Iron Scripter 2018: Prequel 2

Before we get into my commentary and view of the solution I need to emphasise that these comments are mine – *they are not an attempt to provide an "official" answer or define how the factions should approach Iron Scripter*. My hope is that the commentaries will provoke more thought and debate to help define the factions.

The puzzle

This was the information provided for the second of the Iron Scripter 2018 Prequel puzzles:

Greetings Iron Scripters. You've overcome the first challenge on your path to Iron Scripter 2018. Victory will come to the faction that has mastered all aspects of the art of the scripter and your next challenge will move you closer to being a master.

One aspect of mastery of the art is embracing the concept of objects – the subject of this trial.

The oldest parts of the archives deal with the early days of PowerShell when other scripting languages were in use and the factions were new. The following fragment of code was discovered:

```
$objOperatingSystem = Get-WmiObject -Query "SELECT * FROM Win32_OperatingSystem"


 Write-Host "OS Name: "  $objOperatingSystem.Name
 Write-Host "Version: "  $objOperatingSystem.Version
 Write-Host "Service Pack: "  $objOperatingSystem.ServicePackMajorVersion "."
$objOperatingSystem.ServicePackMinorVersion
 Write-Host "OS Manufacturer: "  $objOperatingSystem.Manufacturer
 Write-Host "Windows Directory: "  $objOperatingSystem.WindowsDirectory
 Write-Host "Locale: "  $objOperatingSystem.Locale
 Write-Host "Available Physical Memory: "  $objOperatingSystem.FreePhysicalMemory
 Write-Host "Total Virtual Memory: "  $objOperatingSystem.TotalVirtualMemorySize
 Write-Host "Available Virtual Memory: " $objOperatingSystem.FreeVirtualMemory

 $disks = Get-WmiObject -Query "SELECT * FROM Win32_LogicalDisk"
 foreach ($objDisk in $disks) {
   Write-Host  "Drive: " $objDisk.DeviceID
   Write-Host  "DriveType: " $objDisk.Description
   Write-Host  "Size:" $objDisk.Size
   Write-Host  "FreeSpace: " $objDisk.FreeSpace
   Write-Host  "Compressed:" $objDisk.Compressed
 }
```

This code doesn't fit with the view point of any existing faction – whether there ever was a faction that supported this style is unknown.

Your goal is to take this code. Modify it to output objects rather than text. The solution must be acceptable to your faction:

- Daybreak Faction - beautiful code
- Flawless Faction - flawless code
- Battle Faction - good enough to get the job done

Remember the following when creating your solution:

- Following your faction's aims is the most important aspect of this challenge
- Use best practice if it doesn't conflict with your faction's aims
- Output a single type of named object – you can assign the name yourself
- Calculate the percentage used space on each disk and add it to the output object
- Create and use a format file or type file as appropriate to control the display of the object you'll output
- Ensure the code works with remote machines?
- PowerShell v5.1 is the assumed standard for your code. If you can also make the solution work with PowerShell v6, on Windows, that is a bonus

Good luck and good coding.

The commentary

If you've been around the PowerShell community for any length of time you'll recognise what's happening with the code. The author has probably converted some VBScript code directly into PowerShell without changing the code to output objects, or possibly not even being aware of the desirability of outputting objects.

In blunt terms of getting the job done the code works – it fetches the desired information and presents it on screen.

```
OS Name:  Microsoft Windows 10 Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2
Version:  10.0.16299
Service Pack:  0 . 0
OS Manufacturer:  Microsoft Corporation
Windows Directory:  C:\WINDOWS
Locale:  0809
Available Physical Memory:  4549792
Total Virtual Memory:  19197460
Available Virtual Memory:  5238252
Drive:  C:
DriveType:  Local Fixed Disk
Size: 511210610688
FreeSpace:  138813419520
Compressed: False
Drive:  D:
DriveType:  CD-ROM Disc
Size:
FreeSpace:
Compressed:
Drive:  E:
DriveType:  Removable Disk
Size: 2096857088
FreeSpace:  2096365568
Compressed: False
```

As you can see its not particularly well formatted (anyone who suggests using formatted strings and the -f operator at this point should hand in their faction T-shirt) and the sizes are in bytes which are difficult to compare. It's also outputting text rather than objects.

