Search Health Reports (SRx) – Digging in further with PowerShell

-Brian Pendergrass

After introducing the Search Health Reports (SRx), we continued to extend the battery of PowerShell tests for analyzing and troubleshooting a SharePoint 2013 and SharePoint 2016 on-premises search farm. Because most of these efforts resulted in new or improved tests, we largely suggested and recommended the *RunAllTests* report and *Indexer Disk* report to leverage the SRx in the vast majority of troubleshooting scenarios. However, sometimes you need (or simply want) to dig in a bit further and expose more detail about the Search system.

In this article, I want to share some of the most common building blocks - the lower level custom functionality - used within the tests and help uncover the richness built into the SRx. I encourage you to [download the SRx Core,] initialize the shell by running the .\initSRx.ps1 script, and try out these in your environment..

```
#Options when initializing SRx
 .\initSRx.ps1 -SSA "Name-of-an-SSA"    # > For handling multiple SSAs in a farm
 .\initSRx.ps1 -Verbose                 # > Enables verbose logging in the console
 .\initSRx.ps1 -RebuildSRx              # > Re-initializes the SRx shell, which
                                        #   rebuilds the $SRxEnv and $xSSA objects
                                        #   (e.g. useful after a topology change)
#Run all tests with detailed output:
New-SRxReport -RunAllTests -Details

#Run a specific test (in this case, "OSPingSearchServer")
New-SRxReport -Test OSPingSearchServer
    #Hint: After the -Test parameter, use <tab> key to iterate
    #      through each test names or auto-complete a test name

#New-SRxReport acts as a wrapper for Test-SRx, which invokes a
#PowerShell script with a corresponding name. For example:
Test-SRx -Name OSPingSearchServer
#Invokes: <SRxPath>\lib\tests\core\Test-OSPingSearchServer.ps1
```

```
    # > Test-SRx returns a "standardized" object, which can be
    #   pipelined to other custom functionality
    # > New-SRxReport provides standardized formatting of the
    #   output object from the test

#To run all tests and generate an array of test result objects:
Test-SRx -RunAllTests
#Detailed indexer reports...
Get-SRxIndexReports -DiskReport
#The extended and customized SSA Object ($xSSA)
$xSSA | gm | ? {$_.Name -like "_*"}    #In SRx, custom properties and methods appende
d
                                       #to any out-of-the-box objects are named with
an
                                       #underscore "_" as their prefix to differentia
te
#To get Search Servers details...
 $xSSA._Servers                        # > An array of all search servers discovered
                                       #   during initialization of SRx, where each
                                       #   server is represented as a custom object
$xSSA._GetServer("-Name-of-Server-")   # > Gets a specific server by name
$xSSA._GetServerEx("-Name-of-Server-") # > Similar to above, but returns an extended
                                       #   server object by fetching the applicable
                                       #   registry keys and system properties (from
                                       #   Get-WMI for various classes)


#--------------------------------
#Example of tools available for a specific server...
 $server = $xSSA._GetServer("-Name-of-Server")
 $server.canPing()            # > Wrapper: Test-Connection <severname>
 $server.GetProcesses()       # > Under the covers, runs "Get-Process" with a list of
                              #   applicable Search processes (e.g. noderunner, mssear
ch)
```

```
 $server = $xSSA._GetServerEx("-Name-of-Server")
 $server | gm                #View the other extended properties and methods for this
#Topology Visualization...
 $xSSA._ShowTopologyReport()
#Component fun...
 $xSSA._GetCC()       #Get the list of Crawl Components
 $xSSA._GetCPC()      #Get the list of Content Processing Components
 $xSSA._GetAPC()      #Get the list of Analytics Processing Components
 $xSSA._GetQPC()      #Get the list of Query Processing Components
 $xSSA._GetIndexer()  #Get the list of Index Components

#Assuming you have an component named "CrawlComponent2" or "IndexComponent4", run:
 $xSSA._GetCC(2)      #Gets "CrawlComponent2" from the Active topology
 $xSSA._GetIndexer(4) #Gets "IndexComponent4" from the Active topology

#Each component object also has extended methods and properties...
 $i4 = $xSSA._GetIndexer(4)
 $i4._GetProcess()                # > Wrapper: Get-Process noderunner -computer <nam
e>
 $i4._GetHealthReport()           # > Wrapper: Get-SPEnterpriseSearchStatus -SSA <SS
A>
                                  #              -Component IndexComponent4 -HealthR
eport
 $i4._BuildDiskReportData()       # > Runs underlying processing for an index disk r
eport
 $i4._CellPath                    # > An extended property with the index cell path
 $i4 | gm | ? {$_.Name -like "_*"} # > View the list of properties and methods for th
is
#To get more Content Sources details, which aggregates data from the following:
#  - The out-of-the-box content source object
#  - MSSCrawlHistory
#  - MSSCrawlComponentsState
#  - Web Application info [if in the same farm as the SSA]


 $xSSA._GetContentSource()       # > Wrapper: Get-SPEnterpriseSearchCrawlContentSourc
e
```

```
 $xSSA._GetContentSourceEx()     # > Returns an extended content source object
                                 #   Hint: To view the extended properties, pipe the
                                 #         output object to: | SELECT *
 $xSSA._GetContentSourceReport() #Equivalent to: $xSSA._GetContentSourceEx() | SELECT
*

#Get a specific content source (either by Content Source ID or by name)
 $cs1 = $xSSA._GetContentSource("Local SharePoint Sites")
#And this is the equivalent (assuming "Local SharePoint Sites" has an ID of 1)
 $cs1 = $xSSA._GetContentSource(1)

#Working with a specific content source
 $xCS1 = $xSSA._GetContentSourceEx(1)
 $xCS1.StartAddresses.AbsoluteUri
 $xCS1.StartIncrementalCrawl()
 $xCS1._CrawlStatusDetailed

#Note: Fulfilling the extended properties can incur higher overhead for environments
#      with many content sources. Therefore, when running GetContentSourceEx for all
#      content sources (e.g. by not specifying a parameter value () or by using "*"),
#      this method implements an in-memory caching mechanism for 20 minutes.
#       - However, the cache is bypassed when requesting a specific content source
#Crawl Visualization...
 $xSSA._ShowRecentCrawlVisualization(2,1)  #Displays a visualization of crawl activit
y
                                           #for the last 2 hours where each character
                                           #represents a 1 minute block of time
```