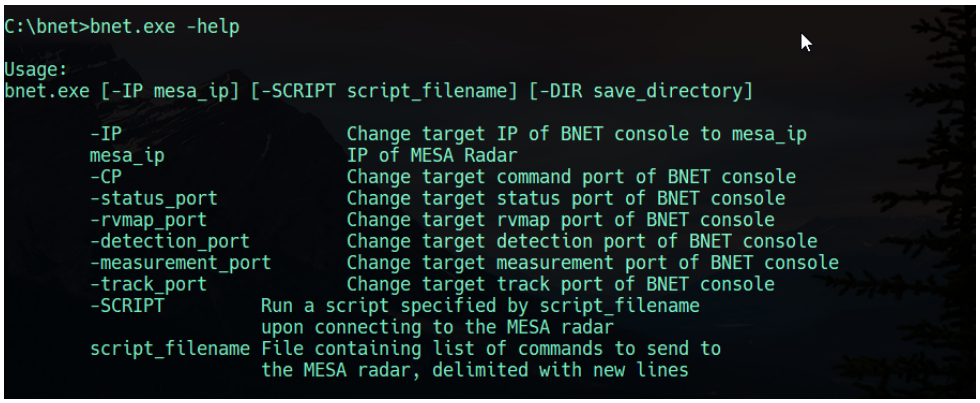
Upon opening an instance of BNET, a startup screen is printed showing a version summary of the associated software, firmware and MESA radar hardware. At the bottom is the BNET command prompt '**>>**' which will allow you to enter a variety of BNET and MESA radar commands as described in the following sections.



2.1 Command Line Options

For ease of use the BNET interface defaults to the IP address and port 169.254.1.10:23. By default the save location for radar data is a time stamped folder that is created in the root folder location of the BNET executable. These defaults can be changed with the following command line flags.

Type bnet.exe -help to see command line options.



2.1.1 Custom IP Address

Calling BNET with the command line flag "**-IP"** followed by the desired custom IP address will allow BNET to interface with a radar unit that does not have the standard address. Below is an example where one connects to a radar unit that has the IP address: 169.254.1.**20**.

2.1.2 Custom Command Port

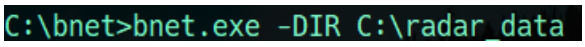
Calling BNET with the command line flag "-CP" followed by the desired custom command port will allow BNET to interface with a radar unit that does not have the standard command port.

2.1.3 Custom Data Ports

Each data port has its own flag that allows the user to override the default port. Note that as long as the IP and command port are correct, BNET will successfully connect to the radar. BNET will only throw a connection exception when it fails to enable (connect) a data port.

2.1.4 Custom Save Location

Calling BNET with the command line flag "**-DIR"**, followed by either a relative or absolute file path allows one to specify a custom save location for radar data. Below is an example starting BNET and saving to a folder on the C drive called "Radar_Data".



2.1.5 Startup Script

It is possible for BNET to start by first executing a script using the command line flag "**-SCRIPT"**. Scripts come in the form of text files that have commands separated by line breaks and are discussed at length in later sections of this manual. Below is an example starting BNET with a custom script called "configure_settings.txt".

2.1.6 Summary of Command Line Options

Option Flag	Option Argument
-IP	IP Address of MESA radar
-CP	Command Port of MESA radar
-status_port	Port for status data
-rvmap_port	Port for rvmap data
-detection_port	Port for detection data
-track_port	Port for track data
-measurement_port	Port for measurement data
-DIR	Direct or relative path to save directory
-SCRIPT	Direct or relative path to script text file

2.2 Sending and Receiving Commands

Commands not specific to a BNET executable version will pass through to the MESA radar unit to which it is connected. To send a command, write the string associated with that command and press enter. For available commands, see the MESA radar manual. Below is an example of sending a command to the MESA radar along with its response.

```
MESA-900365 >> ETH:IP?
Current Ethernet configuration:
  IP Address      : 169.254.1.10
  Netmask         : 255.255.0.0
  Default gateway: 0.0.0.0
Ethernet configuration on next boot:
  IP Address      : 169.254.1.10
  Netmask         : 255.255.0.0
  Default gateway: 0.0.0.0
OK
```

For more information about the types of commands and related responses, please reference the MESA Radar Manual document.

It is possible to access a command history of previously sent commands for a specific instance of BNET by scrolling through the history using the up and down arrow keys.

2.3 Connecting to MESA Radar data ports

By default, the BNET executable interface does not connect to MESA radar data ports, only the command interface port (23). This allows users to subscribe to only the data ports they require and consume less computing resources for a given task. There are four data ports described in the MESA Radar Manual and they each have commands to enable the connection, disable the connection, enable file saving and disable file saving:

- enable_rvmap
- enable_detection
- enable_status
- enable_track
- enable_measurements
- enable_rvmap_logging
- enable_detection_logging
- enable_status_logging
- enable_track_logging
- enable_measurements_logging

- disable_rvmap
- disable_detection
- disable_status
- disable_track
- disable_measurements
- disable_rvmap_logging
- disable_detection_logging
- disable_status_logging
- disable_track_logging
- disable_measurements_logging

2.3.1 Data File Chunking

To allow BNET to save files to disk with a reduced probability of experiencing memory or disk access issues, especially with the high bandwidth RVmaps data packets, an option to create files with concatenated data packets has been created. The two command are:

- enable_chunking
- disable_chunking

By default, the packet 'chunking' option is enabled. When enabled, data packets (status, detections tracks and RVmaps) will be individually concatenated (chunked) by data port and saved when a sufficient number of the respective data packets have been accumulated for each type. The data packets will be in the same file locations as before in the normal mode. The files can be separated by parsing for the respective packet flags.

2.4 Scripting Command Sequences

It is possible to have BNET automatically step through a list of commands written in a text file and delimited by line breaks. Using a BNET command called pause, it is possible to insert timing into these scripts. Scripts can be initiated either upon startup of BNET or during normal operation from the command line interface. During normal operation, the special command is "**SCRIPT:"** and the argument is a relative or absolute path to a text file that contains the command sequence desired. When a script is called from within a script it returns to the script that called it, which allows for reuse of scripts for common tasks, such as enabling all data ports.

Commands specific to BNET as well as commands specific to MESA radar are allowed in scripts. Below is an example text file script for setting a custom azimuth search range and collecting data from the radar.

Script comments are represented by "**##**" symbols at the beginning of a line.

The example script below starts with a comment line, then connects to the RVmap data port, enables logging of the RVmap data, resets the radar to factory defaults, changes the full azimuth default scan (from +/-60 deg 2 deg steps) to a limited +/-30 degree FOV with 5 degree steps, starts a data collection for 1000 milliseconds and then automatically stops the radar.

```
## Example script
enable_rvmap
enable_rvmap_logging
RESET:PARAMETERS
MODE:SEARCH:AZFOVMIN -30
MODE:SEARCH:AZFOVMAX 30
MODE:SEARCH:AZSTEP 5
MODE:SEARCH:START
pause 1000
MODE:SEARCH:STOP
```

2.4.1 Pause Feature (Script Timing)

Scripts normally execute as fast as the MESA radar can respond, but sometimes pauses are required for interfacing with outside hardware, or when human intervention is needed for a test. The pause command has the syntax "**pause**". If there is no argument, BNET will wait for a user to press the enter button on the keyboard. If there is an argument, it parses an integer that tells it how many milliseconds to pause for. Below are examples of using the pause feature:

```
MESA-900365 >> pause 50
MESA-900365 >> pause 1000
MESA-900365 >> pause
Press enter to continue...
MESA-900365 >> pause 10000
MESA-900365 >>
```