Objects are easy to get from this code:

```
Get-WmiObject -Query "SELECT * FROM Win32_OperatingSystem" |
select Name, Version, ServicePackMajorVersion, ServicePackMinorVersion,
Manufacturer, WindowsDirectory, Locale, FreePhysicalMemory, TotalVirtualMemorySize,
FreeVirtualMemory
```

```
Get-WmiObject -Query "SELECT * FROM Win32_LogicalDisk" |
select DeviceID, Description, Size, FreeSpace, Compressed
```

```
Name                    : Microsoft Windows 10
Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2
Version                 : 10.0.16299
ServicePackMajorVersion : 0
ServicePackMinorVersion : 0
Manufacturer            : Microsoft Corporation
WindowsDirectory        : C:\WINDOWS
Locale                  : 0809
FreePhysicalMemory      : 2372952
TotalVirtualMemorySize  : 20187268
FreeVirtualMemory       : 3356476

DeviceID    : C:
Description : Local Fixed Disk
Size        : 511210610688
FreeSpace   : 134462988288
Compressed  : False

DeviceID    : D:
Description : CD-ROM Disc
Size        :
FreeSpace   :
Compressed  :

DeviceID    : E:
Description : Removable Disk
Size        : 2096857088
FreeSpace   : 2096365568
Compressed  : False
```

BUT we're outputting two different types of object. What we were asked for was a single object.

As a quick aside you'll notice that I'm still using -Query for fetching the WMI data. You can switch to using -Class if you want – it's a matter of personal preference. My research over the years indicates that using -Query is slightly faster because if you use -Class and -Filter a query is generated under the covers. Writing WQL queries can be more verbose and a bit trickier.

I'm also going to switch to using the CIM cmdlets.

```
$os = Get-CimInstance -Query 'SELECT * FROM Win32_OperatingSystem'

Get-CimInstance -Query 'SELECT * FROM Win32_LogicalDisk' |
foreach {

  $props = [ordered]@{
     OSName = $os.Name
     Version = $os.Version
     ServicePack = "$($os.ServicePackMajorVersion).$($os.ServicePackMinorVersion)"
     Manufacturer = $os.Manufacturer
     WindowsDirectory = $os.WindowsDirectory
     Locale = $os.Locale
     FreePhysicalMemory = $os.FreePhysicalMemory
     TotalVirtualMemorySize = $os.TotalVirtualMemorySize
     FreeVirtualMemory = $os.FreeVirtualMemory
     Drive = $PSItem.DeviceID
     DriveType = $PSItem.Description
     Size = $PSItem.Size
     FreeSpace = $PSItem.FreeSpace
     Compressed = $PSItem.Compressed
  }

  New-Object -TypeName PSObject -Property $props

}
```

I've also used single quotes round the WMI queries. This will give output like this for each disk:

```
OSName                : Microsoft Windows 10
                        Enterprise|C:\WINDOWS|\Device\Harddisk0\Partition2
Version               : 10.0.16299
ServicePack           : 0.0
Manufacturer          : Microsoft Corporation
WindowsDirectory      : C:\WINDOWS
Locale                : 0809
FreePhysicalMemory    : 2075744
```

```
TotalVirtualMemorySize : 20187268
FreeVirtualMemory      : 3042124
Drive                  : C:
DriveType              : Local Fixed Disk
Size                   : 511210610688
FreeSpace              : 134061481984
Compressed             : False
```

The OS information is repeated each time but you can always filter that out if you just want the disk information. The OS name is a bit odd looking. That's because it's formed from a number of fields:

- Caption
- Windows Directory
- A disk partition – the Recovery Partition on my Windows 10 machine but a non-existent partition on my Windows Server 2016 machines.

