Advanced Scripting   
Working with XML

Last Updated: 2/4/2022 8:05 AM Version 1  
Document Prepared for: CIT361 Student

# Name Daniel Harris ID 235868292

# Instructions

Save a copy of this document. Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

XML is used by various systems to exchange data. Though falling a bit out of favor now due to JSON and YAML you will still find it used in many systems. In this exercise you will load XML data, modify the data, then save the results.

# Requirements

PowerShell

# Setup

Make sure you have access to the psfiles example files. You can get a copy here <http://cf.esage.com/psfiles.zip>

# Task 1—Read an XML File and View its Contents

The xml file you will be working with has the following organization

<lapidary>

<metals>

<metal>

<Symbol>Ag</Symbol>

<Name>Silver (fine)</Name>

<MeltingPoint>1762</MeltingPoint>

<SpecificGravity>10.6</SpecificGravity>

</metal>

<metal>

<Symbol>Al</Symbol>

<Name>Aluminum</Name>

<MeltingPoint>1220</MeltingPoint>

<SpecificGravity>2.7</SpecificGravity>

</metal>

</metals>

<gems>

<gem>

<GID>2</GID>

<Mineral>Almandite</Mineral>

<Hardness>7.5</Hardness>

<RefractiveIndex>1.8</RefractiveIndex>

<CrownAngle>37</CrownAngle>

<PavilionAngle>42</PavilionAngle>

<CriticalAngle>34</CriticalAngle>

<Common>false</Common>

</gem>

<gem>

<GID>3</GID>

<Mineral>Anatase</Mineral>

<Hardness>5.75</Hardness>

<RefractiveIndex>2.524</RefractiveIndex>

<CrownAngle>35</CrownAngle>

<PavilionAngle>41</PavilionAngle>

<CriticalAngle>24</CriticalAngle>

<Common>false</Common>

</gem>

</gems>

</lapidary>

All the heavy lifting work of XML is provided by the .Net System.Xml object. PowerShell exposes each XML node as an pseudo object in PowerShell. You can take any branch of the XML tree and work with it independently by assigning it a variable. You can work with the pseudo objects as collections of objects.

## Steps

1. To create an instance of an xml document all you have to do is pass an XML formatted string, or array of strings to the XML’s constructor. Make your psfiles/data directory your current directory. Read some XML data.  
   $x=[xml](Get-Content .\lapidary.xml)
2. If this throws and error your xml document string is not formatted correctly.
3. Explore the XML document’s pseudo nodes with PowerShell.  
   $x
   1. You should see the root node of the document, which in this case is lapidary.
4. Enter  
   $x.lapidary
   1. You should see two nodes metals and gems
5. Enter  
   $x.lapiary.metals
   1. You should see a collection of metal objects
6. To see the collection of metal objects enter  
   $x.lapidary.metals.metal
   1. You should see a list of metal objects with several properties. List the property names for a metal: Symbol, Name, MeltingPoint, SpecificGravity
7. Format the metal objects as a table to easier see the content.  
   $x.lapidary.metals.metal|ft

# Task 2—Working with Individual Collections

You can work collections easily. The foreach loop is used to iterate through a collection.

## Steps

1. The Lapidary data’s melting point is in Farenheight, Lets convert it to Celsius. Enter (on one line)  
   foreach($m in $x.lapidary.metals.metal){"$($m.name) melts at $(($m.meltingpoint-32) /1.8)C"}
   1. What is the melting point for titanium? 1800C
2. You can work with subsets of data by assigning a variable to node in the xml document. Create a variable that contains the metals.  
   $metals=$x.lapidary.metals.metal
   1. View the results  
      $metals
3. Do the same for gems  
   $gems=$x.lapidary.gems.gem
4. Sort the metals by name  
   $metals|sort Name
   1. What is the melting point of the first metal in the list? 1220
5. What is the heaviest metal? Platinum

# Task 3—Modifying XML data

## Steps

1. Modify the XML data to update the melting point to Celsius to the nearest degree. After the conversion you will cast the result to an [int] to round the data. However, an XML node can only contain strings so you will need to convert that to a string. You will use the $variable that contains the $metals nodes of the document for this task. Enter  
   foreach($m in $metals) {$m.meltingpoint=[string][int](($m.meltingpoint-32) /1.8)}
   1. Did this change the data in the original xml object ($x)? yes
2. Adding elements. Now you will add a new element to the metals node. It will be Unobtainium, it melts at 5000 degrees and has a specific gravity of .75.
   1. First you need a new Metal node to add data to.  
      $nm=$x.CreateElement('Metal')
   2. Create elements for each property of the Metal, and set the value for that property.  
      $symbol=$x.CreateElement('Symbol')  
      $symbol.InnerText='Uo'  
      $Name=$x.CreateElement('Name')  
      $Name.InnerText='Unobtainium'  
      $mp=$x.CreateElement('MeltingPoint')  
      $mp.InnerText=5000  
      $sp=$x.CreateElement('SpecificGravity')  
      $sp.InnerText=.75
   3. Add each property element to the new metal element  
      $nm.AppendChild($symbol)  
      $nm.AppendChild($name)  
      $nm.AppendChild($mp)  
      $nm.AppendChild($sp)
      1. View your new metal  
         $nm
   4. Add the new metal to the Metals node.  
      $x.lapidary.metals.AppendChild($nm)
   5. View your new xml data  
      $x.lapidary.metals.metal
      1. Is Unobtainium there? yes
   6. Remove nodes. Now you will remove the gems from this document.  
      $x.lapidary.gems.RemoveAll()
      1. View the contents of the xml document  
         $x.lapidary
         1. Are the gems gone? yes

# Task 4—Save the Changes

## Steps

1. You can save an XML document using the XML objects save method. The XML object is unaware of PowerShell’s current directory so you will need to construct a path to save the file to. Enter  
   $filename=Join-Path (Get-Location) 'MetalsInC.xml'  
   $x.Save($filename)
2. View the contents of your new file.  
   Get-Content $filename
   1. Does it have Unobtainium and no gems? yes

# Deliverable

Upload this document with completed answers to i-learn.