Advanced Scripting   
Type Modification

Last Updated: 10/6/2021 8:16 PM Version 1  
Document Prepared for: CIT361 Student

# Name Zach Lucas ID 895649438

# 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

PowerShell supports both implicit and explicit type conversion.

# Requirements

* PowerShell

# Task 1—Explicit Conversion

## Steps

1. You can explicitly cast from one type to another in PowerShell by simply putting the desired name (either short or full) type enclosed in brackets [] in front of the item you want to cast. You can cast literals or data in variables.
   1. Explicitly cast a string to an integer using full name:  
      $v=[System.Int32] '123'
   2. Check the type:  
      $v.gettype()
      1. What is the types name? Int32
      2. What is the types full name? System.Int32
   3. Explicitly cast a string to an integer using short alias:  
      $v=[int] '123'
   4. Check the type:  
      $v.gettype()
      1. What is the types name? Int32
      2. What is the types full name? System.Int32
   5. Explicitly cast a variable containing a string to an integer using short alias:  
      $s='123'  
      $v=[int]$s
2. Get a list of short type aliases  
   [psobject].Assembly.GetType('System.Management.Automation.TypeAccelerators')::Get
   1. What is the full name for the **float** type alias? System.Single
   2. What is the alias for **System.Int64**? Int64
3. Can PowerShell convert the string '1,000.55' to an [int]? Yes
   1. What was the Value? 1001
   2. Observations/comments It just rounds it up to the next whole number.
4. What is the numeric (ascii) value of each letter in the string 'Red Barchetta'? 82, 101, 100, 32, 66, 97, 114, 99, 104, 101, 116, 116, 97
   1. PowerShell Command(s) [int[]][char[]]’Red Barchetta’

# Task 2—Implicit Conversion

When an operator expects a certain type, but a different type is provided, PowerShell will attempt to cast it to the correct type. If the cast can not be completed an error is thrown.

## Steps

1. Implicit conversion in expressions.
   1. Enter:  
      4+'6'
      1. What is the result?
      2. Notice that the string was cast to an integer to complete the expression.
   2. Enter:  
      '4' + 6
      1. What is the result? 46
      2. What is the resultant type? String
2. Implicit conversion passing parameters.
   1. Get a list of processeses:  
      get-process
   2. Choose any process
      1. What is the process ID? 614
      2. What type is the process ID? Int32
   3. Use get-process with the process id you chose earlier.  
      Get-Process -pid <yourprocessid>
   4. Store the process ID you choose earlier as a string in a variable:  
      $pid=[string]<yourprocessedid>
   5. Use get processes to get the process by id using the string $pid variable.  
      get-process -pid $pid
      1. Did the string process ID work? It doesn’t because it is read only or constant
   6. Try to get a process that does not exist by enter an ID that does not exist  
      get-process -pid 32000  
      What error is returned? That it cannot find process with identifier 32000
   7. Try to get a process by ID using a string that cannot be converted to a number:  
      Get-Process -pid 'number1'
      1. What error is returned? Cannot convert the value because it is not in the correct format

# Task 3—Methods, Parsing and Convert Class

Objects may contain either constructors that allow different types to be passed in and then converted, Conversion methods, or Parse methods to aid in type conversion.

## Steps

1. “To” Method converters
   1. View the static members of the int type  
      1|get-member
   2. Notice all the methods that start with “To”. They are methods to convert an int to the corresponding type. How many “To” methods are there? 16
   3. Choose any “To” converter method and run it on an inteteger. E.g 10.ToUint16()
      1. What method did you choose? ToDecimal
      2. What was returned? Turned 2,55 to 2.55
      3. What is the fullname of the datatype returned? System.Decimal
2. “Parse” Methods are provided to convert a string to the target type.
   1. Get list of the static members for the int type:  
      [int]|Get-Member -Static
   2. What “Parse” Methods are there? Parse, TryParse
   3. Since you don’t need an instance of the type to use the static methods, to call a method you use the [type] then a double ::, then the method name. Use the parse method of int to convert a string to an integer:  
      [int]::parse('1101')
      1. What was returned? 1101
      2. What type was returned? Int32
   4. Try one that cannot be parsed:  
      [int]::parse('1101x')
      1. What was the result? An error
      2. What was returned? That it is not in the correct format
3. The TryParse method works like the parse method but returns a Boolean that indicates if the parse was successful. To call it you need to pass two parameters the first is the string to parse, the second is the variable you want the parsed value to be placed in. It must be of the datatype you are converting to. If the parse is successful, the variable will contain the converted value. If not, it will be initialized according to the datatype. The result variable must be passed by reference so that the method can change its value.
   1. Try:  
      [int]$result=0  
      [int]::TryParse('1101',[ref]$result)
      1. What was returned? True
      2. What is the value of $result? 1101
   2. Try this one:  
      [int]::TryParse('1101x',[ref]$result)
      1. What was returned? False
      2. What is the value of $result? 0
   3. The Convert class is used to convert one base type to another. There are many methods. View this doc page <https://docs.microsoft.com/en-us/dotnet/api/system.convert.touint64?view=netframework-4.8> How many overloads are there for the ToUint64 method? 19
   4. Familiarize yourself with the breadth of the class.
   5. Using the Convert class convert a binary string to an unsigned integer:  
      **[convert]::ToUInt64('1101',2)**
      1. What is the result? 13

# Task 4—Adding Properties to Existing Classes

All objects in PowerShell can have properties added to them on the fly. I often use this capability to add a property for tracking progress.

# Steps

1. Create a datetime object that represents Christmas:  
   $d=[datetime]'12/25/2020'
2. Verify there no property named holiday:  
   $d.holiday='christmas'
3. Add a property named holiday to the object in $d:  
   Add-Member -InputObject $d -Name Holiday -Type NoteProperty -Value Christmas -Force
4. View the new property:  
   $d|Get-Member
   1. What is the MemberType of Holiday? NoteProperty
5. View the property’s value:  
   $d.Holiday
6. Try changing the value of Holiday:  
   $d.Holiday="Christ's Birth"
   1. Did it change? It changed to Christ’s Birth

# Deliverable

Upload this document with completed answers to i-learn.