Powershell, commandes à distance et dé/sérialisation d’objets

Lors d’accès à distance à des logs Windows (commande Get-WinEvent), je me suis retrouvé confronté à un type d’objets que je n’attendais pas, et à des informations manquantes :

```
TypeName : Deserialized.System.Diagnostics.Eventing.Reader.EventLogRecord
```

Voici la description du membre « Properties » :

```
Properties Property Deserialized.System.Collections.Generic.List`1[[System.Diagnosti
cs.Eventing.Reader.EventProperty, System.Core, Version=4.0.0.0, Culture=neutral, Pub
licKeyToken=b77a5c561934e089]] {get;set;}
```

Problème : voici le contenu du membre « Properties » sur un objet EventLogRecord désérialisé :

```
System.Diagnostics.Eventing.Reader.EventProperty
System.Diagnostics.Eventing.Reader.EventProperty
System.Diagnostics.Eventing.Reader.EventProperty
System.Diagnostics.Eventing.Reader.EventProperty
System.Diagnostics.Eventing.Reader.EventProperty
```

Alors que voici le contenu d’un objet local à ma machine :

```
PS > $e3.Properties

Value
-----
McAfee Endpoint Security
SECURITY_PRODUCT_STATE_ON
```

Voici les membres d'un objet Properties :

```
PS > $e3.Properties | Get-Member

   TypeName : System.Diagnostics.Eventing.Reader.EventProperty

Name        MemberType Definition
----        ---------- ----------
Equals      Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
ToString    Method     string ToString()
Value       Property   System.Object Value {get;}
```

Je soupçonne le mécanisme de sérialisation/dé-sérialisation d'utiliser la méthode ToString(). En effet, sur mon objet local, ToString() d'une propriété donne bien la même chose qu'un objet distant :

```
PS > $e3.Properties[0].ToString()
System.Diagnostics.Eventing.Reader.EventProperty
```

Si comme moi vous n'avez pas besoin du type d'origine mais que vous voulez avoir toutes les informations sans pertes, vous pouvez passer par une conversion en JSon dans le job distant, et utiliser ConvertFrom-Json pour récupérer des PSObjects à utiliser plus tard.

Petite subtilité : en fonction de la profondeur de votre objet, il sera peut-être nécessaire d'ajouter l'argument « depth » avec une valeur numérique (100 maximum, voir l'extrait de code plus bas).

Voici le code complet d'une fonction qui me permet de récupérer les évènements à distance (avec un filtre XML d'entrée auquel on ajoute les dates de début et de fin des logs recherchés) :

```
function Get-RemoteEvents {
    Param(
        [Parameter(Mandatory)]
        [string[]]$servers,
        [Parameter(Mandatory)]
        [pscredential]$cred,
        [Parameter(Mandatory)]
```

```
        [string]$filter,
        [System.DateTime]$EventStartDate,
        [System.DateTime]$EventEndDate
    )
    $endDate = if ($EventEndDate) { $EventEndDate } else { Get-Date }
    $jobs = @()
    foreach ($server in $servers) {
        try {
            $argsList = if ($EventStartDate) {
                @($filter, $EventStartDate, $endDate)
            } else {
                $filter
            }
            $jobs += Invoke-Command -computername $server -scriptblock `
            {
                $dateFilter = switch ($args.Count) {
                    3 {
                        $eventStartDate = $args[1].ToUniversalTime().ToString("yyyy'
-'MM'-'dd'T'HH':'mm':'ss'.'fff'Z'")
                        $eventEndDate = $args[2].ToUniversalTime().ToString("yyyy'-'
MM'-'dd'T'HH':'mm':'ss'.'fff'Z'")
                        ' and *[System[TimeCreated[@SystemTime>=''' + $eventStartDat
e + ''' and @SystemTime<=''' + $eventEndDate + '''']]]'
                    }
                    1 {
                        ""
                    }
                    default {
                        throw [System.ArgumentException] "Bad number of arguments to
 retrieve events"
                    }
                }
                if ($dateFilter -eq "") {
                    Get-WinEvent -FilterXml $args[0] | ConvertTo-Json -Depth 100
                } else {
                    $xmlFilter = $args[0]
                    $selectNodes = $xmlFilter.SelectNodes("//Select")
                    foreach ($n in $selectNodes) {
                        $n.InnerText += $dateFilter
                    }
                    Get-WinEvent -FilterXml $xmlFilter.OuterXml | ConvertTo-Json -De
pth 100
                }
            } -AsJob -Credential $cred -ArgumentList $argsList
        } catch {

        }
    }
    $jobsEnded = $false
    while (-not $jobsEnded) {
        $jeCount = 0 # jobs ended count
        foreach ($job in $jobs) {
            if ($job.State -ne "Running") {
                $jeCount += 1
            }
```

```
        }
        if ($jeCount -eq $jobs.Count) {
            $jobsEnded = $true
        }
        Start-Sleep -Seconds 1
    }
    $allEvents = $(Receive-Job -Job $jobs) | ForEach-Object { $_ | ConvertFrom-Json
} | ForEach-Object { $_ }
    $allEvents = $allEvents | Sort-Object -Property TimeCreated -Descending
    return New-Object psobject @{
        events = $allEvents
        count = $allEvents.Count
        start = $EventStartDate
        end = $endDate
        filter = $filter
    }
}
```