2.5 Command Logs

BNET saves the command history with time stamps to a folder in the root directory of the BNET executable called "log_files". These logs are useful for knowing the exact timing and command sequence that was used to produce associated data. These command logs are generated for all version of BNET, which include the executable, MATLAB and DLL versions. Below is an example log file after using the BNET interface for some time:

```
[20:57:51:502] BNET API: Connected.
[20:57:51:505] BNET:
*IDN?

[20:57:51:509] MESA:
ECHODYNE Corp. DAA Radar
Serial Number: "343434"
MCU Firmware version: 10.6.6.0.5
MCU Firmware build date: Nov 14, 2017 02:53:19
FPGA Firmware version: 6398
FPGA ID: ACD51302
FPGA Time Stamp: Wed Oct 4 16:58:29 2017

Please report issues to help@echodyne.com
OK

[20:57:51:517] BNET API: STATUS data collection set to 1.
[20:57:51:521] BNET API: RVMAP data collection set to 1.
[20:57:51:525] BNET API: DETECTION data collection set to 1.
[20:57:51:528] BNET API: TRACK data collection set to 1.
[20:57:51:532] BNET API: STATUS data logging set to 1.
[20:57:51:536] BNET API: RVMAP data logging set to 1.
[20:57:51:539] BNET API: DETECTION data logging set to 1.
[20:57:51:541] BNET API: TRACK data logging set to 1.
[20:57:51:544] BNET:
MODE:SWT:START

[20:57:51:607] MESA:
OK

[22:57:51:730] BNET API: STATUS data collection set to 0.
[22:57:52:74] BNET API: RVMAP data collection set to 0.
[22:57:52:77] BNET API: DETECTION data collection set to 0.
[22:57:52:81] BNET API: TRACK data collection set to 0.
[22:57:52:84] BNET API: STATUS data logging set to 0.
[22:57:52:87] BNET API: RVMAP data logging set to 0.
[22:57:52:90] BNET API: DETECTION data logging set to 0.
[22:57:52:93] BNET API: TRACK data logging set to 0.
[22:57:52:95] BNET:
MODE:SWT:STOP

[22:57:52:108] MESA:
OK

[22:57:52:111] BNET:
*TST?

[22:57:52:114] MESA:
--------------- SYSTEM INIT STATUS---------------
SDRAM: 0
GLOBAL INTERUPT INIT:  0
```

```
CPU INTERUPT INIT: 0
GLOBAL TIMER: 0
FPGA: 0
SPI: 0
GPIO INIT: 0
GPIO CFG: 0
CPU INTERRUPT ENABLE: 0
GLOBAL INTERRUPT ENABLE: 0
EMMC INIT: 0
ETHERNET INIT: 0
I2C INIT: 0
IMU INIT: 0
CACHCE INIT: 0
--------------- ETHERNET STATUS -----------------
SPEED: 1000 Mb/s
--------------- ERROR LOG ---------------
OK
```

2.6 HPS Firmware Update Utility

To simplify the firmware update procedure, BNET can manage the TFTP commands that have to be sent in order to program new firmware images to the MESA radar unit. The prerequisites are that the user must have the Windows TFTP feature enabled (which is turned off by default) as well as windows firewall exceptions made for the TFTP program. On Linux, the command tftp must be accessible from the console. Once the proper firewall exception rules have been set for the TFTP feature of windows, the BNET command for updating the firmware of the MESA radar unit is “firmware_update”. It takes an argument which is the file name of the HPS image to load. For example, for an HPS image with the file name “521-0013-921_A_Rev16.img”, the command would be: “firmware_update 521-0013-921_A_Rev16.img”. Below is a screenshot of a successful firmware update.

```
MESA-900365 >> firmware_update 521-0013-921_A_Rev16.img
HPS firmware update initiated.
        Loading firmware into image slot: 2
        TFTP -i 169.254.1.10 PUT 521-0013-921_A_Rev16.img
Receiving 521-0013-921_A_Rev16.img into image 2
................................................................
................................................................
................................................................
```

(many rows of dots as the image is uploaded)

```
................................................................
................................................................
................................................................
.........................Transfer successful: 2417440 bytes in 11 second(s)
, 219767 bytes/s
               TFTP complete.
.....................
Received

       Exiting FW Update mode
Update Complete!
Reset or power cycle the MESA radar for these changes to take effect.
MESA-900365 >>
```

The utility loads the firmware image in this example named “521-0013-921_A_Rev16.img” in the root directory of the BNET executable. After this command completes successfully, the unit must be power cycled to complete loading the updated firmware.

***** NOTE: HPS images must always be updated 1st and power cycled before updating FPGA images.**

***** NOTE: Firmware update utilities are not supported on Linux.**

2.7 FPGA Update Utility

The FPGA image can be updated through a similar utility as the HPS firmware. TFTP commands are automatically sent to send a new FPGA image to the MESA radar unit. Windows TFTP must be enabled as well as firewall exceptions made for the TFTP program. On Linux, the command tftp must be accessible from the console. Once the OS has been configured to allow TFTP to access the MESA radar, the command for updating the FPGA image of the MESA radar unit is “fpga_update”. It takes an argument, which is the file name of the FPGA image to load. For example, for an FPGA image with the file name “21-0013-921_F_Rev1.fpga”, the command would be: “fpga_update 21-0013-921_F_Rev1.fpga”. Below is a screen shot of a successful FPGA image update:

```
MESA-900365 >> fpga_update 521-0013-921_F_Rev1.fpga
FPGA firmware update initiated.
        Loading firmware into image slot: 1
        TFTP -i 169.254.1.10 PUT 521-0013-921_F_Rev1.fpga
Receiving 521-0013-921_F_Rev1.fpga into image 1
.............................................................
.............................................................
.............................................................
```

(many rows of dots as the image is uploaded)

```
.............................................................
.............................................................
.....................................................Transfer successful:
3697908 bytes in 15 second(s), 246527 bytes/s
               TFTP complete.
....
Received

        Exiting FW Update mode
Update Complete!
Reset or power cycle the MESA radar for these changes to take effect.
MESA-900365 >>
```

***** NOTE: FPGA images must always be updated 2nd after updating HPS images and power cycling.**

***** NOTE: Firmware update utilities are not supported on Linux.**

2.8 POSIX Time Utility

It is sometimes useful to set the radar to POSIX time. The command to do so using the BNET standalone interface is “set_posix”. This command uses the system clock’s time, converted to POSIX Time, to set the radar’s clock to be days and milliseconds since January 1, 1970.

3 BNET MATLAB Interface

For more complex processing of MESA radar output data than just saving to disk, a MATLAB interface version of the BNET to the MESA radar is provided. It is built using the BNET DLL described later in this document and is the exact same source code as the executable. The MATLAB interface allows for radar data to be accessed directly in MATLAB without loading files that have been saved to disk.

The BNET MATLAB interface takes the form of a MATLAB class object that interfaces with a C++ class that has an instance of the BNET DLL. It has some special features that are useful for testing and control of the MESA radar unit:

- MATLAB Class Object
- Connecting to MESA data ports
- Saving of collected radar data to hard disk
- Command logging
- Data buffering configuration
- Access to data through shared memory
- Access to error messages, error management

3.1 MATLAB Class Object – Member Functions

The BNET MATLAB interface comes in the form of a class object. It can be created using the constructor and is implicitly disconnected and deleted when it is cleared from the MATLAB workspace. Below is a summary of the available member commands, through the file bnet_interface.m:

3.1.1 Class Constructor and Destructor

bnet_interface	
Class constructor, creates an instance of the class object.	
Inputs	N/A
Outputs	BNET Class Object
Example	`bnet = bnet_interface();`

getVersion	
Get the version string from BNET	
Inputs	N/A
Outputs	BNET Version string
Example	`bnet = bnet_interface();` `version = bnet.getVersion();`

delete	
Class destructor, destroys an instance of the class object.	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`bnet.delete();`
	
