

Christmas Cheer Laser, part 8, the hard way

This is an appendix to Christmas Cheer Laser, part 8. If you want to practice burrowing through messy XML data without assistance from Import-Clixml, this is for you.

Answers to the Previous Question

- 9) What laser parameters do you recover from the XML document?

The last hint told us to look for a .xml file in the /etc directory.

```
PS /home/elf> gc /shall/see
Get the .xml children of /etc - an event log to be found. Group all .Id's and the last thing w
ill be in the Properties of the lonely unique event Id.
PS /home/elf>
```

We've already done something like that, so it shouldn't be hard. It's nice that PowerShell will do recursive searches with a wild card in the middle, like /etc/*.xml.

```
dir /etc/*.xml -Recurse -ErrorAction SilentlyContinue
```

```
PS /home/elf> dir /etc/*.xml -Recurse -ErrorAction SilentlyContinue

Directory: /etc/systemd/system/timers.target.wants

Mode                LastWriteTime         Length Name
----                -
--r---             11/18/19  7:53 PM         10006962 EventLog.xml

PS /home/elf>
```

Notice that the EventLog.xml file is over 10 MB. It will take a long time to scroll through this file looking for the correct id. In fact, if we look at the first few lines of the file, we see that each event is very long.

```
gc /etc/systemd/system/timers.target.wants/EventLog.xml | select -
first 20
```

```
PS /home/elf> gc /etc/systemd/system/timers.target.wants/EventLog.xml | select -first 20
<Objs Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Diagnostics.Eventing.Reader.EventLogRecord</T>
      <T>System.Diagnostics.Eventing.Reader.EventRecord</T>
      <T>System.Object</T>
    </TN>
    <ToString>System.Diagnostics.Eventing.Reader.EventLogRecord</ToString>
    <Props>
      <I32 N="Id">3</I32>
      <By N="Version">5</By>
      <Nil N="Qualifiers" />
      <By N="Level">4</By>
      <I32 N="Task">3</I32>
      <I16 N="Opcode">0</I16>
      <I64 N="Keywords">-9223372036854775808</I64>
      <I64 N="RecordId">2194</I64>
      <S N="ProviderName">Microsoft-Windows-Sysmon</S>
      <G N="ProviderId">5770385f-c22a-43e0-bf4c-06f5698ffbd9</G>
      <S N="LogName">Microsoft-Windows-Sysmon/Operational</S>
    </Props>
  </Obj>
</Objs>
```

It turns out that each event is about 210 lines long. A copy of one event is in the text file in the file LaserXML1event.txt. We are lucky though, as the Id we need is in the screenshot above. It is `<I32 N="Id">` The value of the `N="Id"` node in the screenshot is 3. (I believe the I32 stands for 32 bit integer.)

According to the hint, we need to group all the Id values. There will be one value that only occurs once, and it should contain the information we are seeking.

For simpler ways to solve this problem, see "Christmas Cheer Laser part 8.pdf". This way shows the ugly details of burrowing through the XML manually.

PowerShell XML Data Type

There is a lot of XML data, so most languages have methods to handle it. We can load the contents of EventLog.xml file into a variable (PowerShell variable names start with \$) and then change its data type to XML. Many programming languages call this type casting, or casting. `Get-Content` is like `cat` in Linux and `type` in Windows, so it has the aliases `gc`, `type`, and `cat`. (This terminal doesn't have `cat`.)

```
[xml]$xdoc = gc /etc/systemd/system/timers.target/wants/EventLog.xml
```

If you pipe `$xdoc` into `Get-Member`, you will see that it is a `System.Xml.XmlDocument` and it has many methods for dealing with XML data.

Let's explore. The variable itself will tell us that it is a collection of objects called `Objs`.

```
PS /home/elf> $xdoc
Objs
----
Objs
PS /home/elf> █
```

```
PS /home/elf> $xdoc.Objs
Version xmlns                               Obj
-----
1.1.0.1 http://schemas.microsoft.com/powershell/2004/04 {Obj, Obj, Obj, Obj...}
```

Ok. The objects inside of the `Objs` collection are called `Obj`.

We are dealing with a 10MB file, we should limit our output.

```
PS /home/elf> $xdoc.Obj.Obj | select -First 2

RefId      : 0
TN         : TN
ToString   : static string XmlNode(psobject instance)
Props      : Props
MS         : MS

RefId      : 24
TNRef      : TNRef
ToString   : static string XmlNode(psobject instance)
Props      : Props
MS         : MS
```

We can see that underneath Obj, there are properties (nodes in XML) RefId, TN, ToString, Props, and MS. If you look back at the screenshot with the first lines of the file, you will see that our node, <I32 N="Id">3</I32>, lives under the Props node. Use Get-Member to examine the properties of the I32 node.

```
PS /home/elf> $xdoc.Obj.Props.I32 | gm

        TypeName: System.Xml.XmlElement#http://schemas.microsoft.com/powershell/2004/04#I32
Name      MemberType      Definition
-----
ToString  CodeMethod      static string XmlNode(psobject instance)
AppendChild Method          System.Xml.XmlNode AppendChild(System.Xml.XmlNode...
Clone     Method          System.Xml.XmlNode Clone(), System.Object IClonea...
CloneNode Method          System.Xml.XmlNode CloneNode(bool deep)
CreateNavigator Method        System.Xml.XPath.XPathNavigator CreateNavigator()...
Equals    Method          bool Equals(System.Object obj)
GetAttribute Method        string GetAttribute(string name), string GetAttri...
<snip>
WriteContentTo Method      void WriteContentTo(System.Xml.XmlWriter w)
WriteTo    Method      void WriteTo(System.Xml.XmlWriter w)
Item       ParameterizedProperty System.Xml.XmlElement Item(string name) {get;}, S...
#text     Property    string #text {get;set;}
N         Property    string N {get;set;}
```