As the name field contains the Windows Directory which we're already showing and cryptic partition information its better to use the Caption property for the operating system name. I've turned the script into a function so that it becomes easily reusable:

```
function Get-SystemInfo {
$os = Get-CimInstance -Query 'SELECT * FROM Win32_OperatingSystem'

Get-CimInstance -Query 'SELECT * FROM Win32_LogicalDisk' |
foreach {

  $props = [ordered]@{
     OSName = $os.Caption
     Version = $os.Version
     ServicePack = "$($os.ServicePackMajorVersion).$($os.ServicePackMinorVersion)"
     Manufacturer = $os.Manufacturer
     WindowsDirectory = $os.WindowsDirectory
     Locale = $os.Locale
     FreePhysicalMemory = $os.FreePhysicalMemory
     TotalVirtualMemorySize = $os.TotalVirtualMemorySize
     FreeVirtualMemory = $os.FreeVirtualMemory
     Drive = $PSItem.DeviceID
     DriveType = $PSItem.Description
     Size = $PSItem.Size
     FreeSpace = $PSItem.FreeSpace
     Compressed = $PSItem.Compressed
  }

  New-Object -TypeName PSObject -Property $props

}
}
```

This still has the problem regarding the type of the object:

```
PS> Get-SystemInfo | Get-Member


   TypeName: System.Management.Automation.PSCustomObject

Name                   MemberType   Definition
----                   ----------   ----------
Equals                 Method       bool Equals(System.Object obj)
GetHashCode            Method       int GetHashCode()
GetType                Method       type GetType()
ToString               Method       string ToString()
Compressed             NoteProperty bool Compressed=False
Drive                  NoteProperty string Drive=C:
DriveType              NoteProperty string DriveType=Local Fixed Disk
FreePhysicalMemory     NoteProperty uint64 FreePhysicalMemory=2529036
FreeSpace              NoteProperty uint64 FreeSpace=135229014016
FreeVirtualMemory      NoteProperty uint64 FreeVirtualMemory=3396916
Locale                 NoteProperty string Locale=0809
Manufacturer           NoteProperty string Manufacturer=Microsoft Corporation
OSName                 NoteProperty string OSName=Microsoft Windows 10 Enterprise
ServicePack            NoteProperty string ServicePack=0.0
Size                   NoteProperty uint64 Size=511210610688
TotalVirtualMemorySize NoteProperty uint64 TotalVirtualMemorySize=20384276
Version                NoteProperty string Version=10.0.16299
WindowsDirectory       NoteProperty string WindowsDirectory=C:\WINDOWS
```

The task asked for a named type. Now, technically we have a named type in System.Management.Automation.PSCustomObject but what it means is that you give the object you're creating a specific type name so that you can create the format or type data file.

We need to add the PSTypeName property

```
function Get-SystemInfo {
$os = Get-CimInstance -Query 'SELECT * FROM Win32_OperatingSystem'

Get-CimInstance -Query 'SELECT * FROM Win32_LogicalDisk' |
foreach {

  $props = [ordered]@{
     OSName = $os.Caption
     Version = $os.Version
     ServicePack = "$($os.ServicePackMajorVersion).$($os.ServicePackMinorVersion)"
     Manufacturer = $os.Manufacturer
     WindowsDirectory = $os.WindowsDirectory
     Locale = $os.Locale
     FreePhysicalMemory = $os.FreePhysicalMemory
     TotalVirtualMemorySize = $os.TotalVirtualMemorySize
     FreeVirtualMemory = $os.FreeVirtualMemory
     Drive = $PSItem.DeviceID
     DriveType = $PSItem.Description
     Size = $PSItem.Size
     FreeSpace = $PSItem.FreeSpace
     Compressed = $PSItem.Compressed
     PSTypeName = 'SystemInfo'
  }

  New-Object -TypeName PSObject -Property $props

}
}
```

The type name is now set

```
PS> Get-SystemInfo | Get-Member


   TypeName: SystemInfo

Name                   MemberType   Definition
----                   ----------   ----------
Equals                 Method       bool Equals(System.Object obj)
GetHashCode            Method       int GetHashCode()
GetType                Method       type GetType()
ToString               Method       string ToString()
Compressed             NoteProperty bool Compressed=False
Drive                  NoteProperty string Drive=C:
DriveType              NoteProperty string DriveType=Local Fixed Disk
FreePhysicalMemory     NoteProperty uint64 FreePhysicalMemory=2652400
FreeSpace              NoteProperty uint64 FreeSpace=135261724672
FreeVirtualMemory      NoteProperty uint64 FreeVirtualMemory=3415652
Locale                 NoteProperty string Locale=0809
Manufacturer           NoteProperty string Manufacturer=Microsoft Corporation
OSName                 NoteProperty string OSName=Microsoft Windows 10 Enterprise
ServicePack            NoteProperty string ServicePack=0.0
Size                   NoteProperty uint64 Size=511210610688
TotalVirtualMemorySize NoteProperty uint64 TotalVirtualMemorySize=20384276
Version                NoteProperty string Version=10.0.16299
WindowsDirectory       NoteProperty string WindowsDirectory=C:\WINDOWS
```

An interesting option here would be to create a class instead of using New-Object. Could the class populate the disk information on creation? How would remoting be managed? Some interesting thoughts for you to explore.