	`bnet = bnet.interface();`
	`clear bnet;`

3.1.2 Connection Functions

connect	
Connect to a MESA radar unit at a given IP address, port and optionally custom save location for MESA radar data output.	
Inputs	[IP-ADDRESS] IP address of the MESA radar unit as a string [PORT] – Port of the MESA radar unit's interface port as an integer [SAVE_DIRECTORY] – *Optional* – Path to custom save directory as a string
Outputs	Connection status – 0 for success, 1 for failure
Example	`bnet = bnet_interface();` `status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`

isConnected	
Get the current connection state of the BNET object.	
Inputs	N/A
Outputs	Connection status – 1 for Connected, 0 for disconnected

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`connection_state = bnet.isConnected();`

disconnect	
Disconnect from the MESA radar unit.	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.disconnect();`

reconnect	
Reconnect to the MESA radar that the unit was last connected to. Equivalent to disconnecting and connecting back to the original MESA radar that was initially connected to using connect.	
Inputs	N/A
Outputs	Status, 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`status = bnet.reconnect();`

set_dataSource	
Set the data source of the BNET data get functions to be either TCP or a previously saved data folder. The argument 'TCP' is set as default, but can be used to revert a BNET object back into tcp connection mode. Otherwise, the data source can be set to any path to a BNET save data folder. Subsequent calls to get_rvmap, get_detection, get_status, etc. will return packets from the source folder.	
Inputs	source
Outputs	N/A

Example	`bnet = bnet_interface();`
	`bnet.set_dataSource('my_data_folder');`
	`rvmap = bnet.get_rvmap();`

set_collect_rvmap	
Enable or disable collection of RVMAP Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_collect_rvmap(1);`

set_collect_status	
Enable or disable collection of status Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_collect_status(1);`

set_collect_detections	
Enable or disable collection of detection list Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_collect_detections(1);`

set_collect_tracks	
Enable or disable collection of track Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_collect_tracks(1);`

set_collect_measurements	
Enable or disable collection of measurement Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_collect_measurements(1);`

get_collect_rvmap	
Check if we have enabled or disabled collection of RVMAP Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_collect_rvmap();`

get_collect_status	
Check if we have enabled or disabled collection of status Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_collect_status();`

get_collect_detections	
Check if we have enabled or disabled collection of detection list Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_collect_detections();`

get_collect_tracks	
Check if we have enabled or disabled collection of track Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_collect_tracks();`

get_collect_measurements	
Check if we have enabled or disabled collection of measurement Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_collect_measurements();`

3.1.3 File Logging Functions

set_chunking	
Enable or disable chunking of saved of data packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_chunking(1);`

set_save_rvmap	
Enable or disable saving of RVMAP Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_save_rvmap(1);`

set_save_status	
Enable or disable saving of status Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_save_status(1);`

set_save_detections	
Enable or disable saving of detection list Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_save_detections(1);`

set_save_tracks	
Enable or disable saving of track Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_save_tracks(1);`

set_save_measurements	
Enable or disable saving of measurement Packets from the MESA radar unit.	
Inputs	[Enable] 0 to disable and 1 to enable
Outputs	Status – 0 for success, 1 for failure

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`resp = bnet.set_save_measurements(1);`

get_save_rvmap	
Check if we have enabled or disabled saving of RVMAP Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_save_rvmap();`

get_save_status	
Check if we have enabled or disabled saving of status Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_save_status();`

get_save_detections	
Check if we have enabled or disabled saving of detection list Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_save_detections();`

get_save_tracks	
Check if we have enabled or disabled saving of track Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_save_tracks();`

get_save_measurements	
Check if we have enabled or disabled saving of measurement Packets from the MESA radar unit.	
Inputs	N/A
Outputs	[Enabled] 0 to disable and 1 to enable

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`state = bnet.get_save_measurements();`

3.1.4 Folder Setting Functions

newFolder	
Starts a new data collection folder with a new timestamp.	
Inputs	N/A
Outputs	N/A
Example	`bnet = bnet_interface();` `bnet.newFolder();`

saveFolder	
Forces a save of all data that may have not been committed to disk.	
Inputs	N/A
Outputs	N/A
Example	`bnet = bnet_interface();` `bnet.saveFolder();`

getFolder	
Retrieve a string that contains the folder name of the current folder that data is being saved to.	
Inputs	N/A
Outputs	Directory that data is currently being saved to

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`folder_name = bnet.getFolder();`

3.1.5 Buffer Configuration Functions

set_rvmap_bufferLength	
Set the length of the buffer that holds rvmap data. Range velocity maps are saved on system memory up until this length and are overwritten once the length has been reached.	
Inputs	[length] – Length of the buffer
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_rvmap_bufferLength(100);`

get_ rvmap _bufferLength	
Get the length of the buffer for rvmap related data	
Inputs	N/A
Outputs	Length of the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`beam_buf_length = bnet.get_rvmap_bufferLength();`

get_ rvmap _n_buffered	
Get the number of rvmaps that have data saved in the buffer.	
Inputs	N/A
Outputs	Number of packets in the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`n_buffered = bnet.get_rvmap_n_buffered();`

flush_ rvmap _buffer	
Flush out the contents of the buffer that holds rvmap data	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.flush_rvmap_buffer();`

set_detec_bufferLength	
Set the length of the buffer that holds detection data. Detection lists are saved on system memory up until this length, and are overwritten once the length has been reached.	
Inputs	[length] – Length of the buffer
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_detec_bufferLength(100);`

get_ detec _bufferLength	
Get the length of the buffer for detection related data	
Inputs	N/A
Outputs	Length of the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`beam_buf_length = bnet.get_detec_bufferLength();`

get_ detec _n_buffered	
Get the number of detections that have data saved in the buffer.	
Inputs	N/A
Outputs	Number of packets in the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`n_buffered = bnet.get_detec_n_buffered();`

flush_ detec _buffer	
Flush out the contents of the buffer that holds detection data	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.flush_detec_buffer();`

set_track_bufferLength	
Set the length of the buffer that holds track data. Track packets are saved on system memory up until this length and are overwritten once the length has been reached.	
Inputs	[length] – Length of the buffer
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_track_bufferLength(100);`

get_track_bufferLength	
Get the length of the buffer for beam related data	
Inputs	N/A
Outputs	Length of the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`track_buf_length = bnet.get_track_bufferLength();`

get_track_n_buffered	
Get the number of beams that have track data saved in the buffer.	
Inputs	N/A
Outputs	Number of packets in the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`n_buffered = bnet.get_track_n_buffered();`

flush_track_buffer	
Flush out the contents of the buffer that holds beam data	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.flush_track_buffer();`

set_status_bufferLength	
Set the length of the buffer that holds status data. Status packets are saved on system memory up until this length and are overwritten once the length has been reached.	
Inputs	[length] – Length of the buffer
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_status_bufferLength(100);`

get_status_bufferLength	
Get the length of the buffer for status related data	
Inputs	N/A
Outputs	Length of the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`beam_buf_length = bnet.get_status_bufferLength();`

get_status_n_buffered	
Get the number of beams that have data saved in the buffer.	
Inputs	N/A
Outputs	Number of packets in the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`n_buffered = bnet.get_status_n_buffered();`

flush_status_buffer	
Flush out the contents of the buffer that holds status data	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.flush_status_buffer();`

set_measurements_bufferLength	
Set the length of the buffer that holds measurement data. Measurement packets are saved on system memory up until this length and are overwritten once the length has been reached.	
Inputs	[length] – Length of the buffer
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_measurements_bufferLength(100);`

get_measurements_bufferLength	
Get the length of the buffer for measurement related data	
Inputs	N/A
Outputs	Length of the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`meas_buf_length = bnet.get_measurements_bufferLength();`

get_measurements_n_buffered	
Get the number of beams that have measurement data saved in the buffer.	
Inputs	N/A
Outputs	Number of packets in the buffer

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`n_buffered = bnet.get_measurements_n_buffered();`

flush_measurements_buffer	
Flush out the contents of the buffer that holds measurement data	
Inputs	N/A
Outputs	N/A

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.flush_measurements_buffer();`