The I32 node has properties Item, #text, and N. Now look at the contents if I32 with \$xdoc.Obj.Props.I32 | select -first 10

```
PS /home/elf> $xdoc.Obj.Props.I32 | select -first 10

N      #text
-----
Id     3
Task   3
ProcessId 1960
ThreadId 6648
Id     5
Task   5
ProcessId 1960
ThreadId 6640
Id     5
Task   5

PS /home/elf>
```

It looks like each event has 4 I32 nodes, named Id, Task, ProcessID, and ThreadID (they start to repeat after that, as we're into the next object.) This adds a little difficulty. The N= part is inside the tag for the node, so it is an attribute, not another node. We can separate the node with the correct attribute (Id) using the Where-Object cmdlet.

The node we want is in the N column (property) and has the value Id. The #text column gives us the contents between the tags in <I32 N="Id">3</I32>, 3 in our case.

```
$xdoc.Objc.Obj.Props.I32 | where {$_.N -eq "Id"} | select -first 10
PS /home/elf> $xdoc.Objc.Obj.Props.I32 | where {$_.N -eq "Id"} | select -first 10

N #text
- ----
Id 3
Id 5

PS /home/elf>
```

Ah, that is what we need. The command has grabbed all the nodes for <I32 N="Id">. PowerShell has a cmdlet, Group-Object (alias group) that that will put things into groups. For example, it can put all the Id 5 items in the screenshot above into one group and count them (same for the ones, twos, threes, etc.)

Notice the numbers are in the column called "#text", above. We can group by that.

```
$xdoc.Objc.Obj.Props.I32 | where {$_.N -eq "Id"} | group '#text'
PS /home/elf> $xdoc.Objc.Obj.Props.I32 | where {$_.N -eq "Id"} | group '#text'

Count Name Group
-----
1 1 {I32}
39 2 {I32, I32, I32, I32...}
179 3 {I32, I32, I32, I32...}
2 4 {I32, I32}
905 5 {I32, I32, I32, I32...}
98 6 {I32, I32, I32, I32...}

PS /home/elf>
```

The riddle told us to find the Id that only happens once, and it is <I32 N="Id">1</I32>. Now, what is in the node that has N="Id" with a value of 1? Note that we have moved up one level for the next command. We are looking at \$xdoc.Objc.Obj.Props and not \$xdoc.Objc.Obj.Props.I32. This is so we will have the properties we need in the pipeline, and not just the I32 stuff. Since we are doing that, we have to use \$_.I32.N in our Where-Object statement.

Also, lets add a check so the only object we keep is the one with the Id value equal to 1.

```
$_.I32."#text" -eq 1
```

```
$xdoc.Objv.Obj.Props | where {$_.I32.N -eq "Id" -and $_.I32."#text" -eq 1}
```

```
PS /home/elf> $xdoc.Objv.Obj.Props | where {$_.I32.N -eq "Id" -and $_.I32."#text" -eq 1}
I32 : {I32, I32, I32, I32}
By  : {By, By}
Nil : {Nil, Nil, Nil}
I16 : I16
I64 : {I64, I64}
S   : {S, S, S, S...}
G   : G
Obj : {Obj, Obj, Obj, Obj...}
DT  : DT

PS /home/elf>
```

We need the contents, not the node names so pipe into `Format-List -Property *`, or `fl *` to grab the values of all properties.

```
PS /home/elf> $xdoc.Objv.Obj.Props | where {$_.I32.N -eq "Id" -and $_.I32."#text" -eq 1} | fl *
I32           : {I32, I32, I32, I32}
By            : {By, By}
Nil           : {Nil, Nil, Nil}
I16           : I16
I64           : {I64, I64}
S             : {S, S, S, S...}
G            : G
Obj           : {Obj, Obj, Obj, Obj...}
DT            : DT
Name          : Props
LocalName     : Props
NamespaceURI  : http://schemas.microsoft.com/powershell/2004/04
Prefix        :
NodeType      : Element
ParentNode    : Obj
OwnerDocument : #document
IsEmpty       : False
Attributes    : {}
HasAttributes : False
SchemaInfo    : System.Xml.XmlName
InnerXml      : <I32 N="Id"
```

<snip>

```
N="Value">PowerShell.EXE</S></Props></Obj><Obj RefId="18016"><TNRef
RefId="1806" /><ToString>System.Diagnostics.Eventing.Reader.EventProperty</
ToString><Props><S
N="Value">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -c
"$correct_gases_postbody = @{'n O=6`n H=7`n He=3`n N=4`n
Ne=22`n Ar=11`n Xe=10`n F=20`n Kr=8`n
Pn=9`n`n`n"</S></Props></Obj><Obj RefId="18017"><TNRef RefId="1806" /><ToStr
ing>System.Diagnostics.Eventing.Reader.EventProperty</ToString><Props><S
N="Value">C:\</S></Props></Obj><Obj RefId="18018"><TNRef RefId="1806" /><ToStr
ing>System.Diagnostics.Eventing.Reader.EventProperty</ToString><Props><S
N="Value">ELFURESEARCH\allservices</S></Props></Obj><Obj
```

Ugh. Its two pages of text, but we can find the answer.

\$correct_gases_postbody = @{\n O=6\n H=7\n He=3\n N=4\n Ne=22\n Ar=11\n Xe=10\n F=20\n Kr=8\n Rn=9\n}

From here you can go back to the original document, "Christmas Cheer Laser part 8.pdf".