One thing we haven't done yet is calculate the free space as a percentage. That means adding another property. If you have a CD or DVD drive you won't have any free space and will get a divide by zero error so make the free space percentage calculation conditional.

You weren't asked to do this but I always think its neater to present memory information and disk space information in understandable units. Just for fun WMI returns the disk sizes are bytes and the memory sizes in kilobytes! Yay for consistency.

As a last change I've added a computername parameter that defaults to the local machine. This supplies the remoting capability.

```
function Get-SystemInfo {
param (
  [string]$computername = $env:COMPUTERNAME
)
$os = Get-CimInstance -Query 'SELECT * FROM Win32_OperatingSystem' -ComputerName $computername

Get-CimInstance -Query 'SELECT * FROM Win32_LogicalDisk' -ComputerName $computername |
foreach {

  $props = [ordered]@{
     OSName = $os.Caption
     Version = $os.Version
     ServicePack = "$($os.ServicePackMajorVersion).$($os.ServicePackMinorVersion)"
     Manufacturer = $os.Manufacturer
     WindowsDirectory = $os.WindowsDirectory
     Locale = $os.Locale
     'FreePhysicalMemory GB' = [math]::Round(($os.FreePhysicalMemory / 1MB), 2)
     'TotalVirtualMemorySize GB' = [math]::Round(($os.TotalVirtualMemorySize / 1MB), 2)
     'FreeVirtualMemory GB' = [math]::Round(($os.FreeVirtualMemory / 1MB), 2)
     Drive = $PSItem.DeviceID
     DriveType = $PSItem.Description
     'Size GB' = [math]::Round(($PSItem.Size / 1GB), 2)
     'FreeSpace GB' = [math]::Round(($PSItem.FreeSpace / 1GB), 2)
     'FreeSpace %' = if ($PSItem.FreeSpace){
                     [math]::Round((($PSItem.FreeSpace / $PSItem.Size ) * 100), 2)}
                     else {0}
     Compressed = $PSItem.Compressed
     ComputerName = $computername
     PSTypeName = 'SystemInfo'
  }

  New-Object -TypeName PSObject -Property $props

}
}
```

The output from this looks like this:

```
OSName                    : Microsoft Windows 10 Enterprise
Version                   : 10.0.16299
ServicePack               : 0.0
Manufacturer              : Microsoft Corporation
WindowsDirectory          : C:\WINDOWS
Locale                    : 0809
FreePhysicalMemory GB     : 11.94
TotalVirtualMemorySize GB : 18.31
FreeVirtualMemory GB      : 14.01
Drive                     : C:
DriveType                 : Local Fixed Disk
Size GB                   : 476.1
FreeSpace GB              : 125.29
FreeSpace %               : 26.32
Compressed                : False
ComputerName              : W510W10
```

The last part of the coding challenge was to create a format or type file to control the formatting. You normally do this to control the display – think of the default display from Get-Process compared to all of the possible properties you could display. I'll be creating a format file to manage the display. A type file is used to extend, or modify, a given type – you could use a type file to perform some of the calculations in the script but I prefer to do that up front so I have the raw object in a form I can easily use.

First you need to create a format file

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <ViewDefinitions>
    <View>
      <Name>systeminfo</Name>
      <ViewSelectedBy>
        <TypeName>SystemInfo</TypeName>
      </ViewSelectedBy>
      <TableControl>
        <TableHeaders>
          <TableColumnHeader>
```