3.1.6 Data Collection Functions

get_tracks	
Get a track packet from the MESA radar unit that has arrived on the PC's TCP sockets during some type of tracking mode. The data that comes out of this function is the oldest piece of data in the configurable buffer.	
Inputs	N/A
Outputs	tracks – One track data structure, as described in the MESA radar manual

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_collect_tracks(1);`
	`bnet.query('MODE:SWT:START');`
	`tracks = bnet.get_tracks();`

get_meas	
Get a measurement packet from the MESA radar unit that has arrived on the PC's TCP sockets during some type of tracking mode. The data that comes out of this function is the oldest piece of data in the configurable buffer.	
Inputs	N/A
Outputs	meas – One measurements data structure, as described in the MESA radar manual

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_collect_measurements(1);`
	`bnet.query('MODE:SWT:START');`
	`measurements = bnet.get_meas();`

get_status	
Get a status packet from the MESA radar unit that has arrived on the PC's TCP sockets during operation of the unit. The data that comes out of this function is the oldest piece of data in the configurable buffer.	
Inputs	N/A
Outputs	status – One status data structure, as described in the MESA radar manual

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_collect_status(1);`
	`status = bnet.get_status();`

get_rvmap	
Get a rvmap packet from the MESA radar unit that has arrived on the PC's TCP sockets during operation of the unit. The data that comes out of this function is the oldest piece of data in the configurable buffer.	
Inputs	N/A
Outputs	rvmap – One rvmap data structure, as described in the MESA radar manual

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_collect_rvmap(1);`
	`rvmap = bnet.get_rvmap();`

get_detections	
Get a detection packet from the MESA radar unit that has arrived on the PC's TCP sockets during operation of the unit. The data that comes out of this function is the oldest piece of data in the configurable buffer.	
Inputs	N/A
Outputs	detection – One detection data packet structure, as described in the MESA radar manual

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`bnet.set_collect_detections(1);`
	`detection = bnet.get_detections();`

3.1.7 Sending Commands

query	
Send a string to the MESA radar that BNET is connected to. Automatically parse the response and throw an error if a response other than OK is received. For more information on the automatic error handling, see the section below.	
Inputs	cmd – A command string to be sent to the MESA radar, see the manual for a list of available commands
Outputs	status – status code that the command returned reso – Response string from the MESA radar unit

Example	`bnet = bnet_interface();`
	`status = bnet.connect('169.254.1.10', 23, 'C:\RADAR_DATA');`
	`[status, resp] = bnet.query('*IDN?');`

3.2 Example Script

The BNET MATLAB interface is provided with an example script called bnet_example.m that walks through all the functionalities of BNET and demonstrates all the expected outputs of BNET specific functions.

4 BNET Interface API

The bnet_interface C++ class allows for a connection to one MESA-K radar. It's possible to:

- Send commands and receive responses synchronously
- Get MESA-K data from the data ports in useable C++ structures
    - Control over the buffer that this data is stored in
    - Control over whether data is collected
    - Control over whether data is saved to disk
- Log API actions
    - Enable and disable logging

To include the bnet_interface class in your C/C++ project, it's possible to statically or dynamically link to libbnetinterface.a provided in the release package. bnet_interface also depends on std::filesystem, which can be linked using the linker flag -lstdc++fs.

For detailed information on the functions provided in bnet_interface, see the bnet_interface.h header file. Static and dynamic BNET libraries are available in the API folder and can be interfaced with using the provided BNETConfig.cmake file.

4.1 List of libraries:

- bnet_interface	Class for communicating with radar and saving data
- bnet_reader	Class for reading saved radar data
- bnet_plus	(deprecated) Legacy class for communicating with radar and reading data. (Maintained for compatibility reasons)

Binaries are located in the application folder:

- BNET_Executable	Console for communicating with radar and saving data
- MATLAB	MATLAB mex version of bnet_interface
- BNET_Example	Binary compiled using bnet_interface
- BNET_plus_Example	Binary compiled using bnet_plus
- saved_data_converter	Console tool for converting between chunked and unchunked saved data

4.2 Build Environment for Windows Users

BNET is built using cmake and the GNU compiler toolchain. Currently, MSYS2 and Linux builds are supported. BNET can be built using [MSYS2](https://www.msys2.org/, "MSYS2 Homepage") under MinGW 64-bit OR MinGW 64-bit. A clean MSYS2 install does *not* contain all of the required libraries for building and testing with BNET. For all of the instructions below, please use MSYS2 Mingw64 (Not plain MSYS2)

To get git installed on MSYS2 use the following line:

$ yes Y | pacman -S git

After following the initial MSYS2 setup, use the MSYS2 configuration script to get all build dependencies.

$ ./configure_msys2_env.sh

4.3 CMake Generators

It's possible to produce project files for a variety of different build tools. If you're more familiar with build tools other than cmake, look into getting started by using different generators with the -G flag. Some of the generators available:

- Makefiles (Unix, MSYS, NMake, MinGW, etc.)
- Ninja
- Visual Studio (As recent as 2019)
- Xcode
- Eclipse
- CodeBlocks

4.4 Using the provided example

Example for Visual Studio 15:

$ cmake -Bbuild -H. -G"Visual Studio 15 2017"

$ cmake --build build

4.5 Linux

BNET should build without issue on most Linux distributions. If it doesnt work, update GNU compilers to a release that supports C++17. It might also be true that cmake is out of date and will need to be updated. Catch2 can be cloned and built from [here](https://github.com/catchorg/Catch2). It maybe necessary to update the cmake distribution to one that at least supports C++17.

5 Description of BNET File Saving

For BNET function calls that are associated with saving MESA data to disk, to minimize PC I/O file system overhead, individual data packets are concatenated (aka 'chunked) together and saved as large files. Upon disconnect or when the folder structure for saving is changed, BNET will save any accumulated packets that were not yet committed to disk.

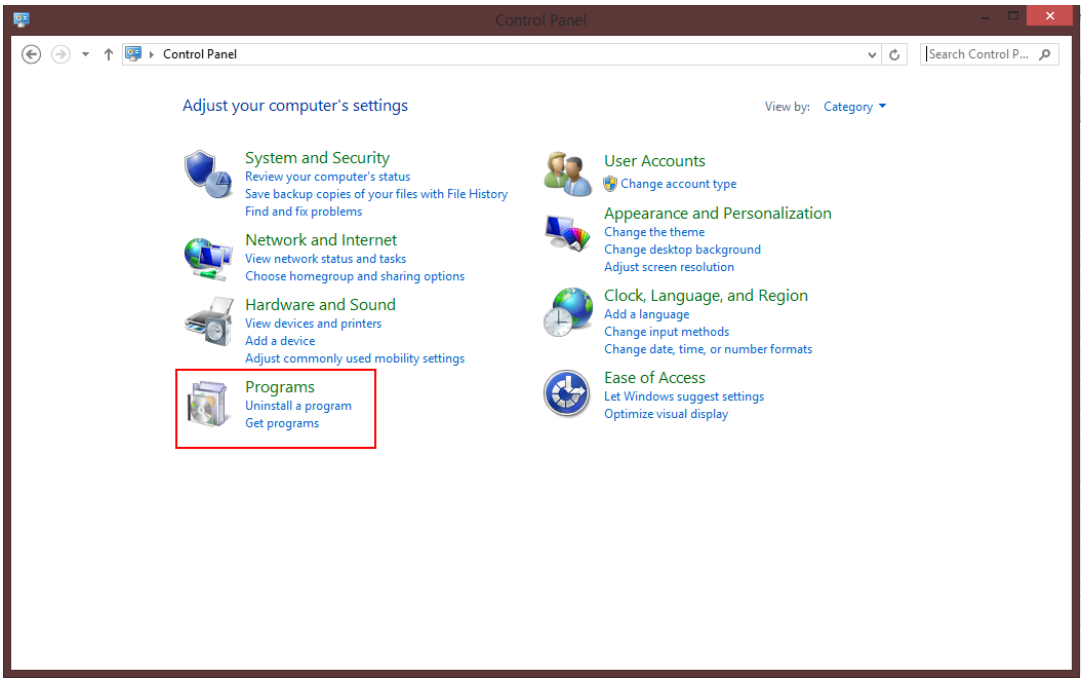
Reading files saved by BNET can be accomplished by converting these large files into many files for each packet within the file, or by setting the data folder as BNET's data source. When a folder is set as BNET's data source, read functions for each data type will behave as the packets are streaming into BNET from a MESA radar. For more information on this function see section 3.1 of this document.

