# Implementations - Universal Connector/MG - Manual TCP Packet Sending

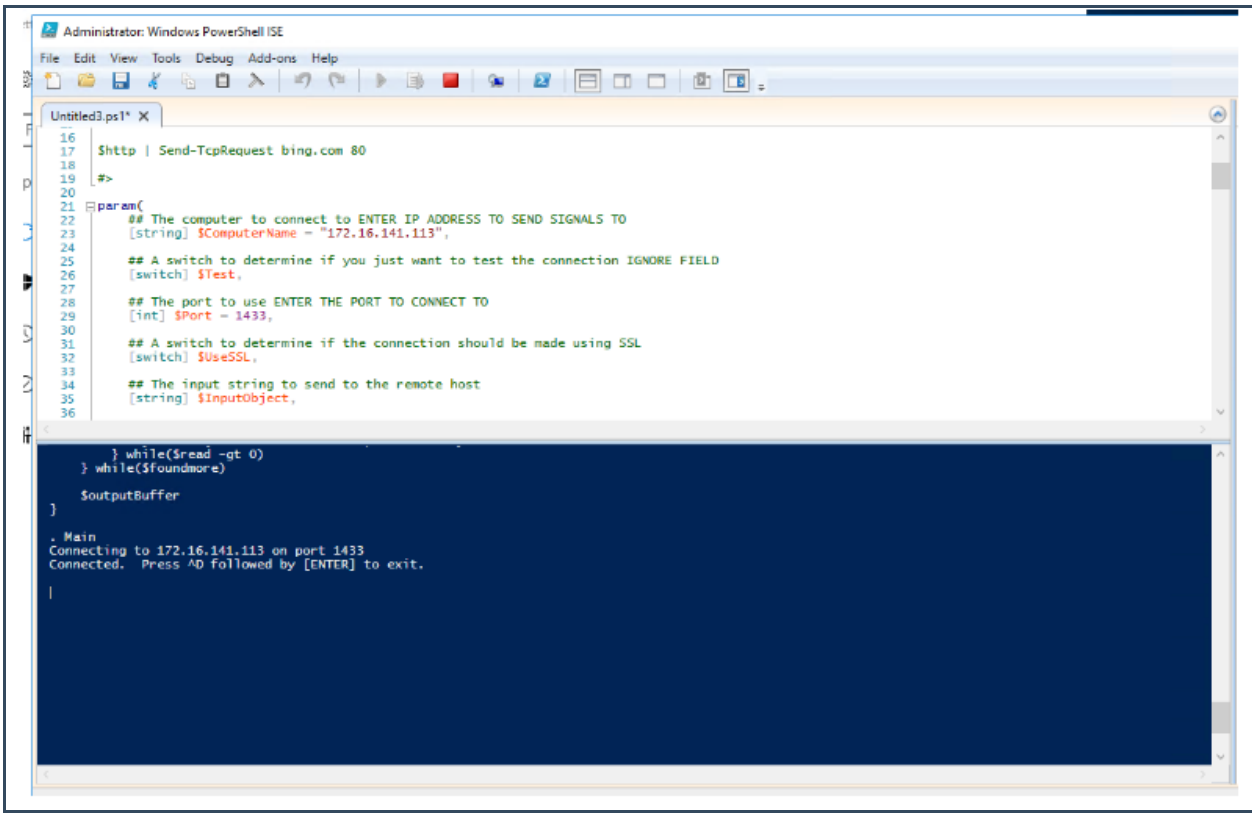
12/21/2023 12:45 pm EST

## Introduction

When troubleshooting Media Gateway or Universal Connector signals, sometimes it is important to isolate the Manitou or Media Gateway programming, and manually send a signal to the intended system. The following instructions detail the process of Manually Sending a TCP signal for isolating MG/Universal Connector function.

## Instructions

1. Launch Powershell ISE as an administrator
2. Select File, New on the Menu bar. this should separate the window into two halves.
3. In the upper field half of the Powershell window, paste the Connection Script Located Below
4. Scroll down to line 23 of the script, Change the IP address to the intended recipient for signaling
5. Scroll down to line 29 of the script, change the Port to the intended port that communication is expected
6. Select the Run button on the Quick launch bar or press F5 to execute the script
7. The script should complete successfully indicating that the IP address has been connected on the indicated port.




8. Powershell is now ready to manually transmit a signal to the target machine on the specified port.

9. It is now necessary to gather what signal the MG/Universal Connector is sending to the intended recipient, and then provide this in Unicode Format.

    - This output should be a manual match of the data being exported by the MG/UC and should be a functional match to the Data Packet settings in the MG.