```
            <Label>Computer Name</Label>
            <Width>10</Width>
            <Alignment>Left</Alignment>
          </TableColumnHeader>
          <TableColumnHeader>
            <Label>Free Physical RAM (GB)</Label>
            <Width>22</Width>
            <Alignment>Right</Alignment>
          </TableColumnHeader>
          <TableColumnHeader>
            <Label>Drive</Label>
            <Width>5</Width>
            <Alignment>Left</Alignment>
          </TableColumnHeader>
          <TableColumnHeader>
            <Label>Size (GB)</Label>
            <Width>10</Width>
            <Alignment>Right</Alignment>
          </TableColumnHeader>
          <TableColumnHeader>
            <Label>Percent Free Space</Label>
            <Width>20</Width>
            <Alignment>Right</Alignment>
          </TableColumnHeader>
        </TableHeaders>
        <TableRowEntries>
          <TableRowEntry>
            <TableColumnItems>
              <TableColumnItem>
                <PropertyName>ComputerName</PropertyName>
              </TableColumnItem>
              <TableColumnItem>
                <PropertyName>FreePhysicalMemory GB</PropertyName>
              </TableColumnItem>
              <TableColumnItem>
                <PropertyName>Drive</PropertyName>
              </TableColumnItem>
              <TableColumnItem>
                <PropertyName>Size GB</PropertyName>
              </TableColumnItem>
              <TableColumnItem>
                <PropertyName>FreeSpace %</PropertyName>
              </TableColumnItem>
            </TableColumnItems>
          </TableRowEntry>
        </TableRowEntries>
      </TableControl>
    </View>
  </ViewDefinitions>
</Configuration>
```

I find the easiest way is to copy existing format data using Export-Formatdata, pretty print the XML and then modify. I've only created a Table view, you can add List and Custom if you like a challenge. The properties I've chosen are arbitrary and you may prefer to use other properties or expand the number of properties used.

The code then becomes:

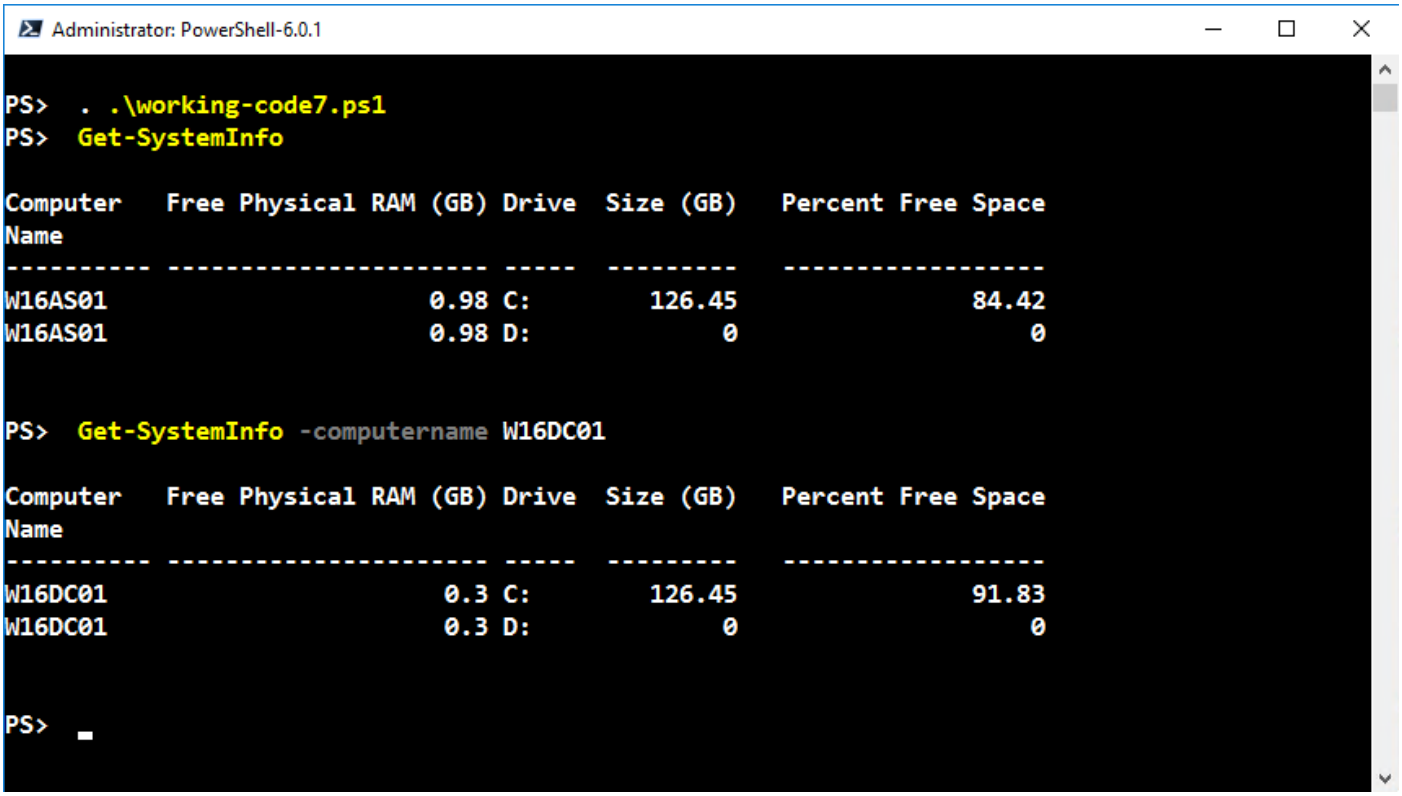
```
Update-FormatData -PrependPath 'C:\MyData\2018 Summit\Iron Scripter
prequels\Puzzle02\SystemInfo.Format.ps1xml'