5.1 Converting Files to Legacy File Logging Scheme

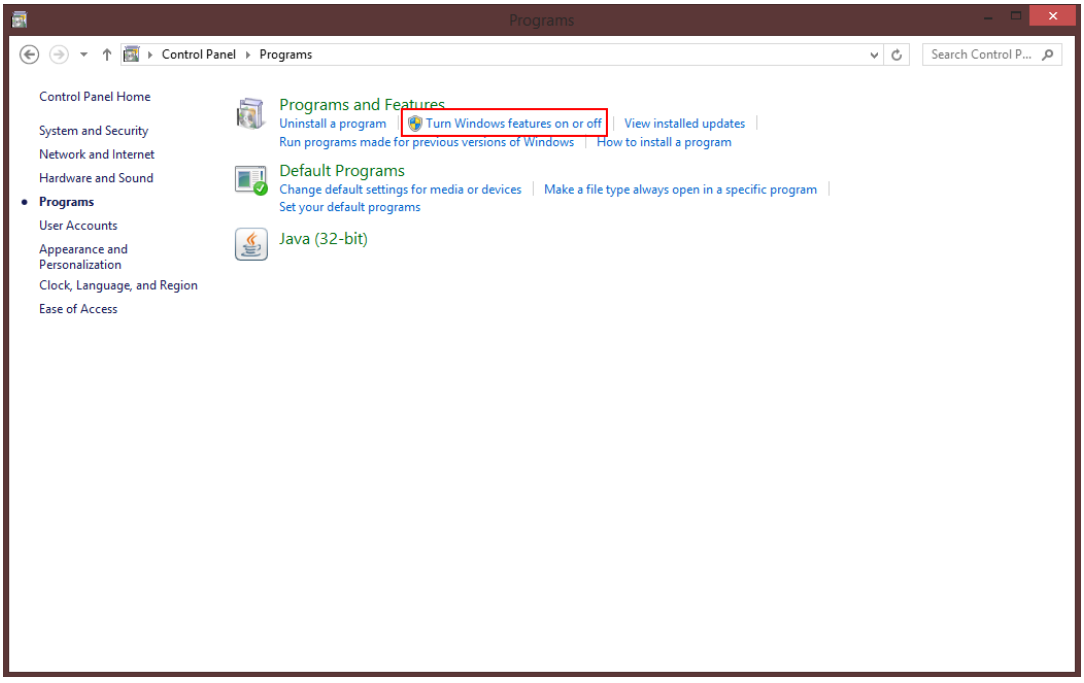
saved_data_converter.exe is provided to convert the large files of multiple packets into multiple files containing single packets. saved_data_converter.exe takes as an argument the BNET saved data folder to convert and will convert the files to legacy. This program should be invoked from the command line.

6 Installing TFTP for Firmware and FPGA Update

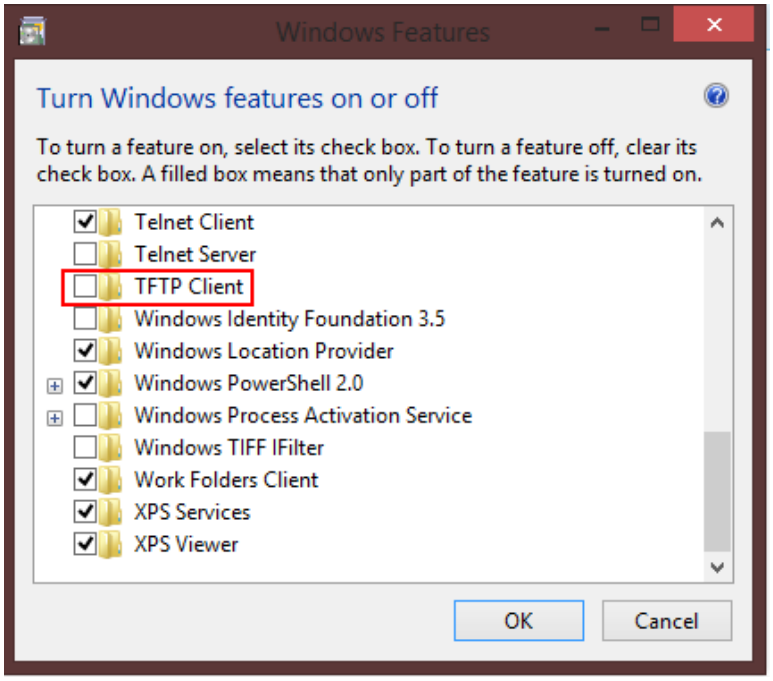
Updating the firmware utilizes the Windows OS TFTP client that is available through the command line after turning on the windows feature. To do this first open the control panel and click on programs:



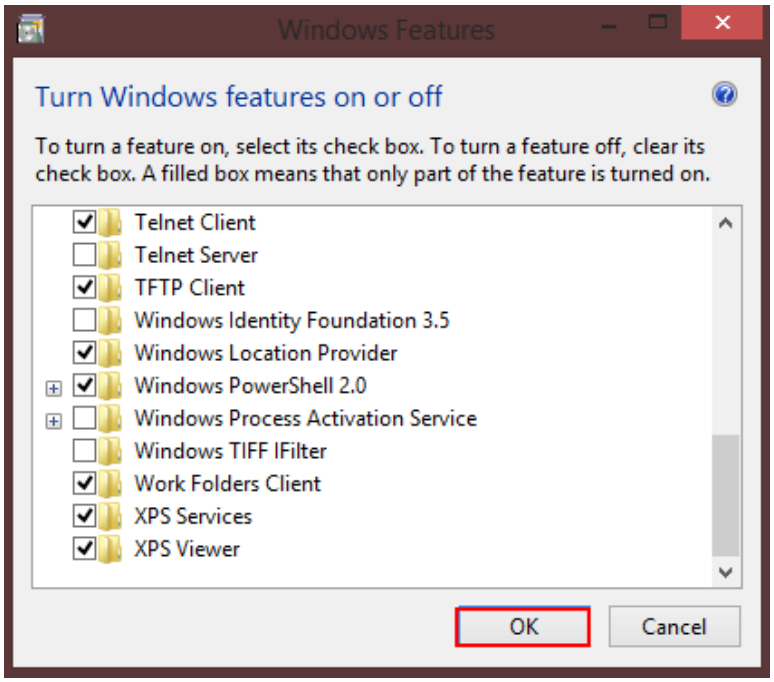
Next click on “Turn Windows features on or off” (Which requires administrator privileges)