10. Input this data into the lower window of Powershell pressing Enter to submit the data packet to the intended host. After a brief wait the cursor should drop an additional line, which indicates a successful transmission of signal.

11. If the signal was not received by the intended host then the issue may reside outside of the MG/UC and need help from the 3rd party vendor or customer technician.

## Connection Script

```
<#

.SYNOPSIS

Send a TCP request to a remote computer, and return the response.
If you do not supply input to this script (via either the pipeline, or the
```

```
-InputObject parameter,) the script operates in interactive mode.

.EXAMPLE

PS >$http = @"
  GET / HTTP/1.1
  Host:bing.com
  `n`n
"@

$http | Send-TcpRequest bing.com 80

#>

param(
    ## The computer to connect to ENTER IP ADDRESS TO SEND SIGNALS TO
    [string] $ComputerName = "000.000.000.000",

    ## A switch to determine if you just want to test the connection IGNORE FIELD
    [switch] $Test,

    ## The port to use ENTER THE PORT TO CONNECT TO
    [int] $Port = 0000,

    ## A switch to determine if the connection should be made using SSL
    [switch] $UseSSL,

    ## The input string to send to the remote host
    [string] $InputObject,

    ## The delay, in milliseconds, to wait between commands
    [int] $Delay = 100
)

Set-StrictMode -Version Latest

[string] $SCRIPT:output = ""

## Store the input into an array that we can scan over. If there was no input,
## then we will be in interactive mode.
$currentInput = $inputObject
if(-not $currentInput)
{
    $currentInput = @($input)
}
$scriptedMode = ([bool] $currentInput) -or $test

function Main
{
    ## Open the socket, and connect to the computer on the specified port
    if(-not $scriptedMode)
    {
        write-host "Connecting to $computerName on port $port"
    }

    try
    {
        $socket = New-Object Net.Sockets.TcpClient($computerName, $port)
    }
    catch
    {
```

```
{
   if($test) { $false }
   else { Write-Error "Could not connect to remote computer: $_" }

   return
}

## If we're just testing the connection, we've made the connection
## successfully, so just return $true
if($test) { $true; return }

## If this is interactive mode, supply the prompt
if(-not $scriptedMode)
{
   write-host "Connected.  Press ^D followed by [ENTER] to exit.`n"
}

$stream = $socket.GetStream()

## If we wanted to use SSL, set up that portion of the connection
if($UseSSL)
{
   $sslStream = New-Object System.Net.Security.SslStream $stream,$false
   $sslStream.AuthenticateAsClient($computerName)
   $stream = $sslStream
}

$writer = new-object System.IO.StreamWriter $stream

while($true)
{
   ## Receive the output that has buffered so far
   $SCRIPT:output += GetOutput

   ## If we're in scripted mode, send the commands,
   ## receive the output, and exit.
   if($scriptedMode)
   {
      foreach($line in $currentInput)
      {
         $writer.WriteLine($line)
         $writer.Flush()
         Start-Sleep -m $Delay
         $SCRIPT:output += GetOutput
      }

      break
   }
   ## If we're in interactive mode, write the buffered
   ## output, and respond to input.
   else
   {
      if($output)
      {
         foreach($line in $output.Split("`n"))
         {
            write-host $line
         }
         $SCRIPT:output = ""
      }
```

```
        ## Read the user's command, quitting if they hit ^D
        $command = read-host
        if($command -eq ([char] 4)) { break; }

        ## Otherwise, Write their command to the remote host
        $writer.WriteLine($command)
        $writer.Flush()
      }
   }

   ## Close the streams
   $writer.Close()
   $stream.Close()

   ## If we're in scripted mode, return the output
   if($scriptedMode)
   {
      $output
   }
}

## Read output from a remote host
function GetOutput
{
   ## Create a buffer to receive the response
   $buffer = new-object System.Byte[] 1024
   $encoding = new-object System.Text.AsciiEncoding

   $outputBuffer = ""
   $foundMore = $false

   ## Read all the data available from the stream, writing it to the
   ## output buffer when done.
   do
   {
      ## Allow data to buffer for a bit
      start-sleep -m 1000

      ## Read what data is available
      $foundmore = $false
      $stream.ReadTimeout = 1000

      do
      {
         try
         {
            $read = $stream.Read($buffer, 0, 1024)

            if($read -gt 0)
            {
               $foundmore = $true
               $outputBuffer += ($encoding.GetString($buffer, 0, $read))
            }
         } catch { $foundMore = $false; $read = 0 }
      } while($read -gt 0)
   } while($foundmore)

   $outputBuffer
}
```

```
. Main
```