function Get-SystemInfo {
param (
  [string]$computername = $env:COMPUTERNAME
)
$os = Get-CimInstance -Query 'SELECT * FROM Win32_OperatingSystem' -ComputerName $computername

Get-CimInstance -Query 'SELECT * FROM Win32_LogicalDisk' -ComputerName $computername |
foreach {

  $props = [ordered]@{
     OSName = $os.Caption
     Version = $os.Version
     ServicePack = "$($os.ServicePackMajorVersion).$($os.ServicePackMinorVersion)"
     Manufacturer = $os.Manufacturer
     WindowsDirectory = $os.WindowsDirectory
     Locale = $os.Locale
     'FreePhysicalMemory GB' = [math]::Round(($os.FreePhysicalMemory / 1MB), 2)
     'TotalVirtualMemorySize GB' = [math]::Round(($os.TotalVirtualMemorySize / 1MB), 2)
     'FreeVirtualMemory GB' = [math]::Round(($os.FreeVirtualMemory / 1MB), 2)
     Drive = $PSItem.DeviceID
     DriveType = $PSItem.Description
     'Size GB' = [math]::Round(($PSItem.Size / 1GB), 2)
     'FreeSpace GB' = [math]::Round(($PSItem.FreeSpace / 1GB), 2)
```

```
        'FreeSpace %' = if ($PSItem.FreeSpace){
                        [math]::Round((($PSItem.FreeSpace / $PSItem.Size ) * 100), 2)}
                        else {0}
        Compressed = $PSItem.Compressed
        ComputerName = $computername
        PSTypeName = 'SystemInfo'
    }

    New-Object -TypeName PSObject -Property $props

}
}
```

Note that I'm prepending the format data. This means that its available before the standard formatting data and can be easily changed and updated while I'm developing. Its output looks like this:

```
PS> Get-SystemInfo

Computer   Free Physical RAM (GB) Drive  Size (GB)   Percent Free Space
Name
---------- --------------------- -----  ---------   ------------------
W510W10                     8.15 C:         476.1                26.02
W510W10                     8.15 D:             0                    0
W510W10                     8.15 E:          1.95                99.98
```

The code works on PowerShell v6 and works against remote machines



Job done.

That solves the puzzle and fulfils the requirements but what about the faction issues. Individual faction members have contributed examples on the forum which you should view so instead of giving my view of the code I'll briefly outline what I'd expect the faction versions to look like.

Battle faction will probably stick with what we have. It meets the requirements and gets the job done. If things change the code can be easily modified. Time to move on to another problem.

Daybreak faction will want to format the code so that it looks good for instance lining up the = signs in the property assignments. They may want to replace the WQL queries with -ClassName as it fits with the faction outlook better. The calculations may be moved into the format file to make the code neater and more elegant. A class may be substituted for the PSObject.

Flawless faction will want to add all the bells and whistles – help file, parameter validation, Write-Debug and Write-Verbose commands as appropriate, try-catch blocks as necessary and anything else that ensures the code executes flawlessly.

Enjoy!

Puzzle 3 will be available by the time you're reading this.