Scroll down to find the TFTP Client

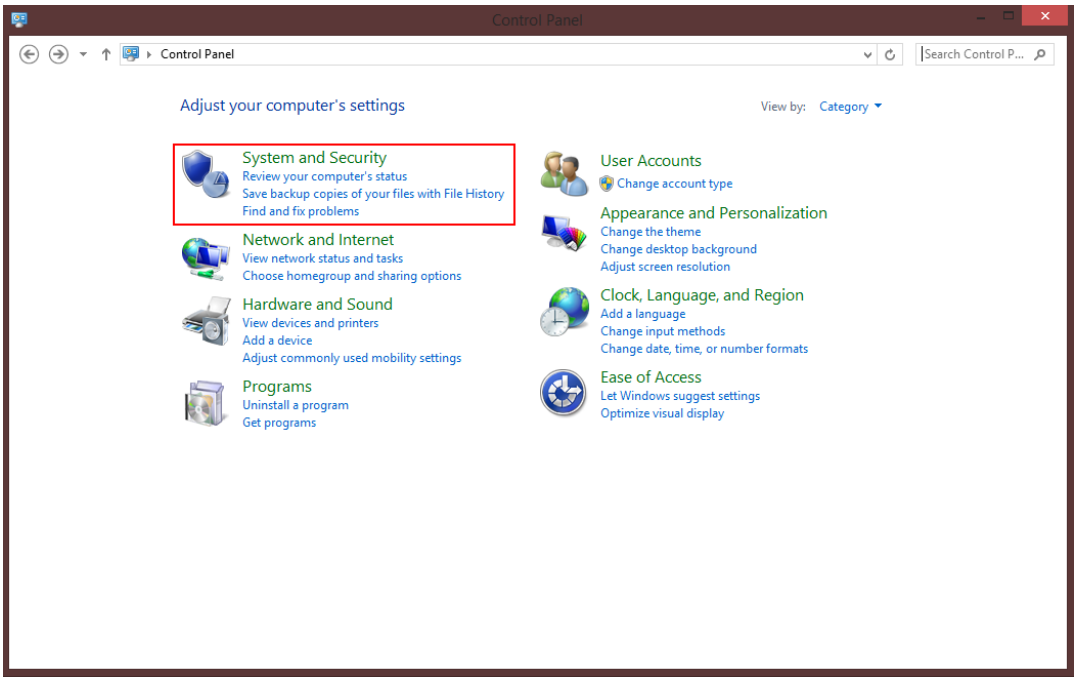


Click the TFTP Client box and press OK, Windows will require that you restart for these changes to take effect.

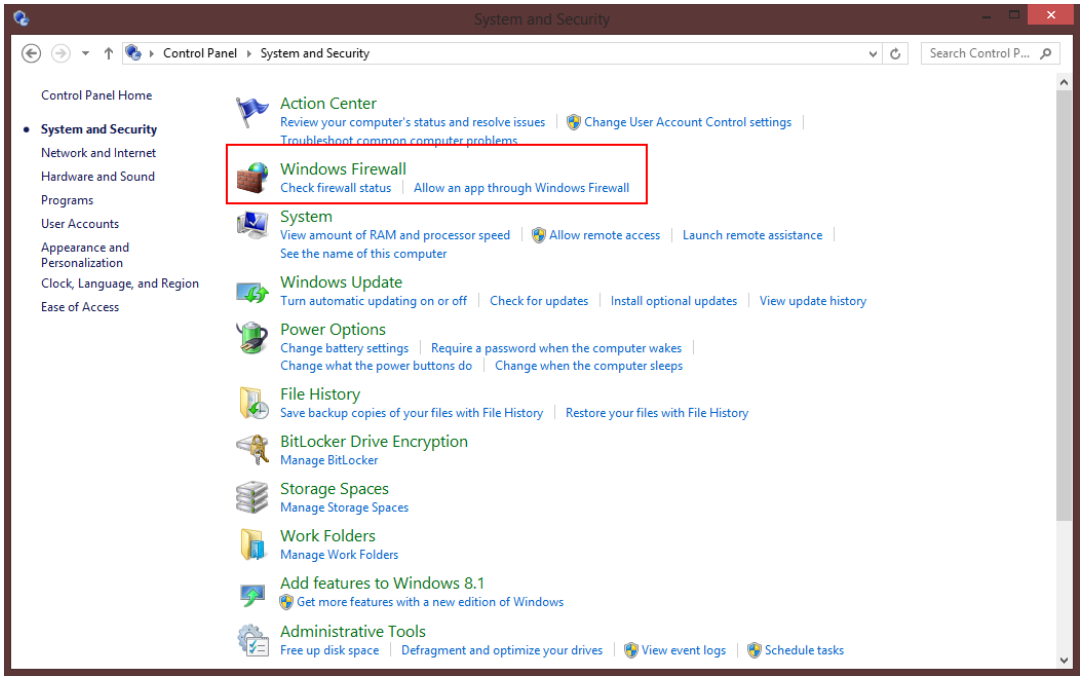


6.1 Creating a Firewall Exception for the TFTP Client

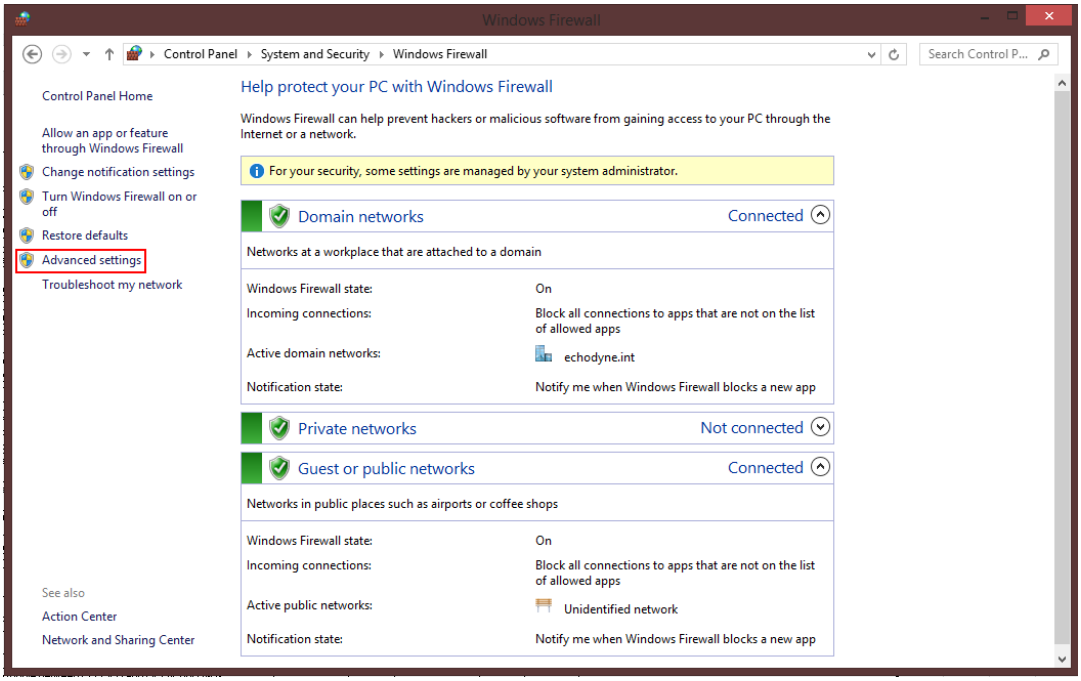
The TFTP Client needs to be able to freely connect to the DAA unit so we must allow it through the firewall. To begin open the control panel and select “System and Security”



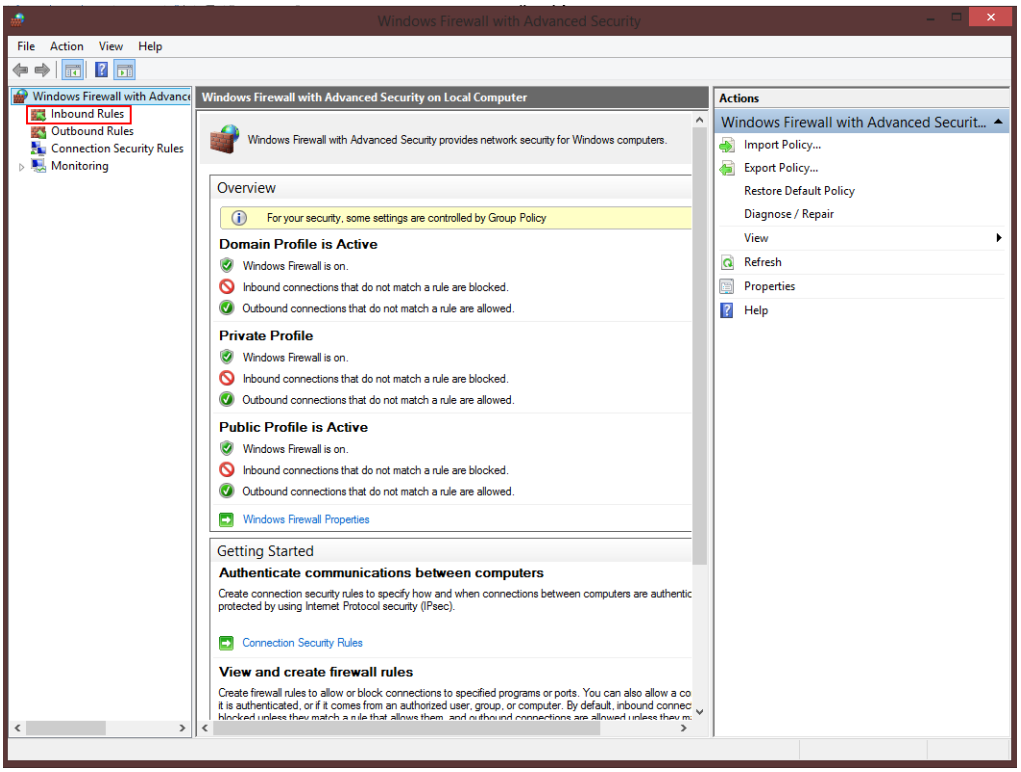
Next select the “Windows Firewall”



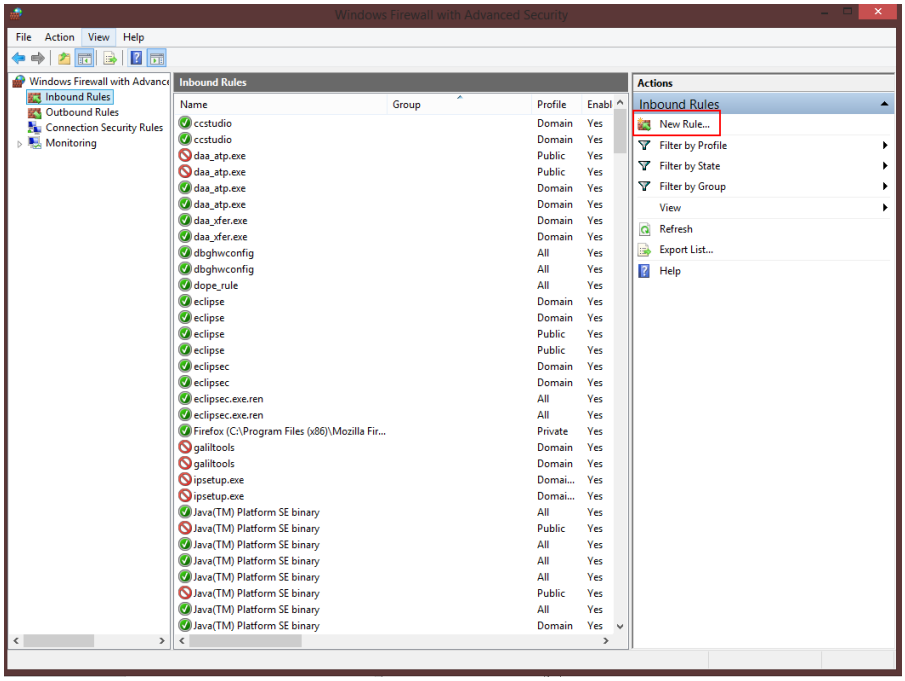
Select “Advanced Settings” so we can create our rule



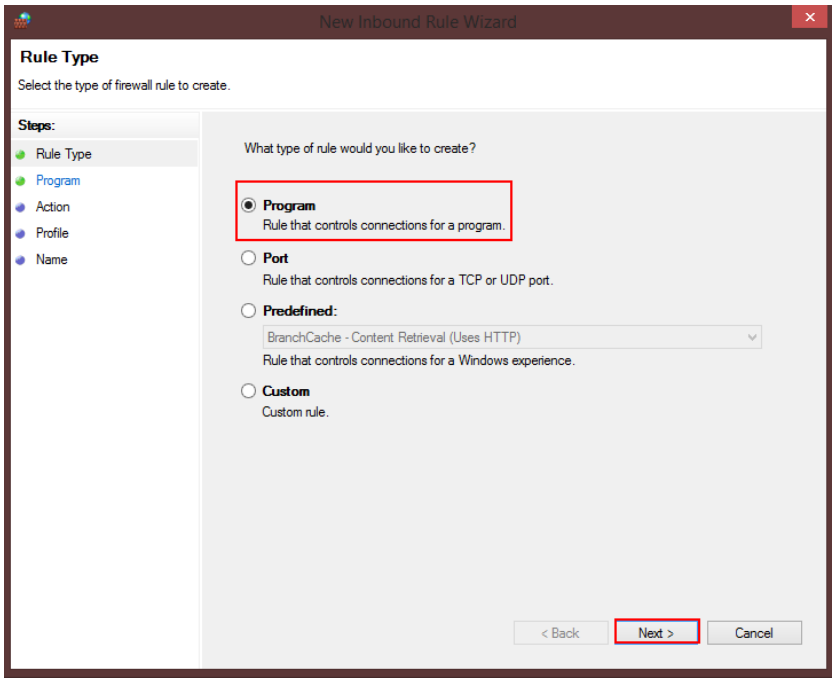
We would like to set an “inbound rule” so select this option:



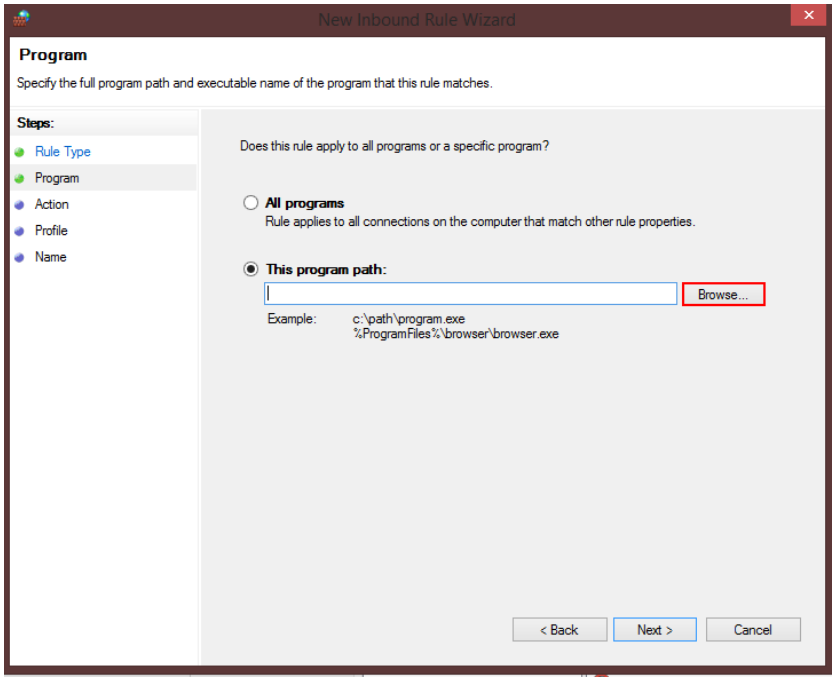
Click “new rule” to start creating the new rule:



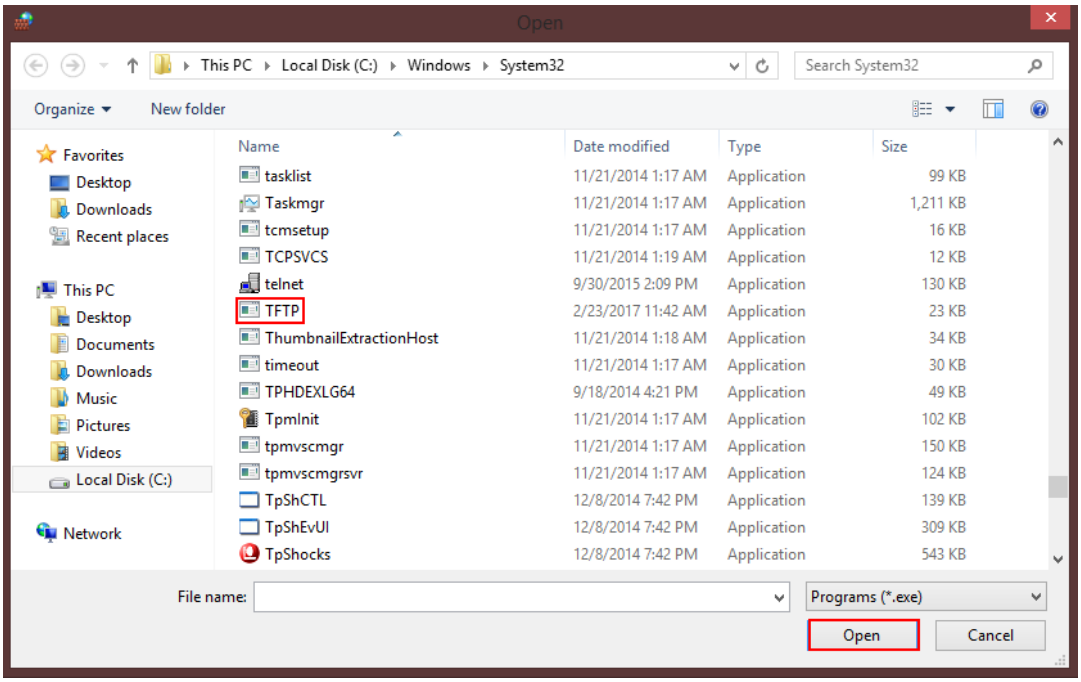
Choose “Program” for the rule type and hit next:



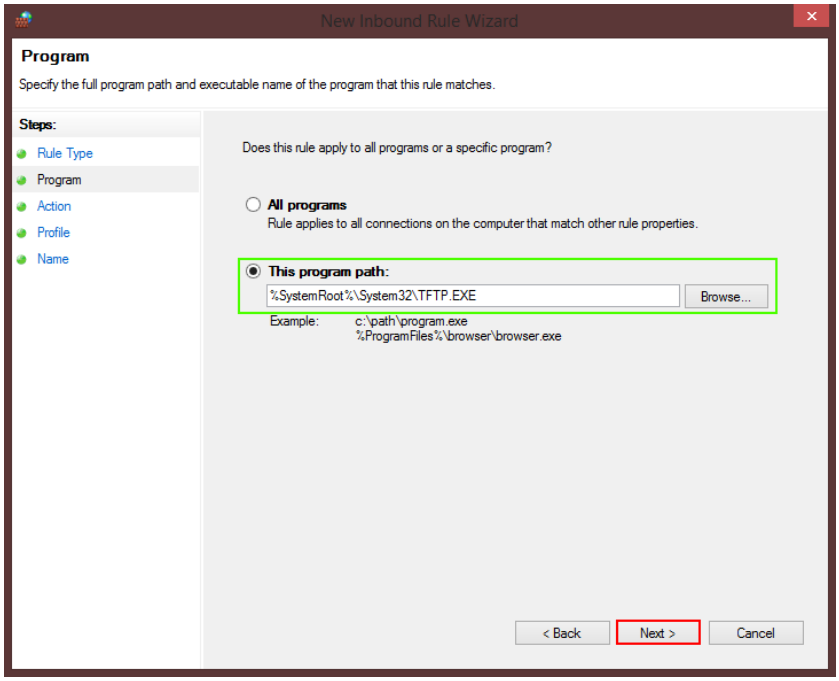
Next select browse so that we can choose TFTP:



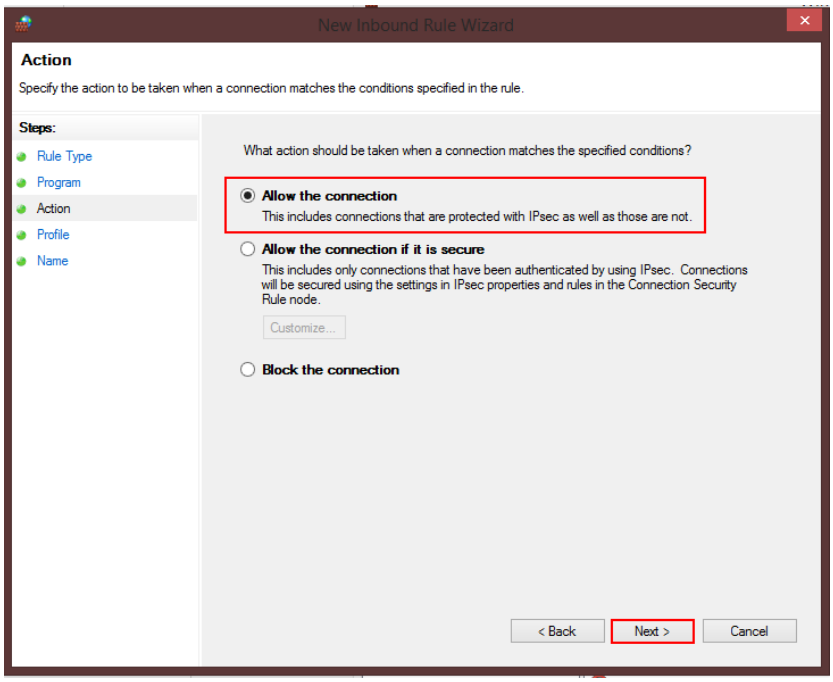
Scroll down to TFTP (if it's sorted alphabetically) and press open



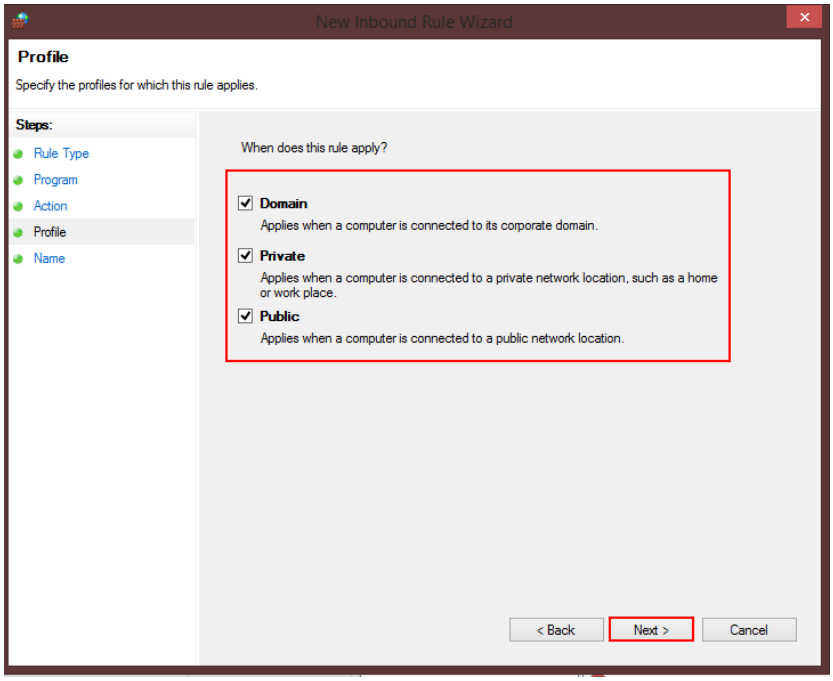
The screen should now look as below. If it does then press next:



We would like to “Allow the connection”



Allow it on the following types of networks:



The last step is to name the rule: