TEC US 2011: PowerShell Deep Dive: Kirk Munro – Defining domain-specific vocabularies using Windows PowerShell

These are the scripts Kirk was using. You can also find his:

- Slides and video here: http://dmitrysotnikov.wordpress.com/2011/09/06/video-kirk-munro-defining-domain-specific- vocabularies-using-Windows-PowerShell /

Visio DSL Examples.ps1

```
# Simple Visio DSV example
Visio {
  DefaultPageBackground 'World'
  DefaultPageTheme -Colors 'Foundry - Dark' -Effects 'Mesh'
  Document {
    $global:myPage = Page 1 {
      Background 'Currency'
      Theme -Colors 'Office - Dark' -Effects 'Outline'
      DiagramStyle 'Flowchart' -Direction 'Bottom to Top' -PlacementDepth 'Shallow'
      ConnectorStyle 'Simple' -Spacing '30mm' -Curvature 'Straight'
      AutoEnlarge
      $root = Get-Item -LiteralPath 'C:\Program Files'
      $global:rootShape = Shape 'Rounded Rectangle' -Label $root.Name
      foreach ($item in ($root | Get-ChildItem | Select -First 4)) {
        $childShape = Shape 'Circle' -Label $item.Name {
          Connect -ToShape $global:rootShape
        }
      }
      Layout -ArrangeShapes -CenterDrawing
    }
  }
} | Out-Null

# Mix and match example

Set-VisioDefaultPageBackground 'World'
Set-VisioDefaultPageTheme -Colors 'Foundry - Dark' -Effects 'Mesh'
$visio = Visio
$document = $visio | Select-VisioDocument
$page = $document | Select-VisioPage -PageId 1 -ScriptBlock {
  Background 'Currency'
  Theme -Colors 'Office - Light' -Effects 'Outline'
  DiagramStyle 'Flowchart' -Direction 'Bottom to Top' -PlacementDepth 'Shallow'
  ConnectorStyle 'Simple' -Spacing '30mm' -Curvature 'Straight'
  AutoEnlarge
  $root = Get-Item -LiteralPath 'C:\Program Files'
  $global:rootShape = Shape 'Rounded Rectangle' -Label $root.Name
  foreach ($item in ($root | Get-ChildItem | Select -First 4)) {
    $childShape = Shape 'Circle' -Label $item.Name {
      Connect -ToShape $global:rootShape
    }
  }
  Layout -ArrangeShapes -CenterDrawing
}

# Real-world DSV example

$defaultVIServer = Connect-VIServer -Server vmguru.com -Credential (Get-Credential vmguru\kmunro)
$visio = Visio {
  Document {
```

```
    Stencil 'C:\users\kmunro\Documents\My Shapes\Virtualization Visio Stencil.vss'
    Page 1 {
      Background 'World'
      DiagramStyle 'Flowchart' -Direction 'Left to Right' -PlacementDepth 'Shallow'
      ConnectorStyle 'Simple' -Spacing '20mm' -Curvature 'Straight'
      $global:vcServerShape = Shape 'Virtual Center Management Console' -Label
$defaultVIServer.Name
      foreach ($cluster in Get-Cluster) {
        $global:clusterShape = Shape 'Cluster' -Label $cluster.Name {
          Connect -ToShape $global:vcServerShape -Right
        }
        $vms = $cluster | Get-VM | Sort-Object -Property Name
        $vmHosts = $cluster | Get-VMHost | Sort-Object -Property Name -Descending
        if (-not $vmHosts) {
          continue
        }
        foreach ($vmHost in $vmHosts) {
          $vmHostShape = Shape 'ESX Host' -Label $vmHost.Name {
            Connect -ToShape $global:clusterShape -Right
          }
          $global:lastShape = $vmHostShape
          $hostVMs = $vms | Where-Object {$_.HostId -eq $vmHost.Id}
          if (-not $hostVMs) {
            continue
          }
          foreach ($vm in $hostVMs) {
            $vmShapeName = 'Other Server'
            switch -regex ($vm.Guest.OSFullName) {
              'Microsoft' {$vmShapeName = 'Microsoft Server'; break}
              'Linux'     {$vmShapeName = 'Linux Server';     break}
            }
            $vmShape = Shape $vmShapeName -Label $vm.Name {
              Connect -ToShape $global:lastShape -Right
            }
            $global:lastShape = $vmShape
          }
        }
      }
      Layout -ArrangeShapes -CenterDrawing
    }
  }
}
Disconnect-VIServer $defaultVIServer
```

Import-VisioAssembly.ps1

```
##############################################################################################################
# File:             Import-VisioAssembly.ps1
#
# Author:           Poshoholic
#
# Publisher:        Poshoholic Studios
#
# Copyright:        © 2011 Poshoholic Studios. All rights reserved.
#
# Usage:            This script file is invoked automatically when you load the Visio module.
#
#                   Please provide feedback on the PowerGUI Forums.
#
##############################################################################################################

Set-StrictMode -Version 2

#region Import the VisioAssembly module.

Import-Module -Name Visio\VisioAssembly
```

```
#endregion

#region Load the Visio assembly.

Import-VisioAssembly

#endregion

# SIG # Begin signature block
# MIIdfwYJKoZIhvcNAQcCoIIdcDCCHWwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU1s31w/IAGmUkavmKBJgUIa68
# l/ygghi8MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAw
# VzELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
# BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5
# MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJH
# bG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
# AQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6a
# rymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCO
# XkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6T
# RGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux
# 2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlD
# SgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
# HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkq
# hkiG9w0BAQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoR
# SLblCKOzyj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQL
# cFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrg
# lfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr
# +WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XS
# QRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4DCCBAcwggLvoAMCAQICCwEA
# AAAAAR5GQJ02MA0GCSqGSIb3DQEBBQUAMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwHhcNMDgxMjE3MTc0ODAyWhcN
# MTExMjE3MTc0ODAyWjBhMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOUXVlc3QgU29m
# dHdhcmUxFzAVBgNVBAMTDlF1ZXN0IFNvZnR3YXJlMSAwHgYJKoZIhvcNAQkBFhFz
# dXBwb3J0QHF1ZXN0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1mza
# 2hKiiqZnaF1sHhuFRS7MEGq9tYhF7AFbJRvTvhCZk9sxK92thKBFyDSOzJauB7Zt
# j+1HwQzpqbbU94EsR09JOf8vB+xQKLCxaBP5YjwhjJzVy+1d6frVWYN1oVxPXRBM
# G7BnFgfRkOdtsg/Qn1Uqn1ENSozyjTuh5iduUy0CAwEAAaOCAUAwggE8MB8GA1Ud
# IwQYMBaAFNJb80smS6Ww5139Vn/28S44TlOgME4GCCsGAQUFBwEBBEIwQDA+Bggr
# BgEFBQcwAoYyaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLm5ldC9jYWNlcnQvT2Jq
# ZWN0U2lnbi5jcnQwOQYDVR0fBDIwMDAuoCygKoYoaHR0cDovL2NybC5nbG9iYWxz
# aWduLm5ldC9PYmplY3RTaWduLmNybDAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIH
# gDATBgNVHSUEDDAKBggrBgEFBQcDAzBLBgNVHSAERDBCMEAGCSsGAQQBoDIBMjAz
# MDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24ubmV0L3JlcG9zaXRv
# cnkvMBEGCWCGSAGG+EIBAQQEAwIEEDANBgkqhkiG9w0BAQUFAAOCAQEAG9hUuQek
# ddDJ/pzfqo9p4hzKBkeKcVsunEeTUMNg90XzgdOYRFJPCD7T+gXXrTs6Y2xFmLJN
# G/2lQsjQ/32cBBN9zZdbX+ExhFfEV9/w0gbw3H/PfYkCRvp9VZlTafIt4MJCt/Zp
# guPQgggpWadScg7jQNyeHEg6H6c3WHO8PMiKcKJp9LuM1PKX9Bjy6F2k8rbdEAyJ
# u0mIiAcnEAc/KwoKBZVT1gnT3rkwgTgNlXw2hqT/Zcf8Jy4IDzbKzL+gYmDCNaju
# wAzhzaA05oZTLwhFV1sdc5MSJVJnMJVLpNO1jrhi5g6Oo6EmezM/kE8nzoXbmTlP
# JjOApuATvUdFlzCCBA0wggL1oAMCAQICCwQAAAAAASOeD6yzMA0GCSqGSIb3DQEB
# BQUAMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAw
# DgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcN
# OTkwMTI4MTMwMDAwWhcNMTcwMTI3MTIwMDAwWjCBgTELMAkGA1UEBhMCQkUxGTAX
# BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExJTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0
# IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMTJ0dsb2JhbFNpZ24gUHJpbWFyeSBPYmpl
# Y3QgUHVibGlzaGluZyBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
# AKKbdSqnE7oJcSQY36EGYikSntyedXPo31ZXaZYTVk/yyLwBWO0mhnILYPUZxVUD
# V5u5EMmh1HRA/2wA6OZTN/632nk+uFI46YEsnw4zUqbNcM5KXWL00WdevJdKB8q8
# 3Y1Hsc3xZVuFAbBLa97Nji71UOijnJ0mmGs2Y0EDcETwX+IldXlQfV+hBqJGDFWV
# RxTTkUaGaJnnJ/SU7JpBUfeW1HqM4USXaHED2FhvvbQQQu4NZnVGi0SW0jAAEgdj
# 90SbAXDKVm+cWJcqJxeLLnFSbUarpysPfxZIZMhS+gYXAAd010WzDPV4lXPoCu7E
# 4HKMHhGqHrtezvm0AO5zvc0CAwEAAaOBrjCBqzAOBgNVHQ8BAf8EBAMCAQYwDwYD
# VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUFVF5GnwMWfnazdjEOhOayXgtf00wMwYD
# VR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nbG9iYWxzaWduLm5ldC9Sb290LmNy
# bDATBgNVHSUEDDAKBggrBgEFBQcDAzAfBgNVHSMEGDAWgBRge2YaRQ2XyolQL30E
# zTSo//z9SzANBgkqhkiG9w0BAQUFAAOCAQEAtXimonwEt3/Jf31qvHH6KTBgwvRi
# Hv5/Qx6bbuKyH3MLhXZbffVOSQYv1Pq3kUDv7W+NjhODVMUqAj0KpNyZC3q9dy/M
```

```
# QMGP88SMTnK6EHzm/2Qrx85sp/zXmnyORo0Bg01CO9ucP58yYVfXF7CzNmbws/1E
# b4E3sZROp1YlifWK1m0RYmJ5XEKQAhjTnCP8COhkRbktfoBbTq/DiimSg3gfkUE0
# r4XF/QeZTixc/sf9F7slJTFNcrW1KUtImjdvE8cRTkpFHn4vMZyr6FKv1meXNIhf
# DidqZlLRWsesMCwgON0r/zrrzhBFgqJ7G6Egc1abKpPmBFEGbBvcL4mUkzCCBBow
# ggMCoAMCAQICCwQAAAAASAZwZBmMA0GCSqGSIb3DQEBBQUAMFcxCzAJBgNVBAYT
# AkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENB
# MRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcNMDkwMzE4MTEwMDAwWhcN
# MjgwMTI4MTIwMDAwWjBUMRgwFgYDVQQLEw9UaW1lc3RhbXBpbmcgQ0ExEzARBgNV
# BAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkdsb2JhbFNpZ24gVGltZXN0YW1waW5n
# IENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwwy3Eg1NaIoz3jYF
# 8Dy69drNDlN7Rp+C8mIT18F3rbuBN35PHpOBwQYi2h1QhMaXlZKpk7Y9q4Z5GVR9
# DhYETMSIlyzGoahfFTrSZCvMPgx66KRWsR67z4TOjTU6NJxsLcB3tTCpH2fmOglE
# OkNyQaKRw0aaH7a5pw+vHHUbZCXnCGwUR/VHGt6O6qJjlX31qK1VomSbcm+5AnM/
# OYo5XMT+j/sRnL0QGUlj0EMii9arkpl0FM8wB75Pvf2Kj55a3208zFqZUJC5rcKX
# Q8Jf7c0zPYfMwaBbqWI7eH1ko6xNHyvXAxFscVSKsKuxHNZ9I9tABzcm21CvOD2m
# B3VvlwIDAQABo4HpMIHmMA4GA1UdDwEB/wQEAwIBBjASBgNVHRMBAf8ECDAGAQH/
# AgEAMB0GA1UdDgQWBBTowvHEMtwzNTe8ZXb1nBcuF0Us/jBLBgNVHSAERDBCMEAG
# CSsGAQQBoDIBHjAzMDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24u
# bmV0L3JlcG9zaXRvcnkvMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwuZ2xv
# YmFsc2lnbi5uZXQvcm9vdC5jcmwwHwYDVR0jBBgwFoAUYHtmGkUNl8qJUC99BM00
# qP/8/UswDQYJKoZIhvcNAQEFBQADggEBAF32yysNAUCEn4V6Q3Bq4MXnqgYA12cT
# yQiRMWVPFKipBdw4nmqgMAq9jceAKO5CRcqU895YRamAMgT1WVxqcAA5J5RN9bRG
# NOgcUzGys1QW6cxCq9XZWTAc+0YnJbiHI7HodYgkgx7Idjd7AUlFSKTt4l3SfJyi
# 3C26EFoSYmWrrgDHEDQ7y3K9FCQM3MN2J7Sn/uFYKfIOFp+ROR2JpuYPHIeM4lis
# kn4kPqrsFOc6MzSLxjusg6sPFGJ6uhotTUsbxTDwC5J5fTx44Pjm0hWWWZk5KzBh
# 6Lj4wKHpIhQReH3E3Im+wLuU4XKu67VAQE/vFx5YXtCoiZaskijpur8wggQuMIID
# FqADAgECAgsBAAAAAAElsLTMATANBgkqhkiG9w0BAQUFADBUMRgwFgYDVQQLEw9U
# aW1lc3RhbXBpbmcgQ0ExEzARBgNVBAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkds
# b2JhbFNpZ24gVGltZXN0YW1waW5nIENBMB4XDTA5MTIyMTA5MzI1NloXDTIwMTIy
# MjA5MzI1NlowUjELMAkGA1UEBhMCQkUxFjAUBgNVBAoTDUdsb2JhbFNpZ24gTlYx
# KzApBgNVBAMTIkdsb2JhbFNpZ24gVGltZSBTdGFtcGluZyBBdXRob3JpdHkwggEi
# MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDNwj1ddyLQwn04MsMVgx9CajtT
# Zt1qNkQNac9ojYlFn34v7kI6M3w+ANOXatha1cNNkgpfBlD9v2zEA6KCYNjtUi4T
# dN6XxkUhe1X26rFkA/x0a7Jfx2xsQxSKJBA3SZWB0kgSpaJ2SVAhf8qFcwo8XbUu
# rZCqXk0yyxeT2X+WwMCJZVbZxbE/mBsn+knuHRvLBowwHDvFp3BbqKsYWv7I9o6/
# AV2PYZg0D1hR/98y6lRlHBQrbPwMkBln7ZvZ2mOb1loko3SOCCMAoZK1HgvRCKBm
# f5Ibo+2AZAJJj7aE79FVjl6pl1rFCAKIlFa/kusqLQY1krU3NjHsw/56O8KFAgMB
# AAGjggEBMIH+MB8GA1UdIwQYMBaAFOjC8cQy3DM1N7xldvWcFy4XRSz+MDwGA1Ud
# HwQ1MDMwMaAvoC2GK2h0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvVGltZXN0YW1w
# aW5nMS5jcmwwHQYDVR0OBBYEFKqqporvpGRz1pXieciP6s+lYCnKMAkGA1UdEwQC
# MAAwDgYDVR0PAQH/BAQDAgeAMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMIMEsGA1Ud
# IAREMEIwQAYJKwYBBAGgMgEeMDMwMQYIKwYBBQUHAgEWJWh0dHA6Ly93d3cuZ2xv
# YmFsc2lnbi5uZXQvcmVwb3NpdG9yeS8wDQYJKoZIhvcNAQEFBQADggEBALyJ7P7m
# NlWTXHnUEXqGgI8XtpOybZuRoVYYEcZV6vYI7a2bnvUrgci73WB7G0eZHm1APh2A
# whPVjgQFL9vnrlKeaIRyoeVKYDz4m9UvRtjDsreTU6ybbEMkJNHx/OlWLjQRWBhD
# 6u//NHRsoMBsf60DGWmIHpVgyru9DLt278cksIHGODHPNq0MOLiQIISbLo8ouZ/2
# ypQnzaw5YVfg45VanHaSMPXeppc9chwqYDKoM02GNTOKXPOk/fcGLOFrSzD1y9ND
# YvhBud59IMsFjI4s9l81/TONQollCDYso4n0WoWLsLl722zLofjSDhu7l3zRJ3m+
# nXw75qdWNNjJkakwggTTMIIDu6ADAgECAgsEAAAAAEjng+vJDANBgkqhkiG9w0B
# AQUFADCBgTELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2Ex
# JTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMT
# J0dsb2JhbFNpZ24gUHJpbWFyeSBPYmplY3QgUHVibGlzaGluZyBDQTAeFw0wNDAx
# MjIxMDAwMDBaFw0xNzAxMjcxMDAwMDBaMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwggEiMA0GCSqGSIb3DQEBAQUA
# A4IBDwAwggEKAoIBAQCwsfKAAHDO7MOMtJftxgmMJm+J32dZgc/eFBNMwrFF4lN1
# QfoHNm+6EXAolHxtcr0HFSVlOgn/hdz6e143hzjkx0sIgJieis1YCQLAwwFJlliI
# iSZZ9W3GucH7GCXt2GJOygpsXXDvztObKQsJxvbuthbUPFSOzF3gr9vdIwkyezKB
# FmIKBst6zzQhtm82trHOy5opNUA+nVh8/62CmPq41YnKNd3LzVcGy5vkv5SogJhf
# d5bwtuerdHlAIaZj6dAHkb2FOLSulqyh/xRz2qVFuE2Gzio879TfKA51qaiIE8Lk
# fGCT8iXMA4SX5k62ny3WtYs0PKvVODrIPcSx+ZTNAgMBAAGjggFnMIIBYzAOBgNV
# HQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQU0lvzSyZL
# pbDnXf1Wf/bxLjhOU6AwSgYDVR0gBEMwQTA/BgkrBgEEAaAyATIwMjAwBggrBgEF
# BQcCARYkaHR0cDovL3d3dy5nbG9iYWxzaWduLm5ldC9yZXBvc2l0b3J5MDkGA1Ud
# HwQyMDAwLqAsoCqGKGh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvcHJpbW9iamVj
# dC5jcmwwTgYIKwYBBQUHAQEEQjBAMD4GCCsGAQUFBzAChjJodHRwOi8vc2VjdXJl
# Lmdsb2JhbHNpZ24ubmV0L2NhY2VydC9QcmltT2JqZWN0LmNydDARBglghkgBhvhC
# AQEEBAMCAAEwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHwYDVR0jBBgwFoAUFVF5GnwM
# WfnazdjEOhOayXgtf00wDQYJKoZIhvcNAQEFBQADggEBAB5q8230jqki/nAIZS6h
# XaszMN1sePpL6q3FjewQemrFWJc5a5LzkeIMpygc0V12josHfBNvrcQ2Q7PBvDFZ
```

```
# zxg42KM7zv/KZ1i/4PGsYT6iOx68AltBrERr9Sbz7V6oZfbKZaY/yvV366WGKlgp
# Vvi+FhBA6dL8VyxjYTdmJTkgLgcDoDYDJZS9fOt+06PCxXYWdTCSuf92QTUhaNEO
# XlyOwwNg5oBA/MBdolRubpJnp4ESh6KjK9u3Tf/k1cflBebV8a78zWYYIfM+R8nl
# lUJhLJ0mgLIPqD0Oyad43250jCxG9nLpPGRrKFXES2Qzy3hUEzjw1XEG1D4NCjUO
# 4LMxggQtMIIEKQIBATByMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxT
# aWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYDVQQDExhHbG9i
# YWxTaWduIE9iamVjdFNpZ24gQ0ECCwEAAAAAAR5GQJ02MAkGBSsOAwIaBQCgeDAY
# BgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgorBgEEAYI3
# AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3DQEJBDEW
# BBTLyk7Ws3VgOsa5ugEHazn2HNOZyTANBgkqhkiG9w0BAQEFAASBgDhG6RzEINcI
# cxUKJfpt8PwTA6Gh+1fp+E/HkQG+m+L+dEFKQ+wW9+jNKGAeWR6jPTpoSj1IKojx
# YH6nQj9fWTQ1J+dRU3WMgHKHK5/SFoD05jl9yKthaiCelYCDZZrt8X539yislAg2
# LoeslfNNd7xs/c4M6iyXZas4C0SuNwFtoYIClzCCApMGCSqGSIb3DQEJBjGCAoQw
# ggKAAgEBMGMwVDEYMBYGA1UECxMPVGltZXN0YW1waW5nIENBMRMwEQYDVQQKEwpH
# bG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxTaWduIFRpbWVzdGFtcGluZyBDQQIL
# AQAAAAABJbC0zAEwCQYFKw4DAhoFAKCB9zAYBgkqhkiG9w0BCQMxCwYJKoZIhvcN
# AQcBMBwGCSqGSIb3DQEJBTEPFw0xMTA0MTcyMjU4NTlaMCMGCSqGSIb3DQEJBDEW
# BBTQhkrx3YG9QoYc5xsifUM0cPG4ezCBlwYLKoZIhvcNAQkQAgwxgYcwgYQwgYEw
# fwQUrt9992u6JBDWfbrxj1uhW0F+SWwwZzBYpFYwVDEYMBYGA1UECxMPVGltZXN0
# YW1waW5nIENBMRMwEQYDVQQKEwpHbG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxT
# aWduIFRpbWVzdGFtcGluZyBDQQILAQAAAAABJbC0zAEwDQYJKoZIhvcNAQEBBQAE
# ggEAe5iEivhQDqJOFWoLqHC3Hk5yP7sVCvQB/MmW6xelv/VvDAKMISv3IIQobmgr
# 4suN/9RidV/FR6enBkczkSlo4fVlCRn+ponoBA6C0Aymi1W5FbayAXvJ6rv3JQTe
# ZlhPOOxzPo46IskTPKDb5Po+pBQRYGuAmWm6Mn+A+5MiWi0sbZM+Ksh0hbkNA4ks
# Jkb+fwqCyUKgBFiXKYcsJU5Wkd/sprU4HtMSAd6dsY72xegrDHVaZQ8H77sWCtY6
# FHTeQr91Ec9wUNBR5xwXo0WSjCe6nmdEv5nZ5UQle2/ooq+pFtj0m3i8rG24lb+s
# lhjLwtItNAUR0Ey15Mvk3DAPnw==
# SIG # End signature block
```

Visio.psd1

```
#############################################################################################################
#####################
# File:             Visio.psd1
#
# Author:           Poshoholic
#
# Publisher:        Poshoholic Studios
#
# Copyright:        © 2011 Poshoholic Studios. All rights reserved.
#
# Usage:            To load this module in your Script Editor:
#
#                   1. Open the Script Editor.
#
#                   2. Select "PowerShell Libraries" from the File menu.
#
#                   3. Check the Visio module.
#
#                   4. Click on OK to close the "PowerShell Libraries" dialog.
#
#                   Alternatively you can load the module from the embedded console by invoking
this:                 #
#                       Import-Module -Name Visio
#
#                   Please provide feedback on the PowerGUI Forums.
#
#############################################################################################################
#####################

@{

# Script module or binary module file associated with this manifest
ModuleToProcess = 'Visio.psm1'

# Version number of this module.
ModuleVersion = '2.0.0.0'
```

```
# ID used to uniquely identify this module
GUID = '{6d05b20c-37c2-4bd2-8d23-6e61ca461d8e}'

# Author of this module
Author = 'Poshoholic'

# Company or vendor of this module
CompanyName = 'Poshoholic Studios'

# Copyright statement for this module
Copyright = '© 2011 Poshoholic Studios. All rights reserved.'

# Description of the functionality provided by this module
Description = 'Commands for Visio document automation using traditional PowerShell commands or a
domain-specific vocabulary (DSV) for Visio'

# Minimum version of the Windows PowerShell engine required by this module
PowerShellVersion = '2.0'

# Minimum version of the .NET Framework required by this module
DotNetFrameworkVersion = '2.0'

# Minimum version of the common language runtime (CLR) required by this module
CLRVersion = '2.0.50727'

# Processor architecture (None, X86, Amd64, IA64) required by this module
ProcessorArchitecture = 'None'

# Modules that must be imported into the global environment prior to importing
# this module
RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to
# importing this module
ScriptsToProcess = @(
  '.\Import-VisioAssembly.ps1'
)

# Type files (.ps1xml) to be loaded when importing this module
TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
FormatsToProcess = @()

# Modules to import as nested modules of the module specified in
# ModuleToProcess
NestedModules = @()

# Functions to export from this module
FunctionsToExport = '*'

# Cmdlets to export from this module
CmdletsToExport = '*'

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module
AliasesToExport = '*'

# List of all modules packaged with this module
ModuleList = @()

# List of all files packaged with this module
FileList = @(
  '.\Visio.psm1'
  '.\Visio.psd1'
```

```
  '.\Import-VisioAssembly.ps1'
  '.\VisioAssembly\VisioAssembly.psm1'
  '.\VisioAssembly\VisioAssembly.psd1'
)

# Private data to pass to the module specified in ModuleToProcess
PrivateData = @{}

}

# SIG # Begin signature block
# MIIdfwYJKoZIhvcNAQcCoIIdcDCCHWwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUr9x+2UvTe5npe78DvQ5FXCfI
# klygghi8MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAw
# VzELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
# BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5
# MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJH
# bG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
# AQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6a
# rymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCO
# XkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ41tbcdG6T
# RGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux
# 2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlD
# SgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
# HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkq
# hkiG9w0BAQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoR
# SLblCKOzyj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQL
# cFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrg
# lfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr
# +WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XS
# QRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4DCCBAcwggLvoAMCAQICCwEA
# AAAAR5GQJ02MA0GCSqGSIb3DQEBBQUAMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwHhcNMDgxMjE3MTc0ODAyWhcN
# MTExMjE3MTc0ODAyWjBhMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOUXVlc3QgU29m
# dHdhcmUxFzAVBgNVBAMTDlF1ZXN0IFNvZnR3YXJlMSAwHgYJKoZIhvcNAQkBFhFz
# dXBwb3J0QHF1ZXN0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1mza
# 2hKiiqZnaF1sHhuFRS7MEGq9tYhF7AFbJRvTvhCZk9sxK92thKBFyDSOzJauB7Zt
# j+1HwQzpqbbU94EsR09JOf8vB+xQKLCxaBP5YjwhjJzVy+1d6frVWYN1oVxPXRBM
# G7BnFgfRkOdtsg/Qn1Uqn1ENSozyjTuh5iduUy0CAwEAAaOCAUAwggE8MB8GA1Ud
# IwQYMBaAFNJb80smS6Ww5139Vn/28S44TlOgME4GCCsGAQUFBwEBBEIwQDA+Bggr
# BgEFBQcwAoYyaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLm5ldC9jYWNlcnQvT2Jq
# ZWN0U2lnbi5jcnQwOQYDVR0fBDIwMDAuoCygKoYoaHR0cDovL2NybC5nbG9iYWxz
# aWduLm5ldC9PYmplY3RTaWduLmNybDAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIH
# gDATBgNVHSUEDDAKBggrBgEFBQcDAzBLBgNVHSAERDBCMEAGCSsGAQQBoDIBMjAz
# MDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24ubmV0L3JlcG9zaXRv
# cnkvMBEGCWCGSAGG+EIBAQQEAwIEEDANBgkqhkiG9w0BAQUFAAOCAQEAG9hUuQek
# ddDJ/pzfqo9p4hzKBkeKcVsunEeTUMNg90XzgdOYRFJPCD7T+gXXrTs6Y2xFmLJN
# G/2lQsjQ/32cBBN9zZdbX+ExhFfEV9/w0gbw3H/PfYkCRvp9VZlTafIt4MJCt/Zp
# guPQgggpWadScg7jQNyeHEg6H6c3WHO8PMiKcKJp9LuM1PKX9Bjy6F2k8rbdEAyJ
# u0mIiAcnEAc/KwoKBZVT1gnT3rkwgTgNlXw2hqT/Zcf8Jy4IDzbKzL+gYmDCNaju
# wAzhzaA05oZTLwhFV1sdc5MSJVJnMJVLpNO1jrhi5g6Oo6EmezM/kE8nzoXbmTlP
# JjOApuATvUdFlzCCBA0wggL1oAMCAQICCwQAAAAASOeD6yzMA0GCSqGSIb3DQEB
# BQUAMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAw
# DgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcN
# OTkwMTI4MTMwMDAwWhcNMTcwMTI3MTIwMDAwWjCBgTELMAkGA1UEBhMCQkUxGTAX
# BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExJTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0
# IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMTJ0dsb2JhbFNpZ24gUHJpbWFyeSBPYmpl
# Y3QgUHVibGlzaGluZyBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
# AKKbdSqnE7oJcSQY36EGYikSntyedXPo31ZXaZYTVk/yyLwBWO0mhnILYPUZxVUD
# V5u5EMmh1HRA/2wA6OZTN/632nk+uFI46YEsnw4zUqbNcM5KXWL00WdevJdKB8q8
# 3Y1Hsc3xZVuFAbBLa97Nji71UOijnJ0mmGs2Y0EDcETwX+IldXlQfV+hBqJGDFWV
# RxTTkUaGaJnnJ/SU7JpBUfeW1HqM4USXaHED2FhvvbQQQu4NZnVGi0SW0jAAEgdj
# 90SbAXDKVm+cWJcqJxeLLnFSbUarpysPfxZIZMhS+gYXAAd010WzDPV4lXPoCu7E
# 4HKMHhGqHrtezvm0AO5zvc0CAwEAAaOBrjCBqzAOBgNVHQ8BAf8EBAMCAQYwDwYD
# VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUFVF5GnwMWfnazdjEOhOayXgtf00wMwYD
# VR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nbG9iYWxzaWduLm5ldC9Sb290LmNy
# bDATBgNVHSUEDDAKBggrBgEFBQcDAzAfBgNVHSMEGDAWgBRge2YaRQ2XyolQL30E
```

```
# zTSo//z9SzANBgkqhkiG9w0BAQUFAAOCAQEAtXimonwEt3/Jf31qvHH6KTBgwvRi
# Hv5/Qx6bbuKyH3MLhXZbffVOSQYv1Pq3kUDv7W+NjhODVMUqAj0KpNyZC3q9dy/M
# QMGP88SMTnK6EHzm/2Qrx85sp/zXmnyORo0Bg01CO9ucP58yYVfXF7CzNmbws/1E
# b4E3sZROp1YlifWK1m0RYmJ5XEKQAhjTnCP8COhkRbktfoBbTq/DiimSg3gfkUE0
# r4XF/QeZTixc/sf9F7slJTFNcrW1KUtImjdvE8cRTkpFHn4vMZyr6FKv1meXNIhf
# DidqZlLRWsesMCwgON0r/zrrzhBFgqJ7G6Egc1abKpPmBFEGbBvcL4mUkzCCBBow
# ggMCoAMCAQICCwQAAAAASAZwZBmMA0GCSqGSIb3DQEBBQUAMFcxCzAJBgNVBAYT
# AkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENB
# MRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcNMDkwMzE4MTEwMDAwWhcN
# MjgwMTI4MTIwMDAwWjBUMRgwFgYDVQQLEw9UaW1lc3RhbXBpbmcgQ0ExEzARBgNV
# BAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkdsb2JhbFNpZ24gVGltZXN0YW1waW5n
# IENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwwy3Eg1NaIoz3jYF
# 8Dy69drNDlN7Rp+C8mIT18F3rbuBN35PHpOBwQYi2h1QhMaXlZKpk7Y9q4Z5GVR9
# DhYETMSIlyzGoahfFTrSZCvMPgx66KRWsR67z4TOjTU6NJxsLcB3tTCpH2fmOglE
# OkNyQaKRw0aaH7a5pw+vHHUbZCXnCGwUR/VHGt6O6qJjlX31qK1VomSbcm+5AnM/
# OYo5XMT+j/sRnL0QGUlj0EMii9arkpl0FM8wB75Pvf2Kj55a3208zFqZUJC5rcKX
# Q8Jf7c0zPYfMwaBbqWI7eH1ko6xNHyvXAxFscVSKsKuxHNZ9I9tABzcm21CvOD2m
# B3VvlwIDAQABo4HpMIHmMA4GA1UdDwEB/wQEAwIBBjASBgNVHRMBAf8ECDAGAQH/
# AgEAMB0GA1UdDgQWBBTowvHEMtwzNTe8ZXb1nBcuF0Us/jBLBgNVHSAERDBCMEAG
# CSsGAQQBoDIBHjAzMDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24u
# bmV0L3JlcG9zaXRvcnkvMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwuZ2xv
# YmFsc2lnbi5uZXQvcm9vdC5jcmwwHwYDVR0jBBgwFoAUYHtmGkUNl8qJUC99BM00
# qP/8/UswDQYJKoZIhvcNAQEFBQADggEBAF32yysNAUCEn4V6Q3Bq4MXnqgYA12cT
# yQiRMWVPFKipBdw4nmqgMAq9jceAKO5CRcqU895YRamAMgT1WVxqcAA5J5RN9bRG
# NOgcUzGys1QW6cxCq9XZWTAc+0YnJbiHI7HodYgkgx7Idjd7AUlFSKTt4l3SfJyi
# 3C26EFoSYmWrrgDHEDQ7y3K9FCQM3MN2J7Sn/uFYKfIOFp+ROR2JpuYPHIeM4lis
# kn4kPqrsFOc6MzSLxjusg6sPFGJ6uhotTUsbxTDwC5J5fTx44Pjm0hWWWZk5KzBh
# 6Lj4wKHpIhQReH3E3Im+wLuU4XKu67VAQE/vFx5YXtCoiZaskijpur8wggQuMIID
# FqADAgECAgsBAAAAAAElsLTMATANBgkqhkiG9w0BAQUFADBUMRgwFgYDVQQLEw9U
# aW1lc3RhbXBpbmcgQ0ExEzARBgNVBAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkds
# b2JhbFNpZ24gVGltZXN0YW1waW5nIENBMB4XDTA5MTIyMTA5MzI1NloXDTIwMTIy
# MjA5MzI1NlowUjELMAkGA1UEBhMCQkUxFjAUBgNVBAoTDUdsb2JhbFNpZ24gTlYx
# KzApBgNVBAMTIkdsb2JhbFNpZ24gVGltZSBTdGFtcGluZyBBdXRob3JpdHkwggEi
# MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDNwj1ddyLQwn04MsMVgx9CajtT
# Zt1qNkQNac9ojYlFn34v7kI6M3w+ANOXatha1cNNkgpfBlD9v2zEA6KCYNjtUi4T
# dN6XxkUhe1X26rFkA/x0a7Jfx2xsQxSKJBA3SZWB0kgSpaJ2SVAhf8qFcwo8XbUu
# rZCqXk0yyxeT2X+WwMCJZVbZxbE/mBsn+knuHRvLBowwHDvFp3BbqKsYWv7I9o6/
# AV2PYZg0D1hR/98y6lRlHBQrbPwMkBln7ZvZ2mOb1loko3SOCCMAoZK1HgvRCKBm
# f5Ibo+2AZAJJj7aE79FVjl6pl1rFCAKIlFa/kusqLQY1krU3NjHsw/56O8KFAgMB
# AAGjggEBMIH+MB8GA1UdIwQYMBaAFOjC8cQy3DM1N7xldvWcFy4XRSz+MDwGA1Ud
# HwQ1MDMwMaAvoC2GK2h0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvVGltZXN0YW1w
# aW5nMS5jcmwwHQYDVR0OBBYEFKqqporvpGRz1pXieciP6s+lYCnKMAkGA1UdEwQC
# MAAwDgYDVR0PAQH/BAQDAgeAMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMIMEsGA1Ud
# IAREMEIwQAYJKwYBBAGgMgEeMDMwMQYIKwYBBQUHAgEWJWh0dHA6Ly93d3cuZ2xv
# YmFsc2lnbi5uZXQvcmVwb3NpdG9yeS8wDQYJKoZIhvcNAQEFBQADggEBALyJ7P7m
# NlWTXHnUEXqGgI8XtpOybZuRoVYYEcZV6vYI7a2bnvUrgci73WB7G0eZHm1APh2A
# whPVjgQFL9vnrlKeaIRyoeVKYDz4m9UvRtjDsreTU6ybbEMkJNHx/OlWLjQRWBhD
# 6u//NHRsoMBsf60DGWmIHpVgyru9DLt278cksIHGODHPNq0MOLiQIISbLo8ouZ/2
# ypQnzaw5YVfg45VanHaSMPXeppc9chwqYDKoM02GNTOKXPOk/fcGLOFrSzD1y9ND
# YvhBud59IMsFjI4s9l81/TONQollCDYso4n0WoWLsLl722zLofjSDhu7l3zRJ3m+
# nXw75qdWNNjJkakwggTTMIIDu6ADAgECAgsEAAAAAEjng+vJDANBgkqhkiG9w0B
# AQUFADCBgTELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2Ex
# JTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMT
# J0dsb2JhbFNpZ24gUHJpbWFyeSBPYmplY3QgUHVibGlzaGluZyBDQTAeFw0wNDAx
# MjIxMDAwMDBaFw0xNzAxMjcxMDAwMDBaMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwggEiMA0GCSqGSIb3DQEBAQUA
# A4IBDwAwggEKAoIBAQCwsfKAAHDO7MOMtJftxgmMJm+J32dZgc/eFBNMwrFF4lN1
# QfoHNm+6EXAolHxtcr0HFSVlOgn/hdz6e143hzjkx0sIgJieis1YCQLAwwFJlliI
# iSZZ9W3GucH7GCXt2GJOygpsXXDvztObKQsJxvbuthbUPFSOzF3gr9vdIwkyezKB
# FmIKBst6zzQhtm82trHOy5opNUA+nVh8/62CmPq41YnKNd3LzVcGy5vkv5SogJhf
# d5bwtuerdHlAIaZj6dAHkb2FOLSulqyh/xRz2qVFuE2Gzio879TfKA51qaiIE8Lk
# fGCT8iXMA4SX5k62ny3WtYs0PKvVODrIPcSx+ZTNAgMBAAGjggFnMIIBYzAOBgNV
# HQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQU0lvzSyZL
# pbDnXf1Wf/bxLjhOU6AwSgYDVR0gBEMwQTA/BgkrBgEEAaAyATIwMjAwBggrBgEF
# BQcCARYkaHR0cDovL3d3dy5nbG9iYWxzaWduLm5ldC9yZXBvc2l0b3J5MDkGA1Ud
# HwQyMDAwLqAsoCqGKGh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvcHJpbW9iamVj
# dC5jcmwwTgYIKwYBBQUHAQEEQjBAMD4GCCsGAQUFBzAChjJodHRwOi8vc2VjdXJl
# Lmdsb2JhbHNpZ24ubmV0L2NhY2VydC9QcmltT2JqZWN0LmNydDARBglghkgBhvhC
# AQEEBAMCAAEwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHwYDVR0jBBgwFoAUFVF5GnwM
```

```
# WfnazdjEOhOayXgtf00wDQYJKoZIhvcNAQEFBQADggEBAB5q8230jqki/nAIZS6h
# XaszMN1sePpL6q3FjewQemrFWJc5a5LzkeIMpygc0V12josHfBNvrcQ2Q7PBvDFZ
# zxg42KM7zv/KZ1i/4PGsYT6iOx68AltBrERr9Sbz7V6oZfbKZaY/yvV366WGKlgp
# Vvi+FhBA6dL8VyxjYTdmJTkgLgcDoDYDJZS9fOt+06PCxXYWdTCSuf92QTUhaNEO
# XlyOwwNg5oBA/MBdolRubpJnp4ESh6KjK9u3Tf/k1cflBebV8a78zWYYIfM+R8nl
# lUJhLJ0mgLIPqD0Oyad43250jCxG9nLpPGRrKFXES2Qzy3hUEzjw1XEG1D4NCjUO
# 4LMxggQtMIIEKQIBATByMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxT
# aWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYDVQQDExhHbG9i
# YWxTaWduIE9iamVjdFNpZ24gQ0ECCwEAAAAAAR5GQJ02MAkGBSsOAwIaBQCgeDAY
# BgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgorBgEEAYI3
# AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3DQEJBDEW
# BBTmefoHdsRwr+XoVY801j2iFs2rODANBgkqhkiG9w0BAQEFAASBgFaKZNgv2//o
# CPrBMys0a91gQvrb31yJlhMypxaewewSgdjESKyEn3iIGlfxsVrR9JGTXfhDlWDc
# ImqHcvjO5Yiaxnle0QrWB4x5+jlUcF6npZthnLvM8gAvSw9Y9KjNiuOvftnQ2S+G
# vSt3kEKvQzhzqbItvV37IIsPnYDO9kKPoYIClzCCApMGCSqGSIb3DQEJBjGCAoQw
# ggKAAgEBMGMwVDEYMBYGA1UECxMPVGltZXN0YW1waW5nIENBMRMwEQYDVQQKEwpH
# bG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxTaWduIFRpbWVzdGFtcGluZyBDQQIL
# AQAAAAABJbC0zAEwCQYFKw4DAhoFAKCB9zAYBgkqhkiG9w0BCQMxCwYJKoZIhvcN
# AQcBMBwGCSqGSIb3DQEJBTEPFw0xMTA0MTkwNjA4NDlaMCMGCSqGSIb3DQEJBDEW
# BBTGih/4DNlWoiffuwjtQCR2grqY+DCBlwYLKoZIhvcNAQkQAgwxgYcwgYQwgYEw
# fwQUrt9992u6JBDWfbrxj1uhW0F+SWwwZzBYpFYwVDEYMBYGA1UECxMPVGltZXN0
# YW1waW5nIENBMRMwEQYDVQQKEwpHbG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxT
# aWduIFRpbWVzdGFtcGluZyBDQQILAQAAAAABJbC0zAEwDQYJKoZIhvcNAQEBBQAE
# ggEAtuR5xnugWWboaIl185ourPlDzDgV6ep+iU1+4W1UQMvkV4YdVvwpyfq/s8H+
# mVkw9Hdatbjz+l/HtNxhEuu8rjDjD9g+05CG34bexg+far18lN81ixoaNUxOd279
# 5XC8yfa98ndDt0zrTJ6ND7ZCdZOJzQs7xcdB3Qpd9kEVtxS4QymG0ReiRY57n1FF
# i7sYdIJ7howMj3xVAGOirHV8GQooYpzu4p+v2NgmpU06Tx/mKBRgrJKHpqNSdsil
# V8ha5LvjeadRowMzHYhWIyBdr3dcf6lUAqgAu07O2SRFz5kahfU35CLRGLbcXeOh
# DCbtTqKlhaqA+MlpbVvT2OQ/Vg==
# SIG # End signature block
```

Visio.psm1

```
########################################################################################################################
# File:             Visio.psm1
#
# Author:           Poshoholic
#
# Publisher:        Poshoholic Studios
#
# Copyright:        © 2011 Poshoholic Studios. All rights reserved.
#
# Usage:            To load this module in your Script Editor:
#
#                   1. Open the Script Editor.
#
#                   2. Select "PowerShell Libraries" from the File menu.
#
#                   3. Check the Visio module.
#
#                   4. Click on OK to close the "PowerShell Libraries" dialog.
#
#                   Alternatively you can load the module from the embedded console by invoking
this:                 #
#                       Import-Module -Name Visio
#
#                   Please provide feedback on the PowerGUI Forums.
#
########################################################################################################################

#Set-StrictMode -Version 2

#region Set the module export mode to explicit.

Export-ModuleMember
```

```
#endregion

#region Visio DSV command definitions.

$script:visioCommandVocabulary = @{
  #region Define the Visio Application DSV.
  Application = @{
    #region Define aliases in the DSV.
    Alias = @{
      DefaultPageBackground = @{
              Scope = 'Script'
              Value = 'Set-VisioDefaultPageBackground'
        Description = 'Set the default Visio page background. This background will be applied to
any new Visio pages that are created.'
      }
      DefaultPageTheme = @{
              Scope = 'Script'
              Value = 'Set-VisioDefaultPageTheme'
        Description = 'Set the default Visio page theme. This theme will be applied to any new
Visio pages that are created.'
      }
      DefaultPageAutoEnlarge = @{
              Scope = 'Script'
              Value = 'Set-VisioDefaultPageAutoEnlarge'
        Description = 'Set the default Visio page auto enlarge property. This setting will be
applied to any new Visio pages that are created.'
      }
      DefaultPageDiagramStyle = @{
              Scope = 'Script'
              Value = 'Set-VisioDefaultPageDiagramStyle'
        Description = 'Set the default Visio page diagram style. This diagram style will be applied
to any new Visio pages that are created.'
      }
      DefaultPageConnectorStyle = @{
              Scope = 'Script'
              Value = 'Set-VisioDefaultPageConnectorStyle'
        Description = 'Set the default Visio page connector style. This connector style will be
applied to any new Visio pages that are created.'
      }
    }
    #endregion
    #region Define functions in the DSV.
    Function = @{
      SmartLayout = @{
        Scope = 'Script'
        Value = {
          [CmdletBinding()]
          [OutputType([System.Void])]
          param(
            [Parameter(Position=0)]
            [ValidateNotNull()]
            [System.Boolean]
            $Enabled
          )
          try {
            if ((-not $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Enabled')) -or $Enabled)
{
              Enable-VisioSmartLayout
            } else {
              Disable-VisioSmartLayout
            }
          } catch {
            throw
          }
        }
      }
      Show = @{
        Scope = 'Script'
        Value = {
          [CmdletBinding()]
```

```
          [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
          param(
            [Parameter()]
            [System.Management.Automation.SwitchParameter]
            $PassThru
          )
          try {
            $showVisioParameters = $PSCmdlet.MyInvocation.BoundParameters
            Show-Visio -Application $visioApplication @showVisioParameters
          } catch {
            throw
          }
        }
      }
      Hide = @{
        Scope = 'Script'
        Value = {
          [CmdletBinding()]
          [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
          param(
            [Parameter()]
            [System.Management.Automation.SwitchParameter]
            $PassThru
          )
          try {
            $hideVisioParameters = $PSCmdlet.MyInvocation.BoundParameters
            Hide-Visio -Application $visioApplication @hideVisioParameters
          } catch {
            throw
          }
        }
      }
      Close = @{
        Scope = 'Script'
        Value = {
          [CmdletBinding()]
          [OutputType([System.Void])]
          param()
          try {
            $stopVisioParameters = $PSCmdlet.MyInvocation.BoundParameters
            Stop-Visio -Application $visioApplication @stopVisioParameters
          } catch {
            throw
          }
        }
      }
      Document = @{
        Scope = 'Script'
        Value = {
          [CmdletBinding(DefaultParameterSetName='UnnamedDocument')]
          [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
          param(
            [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='NamedDocument')]
            [ValidateNotNullOrEmpty()]
            [Alias('Name')]
            [System.String[]]
            $DocumentId,

            [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='DocumentObject')]
            [ValidateNotNullOrEmpty()]
            [Alias('InputObject')]
            [Microsoft.Office.Interop.Visio.IVDocument[]]
            $Document,

            [Parameter(Position=0, ParameterSetName='UnnamedDocument')]
            [Parameter(Position=1, ParameterSetName='NamedDocument')]
            [Parameter(Position=1, ParameterSetName='DocumentObject')]
            [ValidateNotNullOrEmpty()]
```

```
            [System.Management.Automation.ScriptBlock]
            $ScriptBlock
          )
          begin {
            try {
              #region Initialize local variables.
              [System.String]$private:parameterName = $null
              #endregion

              #region Convert the parameters to private scope.
              foreach ($private:parameterName in @('DocumentId','Document','ScriptBlock')) {
                if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
                  Set-Variable -Name $private:parameterName -Scope Local -Option Private
                }
              }
              #endregion
            } catch {
              throw
            }
          }
          process {
            try {
              $private:selectVisioDocumentParameters = $PSCmdlet.MyInvocation.BoundParameters
              if ($PSCmdlet.ParameterSetName -eq 'DocumentObject') {
                Select-VisioDocument @private:selectVisioDocumentParameters
              } else {
                Select-VisioDocument -Application $visioApplication
@private:selectVisioDocumentParameters
              }
            } catch {
              throw
            }
          }
        }
      }
    }
    #endregion
    #region Define nested modules in the DSV.
    Module = @{
      #region Define the Visio Document DSV.
      Document = @{
        #region Define functions in the DSV.
        Function = @{
          Save = @{
            Scope = 'Script'
            Value = {
              [CmdletBinding()]
              [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
              param(
                [Parameter(Position=0, Mandatory=$true)]
                [ValidateNotNullOrEmpty()]
                [Alias('FileName')]
                [System.String]
                $Path,

                [Parameter()]
                [System.Management.Automation.SwitchParameter]
                $Force,

                [Parameter()]
                [System.Management.Automation.SwitchParameter]
                $PassThru
              )
              try {
                $saveVisioDocumentParameters = $PSCmdlet.MyInvocation.BoundParameters
                Save-VisioDocument -Document $visioDocument @saveVisioDocumentParameters
              } catch {
                throw
              }
            }
```

```
        }
        Close = @{
          Scope = 'Script'
          Value = {
            [CmdletBinding()]
            [OutputType([System.Void])]
            param(
              [Parameter()]
              [System.Management.Automation.SwitchParameter]
              $Force
            )
            try {
              $closeVisioDocumentParameters = $PSCmdlet.MyInvocation.BoundParameters
              Close-VisioDocument -Document $visioDocument @closeVisioDocumentParameters
            } catch {
              throw
            }
          }
        }
        Stencil = @{
          Scope = 'Script'
          Value = {
            [CmdletBinding()]
            [OutputType([System.Void])]
            param(
              [Parameter(Position=0, Mandatory=$true)]
              [ValidateNotNullOrEmpty()]
              [System.String]
              $StencilId
            )
            try {
              $selectVisioStencilParameters = $PSCmdlet.MyInvocation.BoundParameters
              Select-VisioStencil -Document $visioDocument @selectVisioStencilParameters
            } catch {
              throw
            }
          }
        }
        Page = @{
          Scope = 'Script'
          Value = {
            [CmdletBinding(DefaultParameterSetName='NamedPage')]
            [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
            param(
              [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='NamedPage')]
              [ValidateNotNullOrEmpty()]
              [System.String[]]
              $PageId,

              [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='PageObject')]
              [ValidateNotNullOrEmpty()]
              [Alias('InputObject')]
              [Microsoft.Office.Interop.Visio.IVPage[]]
              $Page,

              [Parameter(Position=1)]
              [ValidateNotNullOrEmpty()]
              [System.Management.Automation.ScriptBlock]
              $ScriptBlock
            )
            begin {
              try {
                #region Initialize local variables.
                [System.String]$private:parameterName = $null
                #endregion

                #region Convert the parameters to private scope.
                foreach ($private:parameterName in @('PageId','Page','ScriptBlock')) {
```

```
                    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName))
{
                      Set-Variable -Name $private:parameterName -Scope Local -Option Private
                    }
                  }
                  #endregion
                } catch {
                  throw
                }
              }
              process {
                try {
                  $private:selectVisioPageParameters = $PSCmdlet.MyInvocation.BoundParameters
                  if ($PSCmdlet.ParameterSetName -eq 'PageObject') {
                    Select-VisioPage @private:selectVisioPageParameters
                  } else {
                    Select-VisioPage -Document $visioDocument @private:selectVisioPageParameters
                  }
                } catch {
                  throw
                }
              }
            }
          }
        }
        #endregion
        #region Define nested modules in the DSV.
        Module = @{
          #region Define the Visio Page DSV.
          Page = @{
            #region Define functions in the DSV.
            Function = @{
              Name = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0, Mandatory=$true)]
                    [ValidateNotNullOrEmpty()]
                    [System.String]
                    $NewName,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
                    $renameVisioPageParameters = $PSCmdlet.MyInvocation.BoundParameters
                    Rename-VisioPage -Page $visioPage @renameVisioPageParameters
                  } catch {
                    throw
                  }
                }
              }
              AutoEnlarge = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0)]
                    [System.Boolean]
                    $Enabled,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
```

```
                    $passThruParameter = @{}
                    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and
$PassThru) {
                      $passThruParameter['PassThru'] = $true
                    }
                    if ((-not $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Enabled')) -or
$Enabled) {
                      Enable-VisioPageAutoEnlarge -Page $visioPage @passThruParameter
                    } else {
                      Disable-VisioPageAutoEnlarge -Page $visioPage @passThruParameter
                    }
                  } catch {
                    throw
                  }
                }
              }
              DiagramStyle = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0, Mandatory=$true)]

[ValidateScript({@('Default','Radial','Flowchart','Circular','CompactTree','Hierarchy') -contains
($_ -replace ' ')})]
                    [System.String]
                    $Style,

                    [Parameter(Position=1)]

[ValidateScript({@('DownThenRight','RightThenDown','RightThenUp','UpThenRight','UpThenLeft','LeftTh
enUp','LeftThenDown','DownThenLeft','TopToBottom','BottomToTop','LeftToRight','RightToLeft') -
contains ($_ -replace ' ')})]
                    [System.String]
                    $Direction,

                    [Parameter(Position=2)]
                    [ValidateSet('Left','Center','Right','Top','Middle','Bottom')]
                    [System.String]
                    $Alignment,

                    [Parameter(Position=3)]
                    [ValidateSet('Default','Shallow','Medium','Deep')]
                    [System.String]
                    $PlacementDepth,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
                    $setVisioDiagramPageStyleParameters = $PSCmdlet.MyInvocation.BoundParameters
                    Set-VisioPageDiagramStyle -Page $visioPage @setVisioDiagramPageStyleParameters
                  } catch {
                    throw
                  }
                }
              }
              ConnectorStyle = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0)]

[ValidateScript({@('Default','RightAngle','Straight','CenterToCenter','Network','Flowchart','Tree',
'OrganizationChart','Simple','SimpleHorizontalVertical','SimpleVerticalHorizontal') -contains ($_ -
replace ' ')})]
```

```
                    [System.String]
                    $Style,

                    [Parameter(Position=1)]
                    [ValidateNotNullOrEmpty()]
                    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
                    [System.String]
                    $Spacing,

                    [Parameter(Position=2)]
                    [ValidateSet('Default','Straight','Curved')]
                    [System.String]
                    $Curvature,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
                    $setVisioPageConnectorStyleParameters = $PSCmdlet.MyInvocation.BoundParameters
                    Set-VisioPageConnectorStyle -Page $visioPage
@setVisioPageConnectorStyleParameters
                  } catch {
                    throw
                  }
                }
              }
              Theme = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0)]
                    [System.String]
                    $Colors,

                    [Parameter(Position=1)]
                    [System.String]
                    $Effects,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
                    $setVisioPageThemeParameters = $PSCmdlet.MyInvocation.BoundParameters
                    Set-VisioPageTheme -Page $visioPage @setVisioPageThemeParameters
                  } catch {
                    throw
                  }
                }
              }
              Background = @{
                Scope = 'Script'
                Value = {
                  [CmdletBinding()]
                  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                  param(
                    [Parameter(Position=0)]
                    [System.String]
                    $Background,

                    [Parameter()]
                    [System.Management.Automation.SwitchParameter]
                    $PassThru
                  )
                  try {
                    $setVisioPageBackgroundParameters = $PSCmdlet.MyInvocation.BoundParameters
                    Set-VisioPageBackground -Page $visioPage @setVisioPageBackgroundParameters
```

```
                } catch {
                  throw
                }
              }
            }
            Layout = @{
              Scope = 'Script'
              Value = {
                [CmdletBinding()]
                [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
                param(
                  [Parameter()]
                  [System.Management.Automation.SwitchParameter]
                  $ArrangeShapes,

                  [Parameter()]
                  [System.Management.Automation.SwitchParameter]
                  $CenterDrawing,

                  [Parameter()]
                  [System.Management.Automation.SwitchParameter]
                  $ShrinkToFit,

                  [Parameter()]
                  [System.Management.Automation.SwitchParameter]
                  $PassThru
                )
                try {
                  $formatVisioPageLayoutParameters = $PSCmdlet.MyInvocation.BoundParameters
                  Format-VisioPageLayout -Page $visioPage @formatVisioPageLayoutParameters
                } catch {
                  throw
                }
              }
            }
            Shape = @{
              Scope = 'Script'
              Value = {
                [CmdletBinding(DefaultParameterSetName='NewShape')]
                [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                param(
                  [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='ShapeObject')]
                  [ValidateNotNullOrEmpty()]
                  [Alias('InputObject')]
                  [Microsoft.Office.Interop.Visio.IVShape[]]
                  $Shape,

                  [Parameter(Position=0, Mandatory=$true, ParameterSetName='NewShape')]
                  [ValidateNotNullOrEmpty()]
                  [System.String[]]
                  $ShapeId,

                  [Parameter(Mandatory=$true, ParameterSetName='ByName')]
                  [System.Management.Automation.SwitchParameter]
                  $ByName,

                  [Parameter(Mandatory=$true, ParameterSetName='ByLabel')]
                  [System.Management.Automation.SwitchParameter]
                  $ByLabel,

                  [Parameter(Position=0, Mandatory=$true, ParameterSetName='ByName')]
                  [Parameter(ParameterSetName='NewShape')]
                  [ValidateNotNullOrEmpty()]
                  [System.String[]]
                  $Name,

                  [Parameter(Position=0, Mandatory=$true, ParameterSetName='ByLabel')]
                  [Parameter(ParameterSetName='NewShape')]
                  [ValidateNotNullOrEmpty()]
```

```
                    [Alias('Text')]
                    [System.String]
                    $Label,

                    [Parameter(Position=1, ParameterSetName='ShapeObject')]
                    [Parameter(Position=1, ParameterSetName='NewShape')]
                    [Parameter(Position=1, ParameterSetName='ByName')]
                    [Parameter(Position=1, ParameterSetName='ByLabel')]
                    [ValidateNotNullOrEmpty()]
                    [System.Management.Automation.ScriptBlock]
                    $ScriptBlock,

                    [Parameter(ParameterSetName='NewShape')]
                    [ValidateNotNullOrEmpty()]
                    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
                    [System.String]
                    $X,

                    [Parameter(ParameterSetName='NewShape')]
                    [ValidateNotNullOrEmpty()]
                    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
                    [System.String]
                    $Y,

                    [Parameter(ParameterSetName='NewShape')]
                    [System.Management.Automation.SwitchParameter]
                    $SmartLayout
                  )
                  begin {
                    try {
                      #region Initialize local variables.
                      [System.String]$private:parameterName = $null
                      #endregion

                      #region Convert the parameters to private scope.
                      foreach ($private:parameterName in
@('Shape','ShapeId','ByName','ByLabel','Name','Label','ScriptBlock','X','Y','SmartLayout')) {
                        if
($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
                          Set-Variable -Name $private:parameterName -Scope Local -Option Private
                        }
                      }
                      #endregion
                    } catch {
                      throw
                    }
                  }
                  process {
                    try {
                      $private:selectVisioShapeParameters = $PSCmdlet.MyInvocation.BoundParameters
                      if ($PSCmdlet.ParameterSetName -eq 'ShapeObject') {
                        Select-VisioShape @private:selectVisioShapeParameters
                      } else {
                        Select-VisioShape -Page $visioPage @private:selectVisioShapeParameters
                      }
                    } catch {
                      throw
                    }
                  }
                }
              }
            }
            #endregion
            #region Define nested modules in the DSV.
            Module = @{
              #region Define the Visio Shape DSV.
              Shape = @{
                #region Define functions in the Visio Shape DSV.
                Function = @{
                  #Shape = $script:visioCommandVocabulary.Page.Function.Shape
```

```
Name = @{
  Scope = 'Script'
  Value = {
    [CmdletBinding()]
    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
    param(
      [Parameter(Position=0, Mandatory=$true)]
      [ValidateNotNullOrEmpty()]
      [System.String]
      $NewName,

      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
    )
    try {
      $renameVisioShapeParameters = $PSCmdlet.MyInvocation.BoundParameters
      Rename-VisioShape -Shape $visioShape @renameVisioPageParameters
    } catch {
      throw
    }
  }
}
Label = @{
  Scope='Script'
  Value = {
    [CmdletBinding()]
    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
    param(
      [Parameter(Position=0, Mandatory=$true)]
      [ValidateNotNullOrEmpty()]
      [Alias('Text')]
      [System.String]
      $Label,

      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
    )
    try {
      $setVisioShapeLabelParameters = $PSCmdlet.MyInvocation.BoundParameters
      Set-VisioShapeLabel -Shape $visioShape @setVisioShapeLabelParameters
    } catch {
      throw
    }
  }
}
WebHyperlink = @{
  Scope='Script'
  Value = {
    [CmdletBinding()]
    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
    param(
      [Parameter(Position=0, Mandatory=$true)]
      [ValidateNotNullOrEmpty()]
      [System.String]
      $Url,

      [Parameter(Position=1)]
      [ValidateNotNullOrEmpty()]
      [System.String]
      $Description,

      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $Default,

      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
```

```
                    )
                    try {
                      $addVisioShapeWebHyperlinkParameters =
$PSCmdlet.MyInvocation.BoundParameters
                      Add-VisioShapeWebHyperlink -Shape $visioShape
@addVisioShapeWebHyperlinkParameters
                    } catch {
                      throw
                    }
                  }
                }
                FileHyperlink = @{
                  Scope = 'Script'
                  Value = {
                    [CmdletBinding(DefaultParameterSetName='Page')]
                    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                    param(
                      [Parameter(Position=0, Mandatory=$true, ParameterSetName='ExternalFile')]
                      [ValidateNotNullOrEmpty()]
                      [System.String]
                      $FileName,

                      [Parameter(ParameterSetName='Shape')]
                      [Parameter(Position=0, Mandatory=$true, ParameterSetName='Page')]
                      [Parameter(Position=1, Mandatory=$true, ParameterSetName='ExternalFile')]
                      [ValidateNotNullOrEmpty()]
                      [System.String]
                      $PageName,

                      [Parameter(Position=0, Mandatory=$true, ParameterSetName='Shape')]
                      [Parameter(Position=2, Mandatory=$true, ParameterSetName='ExternalFile')]
                      [ValidateNotNullOrEmpty()]
                      [System.String]
                      $ShapeName,

                      [Parameter()]
                      [ValidateNotNullOrEmpty()]
                      [System.String]
                      $Zoom,

                      [Parameter()]
                      [ValidateNotNullOrEmpty()]
                      [System.String]
                      $Description,

                      [Parameter()]
                      [System.Management.Automation.SwitchParameter]
                      $Default,

                      [Parameter()]
                      [System.Management.Automation.SwitchParameter]
                      $PassThru
                    )
                    try {
                      $addVisioShapeFileHyperlinkParameters =
$PSCmdlet.MyInvocation.BoundParameters
                      Add-VisioShapeFileHyperlink -Shape $visioShape
@addVisioShapeFileHyperlinkParameters
                    } catch {
                      throw
                    }
                  }
                }
                Move = @{
                  Scope = 'Script'
                  Value = {
                    [CmdletBinding()]
                    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                    param(
                      [Parameter(Position=0)]
```

```
      [ValidateNotNullOrEmpty()]
      [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
      [System.String]
      $X,

      [Parameter(Position=1)]
      [ValidateNotNullOrEmpty()]
      [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
      [System.String]
      $Y,

      [Parameter(Position=2)]
      [ValidateNotNullOrEmpty()]
      [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
      [System.String]
      $DeltaX,

      [Parameter(Position=3)]
      [ValidateNotNullOrEmpty()]
      [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
      [System.String]
      $DeltaY,

      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
    )
    try {
      $moveVisioShapeParameters = $PSCmdlet.MyInvocation.BoundParameters
      Move-VisioShape -Shape $visioShape @moveVisioShapeParameters
    } catch {
      throw
    }
  }
}
MoveToFront = @{
  Scope = 'Script'
  Value = {
    [CmdletBinding()]
    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
    param(
      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
    )
    try {
      $moveVisioShapeToFrontParameters = $PSCmdlet.MyInvocation.BoundParameters
      Move-VisioShapeToFront -Shape $visioShape @moveVisioShapeToFrontParameters
    } catch {
      throw
    }
  }
}
MoveToBack = @{
  Scope = 'Script'
  Value = {
    [CmdletBinding()]
    [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
    param(
      [Parameter()]
      [System.Management.Automation.SwitchParameter]
      $PassThru
    )
    try {
      $moveVisioShapeToBackParameters = $PSCmdlet.MyInvocation.BoundParameters
      Move-VisioShapeToBack -Shape $visioShape @moveVisioShapeToBackParameters
    } catch {
      throw
    }
  }
```

```
                    }
                    MoveForward = @{
                      Scope = 'Script'
                      Value = {
                        [CmdletBinding()]
                        [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                        param(
                          [Parameter()]
                          [System.Management.Automation.SwitchParameter]
                          $PassThru
                        )
                        try {
                          $moveVisioShapeForwardParameters = $PSCmdlet.MyInvocation.BoundParameters
                          Move-VisioShapeForward -Shape $visioShape @moveVisioShapeForwardParameters
                        } catch {
                          throw
                        }
                      }
                    }
                    MoveBackward = @{
                      Scope = 'Script'
                      Value = {
                        [CmdletBinding()]
                        [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                        param(
                          [Parameter()]
                          [System.Management.Automation.SwitchParameter]
                          $PassThru
                        )
                        try {
                          $moveVisioShapeBackwardParameters = $PSCmdlet.MyInvocation.BoundParameters
                          Move-VisioShapeBackward -Shape $visioShape
@moveVisioShapeBackwardParameters
                        } catch {
                          throw
                        }
                      }
                    }
                    Connect = @{
                      Scope = 'Script'
                      Value = {
                        [CmdletBinding(DefaultParameterSetName='Default')]
                        [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
                        param(
                          [Parameter(Position=0, Mandatory=$true)]
                          [ValidateNotNullOrEmpty()]
                          [Microsoft.Office.Interop.Visio.IVShape]
                          $ToShape,

                          [Parameter(Mandatory=$true, ParameterSetName='Top')]
                          [System.Management.Automation.SwitchParameter]
                          $Top,

                          [Parameter(Mandatory=$true, ParameterSetName='Bottom')]
                          [System.Management.Automation.SwitchParameter]
                          $Bottom,

                          [Parameter(Mandatory=$true, ParameterSetName='Left')]
                          [System.Management.Automation.SwitchParameter]
                          $Left,

                          [Parameter(Mandatory=$true, ParameterSetName='Right')]
                          [System.Management.Automation.SwitchParameter]
                          $Right,

                          [Parameter()]
                          [ValidateNotNullOrEmpty()]
                          [Microsoft.Office.Interop.Visio.IVShape]
                          $Connector,
```

```
                        [Parameter()]
                        [System.Management.Automation.SwitchParameter]
                        $SmartLayout,

                        [Parameter()]
                        [System.Management.Automation.SwitchParameter]
                        $PassThru
                      )
                      try {
                        $connectVisioShapeParameters = $PSCmdlet.MyInvocation.BoundParameters
                        Connect-VisioShape -Shape $visioShape @connectVisioShapeParameters
                      } catch {
                        throw
                      }
                    }
                  }
                }
                #endregion
              }
              #endregion
            }
            #endregion
          }
          #endregion
        }
        #endregion
      }
      #endregion
    }
    #endregion
  }
  #endregion
}

#endregion

#region Create Visio DSV nested modules.

function global:Initialize-Dsv {
  [CmdletBinding()]
  [OutputType([System.Management.Automation.ScriptBlock])]
  param(
    [Parameter(Position=0, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [System.Collections.Hashtable]
    $Dsv
  )
  foreach ($dsvKey in $Dsv.Keys) {
    New-Module -Name "Visio${dsvKey}DSV" -ArgumentList $Dsv.${dsvKey} -ScriptBlock {
      [CmdletBinding()]
      param(
        [Parameter(Position=0,Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [System.Collections.Hashtable]
        $DsvDefinition
      )

      #region Set the module export mode to explicit.
      Export-ModuleMember
      #endregion

      #region Initialize local variables.
      [System.String]$private:parameterName = $null
      #endregion

      #region Convert the parameters to private scope.
      Set-Variable -Name DsvDefinition -Scope Local -Option Private
      Set-Variable -Name ScriptBlock -Scope Local -Option Private
      #endregion
```

```
      #region Create aliases for DSV commands that can be used as defined.
      if ($private:DsvDefinition.ContainsKey('Alias')) {
        foreach ($private:aliasName in $private:DsvDefinition.Alias.Keys) {
          $private:newAliasParameters = $private:DsvDefinition.Alias[$private:aliasName]
          New-Alias -Name $private:aliasName @private:newAliasParameters -Force
        }
      }
      #endregion

      #region Create functions for DSV commands that require additional logic.
      if ($private:DsvDefinition.ContainsKey('Function')) {
        foreach ($private:functionName in $private:DsvDefinition.Function.Keys) {
          $private:scope = $private:DsvDefinition.Function[$private:functionName].Scope
          $private:scriptBlock = $private:DsvDefinition.Function[$private:functionName].Value
          New-Item -Path Function: -Name "${private:scope}:${private:functionName}" -Value
$private:scriptBlock -Force
        }
      }
      #endregion

      #region Create nested DSV modules.
      if ($private:DsvDefinition.ContainsKey('Module')) {
        foreach ($private:moduleName in $private:DsvDefinition.Module.Keys) {
          Initialize-Dsv -Dsv $private:DsvDefinition.Module
        }
      }
      #endregion
    } | Import-Module -PassThru
  }
}

Initialize-Dsv -Dsv $script:visioCommandVocabulary

Remove-Item -LiteralPath Function:\Initialize-Dsv

#New-Module -Name VisioApplicationDSV -ArgumentList $script:visioCommandVocabulary.Application -
ScriptBlock $defineDsvScriptBlock | Import-Module
#& (Get-Module -Name VisioApplicationDSV) {New-Module -Name VisioDocumentDSV -ArgumentList
$script:visioCommandVocabulary.Document -ScriptBlock $defineDsvScriptBlock | Import-Module}
#& (Get-Module -Name VisioApplicationDSV) {& (Get-Module -Name VisioDocumentDSV) {New-Module -Name
VisioPageDSV -ArgumentList $script:visioCommandVocabulary.Page -ScriptBlock $defineDsvScriptBlock |
Import-Module}}
#& (Get-Module -Name VisioApplicationDSV) {& (Get-Module -Name VisioDocumentDSV) {& (Get-Module -
Name VisioPageDSV) {New-Module -Name VisioShapeDSV -ArgumentList
$script:visioCommandVocabulary.Shape -ScriptBlock $defineDsvScriptBlock | Import-Module}}}

#endregion

#region Initialize helper variables.

$script:VisioTemplateCache = @{}

#endregion

#region VisioTemplateCache helper functions.

function Update-VisioTemplateCache {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
```

```
  process {
    try {
      foreach ($item in $Document) {
        Write-Progress -Activity 'Updating template cache' -Status "Checking for templates in
""$($item.Name)"" that are not cached."
        if ($script:VisioTemplateCache.Keys -notcontains $item.Application.ProcessId) {
          $script:VisioTemplateCache[$item.Application.ProcessId] = @{
             Stencils = [String[]]@()
            Templates = New-Object -TypeName 'System.Collections.Generic.Dictionary[System.String,
Microsoft.Office.Interop.Visio.IVMaster]' -ArgumentList
@([System.StringComparer]::CurrentCultureIgnoreCase)
          }
        }
        if ($visioDocumentWindow = Get-VisioDocumentWindow -Document $item) {
          [String[]]$dockedStencils = @()
          $visioDocumentWindow.DockedStencils([REF]$dockedStencils)
          if ($dockedStencils) {
            foreach ($stencilName in $dockedStencils) {
              if ($script:VisioTemplateCache[$item.Application.ProcessId].Stencils -contains
$stencilName) {
                continue
              }
              $script:VisioTemplateCache[$item.Application.ProcessId].Stencils += $stencilName
              foreach ($visioTemplate in $item.Application.Documents.Item($stencilName).Masters) {
                Write-Progress -Activity 'Updating template cache' -Status "Adding template
'$($visioTemplate.Name)' ($($visioTemplate.NameU)) to the cache."
                $visioTemplate =
[System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioTemplate,
[Microsoft.Office.Interop.Visio.MasterClass])

$script:VisioTemplateCache[$item.Application.ProcessId].Templates[$visioTemplate.Name]  =
[Microsoft.Office.Interop.Visio.IVMaster]$visioTemplate

$script:VisioTemplateCache[$item.Application.ProcessId].Templates[$visioTemplate.NameU] =
[Microsoft.Office.Interop.Visio.IVMaster]$visioTemplate
              }
            }
          }
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $item
        }
      }
    } catch {
      throw
    }
  }
}

#endregion

#region Visio functions.

function Start-Visio {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
  param(
    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Invisible
  )
  try {
    if ((-not $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Invisible')) -or (-not
$Invisible)) {
      Write-Progress -Activity 'Starting Visio' -Status 'Opening new instance of Visio
application.'
      New-Object -ComObject 'Visio.Application'
    } else {
      Write-Progress -Activity 'Starting Visio' -Status 'Opening new instance of Visio
application.'
```

```
      New-Object -ComObject 'Visio.InvisibleApp'
    }
  } catch {
    throw
  }
}
Export-ModuleMember -Function Start-Visio

function Stop-Visio {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Force
  )
  process {
    try {
      #region Initialize local variables.
      [Microsoft.Office.Interop.Visio.IVDocument]$visioDocument = $null
      [bool]$dirtyBit = $false
      #endregion

      foreach ($item in $Application) {
        #region Close any open documents (by Force, if specified).
        foreach ($visioDocument in $item.Documents) {
          if ((-not $visioDocument.Saved) -and (-not $Force)) {
            $dirtyBit = $true
          } else {
            $visioDocument.Saved = $true
            $visioDocument.Close()
          }
        }
        #endregion

        #region Return an error if any documents were unsaved and Force was not used.
        if ($dirtyBit) {
          Write-Error 'Cannot close Visio because one or more documents it contains have not been
saved. If you want to close it anyway, use the -Force to close it.'
          return
        }
        #endregion

        #region Close the Visio application.
        $item.Quit()
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Stop-Visio

function Show-Visio {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,
```

```
    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        #region Show the Visio application.
        $visioApplication.Visible = $true
        #endregion

        #region Return the Visio application object if requested.
        if ($PassThru) {
          $visioApplication
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Show-Visio

function Hide-Visio {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        #region Show the Visio application.
        $visioApplication.Visible = $false
        #endregion

        #region Return the Visio application object if requested.
        if ($PassThru) {
          $visioApplication
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Hide-Visio

function Use-Visio {
  [CmdletBinding(DefaultParameterSetName='Default')]
  [OutputType([Microsoft.Office.Interop.Visio.IVApplication])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='Application')]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter(Position=0, ParameterSetName='Default')]
```

```
    [Parameter(Position=1, Mandatory=$true, ParameterSetName='Application')]
    [ValidateNotNullOrEmpty()]
    [System.Management.Automation.ScriptBlock]
    $ScriptBlock,

    [Parameter(ParameterSetName='Default')]
    [System.Management.Automation.SwitchParameter]
    $Invisible,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $AutoExit,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Force,

    [Parameter(ParameterSetName='Application')]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [Microsoft.Office.Interop.Visio.IVApplication]$visioApplication = $null
      [System.String]$private:parameterName = $null
      #endregion

      #region Convert the parameters to private scope.
      foreach ($private:parameterName in
@('ScriptBlock','Application','Invisible','AutoExit','Force','PassThru')) {
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
          Set-Variable -Name $private:parameterName -Scope Local -Option Private
        }
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      switch ($PSCmdlet.ParameterSetName) {
        'Default' {
          #region If no Visio applications were passed in from the pipeline, create one and pass it
in.
          $private:startVisioParameters = @{}
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Invisible') -and $Invisible) {
            $private:startVisioParameters['Invisible'] = $Invisible
          }
          $private:useVisioParameters = @{}
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
            $private:useVisioParameters['ScriptBlock'] = $ScriptBlock
          }
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('AutoExit') -and $AutoExit) {
            $private:useVisioParameters['AutoExit'] = $AutoExit
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Force') -and $Force) {
              $private:useVisioParameters['Force'] = $Force
            }
          }
          if ($private:useVisioParameters.Count) {
            Start-Visio @private:startVisioParameters | Use-Visio @private:useVisioParameters -
PassThru
          } else {
            Start-Visio @private:startVisioParameters
          }
          #endregion
          break
        }
        'Application' {
```

```
          #region If we have a Visio application, invoke our DSV script and process optional
parameters.
          foreach ($visioApplication in $Application) {
            #region Invoke the ScriptBlock.
            & (Get-Module -Name VisioApplicationDSV) $ScriptBlock | Out-Null
            #endregion

            #region If the -AutoExit switch was used, close Visio and return.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('AutoExit') -and $AutoExit) {
              Write-Progress -Activity 'Shutting down Visio' -Status 'Closing the Visio
application.'
              $private:stopVisioParameters = @{
                Application = $visioApplication
              }
              if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Force') -and $Force) {
                $private:stopVisioParameters['Force'] = $true
              }
              Stop-Visio @private:stopVisioParameters
              continue
            }
            #endregion

            #region Return the Visio application object to the client if requested.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
              $visioApplication
            }
            #endregion
          }
          #endregion
          break
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Use-Visio

New-Alias -Name Visio -Value Use-Visio -Description 'Starts Visio and allows you to interact with
Visio documents using a PowerShell-driven domain-specific vocabulary (DSV).' -ErrorAction
SilentlyContinue
if ($?) {
  Export-ModuleMember -Alias Visio
}

#endregion

#region VisioShapesPath functions.

function Get-VisioShapesPath {
  [CmdletBinding()]
  [OutputType([System.String])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        $myShapesPath = $null
        if ($visioApplication | Get-Member -Name MyShapesPath -ErrorAction SilentlyContinue) {
          $myShapesPath = $visioApplication.MyShapesPath
        }
        if ((-not $myShapesPath) -and
([System.Reflection.Assembly]::GetAssembly('Microsoft.Office.Interop.Visio.IVDocument').FullName -
```

```
eq 'Microsoft.Office.Interop.Visio, Version=11.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c')) {
          $myShapesPath = (Get-ItemProperty -Path
'Registry::HKEY_CURRENT_USER\Software\Microsoft\Office\11.0\Visio\Application' -ErrorAction
SilentlyContinue).MyShapesPath
        }
        if ($myShapesPath) {
          $myShapesPath
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Get-VisioShapesPath

#endregion

#region VisioDefault functions.

function Get-VisioDefaultConfiguration {
  [CmdletBinding()]
  [OutputType([System.Collections.Hashtable])]
  param()
  try {
    #region Create the Visio Default Configuration store if it does not exist.
    if (-not ($visioDefaultConfiguration = Get-Variable -Scope Script -Name
VisioDefaultConfiguration -ErrorAction SilentlyContinue)) {
      $visioDefaultConfiguration = New-Variable -Scope Script -Name VisioDefaultConfiguration -
Option Private -Value @{} -PassThru
      $visioDefaultConfiguration.Value['SmartLayout'] = $true
      $visioDefaultConfiguration.Value['Page'       ] = @{}
      $visioDefaultConfiguration.Value.Page['Background'           ] = $null
      $visioDefaultConfiguration.Value.Page['Colors'               ] = 'None'
      $visioDefaultConfiguration.Value.Page['Effects'              ] = 'None'
      $visioDefaultConfiguration.Value.Page['AutoEnlarge'          ] = $false
      $visioDefaultConfiguration.Value.Page['DiagramStyle'         ] = 'Default'
      $visioDefaultConfiguration.Value.Page['DiagramDirection'     ] = $null
      $visioDefaultConfiguration.Value.Page['DiagramAlignment'     ] = $null
      $visioDefaultConfiguration.Value.Page['DiagramPlacementDepth'] = 'Default'
      $visioDefaultConfiguration.Value.Page['ConnectorStyle'       ] = 'Default'
      $visioDefaultConfiguration.Value.Page['ConnectorSpacing'     ] = '7.5mm'
      $visioDefaultConfiguration.Value.Page['ConnectorCurvature'   ] = 'Default'
    }
    #endregion
    $visioDefaultConfiguration.Value
  } catch {
    throw
  }
}
Export-ModuleMember -Function Get-VisioDefaultConfiguration

function Set-VisioDefaultPageBackground {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0)]
    [System.String]
    $Background
  )
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default page background.
    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Background') -and ($Background.Trim()))
{
      $visioDefaultConfiguration.Page.Background = $Background.Trim()
```

```
    } else {
      $visioDefaultConfiguration.Page.Background = 'None'
    }
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Set-VisioDefaultPageBackground

function Set-VisioDefaultPageTheme {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0)]
    [System.String]
    $Colors,

    [Parameter(Position=1)]
    [System.String]
    $Effects
  )
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default page theme.
    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Colors') -and ($newColors =
$Colors.Trim() -replace '[^a-z]')) {
      $visioDefaultConfiguration.Page.Colors = $newColors
    } else {
      $visioDefaultConfiguration.Page.Colors = 'None'
    }
    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Effects') -and ($newEffects =
$Effects.Trim() -replace '[^a-z]')) {
      $visioDefaultConfiguration.Page.Effects = $newEffects
    } else {
      $visioDefaultConfiguration.Page.Effects = 'None'
    }
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Set-VisioDefaultPageTheme

function Enable-VisioDefaultPageAutoEnlarge {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param()
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default page auto enlarge flag.
    $visioDefaultConfiguration.Page.AutoEnlarge = $true
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Enable-VisioDefaultPageAutoEnlarge

function Disable-VisioDefaultPageAutoEnlarge {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param()
  try {
```

```
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default page auto enlarge flag.
    $visioDefaultConfiguration.Page.AutoEnlarge = $false
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Disable-VisioDefaultPageAutoEnlarge

function Set-VisioDefaultPageDiagramStyle {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0, Mandatory=$true)]
    [ValidateScript({@('Default','Radial','Flowchart','Circular','CompactTree','Hierarchy') -
contains ($_ -replace ' ')})]
    [System.String]
    $Style,

    [Parameter(Position=1)]

[ValidateScript({@('DownThenRight','RightThenDown','RightThenUp','UpThenRight','UpThenLeft','LeftTh
enUp','LeftThenDown','DownThenLeft','TopToBottom','BottomToTop','LeftToRight','RightToLeft') -
contains ($_ -replace ' ')})]
    [System.String]
    $Direction,

    [Parameter(Position=2)]
    [ValidateSet('Left','Center','Right','Top','Middle','Bottom')]
    [System.String]
    $Alignment,

    [Parameter(Position=3)]
    [ValidateSet('Default','Shallow','Medium','Deep')]
    [System.String]
    $PlacementDepth
  )
  try {
    #region Initialize local variables.
    [System.String[]]$validDirections = @()
    [System.String[]]$validAlignments = @()
    [System.String]$internalStyle = $null
    [System.String]$internalDirection = $null
    [System.String]$internalAlignment = $null
    #endregion

    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Pre-process the parameters.
    $internalStyle = $Style -replace ' '
    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Direction') -and $Direction) {
      $internalDirection = $Direction -replace ' '
    }
    if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Alignment') -and $Alignment) {
      $internalAlignment = $Alignment
    }
    if ((@('Default','Radial','Circular') -contains $internalStyle) -and ($internalDirection -or
$internalAlignment)) {
      Write-Warning "'Direction' and 'Alignment' are not applicable to '$Style' diagrams. These
parameters will be ignored."
      $internalDirection = $null
      $internalAlignment = $null
    } elseif (@('Flowchart','CompactTree') -contains $internalStyle) {
      if ($internalAlignment) {
```

```
        Write-Warning "'Alignment' is not applicable to '$Style' diagrams. This parameter will be
ignored."
        $internalAlignment = $null
      }
      if (-not $internalDirection) {
        switch -exact ($internalStyle) {
          'Flowchart'   {$internalDirection = 'TopToBottom';   break}
          'CompactTree' {$internalDirection = 'DownThenRight'; break}
        }
      }
    } elseif ($internalStyle -eq 'Hierarchy') {
      if (-not $internalDirection) {$internalDirection = 'TopToBottom'}
      if (-not $internalAlignment) {$internalAlignment = 'Center'}
    }
    if ($internalDirection) {
      if ($internalStyle -eq 'CompactTree') {
        $validDirections =
@('DownThenRight','RightThenDown','RightThenUp','UpThenRight','UpThenLeft','LeftThenUp','LeftThenDo
wn','DownThenLeft')
      } else {
        $validDirections = @('TopToBottom','BottomToTop','LeftToRight','RightToLeft')
      }
      if ($validDirections -notcontains $internalDirection) {
        Write-Error "Cannot bind value 'Direction' to the target. The value was invalid. Valid
values for 'Direction' include '$([string]::Join(""', '"",$validDirections))'."
        return
      }
    }
    if ($internalAlignment) {
      if (@('TopToBottom','BottomToTop') -match $internalDirection) {
        $validAlignments = @('Left','Center','Right')
      } else {
        $validAlignments = @('Top','Middle','Bottom')
      }
      if ($validAlignments -notcontains $internalAlignment) {
        Write-Error "Cannot bind value 'Alignment' to the target. The value was invalid. Valid
values for 'Alignment' in the '$internalDirection' direction include '$([string]::Join(""',
'"",$validAlignments))'."
        return
      }
    }
    #endregion

    #region Set the default page diagram style.
    $visioDefaultConfiguration.Page.DiagramStyle = $internalStyle
    $visioDefaultConfiguration.Page.DiagramDirection = $internalDirection
    $visioDefaultConfiguration.Page.DiagramAlignment = $internalAlignment
    $visioDefaultConfiguration.Page.DiagramPlacementDepth = $PlacementDepth
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Set-VisioDefaultPageDiagramStyle

function Set-VisioDefaultPageConnectorStyle {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0)]

[ValidateScript({@('Default','RightAngle','Straight','CenterToCenter','Network','Flowchart','Tree',
'OrganizationChart','Simple','SimpleHorizontalVertical','SimpleVerticalHorizontal') -contains ($_ -
replace ' ')})]
    [System.String]
    $Style,

    [Parameter(Position=1)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
```

```
    [System.String]
    $Spacing,

    [Parameter(Position=2)]
    [ValidateSet('Default','Straight','Curved')]
    [System.String]
    $Curvature
  )
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default page diagram style.
    $visioDefaultConfiguration.Page.ConnectorStyle = $Style -replace ' '
    $visioDefaultConfiguration.Page.ConnectorSpacing = $Spacing
    $visioDefaultConfiguration.Page.ConnectorCurvature = $Curvature
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Set-VisioDefaultPageConnectorStyle

function Enable-VisioSmartLayout {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param()
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default smart layout flag.
    $visioDefaultConfiguration.SmartLayout = $true
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Enable-VisioSmartLayout

function Disable-VisioSmartLayout {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param()
  try {
    #region Lookup the current Visio default configuration.
    $visioDefaultConfiguration = Get-VisioDefaultConfiguration
    #endregion

    #region Set the default smart layout flag.
    $visioDefaultConfiguration.SmartLayout = $false
    #endregion
  } catch {
    throw
  }
}
Export-ModuleMember -Function Disable-VisioSmartLayout

#endregion

#region VisioDocument functions

function Get-VisioDocument {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
```

```
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter(Position=0)]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $DocumentId
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        #region Get the requested documents.
        if ($visioApplication.Documents) {
          foreach ($visioDocument in $visioApplication.Documents) {
            if ($visioDocument.Type -eq
[Microsoft.Office.Interop.Visio.VisDocumentTypes]::visTypeDrawing) {
              if ((-not $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('DocumentId')) -or
                  ($DocumentId -contains $visioDocument.Name) -or
                  ($DocumentId -contains ($visioDocument.Name -replace '\.vsd$')) -or
                ($DocumentId -contains $visioDocument.FullName) -or
                ($DocumentId -contains ($visioDocument.FullName -replace
"^$([System.Text.RegularExpressions.Regex]::Escape([System.Environment]::GetFolderPath('MyDocuments
')))\\"))) {
                  $visioDocument
                }
              }
            }
          }
          #endregion
        }
      } catch {
        throw
      }
    }
}
Export-ModuleMember -Function Get-VisioDocument

function New-VisioDocument {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter(Position=1)]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $TemplateId
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('TemplateId')) {
          #region Add a new document that uses a specific template.
          if ($TemplateId -notmatch '\.vst$') {
            $visioDocument = $visioApplication.Documents.Add("$TemplateId.vst")
          } else {
            $visioDocument = $visioApplication.Documents.Add($TemplateId)
          }
          #endregion
        } else {
          #region Add a new blank document.
          $visioDocument = $visioApplication.Documents.Add('Basic Diagram.vst')
          #endregion
        }
        #region Process the new document and return it to the caller.
```

```
        if ($visioDocument) {
          Select-VisioStencil -Document $visioDocument -Default
          Update-VisioTemplateCache -Document $visioDocument
          foreach ($visioPage in $visioDocument.Pages) {
            $visioPage = [System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioPage,
[Microsoft.Office.Interop.Visio.PageClass])
            Reset-VisioPageConfiguration -Page $visioPage
          }
          $visioDocument
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function New-VisioDocument

function Open-VisioDocument {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter(Position=1, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $DocumentId
  )
  process {
    try {
      foreach ($visioApplication in $Application) {
        foreach ($item in $DocumentId) {
          if ($visioDocument = Get-VisioDocument -Application $visioApplication -DocumentId $item)
{
            #region If the document is open already, return it.
            $visioDocument
            #endregion
          } else {
            #region If the path is not relative or absolute, resolve it using the MyDocuments
folder.
            $path = $item
            if ($item -notmatch '^(\.\.?\\|\\\\|[a-z]:)') {
              $path = Join-Path -Path [System.Environment]::GetFolderPath('MyDocuments') -ChildPath
$path
              if ($path -notmatch '\.vsd$') {
                $path += '.vsd'
              }
            }
            #endregion

            #region Now get all match files in case there is a wildcard character and open them.
            foreach ($file in Get-Item -Path $path) {
              if ($visioDocument = $visioApplication.Documents.OpenEx($file.FullName, 36)) {
                Select-VisioStencil -Document $visioDocument -Default
                Update-VisioTemplateCache -Document $visioDocument -PassThru
              }
            }
            #endregion
          }
        }
      }
    } catch {
      throw
    }
```

```
  }
}
Export-ModuleMember -Function Open-VisioDocument

function Save-VisioDocument {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument]
    $Document,

    [Parameter(Position=1, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNullOrEmpty()]
    [ValidateScript({Test-Path -LiteralPath $_ -IsValid})]
    [Alias('FullName')]
    [Alias('As')]
    [System.String]
    $Path,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Force,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      #region Qualify the Path parameter as required.
      if ($Path -notmatch '^(\.\.?\\|\\\\|[a-z]:)') {
        $Path = Join-Path -Path [System.Environment]::GetFolderPath('MyDocuments') -ChildPath $Path
        if ($Path -notmatch '\.vs[std]$') {
          $Path += '.vsd'
        }
      }
      #endregion

      #region Save the Visio document and return it if requested.
      $fileExists = Test-Path -LiteralPath $Path -ErrorAction SilentlyContinue
      if (($Path -eq $Document.FullName) -and $fileExists) {
        Write-Progress -Activity 'Saving Visio document' -Status "Saving the
""$($Document.FullName)"" Visio document."
        $Document.Save() | Out-Null
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $Document
        }
      } else {
        Write-Progress -Activity 'Saving Visio document' -Status "Saving the
""$($Document.FullName)"" Visio document as ""$Path""."
        if ($fileExists -and -not $Force) {
          Write-Error "The file name ""$Path"" already exists. To overwrite the file, you must use
the -Force. The document ""$($Document.Name)"" has not been saved."
        } else {
          if ($fileExists) {
            Remove-Item -LiteralPath $Path -Force
          }
          $Document.SaveAs($Path) | Out-Null
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
            $Document
          }
        }
      }
      #endregion
    } catch {
      throw
    }
```

```
  }
}
Export-ModuleMember -Function Save-VisioDocument

function Close-VisioDocument {
  [CmdletBinding()]
  [OutputType([System.Void])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Force
  )
  process {
    try {
      foreach ($visioDocument in $Document) {
        #region Close the Visio document.
        Write-Progress -Activity 'Closing Visio document' -Status "Closing the
'$($visioDocument.FullName)' Visio document."
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Force') -and $Force) {
          $visioDocument.Saved = $true
        }
        $visioDocument.Close()
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Close-VisioDocument

# Syntax:
#     Select-VisioDocument [-Application] <Microsoft.Office.Interop.Visio.IVApplication> [[-
ScriptBlock] <ScriptBlock>]
#     Select-VisioDocument [-Document] <Microsoft.Office.Interop.Visio.IVDocument> [[-ScriptBlock]
<ScriptBlock>]
#     Select-VisioDocument [-Application] <Microsoft.Office.Interop.Visio.IVApplication> [-
DocumentId[]] <string> [[-ScriptBlock] <ScriptBlock>]
function Select-VisioDocument {
  [CmdletBinding(DefaultParameterSetName='UnnamedDocument')]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='DocumentObject')]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='UnnamedDocument')]
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='NamedDocument')]
    [ValidateNotNullOrEmpty()]
    [Microsoft.Office.Interop.Visio.IVApplication[]]
    $Application,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='NamedDocument')]
    [ValidateNotNullOrEmpty()]
    [Alias('Name')]
    [System.String[]]
    $DocumentId,

    [Parameter(Position=1, ParameterSetName='UnnamedDocument')]
```

```
    [Parameter(Position=1, ParameterSetName='DocumentObject')]
    [Parameter(Position=2, ParameterSetName='NamedDocument')]
    [ValidateNotNullOrEmpty()]
    [System.Management.Automation.ScriptBlock]
    $ScriptBlock
  )
  begin {
    try {
      #region Initialize local variables.
      [System.String]$private:parameterName = $null
      #endregion

      #region Convert the parameters to private scope.
      foreach ($private:parameterName in @('Document','Application','DocumentId','ScriptBlock')) {
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
          Set-Variable -Name $private:parameterName -Scope Local -Option Private
        }
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      switch ($PSCmdlet.ParameterSetName) {
        'UnnamedDocument' {
          #region If a new document was requested, create one and pass it in.
          $private:newVisioDocumentParameters = @{
            Application = $Application
          }
          $private:selectVisioDocumentParameters = @{}
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
            $private:selectVisioDocumentParameters['ScriptBlock'] = $ScriptBlock
          }
          New-VisioDocument @private:newVisioDocumentParameters | Select-VisioDocument
@private:selectVisioDocumentParameters
          #endregion
          break
        }
        'NamedDocument' {
          #region If a specific document was requested, open it if it's not open and pass it in.
          $private:openVisioDocumentParameters = @{
            Application = $Application
             DocumentId = $DocumentId
          }
          $private:selectVisioDocumentParameters = @{}
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
            $private:selectVisioDocumentParameters['ScriptBlock'] = $ScriptBlock
          }
          Open-VisioDocument @private:openVisioDocumentParameters | Select-VisioDocument
@private:selectVisioDocumentParameters
          #endregion
          break
        }
        'DocumentObject' {
          #region Process the document object.
          foreach ($visioDocument in $Document) {
            #region Activate the Visio document window if necessary.
            if ($visioDocument -ne $visioDocument.Application.ActiveDocument) {
              $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioDocument
              $visioDocumentWindow.Activate()
            } else {
              $visioDocumentWindow = $visioDocument.Application.ActiveWindow
            }
            #endregion

            #region Invoke the ScriptBlock.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
              & (Get-Module -Name VisioDocumentDSV) $ScriptBlock | Out-Null
```

```
            }
            #endregion

            #region Return the Visio document object to the client.
            $visioDocument
            #endregion
          }
          #endregion
          break
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Select-VisioDocument

#endregion

#region VisioDocumentWindow functions

function Get-VisioDocumentWindow {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVWindow])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document
  )
  process {
    try {
      foreach ($visioDocument in $Document) {
        Write-Progress -Activity 'Searching for document window' -Status "Searching for the
document window associated with the Visio document called ""$($visioDocument.Name)""."
        $visioDocumentWindow = $null
        if ($Document -eq $visioDocument.Application.ActiveDocument) {
          $visioDocumentWindow = $visioDocument.Application.ActiveWindow
        } else {
          foreach ($window in $visioDocument.Application.Windows) {
            if ($window.Document -eq $visioDocument) {
              $visioDocumentWindow = $window
              break
            }
          }
        }
        if ($visioDocumentWindow) {
          $visioDocumentWindow
        } else {
          Write-Error "A window associated with the Visio document called
""$($visioDocument.Name)"" could not be found."
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Get-VisioDocumentWindow

#endregion

#region VisioStencil functions

function Select-VisioStencil {
  [CmdletBinding(DefaultParameterSetName='Id')]
  [OutputType([Microsoft.Office.Interop.Visio.IVDocument])]
  param(
```

```
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='Id')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $StencilId,

    [Parameter(Mandatory=$true, ParameterSetName='Default')]
    [System.Management.Automation.SwitchParameter]
    $Default,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioDocument in $Document) {
        #region Activate the Visio document window if necessary.
        if ($visioDocument -ne $visioApplication.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioDocument
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioApplication.ActiveWindow
        }
        #endregion

        #region Load the Visio stencil(s) into the Visio document.
        switch ($PSCmdlet.ParameterSetName) {
          'Id' {
            foreach ($item in $StencilId) {
              if ($item -notmatch '\.vss$') {
                $item += '.vss'
              }
              $myShapesPath = Get-VisioShapesPath -Application $visioDocument.Application
              if (($item -notmatch '\\\\') -and (-not (Split-Path -Path $item -Qualifier -
ErrorAction SilentlyContinue)) -and $myShapesPath) {
                $item = Join-Path -Path $myShapesPath -ChildPath $item
              }
              Write-Progress -Activity 'Loading Visio stencil' -Status "Loading the Visio stencil
'$($item)' into the Visio document."
              $visioDocument.Application.Documents.OpenEx($item, 4) | Out-Null
            }
            break
          }
          default {
              if (@('Microsoft.Office.Interop.Visio, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c','Microsoft.Office.Interop.Visio, Version=11.0.0.0,
Culture=neutral, PublicKeyToken=71e9bce111e9429c') -notcontains
[System.Reflection.Assembly]::GetAssembly('Microsoft.Office.Interop.Visio.IVPage')) {
              #region Load the background and borders and tiles stencils into the document.
              foreach ($item in @('BCKGRN_M.VSS','BORDRS_M.VSS')) {
                Write-Progress -Activity 'Loading Visio stencil' -Status "Loading the Visio stencil
'$($item)' into the Visio document."
                $visioDocument.Application.Documents.OpenEx($item, 4) | Out-Null
              }
              #endregion
            }
            break
          }
        }
        #endregion

        #region Update the template cache when loading additional stencils.
        Update-VisioTemplateCache -Document $visioDocument
        #endregion
```

```
        #region Return the Visio document object if requested.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $visioDocument
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Select-VisioStencil

#endregion

#region VisioPage functions

function Get-VisioPage {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter(Position=1, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('Name')]
    [System.String[]]
    $PageId,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Force
  )
  process {
    try {
      foreach ($visioDocument in $Document) {
        foreach ($item in $PageId) {
          $found = $false
          if ($item -match '^\d+$') {
            if ($visioDocument.Pages.Count -ge $item) {
              $found = $true
              $visioPage = $visioDocument.Pages.ItemU([int]$item)
              $visioPage =
[System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioPage,
[Microsoft.Office.Interop.Visio.PageClass])
                [Microsoft.Office.Interop.Visio.IVPage]$visioPage

            }
          } else {
            foreach ($visioPage in $visioDocument.Pages) {
              if (($visioPage.Name -like $item) -or ($visioPage.NameU -like $item)) {
                $found = $true
                $visioPage =
[System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioPage,
[Microsoft.Office.Interop.Visio.PageClass])
                  [Microsoft.Office.Interop.Visio.IVPage]$visioPage
              }
            }
          }
          if (-not $found) {
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Force') -and $Force) {
              New-VisioPage -Document $visioDocument -PageId $item
            } else {
              Write-Error "Page $item not found."
            }
```

```
          }
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Get-VisioPage

function New-VisioPage {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter(Position=1)]
    [ValidateNotNullOrEmpty()]
    [Alias('Name')]
    [System.String[]]
    $PageId
  )
  process {
    try {
      foreach ($visioDocument in $Document) {
        if (-not $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PageId')) {
          #region Return a new page.
          Write-Progress -Activity 'Adding page to Visio document' -Status 'Adding page to the
Visio document.'
          $visioDocument.Pages.Add() | Reset-VisioPageConfiguration -PassThru
          #endregion
        } else {
          foreach ($item in $PageId) {
            if ($item -match '^\d+$') {
              #region Create as many new pages as necessary and return the one requested.
              for ($index -eq 1; $index -le $item; $index++) {
                $pageCount = $visioDocument.Pages.Count
                foreach ($visioPage in $visioDocument.Pages) {
                  if ($visioPage.Type -ne
[Microsoft.Office.Interop.Visio.VisPageTypes]::visTypeForeground) {
                    $pageCount--
                  }
                }
                if ($index -gt $pageCount) {
                  Write-Progress -Activity 'Adding page to Visio document' -Status "Adding page
$index to the Visio document."
                  $visioPage = $visioDocument.Pages.Add() | Reset-VisioPageConfiguration -PassThru
                  if ($index -eq $item) {
                    $visioPage
                  }
                }
              }
              #endregion
            } else {
              #region Create a new page and set its name to the one provided.
              Write-Progress -Activity 'Adding page to Visio document' -Status 'Adding page to the
Visio document.'
              $visioDocument.Pages.Add() | Reset-VisioPageConfiguration -PassThru | Rename-
VisioPage -NewName ($item -replace '(^\s+|\s+$)')
              #endregion
            }
          }
        }
      }
    } catch {
      throw
```

```
    }
  }
}
Export-ModuleMember -Function New-VisioPage

function Rename-VisioPage {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $NewName,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Rename the Visio page.
        $visioPage.Name = $NewName -replace '(^\s+|\s+$)'
        #endregion

        #region Return the Visio page object if requested.
        if ($PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Rename-VisioPage

function Select-VisioPage {
  [CmdletBinding(DefaultParameterSetName='NamedPage')]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='PageObject')]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,
```

```
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true, ParameterSetName='NamedPage')]
    [ValidateNotNullOrEmpty()]
    [Microsoft.Office.Interop.Visio.IVDocument[]]
    $Document,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='NamedPage')]
    [ValidateNotNullOrEmpty()]
    [Alias('Name')]
    [System.String[]]
    $PageId,

    [Parameter(Position=1, ParameterSetName='PageObject')]
    [Parameter(Position=2, ParameterSetName='NamedPage')]
    [ValidateNotNullOrEmpty()]
    [System.Management.Automation.ScriptBlock]
    $ScriptBlock
  )
  begin {
    try {
      #region Initialize local variables.
      [System.String]$private:parameterName = $null
      #endregion

      #region Convert the parameters to private scope.
      foreach ($private:parameterName in @('Page','Document','PageId','ScriptBlock')) {
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
          Set-Variable -Name $private:parameterName -Scope Local -Option Private
        }
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      switch ($PSCmdlet.ParameterSetName) {
        'NamedPage' {
          #region If a specific page was requested, get it and pass it in.
          $private:getVisioPageParameters = @{
            Document = $Document
              PageId = $PageId
               Force = $true
          }
          $private:selectVisioPageParameters = @{}
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
            $private:selectVisioPageParameters['ScriptBlock'] = $ScriptBlock
          }
          Get-VisioPage @private:getVisioPageParameters | Select-VisioPage
@private:selectVisioPageParameters
          #endregion
          break
        }
        'PageObject' {
          #region Process the page object.
          foreach ($visioPage in $Page) {
            #region Activate the Visio document window if necessary.
            if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
              $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
              $visioDocumentWindow.Activate()
            } else {
              $visioDocumentWindow = $visioPage.Application.ActiveWindow
            }
            #endregion

            #region Activate the Visio page if necessary.
            if ($visioDocumentWindow.Page -ne $visioPage) {
              $visioDocumentWindow.Page = $visioPage
            }
            #endregion
```

```
            #region Invoke the ScriptBlock.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
              & (Get-Module -Name VisioPageDSV) $ScriptBlock | Out-Null
            }
            #endregion

            #region Return the Visio page object to the client.
            $visioPage
            #endregion
          }
          #endregion
          break
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Select-VisioPage

function Enable-VisioPageAutoEnlarge {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the auto enlarge property on the Visio page.
        $visioPage.PageSheet.CellsU('ResizePage').ResultIU = 1
        #endregion

        #region Return the Visio page object if requested.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Enable-VisioPageAutoEnlarge
```

```
function Disable-VisioPageAutoEnlarge {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the auto enlarge property on the Visio page.
        $visioPage.PageSheet.CellsU('ResizePage').ResultIU = 0
        #endregion

        #region Return the Visio page object if requested.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Disable-VisioPageAutoEnlarge

function Set-VisioPageDiagramStyle {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1, Mandatory=$true)]
    [ValidateScript({@('Default','Radial','Flowchart','Circular','CompactTree','Hierarchy') -
contains ($_ -replace ' ')})]
    [System.String]
    $Style,

    [Parameter(Position=2)]

[ValidateScript({@('DownThenRight','RightThenDown','RightThenUp','UpThenRight','UpThenLeft','LeftTh
```

```
enUp','LeftThenDown','DownThenLeft','TopToBottom','BottomToTop','LeftToRight','RightToLeft') -
contains ($_ -replace ' ')})]
    [System.String]
    $Direction,

    [Parameter(Position=3)]
    [ValidateSet('Left','Center','Right','Top','Middle','Bottom')]
    [System.String]
    $Alignment,

    [Parameter(Position=4)]
    [ValidateSet('Default','Shallow','Medium','Deep')]
    [System.String]
    $PlacementDepth,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [System.Boolean]$cancelled = $false
      [System.String[]]$validDirections = @()
      [System.String[]]$validAlignments = @()
      [System.String]$internalStyle = $null
      [System.String]$internalDirection = $null
      [System.String]$internalAlignment = $null
      [System.Int32]$newRoutingStyle = 0
      [System.Int32]$newPlacementStyle = 0
      [System.Int32]$newPlacementDepth = 0
      #endregion

      #region Pre-process the parameters.
      $internalStyle = $Style -replace ' '
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Direction') -and $Direction) {
        $internalDirection = $Direction -replace ' '
      }
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Alignment') -and $Alignment) {
        $internalAlignment = $Alignment
      }
      if ((@('Default','Radial','Circular') -contains $internalStyle) -and ($internalDirection -or
$internalAlignment)) {
        Write-Warning "'Direction' and 'Alignment' are not applicable to '$Style' diagrams. These
parameters will be ignored."
        $internalDirection = $null
        $internalAlignment = $null
      } elseif (@('Flowchart','CompactTree') -contains $internalStyle) {
        if ($internalAlignment) {
          Write-Warning "'Alignment' is not applicable to '$Style' diagrams. This parameter will be
ignored."
          $internalAlignment = $null
        }
        if (-not $internalDirection) {
          switch -exact ($internalStyle) {
            'Flowchart'   {$internalDirection = 'TopToBottom';   break}
            'CompactTree' {$internalDirection = 'DownThenRight'; break}
          }
        }
      } elseif ($internalStyle -eq 'Hierarchy') {
        if (-not $internalDirection) {$internalDirection = 'TopToBottom'}
        if (-not $internalAlignment) {$internalAlignment = 'Center'}
      }
      if ($internalDirection) {
        if ($internalStyle -eq 'CompactTree') {
          $validDirections =
@('DownThenRight','RightThenDown','RightThenUp','UpThenRight','UpThenLeft','LeftThenUp','LeftThenDo
wn','DownThenLeft')
        } else {
          $validDirections = @('TopToBottom','BottomToTop','LeftToRight','RightToLeft')
```

```
        }
        if ($validDirections -notcontains $internalDirection) {
          Write-Error "Cannot bind value 'Direction' to the target. The value was invalid. Valid
values for 'Direction' include '$([string]::Join(""', '"",$validDirections))'."
          $cancelled = $true
          return
        }
      }
      if ($internalAlignment) {
        if (@('TopToBottom','BottomToTop') -match $internalDirection) {
          $validAlignments = @('Left','Center','Right')
        } else {
          $validAlignments = @('Top','Middle','Bottom')
        }
        if ($validAlignments -notcontains $internalAlignment) {
          Write-Error "Cannot bind value 'Alignment' to the target. The value was invalid. Valid
values for 'Alignment' in the '$internalDirection' direction include '$([string]::Join(""',
'"",$validAlignments))'."
          $cancelled = $true
          return
        }
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      if (-not $cancelled) {
        foreach ($visioPage in $Page) {
          #region Reset page variables.
          $newRoutingStyle = 0
          $newPlacementStyle = 0
          $newPlacementDepth = 0
          #endregion

          #region Activate the Visio document window if necessary.
          if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
            $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
            $visioDocumentWindow.Activate()
          } else {
            $visioDocumentWindow = $visioPage.Application.ActiveWindow
          }
          #endregion

          #region Activate the Visio page if necessary.
          if ($visioDocumentWindow.Page -ne $visioPage) {
            $visioDocumentWindow.Page = $visioPage
          }
          #endregion

          #region Set the diagram style of the Visio page.
          switch ($internalStyle) {
            'Default' {
              $newRoutingStyle = 1
              switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
                {@(3,4,10,11)   -contains $_} {$newRoutingStyle = 3;  break}
                {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 5;  break}
                {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 7;  break}
                {@(17,18,19,20) -contains $_} {$newRoutingStyle = 17; break}
                default                       {$newRoutingStyle = $_; break}
              }
              $visioPage.PageSheet.CellsU('PlaceStyle').ResultIU = 0
              $visioPage.PageSheet.CellsU('RouteStyle').ResultIU = $newRoutingStyle
              break
            }
            {@('Radial','Circular') -contains $_} {
              $newPlacementStyle = $(if ($internalStyle -eq 'Radial') {3} else {6})
              $newRoutingStyle = $(if ($internalStyle -eq 'Radial') {1} else {16})
```

```
    switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
      {@(3,4,10,11)   -contains $_} {$newRoutingStyle = 3;  break}
      {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 5;  break}
      {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 7;  break}
      {@(17,18,19,20) -contains $_} {$newRoutingStyle = 17; break}
      default                       {$newRoutingStyle = $_; break}
    }
    $visioPage.PageSheet.CellsU('PlaceStyle').ResultIU = $newPlacementStyle
    $visioPage.PageSheet.CellsU('RouteStyle').ResultIU = $newRoutingStyle
    break
  }
  'Flowchart' {
    $newPlacementStyle = 1
    switch -exact ($Direction) {
      'TopToBottom' {$newPlacementStyle = 1; break}
      'BottomToTop' {$newPlacementStyle = 4; break}
      'LeftToRight' {$newPlacementStyle = 2; break}
      'RightToLeft' {$newPlacementStyle = 5; break}
    }
    $visioPage.PageSheet.CellsU('PlaceStyle').ResultIU = $newPlacementStyle
    break
  }
  'CompactTree' {
    $newPlacementStyle = 7
    switch -exact ($Direction) {
      'DownThenRight' {$newPlacementStyle = 7;  break}
      'RightThenDown' {$newPlacementStyle = 8;  break}
      'RightThenUp'   {$newPlacementStyle = 9;  break}
      'UpThenRight'   {$newPlacementStyle = 10; break}
      'UpThenLeft'    {$newPlacementStyle = 11; break}
      'LeftThenUp'    {$newPlacementStyle = 12; break}
      'LeftThenDown'  {$newPlacementStyle = 13; break}
      'DownThenLeft'  {$newPlacementStyle = 14; break}
    }
    switch -regex ($Direction) {
      '^Down'  {
        switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
          {@(3,4,10,11)   -contains $_} {$newRoutingStyle = 3;  break}
          {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 5;  break}
          {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 7;  break}
          {@(17,18,19,20) -contains $_} {$newRoutingStyle = 17; break}
          default                       {$newRoutingStyle = $_; break}
        }
      }
      '^Right' {
        switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
          {@(0,3,4,10,11) -contains $_} {$newRoutingStyle = 4;  break}
          {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 6;  break}
          {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 8;  break}
          {@(17,18,19,20) -contains $_} {$newRoutingStyle = 18; break}
          default                       {$newRoutingStyle = $_; break}
        }
      }
      '^Up'    {
        switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
          {@(0,3,4,10,11) -contains $_} {$newRoutingStyle = 10; break}
          {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 12; break}
          {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 14; break}
          {@(17,18,19,20) -contains $_} {$newRoutingStyle = 19; break}
          default                       {$newRoutingStyle = $_; break}
        }
      }
      '^Left'  {
        switch ($visioPage.PageSheet.CellsU('RouteStyle').ResultIU) {
          {@(0,3,4,10,11) -contains $_} {$newRoutingStyle = 11; break}
          {@(5,6,12,13)   -contains $_} {$newRoutingStyle = 13; break}
          {@(7,8,14,15)   -contains $_} {$newRoutingStyle = 15; break}
          {@(17,18,19,20) -contains $_} {$newRoutingStyle = 20; break}
          default                       {$newRoutingStyle = $_; break}
        }
```

```
                }
              }
              $visioPage.PageSheet.CellsU('PlaceStyle').ResultIU = $newPlacementStyle
              $visioPage.PageSheet.CellsU('RouteStyle').ResultIU = $newRoutingStyle
              break
            }
            'Hierarchy' {
              $newPlacementStyle = 16
              switch -exact ("$Direction,$Alignment") {
                'TopToBottom,Left'   {$newPlacementStyle = 16; break}
                'TopToBottom,Center' {$newPlacementStyle = 17; break}
                'TopToBottom,Right'  {$newPlacementStyle = 18; break}
                'BottomToTop,Left'   {$newPlacementStyle = 19; break}
                'BottomToTop,Center' {$newPlacementStyle = 20; break}
                'BottomToTop,Right'  {$newPlacementStyle = 21; break}
                'LeftToRight,Top'    {$newPlacementStyle = 22; break}
                'LeftToRight,Middle' {$newPlacementStyle = 23; break}
                'LeftToRight,Bottom' {$newPlacementStyle = 24; break}
                'RightToLeft,Top'    {$newPlacementStyle = 25; break}
                'RightToLeft,Middle' {$newPlacementStyle = 26; break}
                'RightToLeft,Bottom' {$newPlacementStyle = 27; break}
              }
              $newRoutingStyle = 3
              switch ($newPlacementStyle) {
                {@(16,17,18) -contains $_} {$newRoutingStyle = 3;  break}
                {@(22,23,24) -contains $_} {$newRoutingStyle = 4;  break}
                {@(19,20,21) -contains $_} {$newRoutingStyle = 10; break}
                {@(25,26,27) -contains $_} {$newRoutingStyle = 11; break}
              }
              $visioPage.PageSheet.CellsU('PlaceStyle').ResultIU = $newPlacementStyle
              $visioPage.PageSheet.CellsU('RouteStyle').ResultIU = $newRoutingStyle
              break
            }
          }
          if ($PlacementDepth) {
            $newPlacementDepth = 0
            switch -exact ($PlacementDepth) {
              'Shallow' {$newPlacementDepth = 3; break}
              'Medium'  {$newPlacementDepth = 1; break}
              'Deep'    {$newPlacementDepth = 2; break}
            }
            $visioPage.PageSheet.CellsU('PlaceDepth').ResultIU = $newPlacementDepth
          }
          #endregion

          #region Return the Visio page object if requested.
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
            $visioPage
          }
          #endregion
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Set-VisioPageDiagramStyle

function Set-VisioPageConnectorStyle {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1)]
```

```
[ValidateScript({@('Default','RightAngle','Straight','CenterToCenter','Network','Flowchart','Tree',
'OrganizationChart','Simple','SimpleHorizontalVertical','SimpleVerticalHorizontal') -contains ($_ -
replace ' ')})]
    [System.String]
    $Style,

    [Parameter(Position=2)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $Spacing,

    [Parameter(Position=3)]
    [ValidateSet('Default','Straight','Curved')]
    [System.String]
    $Curvature,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [System.String]$internalStyle = $null
      #endregion

      #region Pre-process the parameters.
      $internalStyle = $Style -replace ' '
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Initialize page connector style variables.
        [System.Int32]$newRoutingStyle = 0
        [System.Double]$spacingValue = 0.0
        [System.String]$spacingUnits = 'mm'
        [System.Double]$newSpacingValue = 0.0
        [System.Int32]$newCurvature = 0
        #endregion

        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the connector style of the Visio page.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Style')) {
          $newRoutingStyle = 0
          switch -exact ($internalStyle) {
            'Default' {
              $newRoutingStyle = 0
            }
            'RightAngle' {
              $newRoutingStyle = 1
```

```
            break
          }
          'Straight' {
            $newRoutingStyle = 3
            break
          }
          'CenterToCenter' {
            $newRoutingStyle = 16
            break
          }
          'Network' {
            $newRoutingStyle = 9
            break
          }
          'Flowchart'{
            switch ($visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
              {@(0,1,7,14,16,17,18) -contains $_} {$newRoutingStyle = 5;  break}
              {@(2,8,9,22,23,24)    -contains $_} {$newRoutingStyle = 6;  break}
              {@(4,10,11,19,20,21)  -contains $_} {$newRoutingStyle = 12; break}
              {@(5,12,13,25,26,27)  -contains $_} {$newRoutingStyle = 13; break}
            }
            break
          }
          'Tree' {
            switch ($visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
              {@(0,1,7,14,16,17,18) -contains $_} {$newRoutingStyle = 7;  break}
              {@(2,8,9,22,23,24)    -contains $_} {$newRoutingStyle = 8;  break}
              {@(4,10,11,19,20,21)  -contains $_} {$newRoutingStyle = 14; break}
              {@(5,12,13,25,26,27)  -contains $_} {$newRoutingStyle = 15; break}
            }
            break
          }
          'OrganizationChart' {
            switch ($visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
              {@(0,1,7,14,16,17,18) -contains $_} {$newRoutingStyle = 3;  break}
              {@(2,8,9,22,23,24)    -contains $_} {$newRoutingStyle = 4;  break}
              {@(4,10,11,19,20,21)  -contains $_} {$newRoutingStyle = 10; break}
              {@(5,12,13,25,26,27)  -contains $_} {$newRoutingStyle = 11; break}
            }
            break
          }
          'Simple' {
            switch ($visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
              {@(0,1,7,14,16,17,18) -contains $_} {$newRoutingStyle = 17; break}
              {@(2,8,9,22,23,24)    -contains $_} {$newRoutingStyle = 18; break}
              {@(4,10,11,19,20,21)  -contains $_} {$newRoutingStyle = 19; break}
              {@(5,12,13,25,26,27)  -contains $_} {$newRoutingStyle = 20; break}
            }
            break
          }
          'SimpleHorizontalVertical' {
            $newRoutingStyle = 21
            break
          }
          'SimpleVerticalHorizontal' {
            $newRoutingStyle = 22
            break
          }
        }
        $visioPage.PageSheet.CellsU('RouteStyle').ResultIU = $newRoutingStyle
      }
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Spacing')) {
        $spacingValue = $Spacing -replace ' ','' -replace '^(\d+(\.\d+)?) ?([a-z]+)?$','$1'
        $spacingUnits = $Spacing -replace ' ','' -replace '^(\d+(\.\d+)?) ?([a-z]+)?$','$3'
        if ($spacingUnits) {
          $newSpacingValue = $visioPage.Application.ConvertResult($spacingValue, $spacingUnits,
''')
        } else {
          $newSpacingValue = $spacingValue
        }
```

```
          $visioPage.PageSheet.CellsU('AvenueSizeX').ResultIU = $newSpacingValue
          $visioPage.PageSheet.CellsU('AvenueSizeY').ResultIU = $newSpacingValue
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Curvature')) {
          $newCurvature = 0
          switch -exact ($Curvature) {
            'Default'  {$newCurvature = 0; break}
            'Straight' {$newCurvature = 1; break}
            'Curved'   {$newCurvature = 2; break}
          }
          $visioPage.PageSheet.CellsU('LineRouteExt').ResultIU = $newCurvature
        }
        #endregion

        #region Return the Visio page object if requested.
        if ($PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Set-VisioPageConnectorStyle

function Set-VisioPageTheme {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1)]
    [System.String]
    $Colors,

    [Parameter(Position=2)]
    [System.String]
    $Effects,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [System.Boolean]$cancelled = $false
      #endregion

      #region Raise a warning if the running instance of Visio does not support themes.
      if
([System.Reflection.Assembly]::GetAssembly('Microsoft.Office.Interop.Visio.IVPage').FullName -eq
'Microsoft.Office.Interop.Visio, Version=11.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c') {
        Write-Warning 'Visio 2003 does not support themes. This command will be ignored.'
        $cancelled = $true
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
```

```
      #region Return if cancelled.
      if ($cancelled) {
        return
      }
      #endregion

      foreach ($visioPage in $Page) {
        #region Initialize page connector style variables.
        [System.String]$colorThemeId = $(if ((-not
$PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Colors')) -or (-not $Colors)) {'None'} else
{$Colors -replace '^visThemeColors' -replace '\W'})
        [System.String]$effectThemeId = $(if ((-not
$PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Effects')) -or (-not $Effects)) {'None'} else
{$Effects -replace '^visThemeEffects' -replace '\W'})
        [System.Boolean]$colorThemeFound = $false
        [System.Boolean]$effectThemeFound = $false
        [System.String[]]$localizedColorThemes = @()
        [System.String[]]$universalColorThemes = @()
        [System.String[]]$localizedEffectThemes = @()
        [System.String[]]$universalEffectThemes = @()
        #endregion

        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the theme of the Visio page.

$visioPage.Document.GetThemeNames([Microsoft.Office.Interop.Visio.VisThemeTypes]::visThemeTypeColor
,([REF]$localizedColorThemes))

$visioPage.Document.GetThemeNamesU([Microsoft.Office.Interop.Visio.VisThemeTypes]::visThemeTypeColo
r,([REF]$universalColorThemes))
        $localizedColorThemes = $localizedColorThemes -replace '\W',''
        $universalColorThemes = $universalColorThemes -replace 'visThemeNone','visThemeColorsNone'
-replace '\W',''
        if (($localizedColorThemes -contains $colorThemeId) -or ($universalColorThemes -contains
$(if ($colorThemeId -match '^visThemeColors') {$colorThemeId} else
{"visThemeColors$colorThemeId"}))) {
          for ($index = 0; $index -lt $universalColorThemes.Count; $index++) {
            if (($localizedColorThemes[$index] -eq $colorThemeId) -or
($universalColorThemes[$index] -eq $(if ($colorThemeId -match '^visThemeColors') {$colorThemeId}
else {"visThemeColors$colorThemeId"}))) {
              $colorThemeId = $universalColorThemes[$index]
              $colorThemeFound = $true
              break
            }
          }
        }
        if (-not $colorThemeFound) {
          Write-Error "The color theme called '$Colors' was not found. These colors will not be
applied to page $($visioPage.Name)."
          $cancelled = $true
          return
        }

$visioPage.Document.GetThemeNames([Microsoft.Office.Interop.Visio.VisThemeTypes]::visThemeTypeEffec
t,([REF]$localizedEffectThemes))
```

```
$visioPage.Document.GetThemeNamesU([Microsoft.Office.Interop.Visio.VisThemeTypes]::visThemeTypeEffe
ct,([REF]$universalEffectThemes))
        $localizedEffectThemes = $localizedEffectThemes -replace '\W',''
        $universalEffectThemes = $universalEffectThemes -replace '\W',''
        if (($localizedEffectThemes -contains $effectThemeId) -or ($universalEffectThemes -contains
$(if ($effectThemeId -match '^visThemeEffects') {$effectThemeId} else
{"visThemeEffects$effectThemeId"}))) {
          for ($index = 0; $index -lt $universalEffectThemes.Count; $index++) {
            if (($localizedEffectThemes[$index] -eq $effectThemeId) -or
($universalEffectThemes[$index] -eq $(if ($effectThemeId -match '^visThemeEffects')
{$effectThemeId} else {"visThemeEffects$effectThemeId"}))) {
              $effectThemeId = $universalEffectThemes[$index]
              $effectThemeFound = $true
              break
            }
          }
        }
        if (-not $effectThemeFound) {
          Write-Error "The effect theme called '$Effects' was not found. These effects will not be
applied to page $($visioPage.Name)."
          $cancelled = $true
          return
        }
        $visioPage.ThemeColors  = [Microsoft.Office.Interop.Visio.VisThemeColors]$colorThemeId
        $visioPage.ThemeEffects = [Microsoft.Office.Interop.Visio.VisThemeEffects]$effectThemeId
        #endregion

        #region Return the Visio page object if requested.
        if ($PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Set-VisioPageTheme

function Set-VisioPageBackground {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1)]
    [System.String]
    $Background,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion
```

```
        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the background of the Visio page.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Background') -and $Background) {
          $backgroundPage = $null
          foreach ($item in $visioPage.Document.Pages) {
            if (($item.Type -eq [Microsoft.Office.Interop.Visio.VisPageTypes]::visTypeBackground) -
and
                (($item.Name -eq $Background) -or ($item.NameU -eq $(if ($Background -match
'^Background ') {$Background} else {"Background $Background"})))) {
              $backgroundPage = $item
              break
            }
          }
          if (-not $backgroundPage) {
            $key = $(if ($Background -match '^Background ') {$Background} else {"Background
$Background"})
            $backgroundTemplate =
$script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$key]
            if (-not $backgroundTemplate) {
              $backgroundTemplate =
$script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$Background]
            }
            if (-not $backgroundTemplate) {
              Write-Error "The shape type '$Background' was not found. This shape will not be set
as a background for page $($visioPage.Name)."
              continue
            }
            $visioPage.BackPage = ''
            $visioPage.Drop($backgroundTemplate,0,0) | Out-Null
            $visioPage.BackPage.Name = $backgroundTemplate.Name
            $visioPage.BackPage.NameU = $backgroundTemplate.NameU
          } else {
            $visioPage.BackPage = $backgroundPage.Name
          }
          if ($visioPage.BackPage) {
            if ($visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU -lt
$visioPage.PageSheet.CellsU('PageWidth').ResultIU) {
              $visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU =
$visioPage.PageSheet.CellsU('PageWidth').ResultIU
            }
            if ($visioPage.BackPage.PageSheet.CellsU('PageHeight').ResultIU -lt
$visioPage.PageSheet.CellsU('PageHeight').ResultIU) {
              $visioPage.BackPage.PageSheet.CellsU('PageHeight') =
$visioPage.PageSheet.CellsU('PageHeight').ResultIU
            }
          }
        } else {
          $visioPage.BackPage = ''
        }
        #endregion

        #region Return the Visio page object if requested.
        if ($PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Set-VisioPageBackground
```

```powershell
function Format-VisioPageLayout {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $ArrangeShapes,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $CenterDrawing,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $ShrinkToFit,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        #region Activate the Visio document window if necessary.
        if ($visioPage.Document -ne $visioPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio page if necessary.
        if ($visioDocumentWindow.Page -ne $visioPage) {
          $visioDocumentWindow.Page = $visioPage
        }
        #endregion

        #region Set the Visio page layout.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ArrangeShapes') -and
$ArrangeShapes) {
          $visioPage.Layout()
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('CenterDrawing') -and
$CenterDrawing) {
          $visioPage.CenterDrawing()
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ShrinkToFit') -and $ShrinkToFit) {
          $visioPage.ResizeToFitContents()
        }
        if ($visioPage.BackPage) {
          if ($visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU -lt
$visioPage.PageSheet.CellsU('PageWidth').ResultIU) {
            $visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU =
$visioPage.PageSheet.CellsU('PageWidth').ResultIU
          }
          if ($visioPage.BackPage.PageSheet.CellsU('PageHeight').ResultIU -lt
$visioPage.PageSheet.CellsU('PageHeight').ResultIU) {
            $visioPage.BackPage.PageSheet.CellsU('PageHeight').ResultIU =
$visioPage.PageSheet.CellsU('PageHeight').ResultIU
          }
        }
        #endregion
```

```
        #region Return the Visio page object if requested.
        if ($PassThru) {
          $visioPage
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Format-VisioPageLayout

#endregion

#region VisioPageConfiguration functions

function Reset-VisioPageConfiguration {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVPage])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      $visioDefaultConfiguration = Get-VisioDefaultConfiguration
      foreach ($visioPage in $Page) {
        Write-Progress -Activity 'Applying default Visio page configuration' -Status 'Setting page
background.'
        Set-VisioPageBackground -Page $visioPage -Background
$visioDefaultConfiguration.Page.Background
        Write-Progress -Activity 'Applying default Visio page configuration' -Status 'Setting page
theme.'
        Set-VisioPageTheme -Page $visioPage -Colors $visioDefaultConfiguration.Page.Colors -Effects
$visioDefaultConfiguration.Page.Effects
        Write-Progress -Activity 'Applying default Visio page configuration' -Status 'Setting page
auto-enlarge flag.'
        if ($visioDefaultConfiguration.Page.AutoEnlarge) {
          Enable-VisioPageAutoEnlarge -Page $visioPage
        } else {
          Disable-VisioPageAutoEnlarge -Page $visioPage
        }
        Write-Progress -Activity 'Applying default Visio page configuration' -Status 'Setting page
diagram style.'
        $setVisioPageDiagramStyleParameters = @{
           Page = $visioPage
          Style = $visioDefaultConfiguration.Page.DiagramStyle
        }
        if ($visioDefaultConfiguration.Page.DiagramDirection) {
          $setVisioPageDiagramStyleParameters['Direction'] =
$visioDefaultConfiguration.Page.DiagramDirection
        }
        if ($visioDefaultConfiguration.Page.DiagramAlignment) {
          $setVisioPageDiagramStyleParameters['Alignment'] =
$visioDefaultConfiguration.Page.DiagramAlignment
        }
        if ($visioDefaultConfiguration.Page.DiagramPlacementDepth) {
          $setVisioPageDiagramStyleParameters['PlacementDepth'] =
$visioDefaultConfiguration.Page.DiagramPlacementDepth
        }
        Set-VisioPageDiagramStyle @setVisioPageDiagramStyleParameters
```

```
        Write-Progress -Activity 'Applying default Visio page configuration' -Status 'Setting page
connector style.'
        $setVisioPageConnectorStyleParameters = @{
          Page = $visioPage
        }
        if ($visioDefaultConfiguration.Page.ConnectorStyle) {
          $setVisioPageConnectorStyleParameters['Style'] =
$visioDefaultConfiguration.Page.ConnectorStyle
        }
        if ($visioDefaultConfiguration.Page.ConnectorSpacing) {
          $setVisioPageConnectorStyleParameters['Spacing'] =
$visioDefaultConfiguration.Page.ConnectorSpacing
        }
        if ($visioDefaultConfiguration.Page.ConnectorCurvature) {
          $setVisioPageConnectorStyleParameters['Curvature'] =
$visioDefaultConfiguration.Page.ConnectorCurvature
        }
        Set-VisioPageConnectorStyle @setVisioPageConnectorStyleParameters
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PassThru') -and $PassThru) {
          $visioPage
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Reset-VisioPageConfiguration

#endregion

#region VisioShape functions

function Get-VisioShape {
  [CmdletBinding(DefaultParameterSetName='ByName')]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='ByName')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $Name,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='ByLabel')]
    [ValidateNotNullOrEmpty()]
    [Alias('Text')]
    [System.String[]]
    $Label
  )
  process {
    try {
      foreach ($visioPage in $Page) {
        switch ($PSCmdlet.ParameterSetName) {
          'ByName' {
            foreach ($item in $Name) {
              foreach ($visioShape in $visioPage.Shapes) {
                $visioShape =
[System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioShape,
[Microsoft.Office.Interop.Visio.ShapeClass])
                if (($visioShape.Name -like $item) -or ($visioShape.NameU -like $item)) {
                  [Microsoft.Office.Interop.Visio.IVShape]$visioShape
                }
              }
            }
            break
```

```
          }
          'ByLabel' {
            foreach ($item in $Label) {
              foreach ($visioShape in $visioPage.Shapes) {
                $visioShape =
[System.Runtime.InteropServices.Marshal]::CreateWrapperOfType($visioShape,
[Microsoft.Office.Interop.Visio.ShapeClass])
                if ($visioShape.Text -like $item) {
                  [Microsoft.Office.Interop.Visio.IVShape]$visioShape
                }
              }
            }
            break
          }
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Get-VisioShape

function New-VisioShape {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $ShapeId,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $Name,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [Alias('Text')]
    [System.String]
    $Label,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $X,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $Y,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $SmartLayout
  )
  begin {
    try {
      #region Initialize local variables.
      [System.Boolean]$private:useSmartLayout = $false
```

```
      [System.Collections.Hashtable]$private:visioDefaultConfiguration = Get-
VisioDefaultConfiguration
      #endregion

      #region Determine if we are using a smart layout or not.
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('SmartLayout')) {
        $private:useSmartLayout = $SmartLayout
      } else {
        $private:useSmartLayout = $visioDefaultConfiguration.SmartLayout
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      foreach ($visioPage in $Page) {
        foreach ($item in $ShapeId) {
          #region Verify that the shape id matches a shape that is available on our loaded
stencils.
          if (-not
$script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates.ContainsKey($item)) {
            Write-Error "The shape type '$item' was not found. This shape will not be added to page
$($visioPage.Name)."
            continue
          }
          #endregion

          #region Add the shape to the Visio diagram.
          Write-Progress -Activity 'Adding shape to Visio diagram' -Status "Adding a shape of type
'$item' to the Visio diagram."

          #region Determine the coordinates where the shape should be placed.
          $xCoordinate = 0.0
          $yCoordinate = 0.0
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('X')) {
            $positionValue = $X -replace '^(\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $positionUnits = $X -replace '^(\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $xCoordinate = $visioPage.Application.ConvertResult([double]$positionValue,
$positionUnits, '')
          } elseif (@(5,11,12,13,14,16,19,25,26,27) -contains
$visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
            $xCoordinate =
$script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$item].Shapes.Item(1).CellsU
('Width').ResultIU / 2
          } else {
            $xCoordinate = $visioPage.PageSheet.CellsU('PageWidth').ResultIU -
($script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$item].Shapes.Item(1).Cells
U('Width').ResultIU / 2)
          }
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Y')) {
            $positionValue = $Y -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $positionUnits = $Y -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $yCoordinate = $visioPage.Application.ConvertResult([double]$positionValue,
$positionUnits, '')
          } elseif (@(4,9,10,11,12,19,20,21,22,25) -contains
$visioPage.PageSheet.CellsU('PlaceStyle').ResultIU) {
            $yCoordinate = $visioPage.PageSheet.CellsU('PageHeight').ResultIU -
($script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$item].Shapes.Item(1).Cells
U('Height').ResultIU / 2)
          } else {
            $yCoordinate =
$script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$item].Shapes.Item(1).CellsU
('Height').ResultIU / 2
          }
          #endregion

          #region Drop the new shape on the page.
```

```
          if ($visioShape =
$visioPage.Drop($script:VisioTemplateCache[$visioPage.Application.ProcessId].Templates[$item],$xCoo
rdinate,$yCoordinate)) {
            #region Set the shape name if a custom name was provided.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Name')) {
              Rename-VisioShape -VisioShape $visioShape -NewName $Name
            }
            #endregion

            #region Set the shape label if a custom label was provided.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Label')) {
              Set-VisioShapeLabel -Shape $visioShape -Label $Label
            }
            #endregion

            #region Re-layout the page if smart layout is enabled.
            if ($useSmartLayout) {
              $visioPage.Layout()
              $visioPage.CenterDrawing()
            }
            #endregion

            #region Update the size of the background page if smart layout or auto-enlarge are
enabled.
            if ($visioPage.BackPage -and
                ($useSmartLayout -or ($visioPage.PageSheet.CellsU('ResizePage').ResultIU -eq 1))) {
              if ($visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU -lt
$visioPage.PageSheet.CellsU('PageWidth').ResultIU) {
                $visioPage.BackPage.PageSheet.CellsU('PageWidth').ResultIU =
$visioPage.PageSheet.CellsU('PageWidth').ResultIU
              }
              if ($visioPage.BackPage.PageSheet.CellsU('PageHeight').ResultIU -lt
$visioPage.PageSheet.CellsU('PageHeight').ResultIU) {
                $visioPage.BackPage.PageSheet.CellsU('PageHeight').ResultIU =
$visioPage.PageSheet.CellsU('PageHeight').ResultIU
              }
            }
            #endregion

            #region Return the shape object to the caller.
            $visioShape
            #endregion
          }
          #endregion

          #endregion
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function New-VisioShape

function Select-VisioShape {
  [CmdletBinding(DefaultParameterSetName='NewShape')]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true,
ParameterSetName='ShapeObject')]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape[]]
    $Shape,

    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true, ParameterSetName='NewShape')]
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true, ParameterSetName='ByName')]
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true, ParameterSetName='ByLabel')]
    [ValidateNotNullOrEmpty()]
```

```
    [Microsoft.Office.Interop.Visio.IVPage[]]
    $Page,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='NewShape')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $ShapeId,

    [Parameter(Mandatory=$true, ParameterSetName='ByName')]
    [System.Management.Automation.SwitchParameter]
    $ByName,

    [Parameter(Mandatory=$true, ParameterSetName='ByLabel')]
    [System.Management.Automation.SwitchParameter]
    $ByLabel,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='ByName')]
    [Parameter(ParameterSetName='NewShape')]
    [ValidateNotNullOrEmpty()]
    [System.String[]]
    $Name,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='ByLabel')]
    [Parameter(ParameterSetName='NewShape')]
    [ValidateNotNullOrEmpty()]
    [Alias('Text')]
    [System.String]
    $Label,

    [Parameter(Position=1, ParameterSetName='ShapeObject')]
    [Parameter(Position=2, ParameterSetName='NewShape')]
    [Parameter(Position=2, ParameterSetName='ByName')]
    [Parameter(Position=2, ParameterSetName='ByLabel')]
    [ValidateNotNullOrEmpty()]
    [System.Management.Automation.ScriptBlock]
    $ScriptBlock,

    [Parameter(ParameterSetName='NewShape')]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $X,

    [Parameter(ParameterSetName='NewShape')]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $Y,

    [Parameter(ParameterSetName='NewShape')]
    [System.Management.Automation.SwitchParameter]
    $SmartLayout
  )
  begin {
    try {
      #region Initialize local variables.
      [System.String]$private:parameterName = $null
      [System.Boolean]$private:useSmartLayout = $false
      [System.Collections.Hashtable]$private:visioDefaultConfiguration = Get-
VisioDefaultConfiguration
      #endregion

      #region Convert the parameters to private scope.
      foreach ($private:parameterName in
@('Shape','Page','ShapeId','ByName','ByLabel','Name','Label','ScriptBlock','X','Y','SmartLayout'))
{
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:parameterName)) {
          Set-Variable -Name $private:parameterName -Scope Local -Option Private
        }
      }
```

```
      #endregion

      #region Determine if we are using a smart layout or not.
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('SmartLayout')) {
        $private:useSmartLayout = $SmartLayout
      } else {
        $private:useSmartLayout = $visioDefaultConfiguration.SmartLayout
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      switch ($PSCmdlet.ParameterSetName) {
        'ByName' {
          #region If a specific shape was requested, use it.
          $private:getVisioShapeParameters = @{
             Page = $Page
             Name = $Name
          }
          $private:selectVisioShapeParameters = @{}
          foreach ($private:optionalParameterName in @('ScriptBlock')) {
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:optionalParameterName))
{
              $private:selectVisioShapeParameters[$private:optionalParameterName] =
$PSCmdlet.MyInvocation.BoundParameters[$private:optionalParameterName]
            }
          }
          Get-VisioShape @private:getVisioShapeParameters | Select-VisioShape
@private:selectVisioShapeParameters
          #endregion
          break
        }
        'ByLabel' {
          #region If a specific shape was requested, use it.
          $private:getVisioShapeParameters = @{
             Page = $Page
            Label = $Label
          }
          $private:selectVisioShapeParameters = @{}
          foreach ($private:optionalParameterName in @('ScriptBlock')) {
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:optionalParameterName))
{
              $private:selectVisioShapeParameters[$private:optionalParameterName] =
$PSCmdlet.MyInvocation.BoundParameters[$private:optionalParameterName]
            }
          }
          Get-VisioShape @private:getVisioShapeParameters | Select-VisioShape
@private:selectVisioShapeParameters
          #endregion
          break
        }
        'NewShape' {
          #region If a specific shape was requested, use it.
          $private:newVisioShapeParameters = @{
               Page = $Page
            ShapeId = $ShapeId
          }
          foreach ($private:optionalParameterName in @('Name','Label','X','Y','SmartLayout')) {
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:optionalParameterName))
{
              $private:newVisioShapeParameters[$private:optionalParameterName] =
$PSCmdlet.MyInvocation.BoundParameters[$private:optionalParameterName]
            }
          }
          $private:selectVisioShapeParameters = @{}
          foreach ($private:optionalParameterName in @('ScriptBlock')) {
```

```
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey($private:optionalParameterName))
{
              $private:selectVisioShapeParameters[$private:optionalParameterName] =
$PSCmdlet.MyInvocation.BoundParameters[$private:optionalParameterName]
            }
          }
          New-VisioShape @private:newVisioShapeParameters | Select-VisioShape
@private:selectVisioShapeParameters
          #endregion
          break
        }
        'ShapeObject' {
          #region Process the shape object.
          foreach ($visioShape in $Shape) {
            #region Activate the Visio document window if necessary.
            if ($visioShape.Document -ne $visioShape.Application.ActiveDocument) {
              $visioDocumentWindow = Get-VisioDocumentWindow -Document $visioShape.Document
              $visioDocumentWindow.Activate()
            } else {
              $visioDocumentWindow = $visioShape.Application.ActiveWindow
            }
            #endregion

            #region Activate the Visio page if necessary.
            if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
              $visioDocumentWindow.Page = $visioShape.ContainingPage
            }
            #endregion

            #region Invoke the ScriptBlock.
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('ScriptBlock')) {
              & (Get-Module -Name VisioShapeDSV) $ScriptBlock | Out-Null
            }
            #endregion

            #region Return the Visio shape object to the client.
            $visioShape
            #endregion
          }
          #endregion
          break
        }
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Select-VisioShape

function Rename-VisioShape {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter(Position=1, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $NewName,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
```

```
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Rename the Visio shape.
        $visioShape.Name = $NewName -replace '(^\s+|\s+$)'
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Rename-VisioShape

function Set-VisioShapeLabel {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter(Position=1, Mandatory=$true, ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('Text')]
    [System.String]
    $Label,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion
```

```
        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Set the Visio shape label if necessary.
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Label')) {
          $visioShape.Text = $Label -replace '(^\s+|\s+$)'
        }
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Set-VisioShapeLabel

function Add-VisioShapeWebHyperlink {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter(Position=1, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $Url,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $Description,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Default,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
```

```
        }
        #endregion

        #region Add the hyperlink to the Visio shape.
        $visioHyperlink = $visioShape.AddHyperLink()
        $visioHyperlink.Address = $Url -replace '(^\s+|\s+$)'
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Description')) {
          $visioHyperlink.Description = $Description -replace '(^\s+|\s+$)'
        }
        $visioHyperlink.IsDefaultLink =
($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Default') -and $Default)
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Add-VisioShapeWebHyperlink

function Add-VisioShapeFileHyperlink {
  [CmdletBinding(DefaultParameterSetName='Page')]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='ExternalFile')]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $FileName,

    [Parameter(ParameterSetName='Shape')]
    [Parameter(Position=1, Mandatory=$true, ParameterSetName='Page')]
    [Parameter(Position=2, Mandatory=$true, ParameterSetName='ExternalFile')]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $PageName,

    [Parameter(Position=1, Mandatory=$true, ParameterSetName='Shape')]
    [Parameter(Position=3, Mandatory=$true, ParameterSetName='ExternalFile')]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $ShapeName,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $Zoom,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [System.String]
    $Description,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $Default,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
```

```
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Add the hyperlink to the Visio shape.
        $visioHyperlink = $visioShape.AddHyperLink()
        switch ($PSCmdlet.ParameterSetName) {
          'Page' {
            $visioHyperlink.SubAddress = $PageName -replace '(^\s+|\s+$)'
            break
          }
          'Shape' {
            if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('PageName')) {
              $visioHyperlink.SubAddress = "$($PageName -replace '(^\s+|\s+$)'))/$($ShapeName -
replace '(^\s+|\s+$)'))"
            } else {
              $visioHyperlink.SubAddress = $ShapeName -replace '(^\s+|\s+$)'
            }
            break
          }
          'ExternalFile' {
            $visioHyperlink.Address = $Filename
            $visioHyperlink.SubAddress = "$($PageName -replace '(^\s+|\s+$)'))/$($ShapeName -
replace '(^\s+|\s+$)'))"
            break
          }
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Zoom')) {
          $visioHyperlink.ExtraInfo = "zoom=$($Zoom -replace '(^\s+|\s+$)' -replace '\W*%$')"
        }
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Description')) {
          $visioHyperlink.Description = $Description -replace '(^\s+|\s+$)'
        }
        $visioHyperlink.IsDefaultLink =
($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Default') -and $Default)
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Add-VisioShapeFileHyperlink

function Move-VisioShape {
  [CmdletBinding()]
```

```
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $X,

    [Parameter(Position=1)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $Y,

    [Parameter(Position=2)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $DeltaX,

    [Parameter(Position=3)]
    [ValidateNotNullOrEmpty()]
    [ValidatePattern('^(-?\d+(\.\d+)?) ?([a-z]+)?$')]
    [System.String]
    $DeltaY,

    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('X') -or
            $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Y') -or
            $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('DeltaX') -or
            $PSCmdlet.MyInvocation.BoundParameters.ContainsKey('DeltaY')) {
          #region Initialize local variables.
          [System.Double]$value = 0.0
          [System.String]$units = $null
          [System.Double]$xDestination = 0.0
          [System.Double]$yDestination = 0.0
          #endregion

          #region Activate the Visio document window if necessary.
          if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
            $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
            $visioDocumentWindow.Activate()
          } else {
            $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
          }
          #endregion

          #region Activate the Visio shape if necessary.
          if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
            $visioDocumentWindow.Page = $visioShape.ContainingPage
          }
          #endregion

          #region Move the Visio shape.
          $xDestination = $visioShape.CellsU('PinX').ResultIU
          $yDestination = $visioShape.CellsU('PinY').ResultIU
```

```
          if ($X) {
            $value = $X -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $units = $X -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $xDestination = $visioShape.Application.ConvertResult($value,$(if ($units) {$units}
else {$null}),'')
          }
          if ($DeltaX) {
            $value = $DeltaX -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $units = $DeltaX -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $xDestination += $visioShape.Application.ConvertResult($value,$(if ($units) {$units}
else {$null}),'')
          }
          if ($Y) {
            $value = $Y -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $units = $Y -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $yDestination = $visioShape.Application.ConvertResult($value,$(if ($units) {$units}
else {$null}),'')
          }
          if ($DeltaY) {
            $value = $DeltaY -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$1'
            $units = $DeltaY -replace '^(-?\d+(\.\d+)?) ?([a-z]+)?$','$3'
            $yDestination += $visioShape.Application.ConvertResult($value,$(if ($units) {$units}
else {$null}),'')
          }
          $visioShape.SetCenter($xDestination,$yDestination)
          #endregion
        }

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Move-VisioShape

function Move-VisioShapeToFront {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
```

```
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Move the Visio shape to the front.
        $visioShape.BringToFront()
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Move-VisioShapeToFront

function Move-VisioShapeToBack {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Move the Visio shape to the back.
        $visioShape.SendToBack()
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
```

```
}
Export-ModuleMember -Function Move-VisioShapeToBack

function Move-VisioShapeForward {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Move the Visio shape forward one level.
        $visioShape.BringForward()
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Move-VisioShapeForward

function Move-VisioShapeBackward {
  [CmdletBinding()]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  process {
    try {
```

```
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion

        #region Move the Visio shape backward one level.
        $visioShape.SendBackward()
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Move-VisioShapeBackward

function Connect-VisioShape {
  [CmdletBinding(DefaultParameterSetName='Default')]
  [OutputType([Microsoft.Office.Interop.Visio.IVShape])]
  param(
    [Parameter(Position=0, Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [Microsoft.Office.Interop.Visio.IVShape]
    $ToShape,

    [Parameter(Mandatory=$true, ParameterSetName='Top')]
    [System.Management.Automation.SwitchParameter]
    $Top,

    [Parameter(Mandatory=$true, ParameterSetName='Bottom')]
    [System.Management.Automation.SwitchParameter]
    $Bottom,

    [Parameter(Mandatory=$true, ParameterSetName='Left')]
    [System.Management.Automation.SwitchParameter]
    $Left,

    [Parameter(Mandatory=$true, ParameterSetName='Right')]
    [System.Management.Automation.SwitchParameter]
    $Right,

    [Parameter(Mandatory=$true, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [Alias('InputObject')]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Shape,

    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [Microsoft.Office.Interop.Visio.IVShape]
    $Connector,
```

```
    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $SmartLayout,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [System.UInt16]$private:direction = 0
      [System.Boolean]$private:directionSupported = $true
      [System.Boolean]$private:useSmartLayout = $false
      [System.Collections.Hashtable]$private:visioDefaultConfiguration = Get-
VisioDefaultConfiguration
      #endregion

      #region Determine the direction that will be used for the connector.
      switch ($PSCmdlet.ParameterSetName) {
        'Top'    {$private:direction = 1; break}
        'Bottom' {$private:direction = 2; break}
        'Left'   {$private:direction = 3; break}
        'Right'  {$private:direction = 4; break}
        default  {$private:direction = 0; break}
      }
      if (($private:direction -ne 0) -and

([System.Reflection.Assembly]::GetAssembly('Microsoft.Office.Interop.Visio.IVPage').FullName -eq
'Microsoft.Office.Interop.Visio, Version=11.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c')) {
        Write-Warning 'Visio 2003 does not support specifying a direction when connecting shapes.
The direction will be ignored.'
        $private:directionSupported = $false
      }

      #endregion

      #region Determine if we are using a smart layout or not.
      if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('SmartLayout')) {
        $private:useSmartLayout = $SmartLayout
      } else {
        $private:useSmartLayout = $visioDefaultConfiguration.SmartLayout
      }
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      foreach ($visioShape in $Shape) {
        #region Activate the Visio document window if necessary.
        if ($visioShape.ContainingPage.Document -ne
$visioShape.ContainingPage.Application.ActiveDocument) {
          $visioDocumentWindow = Get-VisioDocumentWindow -Document
$visioShape.ContainingPage.Document
          $visioDocumentWindow.Activate()
        } else {
          $visioDocumentWindow = $visioShape.ContainingPage.Application.ActiveWindow
        }
        #endregion

        #region Activate the Visio shape if necessary.
        if ($visioDocumentWindow.Page -ne $visioShape.ContainingPage) {
          $visioDocumentWindow.Page = $visioShape.ContainingPage
        }
        #endregion
```

```
        #region Connect the Visio shapes.
        if ($private:directionSupported) {
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Connector')) {
            $ToShape.AutoConnect($visioShape, $direction, $Connector)
          } else {
            $ToShape.AutoConnect($visioShape, $direction,
$visioShape.Application.ConnectorToolDataObject)
          }
        } else {
          if ($PSCmdlet.MyInvocation.BoundParameters.ContainsKey('Connector')) {
            $visioConnector = $Connector
          } else {
            $visioConnector =
$visioShape.ContainingPage.Drop($visioShape.Application.ConnectorToolDataObject,0,0)
          }
          $visioConnector.CellsU('BeginX').GlueTo($ToShape.CellsU('PinX')) | Out-Null
          $visioConnector.CellsU('EndX').GlueTo($visioShape.CellsU('PinX')) | Out-Null
        }
        #endregion

        #region Center the Visio drawing and refresh the layout.
        if ($useSmartLayout) {
          $visioShape.ContainingPage.Layout()
          $visioShape.ContainingPage.CenterDrawing()
        }
        #endregion

        #region Return the Visio shape object if requested.
        if ($PassThru) {
          $visioShape
        }
        #endregion
      }
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Connect-VisioShape

#endregion

# SIG # Begin signature block
# MIIdfwYJKoZIhvcNAQcCoIIdcDCCHWwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUtYJe6H74fvAfHAuXmnEIPzFz
# 0/Kgghi8MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAw
# VzELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
# BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5
# MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJH
# bG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
# AQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6a
# rymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCO
# XkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6T
# RGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux
# 2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlD
# SgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
# HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkq
# hkiG9w0BAQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoR
# SLblCKOzyj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQL
# cFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrg
# lfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr
# +WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XS
# QRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4DCCBAcwggLvoAMCAQICCwEA
# AAAAR5GQJ02MA0GCSqGSIb3DQEBBQUAMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwHhcNMDgxMjE3MTc0ODAyWhcN
# MTExMjE3MTc0ODAyWjBhMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOUXVlc3QgU29m
# dHdhcmUxFzAVBgNVBAMTDlF1ZXN0IFNvZnR3YXJlMSAwHgYJKoZIhvcNAQkBFhFz
```

dXBwb3J0QHF1ZXN0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1mza
2hKiiqZnaF1sHhuFRS7MEGq9tYhF7AFbJRvTvhCZk9sxK92thKBFyDSOzJauB7Zt
j+1HwQzpqbbU94EsR09JOf8vB+xQKLCxaBP5YjwhjJzVy+1d6frVWYN1oVxPXRBM
G7BnFgfRkOdtsg/Qn1Uqn1ENSozyjTuh5iduUy0CAwEAAaOCAUAwggE8MB8GA1Ud
IwQYMBaAFNJb80smS6Ww5139Vn/28S44TlOgME4GCCsGAQUFBwEBBEIwQDA+Bggr
BgEFBQcwAoYyaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLm5ldC9jYWNlcnQvT2Jq
ZWN0U2lnbi5jcnQwOQYDVR0fBDIwMDAuoCygKoYoaHR0cDovL2NybC5nbG9iYWxz
aWduLm5ldC9PYmplY3RTaWduLmNybDAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIH
gDATBgNVHSUEDDAKBggrBgEFBQcDAzBLBgNVHSAERDBCMEAGCSsGAQQBoDIBMjAz
MDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24ubmV0L3JlcG9zaXRv
cnkvMBEGCWCGSAGG+EIBAQQEAwIEEDANBgkqhkiG9w0BAQUFAAOCAQEAG9hUuQek
ddDJ/pzfqo9p4hzKBkeKcVsunEeTUMNg90XzgdOYRFJPCD7T+gXXrTs6Y2xFmLJN
G/2lQsjQ/32cBBN9zZdbX+ExhFfEV9/w0gbw3H/PfYkCRvp9VZlTafIt4MJCt/Zp
guPQgggpWadScg7jQNyeHEg6H6c3WHO8PMiKcKJp9LuM1PKX9Bjy6F2k8rbdEAyJ
u0mIiAcnEAc/KwoKBZVT1gnT3rkwgTgNlXw2hqT/Zcf8Jy4IDzbKzL+gYmDCNaju
wAzhzaA05oZTLwhFV1sdc5MSJVJnMJVLpNO1jrhi5g6Oo6EmezM/kE8nzoXbmTlP
JjOApuATvUdFlzCCBA0wggL1oAMCAQICCwQAAAAASOeD6yzMA0GCSqGSIb3DQEB
BQUAMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAw
DgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcN
OTkwMTI4MTMwMDAwWhcNMTcwMTI3MTIwMDAwWjCBgTELMAkGA1UEBhMCQkUxGTAX
BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExJTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0
IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMTJ0dsb2JhbFNpZ24gUHJpbWFyeSBPYmpl
Y3QgUHVibGlzaGluZyBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
AKKbdSqnE7oJcSQY36EGYikSntyedXPo31ZXaZYTVk/yyLwBWO0mhnILYPUZxVUD
V5u5EMmh1HRA/2wA6OZTN/632nk+uFI46YEsnw4zUqbNcM5KXWL00WdevJdKB8q8
3Y1Hsc3xZVuFAbBLa97Nji71UOijnJ0mmGs2Y0EDcETwX+IldXlQfV+hBqJGDFWV
RxTTkUaGaJnnJ/SU7JpBUfeW1HqM4USXaHED2FhvvbQQQu4NZnVGi0SW0jAAEgdj
90SbAXDKVm+cWJcqJxeLLnFSbUarpysPfxZIZMhS+gYXAAd010WzDPV4lXPoCu7E
4HKMHhGqHrtezvm0AO5zvc0CAwEAAaOBrjCBqzAOBgNVHQ8BAf8EBAMCAQYwDwYD
VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUFVF5GnwMWfnazdjEOhOayXgtf00wMwYD
VR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nbG9iYWxzaWduLm5ldC9Sb290LmNy
bDATBgNVHSUEDDAKBggrBgEFBQcDAzAfBgNVHSMEGDAWgBRge2YaRQ2XyolQL30E
zTSo//z9SzANBgkqhkiG9w0BAQUFAAOCAQEAtXimonwEt3/Jf31qvHH6KTBgwvRi
Hv5/Qx6bbuKyH3MLhXZbffVOSQYv1Pq3kUDv7W+NjhODVMUqAj0KpNyZC3q9dy/M
QMGP88SMTnK6EHzm/2Qrx85sp/zXmnyORo0Bg01CO9ucP58yYVfXF7CzNmbws/1E
b4E3sZROp1YlifWK1m0RYmJ5XEKQAhjTnCP8COhkRbktfoBbTq/DiimSg3gfkUE0
r4XF/QeZTixc/sf9F7slJTFNcrW1KUtImjdvE8cRTkpFHn4vMZyr6FKv1meXNIhf
DidqZlLRWsesMCwgON0r/zrrzhBFgqJ7G6Egc1abKpPmBFEGbBvcL4mUkzCCBBow
ggMCoAMCAQICCwQAAAAASAZwZBmMA0GCSqGSIb3DQEBBQUAMFcxCzAJBgNVBAYT
AkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENB
MRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcNMDkwMzE4MTEwMDAwWhcN
MjgwMTI4MTIwMDAwWjBUMRgwFgYDVQQLEw9UaW1lc3RhbXBpbmcgQ0ExEzARBgNV
BAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkdsb2JhbFNpZ24gVGltZXN0YW1waW5n
IENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwwy3Eg1NaIoz3jYF
8Dy69drNDlN7Rp+C8mIT18F3rbuBN35PHpOBwQYi2h1QhMaXlZKpk7Y9q4Z5GVR9
DhYETMSIlyzGoahfFTrSZCvMPgx66KRWsR67z4TOjTU6NJxsLcB3tTCpH2fmOglE
OkNyQaKRw0aaH7a5pw+vHHUbZCXnCGwUR/VHGt6O6qJjlX31qK1VomSbcm+5AnM/
OYo5XMT+j/sRnL0QGUlj0EMii9arkpl0FM8wB75Pvf2Kj55a3208zFqZUJC5rcKX
Q8Jf7c0zPYfMwaBbqWI7eH1ko6xNHyvXAxFscVSKsKuxHNZ9I9tABzcm21CvOD2m
B3VvlwIDAQABo4HpMIHmMA4GA1UdDwEB/wQEAwIBBjASBgNVHRMBAf8ECDAGAQH/
AgEAMB0GA1UdDgQWBBTowvHEMtwzNTe8ZXb1nBcuF0Us/jBLBgNVHSAERDBCMEAG
CSsGAQQBoDIBHjAzMDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24u
bmV0L3JlcG9zaXRvcnkvMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwuZ2xv
YmFsc2lnbi5uZXQvcm9vdC5jcmwwHwYDVR0jBBgwFoAUYHtmGkUNl8qJUC99BM00
qP/8/UswDQYJKoZIhvcNAQEFBQADggEBAF32yysNAUCEn4V6Q3Bq4MXnqgYA12cT
yQiRMWVPFKipBdw4nmqgMAq9jceAKO5CRcqU895YRamAMgT1WVxqcAA5J5RN9bRG
NOgcUzGys1QW6cxCq9XZWTAc+0YnJbiHI7HodYgkgx7Idjd7AUlFSKTt4l3SfJyi
3C26EFoSYmWrrgDHEDQ7y3K9FCQM3MN2J7Sn/uFYKfIOFp+ROR2JpuYPHIeM4lis
kn4kPqrsFOc6MzSLxjusg6sPFGJ6uhotTUsbxTDwC5J5fTx44Pjm0hWWWZk5KzBh
6Lj4wKHpIhQReH3E3Im+wLuU4XKu67VAQE/vFx5YXtCoiZaskijpur8wggQuMIID
FqADAgECAgsBAAAAAAElsLTMATANBgkqhkiG9w0BAQUFADBUMRgwFgYDVQQLEw9U
aW1lc3RhbXBpbmcgQ0ExEzARBgNVBAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkds
b2JhbFNpZ24gVGltZXN0YW1waW5nIENBMB4XDTA5MTIyMTA5MzI1NloXDTIwMTIy
MjA5MzI1NlowUjELMAkGA1UEBhMCQkUxFjAUBgNVBAoTDUdsb2JhbFNpZ24gTlYx
KzApBgNVBAMTIkdsb2JhbFNpZ24gVGltZSBTdGFtcGluZyBBdXRob3JpdHkwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDNwj1ddyLQwn04MsMVgx9CajtT
Zt1qNkQNac9ojYlFn34v7kI6M3w+ANOXatha1cNNkgpfBlD9v2zEA6KCYNjtUi4T
dN6XxkUhe1X26rFkA/x0a7Jfx2xsQxSKJBA3SZWB0kgSpaJ2SVAhf8qFcwo8XbUu
rZCqXk0yyxeT2X+WwMCJZVbZxbE/mBsn+knuHRvLBowwHDvFp3BbqKsYWv7I9o6/
AV2PYZg0D1hR/98y6lRlHBQrbPwMkBln7ZvZ2mOb1loko3SOCCMAoZK1HgvRCKBm

```
# f5Ibo+2AZAJJj7aE79FVjl6pl1rFCAKIlFa/kusqLQY1krU3NjHsw/56O8KFAgMB
# AAGjggEBMIH+MB8GA1UdIwQYMBaAFOjC8cQy3DM1N7xldvWcFy4XRSz+MDwGA1Ud
# HwQ1MDMwMaAvoC2GK2h0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvVGltZXN0YW1w
# aW5nMS5jcmwwHQYDVR0OBBYEFKqqporvpGRz1pXieciP6s+lYCnKMAkGA1UdEwQC
# MAAwDgYDVR0PAQH/BAQDAgeAMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMIMEsGA1Ud
# IAREMEIwQAYJKwYBBAGgMgEeMDMwMQYIKwYBBQUHAgEWJWh0dHA6Ly93d3cuZ2xv
# YmFsc2lnbi5uZXQvcmVwb3NpdG9yeS8wDQYJKoZIhvcNAQEFBQADggEBALyJ7P7m
# NlWTXHnUEXqGgI8XtpOybZuRoVYYEcZV6vYI7a2bnvUrgci73WB7G0eZHm1APh2A
# whPVjgQFL9vnrlKeaIRyoeVKYDz4m9UvRtjDsreTU6ybbEMkJNHx/OlWLjQRWBhD
# 6u//NHRsoMBsf60DGWmIHpVgyru9DLt278cksIHGODHPNq0MOLiQIISbLo8ouZ/2
# ypQnzaw5YVfg45VanHaSMPXeppc9chwqYDKoM02GNTOKXPOk/fcGLOFrSzD1y9ND
# YvhBud59IMsFjI4s9l81/TONQollCDYso4n0WoWLsLl722zLofjSDhu7l3zRJ3m+
# nXw75qdWNNjJkakwggTTMIIDu6ADAgECAgsEAAAAAEjng+vJDANBgkqhkiG9w0B
# AQUFADCBgTELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2Ex
# JTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMT
# J0dsb2JhbFNpZ24gUHJpbWFyeSBPYmplY3QgUHVibGlzaGluZyBDQTAeFw0wNDAx
# MjIxMDAwMDBaFw0xNzAxMjcxMDAwMDBaMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwggEiMA0GCSqGSIb3DQEBAQUA
# A4IBDwAwggEKAoIBAQCwsfKAAHDO7MOMtJftxgmMJm+J32dZgc/eFBNMwrFF4lN1
# QfoHNm+6EXAolHxtcr0HFSVlOgn/hdz6e143hzjkx0sIgJieis1YCQLAwwFJlliI
# iSZZ9W3GucH7GCXt2GJOygpsXXDvztObKQsJxvbuthbUPFSOzF3gr9vdIwkyezKB
# FmIKBst6zzQhtm82trHOy5opNUA+nVh8/62CmPq41YnKNd3LzVcGy5vkv5SogJhf
# d5bwtuerdHlAIaZj6dAHkb2FOLSulqyh/xRz2qVFuE2Gzio879TfKA51qaiIE8Lk
# fGCT8iXMA4SX5k62ny3WtYs0PKvVODrIPcSx+ZTNAgMBAAGjggFnMIIBYzAOBgNV
# HQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQU0lvzSyZL
# pbDnXf1Wf/bxLjhOU6AwSgYDVR0gBEMwQTA/BgkrBgEEAaAyATIwMjAwBggrBgEF
# BQcCARYkaHR0cDovL3d3dy5nbG9iYWxzaWduLm5ldC9yZXBvc2l0b3J5MDkGA1Ud
# HwQyMDAwLqAsoCqGKGh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvcHJpbW9iamVj
# dC5jcmwwTgYIKwYBBQUHAQEEQjBAMD4GCCsGAQUFBzAChjJodHRwOi8vc2VjdXJl
# Lmdsb2JhbHNpZ24ubmV0L2NhY2VydC9QcmltT2JqZWN0LmNydDARBglghkgBhvhC
# AQEEBAMCAAEwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHwYDVR0jBBgwFoAUFVF5GnwM
# WfnazdjEOhOayXgtf00wDQYJKoZIhvcNAQEFBQADggEBAB5q8230jqki/nAIZS6h
# XaszMN1sePpL6q3FjewQemrFWJc5a5LzkeIMpygc0V12josHfBNvrcQ2Q7PBvDFZ
# zxg42KM7zv/KZ1i/4PGsYT6iOx68AltBrERr9Sbz7V6oZfbKZaY/yvV366WGKlgp
# Vvi+FhBA6dL8VyxjYTdmJTkgLgcDoDYDJZS9fOt+06PCxXYWdTCSuf92QTUhaNEO
# XlyOwwNg5oBA/MBdolRubpJnp4ESh6KjK9u3Tf/k1cflBebV8a78zWYYIfM+R8nl
# lUJhLJ0mgLIPqD0Oyad43250jCxG9nLpPGRrKFXES2Qzy3hUEzjw1XEG1D4NCjUO
# 4LMxggQtMIIEKQIBATByMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxT
# aWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYDVQQDExhHbG9i
# YWxTaWduIE9iamVjdFNpZ24gQ0ECCwEAAAAAAR5GQJ02MAkGBSsOAwIaBQCgeDAY
# BgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgorBgEEAYI3
# AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3DQEJBDEW
# BBSc/jROV5d39PvZ2Y9F+FjTqQKbEzANBgkqhkiG9w0BAQEFAASBgK9/yMLIw1kO
# WPFhdX+8ZfqmCIINzryUkpvHwAcxV7FJY2Lzr3ZOVJ14Drs7pel3gq0K/PllJbyb
# cd8MbMzC3BOj1k7FUJpoSx/FYog+n/NrNd9Q3N9ZXfwAg7CIgMaif1PEd5+mBQRf
# xiKWTo5diAJx59VXnehWLxAPonomxZs8oYIClzCCApMGCSqGSIb3DQEJBjGCAoQw
# ggKAAgEBMGMwVDEYMBYGA1UECxMPVGltZXN0YW1waW5nIENBMRMwEQYDVQQKEwpH
# bG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxTaWduIFRpbWVzdGFtcGluZyBDQQIL
# AQAAAAABJbC0zAEwCQYFKw4DAhoFAKCB9zAYBgkqhkiG9w0BCQMxCwYJKoZIhvcN
# AQcBMBwGCSqGSIb3DQEJBTEPFw0xMTA0MTkwNjA4NDBaMCMGCSqGSIb3DQEJBDEW
# BBTNweDlA/d07tGGgBH8sQBRYsOVYzCBlwYLKoZIhvcNAQkQAgwxgYcwgYQwgYEw
# fwQUrt9992u6JBDWfbrxj1uhW0F+SWwwZzBYpFYwVDEYMBYGA1UECxMPVGltZXN0
# YW1waW5nIENBMRMwEQYDVQQKEwpHbG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxT
# aWduIFRpbWVzdGFtcGluZyBDQQILAQAAAAABJbC0zAEwDQYJKoZIhvcNAQEBBQAE
# ggEAirS6gL6mQ1nE86e50c/I0dNZCmEmuq7kES4SL0r4Bcd9mcBbxthXB5Nx22rg
# BACXWFrDA2OcPsiByERMjW5DWZ8dSdcs/4q4Jy4y45C11wFeZYpL6oq/r3E6rhz3
# eHpIh36wRw/P1As9LrW+R7t/1CodCxCGAVN43LSBFvb4tOooQOVyHnkR6GSLpno4
# K/NlGCGHtEIZb07fyUtLS9FKmG38S0Gl0ZtEel2lbGoKbSNAWrhKVyLFlIuq27Up
# HnLSetuXdO/XJywsNcazY9FTeYzFe0dQCu8+OCZstGvLho4acG3xrncnpaltcrdF
# bmWyZT/qsEE7pr4j1av8R89Cjg==
# SIG # End signature block
```

VisioAssembly.psd1

```
#############################################################################################################
#####################
```

```
# File:             VisioAssembly.psd1
#
# Author:           Poshoholic
#
# Publisher:        Poshoholic Studios
#
# Copyright:        © 2011 Poshoholic Studios. All rights reserved.
#
# Usage:            To load this module in your Script Editor:
#
#                   1. Open the Script Editor.
#
#                   2. Select "PowerShell Libraries" from the File menu.
#
#                   3. Check the VisioAssembly module.
#
#                   4. Click on OK to close the "PowerShell Libraries" dialog.
#
#                   Alternatively you can load the module from the embedded console by invoking
this:                 #
#                       Import-Module -Name Visio\VisioAssembly
#
#                   Please provide feedback on the PowerGUI Forums.
#
################################################################################################
####################

@{

# Script module or binary module file associated with this manifest
ModuleToProcess = 'VisioAssembly.psm1'

# Version number of this module.
ModuleVersion = '1.0.0.0'

# ID used to uniquely identify this module
GUID = '{79ec367b-a879-4156-a303-d403b6bd73ab}'

# Author of this module
Author = 'Poshoholic'

# Company or vendor of this module
CompanyName = 'Poshoholic Studios'

# Copyright statement for this module
Copyright = '© 2011 Poshoholic Studios. All rights reserved.'

# Description of the functionality provided by this module
Description = 'Commands for loading Visio assemblies for Microsoft Office document automation'

# Minimum version of the Windows PowerShell engine required by this module
PowerShellVersion = '2.0'

# Minimum version of the .NET Framework required by this module
DotNetFrameworkVersion = '2.0'

# Minimum version of the common language runtime (CLR) required by this module
CLRVersion = '2.0.50727'

# Processor architecture (None, X86, Amd64, IA64) required by this module
ProcessorArchitecture = 'None'

# Modules that must be imported into the global environment prior to importing
# this module
RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
RequiredAssemblies = @()

# Script files (.ps1) that are run in the caller's environment prior to
```

```
# importing this module
ScriptsToProcess = @()

# Type files (.ps1xml) to be loaded when importing this module
TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
FormatsToProcess = @()

# Modules to import as nested modules of the module specified in
# ModuleToProcess
NestedModules = @()

# Functions to export from this module
FunctionsToExport = '*'

# Cmdlets to export from this module
CmdletsToExport = '*'

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module
AliasesToExport = '*'

# List of all modules packaged with this module
ModuleList = @()

# List of all files packaged with this module
FileList = @(
  '.\VisioAssembly.psm1'
  '.\VisioAssembly.psd1'
)

# Private data to pass to the module specified in ModuleToProcess
PrivateData = @{}

}

# SIG # Begin signature block
# MIIdfwYJKoZIhvcNAQcCoIIdcDCCHWwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUZ1xR8TO6y4M7o87oWnwFGiOL
# eFCgghi8MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAw
# VzELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
# BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5
# MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJH
# bG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
# AQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6a
# rymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCO
# XkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6T
# RGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux
# 2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlD
# SgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
# HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkq
# hkiG9w0BAQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoR
# SLblCKOzyj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQL
# cFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrg
# lfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr
# +WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XS
# QRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4DCCBAcwggLvoAMCAQICCwEA
# AAAAR5GQJ02MA0GCSqGSIb3DQEBBQUAMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwHhcNMDgxMjE3MTc0ODAyWhcN
# MTExMjE3MTc0ODAyWjBhMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOUXVlc3QgU29m
# dHdhcmUxFzAVBgNVBAMTDlF1ZXN0IFNvZnR3YXJlMSAwHgYJKoZIhvcNAQkBFhFz
# dXBwb3J0QHF1ZXN0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1mza
# 2hKiiqZnaF1sHhuFRS7MEGq9tYhF7AFbJRvTvhCZk9sxK92thKBFyDSOzJauB7Zt
# j+1HwQzpqbbU94EsR09JOf8vB+xQKLCxaBP5YjwhjJzVy+1d6frVWYN1oVxPXRBM
```

```
# G7BnFgfRkOdtsg/Qn1Uqn1ENSozyjTuh5iduUy0CAwEAAaOCAUAwggE8MB8GA1Ud
# IwQYMBaAFNJb80smS6Ww5139Vn/28S44TlOgME4GCCsGAQUFBwEBBEIwQDA+Bggr
# BgEFBQcwAoYyaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLm5ldC9jYWNlcnQvT2Jq
# ZWN0U2lnbi5jcnQwOQYDVR0fBDIwMDAuoCygKoYoaHR0cDovL2NybC5nbG9iYWxz
# aWduLm5ldC9PYmplY3RTaWduLmNybDAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIH
# gDATBgNVHSUEDDAKBggrBgEFBQcDAzBLBgNVHSAERDBCMEAGCSsGAQQBoDIBMjAz
# MDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24ubmV0L3JlcG9zaXRv
# cnkvMBEGCWCGSAGG+EIBAQQEAwIEEDANBgkqhkiG9w0BAQUFAAOCAQEAG9hUuQek
# ddDJ/pzfqo9p4hzKBkeKcVsunEeTUMNg90XzgdOYRFJPCD7T+gXXrTs6Y2xFmLJN
# G/2lQsjQ/32cBBN9zZdbX+ExhFfEV9/w0gbw3H/PfYkCRvp9VZlTafIt4MJCt/Zp
# guPQgggpWadScg7jQNyeHEg6H6c3WHO8PMiKcKJp9LuM1PKX9Bjy6F2k8rbdEAyJ
# u0mIiAcnEAc/KwoKBZVT1gnT3rkwgTgNlXw2hqT/Zcf8Jy4IDzbKzL+gYmDCNaju
# wAzhzaA05oZTLwhFV1sdc5MSJVJnMJVLpNO1jrhi5g6Oo6EmezM/kE8nzoXbmTlP
# JjOApuATvUdFlzCCBA0wggL1oAMCAQICCwQAAAAAASOeD6yzMA0GCSqGSIb3DQEB
# BQUAMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAw
# DgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcN
# OTkwMTI4MTMwMDAwWhcNMTcwMTI3MTIwMDAwWjCBgTELMAkGA1UEBhMCQkUxGTAX
# BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExJTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0
# IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMTJ0dsb2JhbFNpZ24gUHJpbWFyeSBPYmpl
# Y3QgUHVibGlzaGluZyBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
# AKKbdSqnE7oJcSQY36EGYikSntyedXPo31ZXaZYTVk/yyLwBWO0mhnILYPUZxVUD
# V5u5EMmh1HRA/2wA6OZTN/632nk+uFI46YEsnw4zUqbNcM5KXWL00WdevJdKB8q8
# 3Y1Hsc3xZVuFAbBLa97Nji71UOijnJ0mmGs2Y0EDcETwX+IldXlQfV+hBqJGDFWV
# RxTTkUaGaJnnJ/SU7JpBUfeW1HqM4USXaHED2FhvvbQQQu4NZnVGi0SW0jAAEgdj
# 90SbAXDKVm+cWJcqJxeLLnFSbUarpysPfxZIZMhS+gYXAAd010WzDPV4lXPoCu7E
# 4HKMHhGqHrtezvm0AO5zvc0CAwEAAaOBrjCBqzAOBgNVHQ8BAf8EBAMCAQYwDwYD
# VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUFVF5GnwMWfnazdjEOhOayXgtf00wMwYD
# VR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nbG9iYWxzaWduLm5ldC9Sb290LmNy
# bDATBgNVHSUEDDAKBggrBgEFBQcDAzAfBgNVHSMEGDAWgBRge2YaRQ2XyolQL30E
# zTSo//z9SzANBgkqhkiG9w0BAQUFAAOCAQEAtXimonwEt3/Jf31qvHH6KTBgwvRi
# Hv5/Qx6bbuKyH3MLhXZbffVOSQYv1Pq3kUDv7W+NjhODVMUqAj0KpNyZC3q9dy/M
# QMGP88SMTnK6EHzm/2Qrx85sp/zXmnyORo0Bg01CO9ucP58yYVfXF7CzNmbws/1E
# b4E3sZROp1YlifWK1m0RYmJ5XEKQAhjTnCP8COhkRbktfoBbTq/DiimSg3gfkUE0
# r4XF/QeZTixc/sf9F7slJTFNcrW1KUtImjdvE8cRTkpFHn4vMZyr6FKv1meXNIhf
# DidqZlLRWsesMCwgON0r/zrrzhBFgqJ7G6Egc1abKpPmBFEGbBvcL4mUkzCCBBow
# ggMCoAMCAQICCwQAAAAAASAZwZBmMA0GCSqGSIb3DQEBBQUAMFcxCzAJBgNVBAYT
# AkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENB
# MRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcNMDkwMzE4MTEwMDAwWhcN
# MjgwMTI4MTIwMDAwWjBUMRgwFgYDVQQLEw9UaW1lc3RhbXBpbmcgQ0ExEzARBgNV
# BAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkdsb2JhbFNpZ24gVGltZXN0YW1waW5n
# IENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwwy3Eg1NaIoz3jYF
# 8Dy69drNDlN7Rp+C8mIT18F3rbuBN35PHpOBwQYi2h1QhMaXlZKpk7Y9q4Z5GVR9
# DhYETMSIlyzGoahfFTrSZCvMPgx66KRWsR67z4TOjTU6NJxsLcB3tTCpH2fmOglE
# OkNyQaKRw0aaH7a5pw+vHHUbZCXnCGwUR/VHGt6O6qJjlX31qK1VomSbcm+5AnM/
# OYo5XMT+j/sRnL0QGUlj0EMii9arkpl0FM8wB75Pvf2Kj55a3208zFqZUJC5rcKX
# Q8Jf7c0zPYfMwaBbqWI7eH1ko6xNHyvXAxFscVSKsKuxHNZ9I9tABzcm21CvOD2m
# B3VvlwIDAQABo4HpMIHmMA4GA1UdDwEB/wQEAwIBBjASBgNVHRMBAf8ECDAGAQH/
# AgEAMB0GA1UdDgQWBBTowvHEMtwzNTe8ZXb1nBcuF0Us/jBLBgNVHSAERDBCMEAG
# CSsGAQQBoDIBHjAzMDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24u
# bmV0L3JlcG9zaXRvcnkvMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwuZ2xv
# YmFsc2lnbi5uZXQvcm9vdC5jcmwwHwYDVR0jBBgwFoAUYHtmGkUNl8qJUC99BM00
# qP/8/UswDQYJKoZIhvcNAQEFBQADggEBAF32yysNAUCEn4V6Q3Bq4MXnqgYA12cT
# yQiRMWVPFKipBdw4nmqgMAq9jceAKO5CRcqU895YRamAMgT1WVxqcAA5J5RN9bRG
# NOgcUzGys1QW6cxCq9XZWTAc+0YnJbiHI7HodYgkgx7Idjd7AUlFSKTt4l3SfJyi
# 3C26EFoSYmWrrgDHEDQ7y3K9FCQM3MN2J7Sn/uFYKfIOFp+ROR2JpuYPHIeM4lis
# kn4kPqrsFOc6MzSLxjusg6sPFGJ6uhotTUsbxTDwC5J5fTx44Pjm0hWWWZk5KzBh
# 6Lj4wKHpIhQReH3E3Im+wLuU4XKu67VAQE/vFx5YXtCoiZaskijpur8wggQuMIID
# FqADAgECAgsBAAAAAAElsLTMATANBgkqhkiG9w0BAQUFADBUMRgwFgYDVQQLEw9U
# aW1lc3RhbXBpbmcgQ0ExEzARBgNVBAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkds
# b2JhbFNpZ24gVGltZXN0YW1waW5nIENBMB4XDTA5MTIyMTA5MzI1NloXDTIwMTIy
# MjA5MzI1NlowUjELMAkGA1UEBhMCQkUxFjAUBgNVBAoTDUdsb2JhbFNpZ24gTlYx
# KzApBgNVBAMTIkdsb2JhbFNpZ24gVGltZSBTdGFtcGluZyBBdXRob3JpdHkwggEi
# MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDNwj1ddyLQwn04MsMVgx9CajtT
# Zt1qNkQNac9ojYlFn34v7kI6M3w+ANOXatha1cNNkgpfBlD9v2zEA6KCYNjtUi4T
# dN6XxkUhe1X26rFkA/x0a7Jfx2xsQxSKJBA3SZWB0kgSpaJ2SVAhf8qFcwo8XbUu
# rZCqXk0yyxeT2X+WwMCJZVbZxbE/mBsn+knuHRvLBowwHDvFp3BbqKsYWv7I9o6/
# AV2PYZg0D1hR/98y6lRlHBQRbPwMkBln7ZvZ2mOb1loko3SOCCMAoZK1HgvRCKBm
# f5Ibo+2AZAJJj7aE79FVjl6pl1rFCAKIlFa/kusqLQY1krU3NjHsw/56O8KFAgMB
# AAGjggEBMIH+MB8GA1UdIwQYMBaAFOjC8cQy3DM1N7xldvWcFy4XRSz+MDwGA1Ud
# HwQ1MDMwMaAvoC2GK2h0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvVGltZXN0YW1w
```

```
# aW5nMS5jcmwwHQYDVR0OBBYEFKqqporvpGRz1pXieciP6s+lYCnKMAkGA1UdEwQC
# MAAwDgYDVR0PAQH/BAQDAgeAMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMIMEsGA1Ud
# IAREMEIwQAYJKwYBBAGgMgEeMDMwMQYIKwYBBQUHAgEWJWh0dHA6Ly93d3cuZ2xv
# YmFsc2lnbi5uZXQvcmVwb3NpdG9yeS8wDQYJKoZIhvcNAQEFBQADggEBALyJ7P7m
# NlWTXHnUEXqGgI8XtpOybZuRoVYYEcZV6vYI7a2bnvUrgci73WB7G0eZHm1APh2A
# whPVjgQFL9vnrlKeaIRyoeVKYDz4m9UvRtjDsreTU6ybbEMkJNHx/OlWLjQRWBhD
# 6u//NHRsoMBsf60DGWmIHpVgyru9DLt278cksIHGODHPNq0MOLiQIISbLo8ouZ/2
# ypQnzaw5YVfg45VanHaSMPXeppc9chwqYDKoM02GNTOKXPOk/fcGLOFrSzD1y9ND
# YvhBud59IMsFjI4s9l81/TONQollCDYso4n0WoWLsLl722zLofjSDhu7l3zRJ3m+
# nXw75qdWNNjJkakwggTTMIIDu6ADAgECAgsEAAAAAEjng+vJDANBgkqhkiG9w0B
# AQUFADCBgTELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2Ex
# JTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMT
# J0dsb2JhbFNpZ24gUHJpbWFyeSBPYmplY3QgUHVibGlzaGluZyBDQTAeFw0wNDAx
# MjIxMDAwMDBaFw0xNzAxMjcxMDAwMDBaMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwggEiMA0GCSqGSIb3DQEBAQUA
# A4IBDwAwggEKAoIBAQCwsfKAAHDO7MOMtJftxgmMJm+J32dZgc/eFBNMwrFF4lN1
# QfoHNm+6EXAolHxtcr0HFSVlOgn/hdz6e143hzjkx0sIgJieis1YCQLAwwFJlliI
# iSZZ9W3GucH7GCXt2GJOygpsXXDvztObKQsJxvbuthbUPFSOzF3gr9vdIwkyezKB
# FmIKBst6zzQhtm82trHOy5opNUA+nVh8/62CmPq41YnKNd3LzVcGy5vkv5SogJhf
# d5bwtuerdHlAIaZj6dAHkb2FOLSulqyh/xRz2qVFuE2Gzio879TfKA51qaiIE8Lk
# fGCT8iXMA4SX5k62ny3WtYs0PKvVODrIPcSx+ZTNAgMBAAGjggFnMIIBYzAOBgNV
# HQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQU0lvzSyZL
# pbDnXf1Wf/bxLjhOU6AwSgYDVR0gBEMwQTA/BgkrBgEEAaAyATIwMjAwBggrBgEF
# BQcCARYkaHR0cDovL3d3dy5nbG9iYWxzaWduLm5ldC9yZXBvc2l0b3J5MDkGA1Ud
# HwQyMDAwLqAsoCqGKGh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvcHJpbW9iamVj
# dC5jcmwwTgYIKwYBBQUHAQEEQjBAMD4GCCsGAQUFBzAChjJodHRwOi8vc2VjdXJl
# Lmdsb2JhbHNpZ24ubmV0L2NhY2VydC9QcmltT2JqZWN0LmNydDARBglghkgBhvhC
# AQEEBAMCAAEwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHwYDVR0jBBgwFoAUFVF5GnwM
# WfnazdjEOhOayXgtf00wDQYJKoZIhvcNAQEFBQADggEBAB5q8230jqki/nAIZS6h
# XaszMN1sePpL6q3FjewQemrFWJc5a5LzkeIMpygc0V12josHfBNvrcQ2Q7PBvDFZ
# zxg42KM7zv/KZ1i/4PGsYT6iOx68AltBrERr9Sbz7V6oZfbKZaY/yvV366WGKlgp
# Vvi+FhBA6dL8VyxjYTdmJTkgLgcDoDYDJZS9fOt+06PCxXYWdTCSuf92QTUhaNEO
# XlyOwwNg5oBA/MBdolRubpJnp4ESh6KjK9u3Tf/k1cflBebV8a78zWYYIfM+R8nl
# lUJhLJ0mgLIPqD0Oyad43250jCxG9nLpPGRrKFXES2Qzy3hUEzjw1XEG1D4NCjUO
# 4LMxggQtMIIEKQIBATByMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxT
# aWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYDVQQDExhHbG9i
# YWxTaWduIE9iamVjdFNpZ24gQ0ECCwEAAAAAAR5GQJ02MAkGBSsOAwIaBQCgeDAY
# BgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgorBgEEAYI3
# AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3DQEJBDEW
# BBSLjrUp7sb0Tx/KLAmlgbBrmhVzBjANBgkqhkiG9w0BAQEFAASBgHtIJI+RVtnS
# GJOnmhSahkxXar+yb/jYjqrB74gjlaz7s20GjsNMQH9s8naC5NoZ81aRaa3Zcu4e
# ++XmkkeiSW1JsIUtV0L+r3JHpW5m2vFnjlfHLnpgX1lLgYhKY6HslXHJX370tInT
# yxr5n13qk4fMq6uBGbiWAaABT+rDr2TooYIClzCCApMGCSqGSIb3DQEJBjGCAoQw
# ggKAAgEBMGMwVDEYMBYGA1UECxMPVGltZXN0YW1waW5nIENBMRMwEQYDVQQKEwpH
# bG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxTaWduIFRpbWVzdGFtcGluZyBDQQIL
# AQAAAAABJbC0zAEwCQYFKw4DAhoFAKCB9zAYBgkqhkiG9w0BCQMxCwYJKoZIhvcN
# AQcBMBwGCSqGSIb3DQEJBTEPFw0xMTA0MTcyMjE5MjJaMCMGCSqGSIb3DQEJBDEW
# BBTU/NehKhb51ZvWNO9fE4HFVoGhxjCBlwYLKoZIhvcNAQkQAgwxgYcwgYQwgYEw
# fwQUrt9992u6JBDWfbrxj1uhW0F+SWwwZzBYpFYwVDEYMBYGA1UECxMPVGltZXN0
# YW1waW5nIENBMRMwEQYDVQQKEwpHbG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxT
# aWduIFRpbWVzdGFtcGluZyBDQQILAQAAAAABJbC0zAEwDQYJKoZIhvcNAQEBBQAE
# ggEAC37dts2J269FmZ6+w6DPJvqLT+jKVtgpE6PyeWRzVZcx2gXHd8O0ZQa6iWjs
# YyxlFF9hlEofcNhkrk11eWMmYp7kbA0raTnf9zHyEZbLclM4luX+DD+GYCi5HoTP
# WvRjTr65jcn0nVDjmZ2GcuYGw2z7Mi4UyjSOTC8t0ZUl1gXoBLclm9Ft0pwOEKIh
# Ef/zeG+DV9s8JEpSdCGm108cDOM8bZ1XcLy50HmCCc5ULCgMlZB6PeODwcU++I1d
# QzI49wKqy5M3Gwu2/40HH2igbUOAiIng2XhLu0wfDKZHv2Vy8XFKQkedLenfT8j4
# rPg34TaLnc1fOS0Z9zi6xyaRaQ==
# SIG # End signature block
```

VisioAssembly.psm1

```
#############################################################################################################
#####################
# File:             VisioAssembly.psm1
#
# Author:           Poshoholic
#
```

```
# Publisher:        Poshoholic Studios
#
# Copyright:        © 2011 Poshoholic Studios. All rights reserved.
#
# Usage:            To load this module in your Script Editor:
#
#                   1. Open the Script Editor.
#
#                   2. Select "PowerShell Libraries" from the File menu.
#
#                   3. Check the VisioAssembly module.
#
#                   4. Click on OK to close the "PowerShell Libraries" dialog.
#
#                   Alternatively you can load the module from the embedded console by invoking
this:                 #
#                       Import-Module -Name Visio\VisioAssembly
#
#                   Please provide feedback on the PowerGUI Forums.
#
##############################################################################################
####################

Set-StrictMode -Version 2

#region Set the module export mode to explicit.

Export-ModuleMember

#endregion

#region OfficeAssembly functions.

function Import-OfficeAssembly {
  [CmdletBinding()]
  [OutputType([System.Reflection.Assembly])]
  param(
    [Parameter(Position=0, Mandatory=$true)]

[ValidateSet('Outlook','Word','Excel','PowerPoint','Access','OneNote','Visio','Project','Publisher'
,'InfoPath')]
    [Alias('Name')]
    [System.String]
    $ProductName,

    [Parameter()]
    [System.Management.Automation.SwitchParameter]
    $PassThru
  )

  #region Initialize local variables.
  [System.Reflection.Assembly]$officeAssembly = $null
  #endregion

  #region Define helper functions.
  function private:Load-OfficeAssembly {
    #region Initialize local variables.
    [System.Reflection.Assembly]$assembly = $null
    #endregion

    #region Load the Office assembly.
    foreach ($supportedVersion in @('15','14','12','11')) {
      trap {
        continue
      }
      if ($assembly = [System.Reflection.Assembly]::Load("Microsoft.Office.Interop.${ProductName},
Version=${supportedVersion}.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c")) {
        break
      }
    }
```

```
    #endregion

    #region Return the assembly that was loaded to the client.
    if ($assembly) {
      $assembly
    }
    #endregion
  }
  #endregion

  #region Load the Office assembly if it is installed on the local system.
  $officeAssembly = Load-OfficeAssembly
  #endregion

  #region Throw an error if the Office product is not installed.
  if (-not $officeAssembly) {
    throw $(New-Object System.InvalidOperationException "${ProductName} 2003 or later is
required.")
    return
  }
  #endregion

  #region Return the Office assembly object if requested.
  if ($PassThru) {
    $officeAssembly
  }
  #endregion
}
Export-ModuleMember -Function Import-OfficeAssembly

#endregion

#region VisioAssembly functions.

function Import-VisioAssembly {
  <#
  .ForwardHelpTargetName Import-OfficeAssembly
  .ForwardHelpCategory Function
  #>
  [CmdletBinding()]
  [OutputType([System.Reflection.Assembly])]
  param(
    [Switch]
    $PassThru
  )
  begin {
    try {
      #region Initialize local variables.
      [System.String]$commandName = 'Import-OfficeAssembly'
      [System.Int32]$outBuffer = $null
      [System.Management.Automation.CommandInfo]$wrappedCmd = $null
      #endregion

      #region Ensure that objects are sent through the pipeline one at a time.
      if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer)) {
        $PSBoundParameters['OutBuffer'] = 1
      }
      #endregion

      #region Look up the command being proxied.
      $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand($commandName,
[System.Management.Automation.CommandTypes]::Function)
      #endregion

      #region If the command was not found, throw an appropriate command not found exception.
      if (-not $wrappedCmd) {
        [System.String]$errorMessage =
$PSCmdlet.GetResourceString('DiscoveryExceptions','CommandNotFoundException')
        [System.Management.Automation.CommandNotFoundException]$exception = New-Object -TypeName
System.Management.Automation.CommandNotFoundException -ArgumentList ($errorMessage -f $commandName)
```

```
        $exception.CommandName = $commandName
        [System.Management.Automation.ErrorRecord]$errorRecord = New-Object -TypeName
System.Management.Automation.ErrorRecord -ArgumentList
$exception,'DiscoveryExceptions',([System.Management.Automation.ErrorCategory]::ObjectNotFound),$co
mmandName
        $PSCmdlet.ThrowTerminatingError($errorRecord)
      }
      #endregion

      #region Create the proxy command script block.
      $scriptCmd = {& $wrappedCmd -ProductName Visio @PSBoundParameters}
      #endregion

      #region Use the script block to create the steppable pipeline, then invoke its begin block.
      $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
      $steppablePipeline.Begin($PSCmdlet)
      #endregion
    } catch {
      throw
    }
  }
  process {
    try {
      #region Process the element that was just received from the previous stage in the pipeline.
      $steppablePipeline.Process($_)
      #endregion
    } catch {
      throw
    }
  }
  end {
    try {
      #region Close the pipeline.
      $steppablePipeline.End()
      #endregion
    } catch {
      throw
    }
  }
}
Export-ModuleMember -Function Import-VisioAssembly

#endregion

# SIG # Begin signature block
# MIIdfwYJKoZIhvcNAQcCoIIdcDCCHWwCAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU3cmZQgpHmcA8lP2MogLjIFHO
# u8Kgghi8MIIDdTCCAl2gAwIBAgILBAAAAAABFUtaw5QwDQYJKoZIhvcNAQEFBQAw
# VzELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExEDAOBgNV
# BAsTB1Jvb3QgQ0ExGzAZBgNVBAMTEkdsb2JhbFNpZ24gUm9vdCBDQTAeFw05ODA5
# MDExMjAwMDBaFw0yODAxMjgxMjAwMDBaMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENBMRswGQYDVQQDExJH
# bG9iYWxTaWduIFJvb3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
# AQDaDuaZjc6j40+Kfvvxi4Mla+pIH/EqsLmVEQS98GPR4mdmzxzdzxtIK+6NiY6a
# rymAZavpxy0Sy6scTHAHoT0KMM0VjU/43dSMUBUc71DuxC73/OlS8pF94G3VNTCO
# XkNz8kHp1Wrjsok6Vjk4bwY8iGlbKk3Fp1S4bInMm/k8yuX9ifUSPJJ4ltbcdG6T
# RGHRjcdGsnUOhugZitVtbNV4FpWi6cgKOOvyJBNPc1STE4U6G7weNLWLBYy5d4ux
# 2x8gkasJU26Qzns3dLlwR5EiUWMWea6xrkEmCMgZK9FGqkjWZCrXgzT/LCrBbBlD
# SgeF59N89iFo7+ryUp9/k5DPAgMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV
# HRMBAf8EBTADAQH/MB0GA1UdDgQWBBRge2YaRQ2XyolQL30EzTSo//z9SzANBgkq
# hkiG9w0BAQUFAAOCAQEA1nPnfE920I2/7LqivjTFKDK1fPxsnCwrvQmeU79rXqoR
# SLblCKOzyj1hTdNGCbM+w6DjY1Ub8rrvrTnhQ7k4o+YviiY776BQVvnGCv04zcQL
# cFGUl5gE38NflNUVyRRBnMRddWQVDf9VMOyGj/8N7yy5Y0b2qvzfvGn9LhJIZJrg
# lfCm7ymPAbEVtQwdpf5pLGkkeB6zpxxxYu7KyJesF12KwvhHhm4qxFYxldBniYUr
# +WymXUadDKqC5JlR3XC321Y9YeRq4VzW9v493kHMB65jUr9TU/Qr6cf9tveCX4XS
# QRjbgbMEHMUfpIBvFSDJ3gyICh3WZlXi/EjJKSZp4DCCBAcwggLvoAMCAQICCwEA
# AAAAR5GQJ02MA0GCSqGSIb3DQEBBQUAMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwHhcNMDgxMjE3MTc0ODAyWhcN
```

```
# MTExMjE3MTc0ODAyWjBhMQswCQYDVQQGEwJVUzEXMBUGA1UEChMOUXVlc3QgU29m
# dHdhcmUxFzAVBgNVBAMTDlF1ZXN0IFNvZnR3YXJlMSAwHgYJKoZIhvcNAQkBFhFz
# dXBwb3J0QHF1ZXN0LmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA1mza
# 2hKiiqZnaF1sHhuFRS7MEGq9tYhF7AFbJRvTvhCZk9sxK92thKBFyDSOzJauB7Zt
# j+1HwQzpqbbU94EsR09JOf8vB+xQKLCxaBP5YjwhjJzVy+1d6frVWYN1oVxPXRBM
# G7BnFgfRkOdtsg/Qn1Uqn1ENSozyjTuh5iduUy0CAwEAAaOCAUAwggE8MB8GA1Ud
# IwQYMBaAFNJb80smS6Ww5139Vn/28S44TlOgME4GCCsGAQUFBwEBBEIwQDA+Bggr
# BgEFBQcwAoYyaHR0cDovL3NlY3VyZS5nbG9iYWxzaWduLm5ldC9jYWNlcnQvT2Jq
# ZWN0U2lnbi5jcnQwOQYDVR0fBDIwMDAuoCygKoYoaHR0cDovL2NybC5nbG9iYWxz
# aWduLm5ldC9PYmplY3RTaWduLmNybDAJBgNVHRMEAjAAMA4GA1UdDwEB/wQEAwIH
# gDATBgNVHSUEDDAKBggrBgEFBQcDAzBLBgNVHSAERDBCMEAGCSsGAQQBoDIBMjAz
# MDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24ubmV0L3JlcG9zaXRv
# cnkvMBEGCWCGSAGG+EIBAQQEAwIEEDANBgkqhkiG9w0BAQUFAAOCAQEAG9hUuQek
# ddDJ/pzfqo9p4hzKBkeKcVsunEeTUMNg90XzgdOYRFJPCD7T+gXXrTs6Y2xFmLJN
# G/2lQsjQ/32cBBN9zZdbX+ExhFfEV9/w0gbw3H/PfYkCRvp9VZlTafIt4MJCt/Zp
# guPQgggpWadScg7jQNyeHEg6H6c3WHO8PMiKcKJp9LuM1PKX9Bjy6F2k8rbdEAyJ
# u0mIiAcnEAc/KwoKBZVT1gnT3rkwgTgNlXw2hqT/Zcf8Jy4IDzbKzL+gYmDCNaju
# wAzhzaA05oZTLwhFV1sdc5MSJVJnMJVLpNO1jrhi5g6Oo6EmezM/kE8nzoXbmTlP
# JjOApuATvUdFlzCCBA0wggL1oAMCAQICCwQAAAAASOeD6yzMA0GCSqGSIb3DQEB
# BQUAMFcxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAw
# DgYDVQQLEwdSb290IENBMRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcN
# OTkwMTI4MTMwMDAwWhcNMTcwMTI3MTIwMDAwWjCBgTELMAkGA1UEBhMCQkUxGTAX
# BgNVBAoTEEdsb2JhbFNpZ24gbnYtc2ExJTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0
# IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMTJ0dsb2JhbFNpZ24gUHJpbWFyeSBPYmpl
# Y3QgUHVibGlzaGluZyBDQTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEB
# AKKbdSqnE7oJcSQY36EGYikSntyedXPo31ZXaZYTVk/yyLwBWO0mhnILYPUZxVUD
# V5u5EMmh1HRA/2wA6OZTN/632nk+uFI46YEsnw4zUqbNcM5KXWL00WdevJdKB8q8
# 3Y1Hsc3xZVuFAbBLa97Nji71UOijnJ0mmGs2Y0EDcETwX+IldXlQfV+hBqJGDFWV
# RxTTkUaGaJnnJ/SU7JpBUfeW1HqM4USXaHED2FhvvbQQQu4NZnVGi0SW0jAAEgdj
# 90SbAXDKVm+cWJcqJxeLLnFSbUarpysPfxZIZMhS+gYXAAd010WzDPV4lXPoCu7E
# 4HKMHhGqHrtezvm0AO5zvc0CAwEAAaOBrjCBqzAOBgNVHQ8BAf8EBAMCAQYwDwYD
# VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUFVF5GnwMWfnazdjEOhOayXgtf00wMwYD
# VR0fBCwwKjAooCagJIYiaHR0cDovL2NybC5nbG9iYWxzaWduLm5ldC9Sb290LmNy
# bDATBgNVHSUEDDAKBggrBgEFBQcDAzAfBgNVHSMEGDAWgBRge2YaRQ2XyolQL30E
# zTSo//z9SzANBgkqhkiG9w0BAQUFAAOCAQEAtXimonwEt3/Jf31qvHH6KTBgwvRi
# Hv5/Qx6bbuKyH3MLhXZbffVOSQYv1Pq3kUDv7W+NjhODVMUqAj0KpNyZC3q9dy/M
# QMGP88SMTnK6EHzm/2Qrx85sp/zXmnyORo0Bg01CO9ucP58yYVfXF7CzNmbws/1E
# b4E3sZROp1YlifWK1m0RYmJ5XEKQAhjTnCP8COhkRbktfoBbTq/DiimSg3gfkUE0
# r4XF/QeZTixc/sf9F7slJTFNcrW1KUtImjdvE8cRTkpFHn4vMZyr6FKv1meXNIhf
# DidqZlLRWsesMCwgON0r/zrrzhBFgqJ7G6Egc1abKpPmBFEGbBvcL4mUkzCCBBow
# ggMCoAMCAQICCwQAAAAASAZwZBmMA0GCSqGSIb3DQEBBQUAMFcxCzAJBgNVBAYT
# AkJFMRkwFwYDVQQKExBHbG9iYWxTaWduIG52LXNhMRAwDgYDVQQLEwdSb290IENB
# MRswGQYDVQQDExJHbG9iYWxTaWduIFJvb3QgQ0EwHhcNMDkwMzE4MTEwMDAwWhcN
# MjgwMTI4MTIwMDAwWjBUMRgwFgYDVQQLEw9UaW1lc3RhbXBpbmcgQ0ExEzARBgNV
# BAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkdsb2JhbFNpZ24gVGltZXN0YW1waW5n
# IENBMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwwy3Eg1NaIoz3jYF
# 8Dy69drNDlN7Rp+C8mIT18F3rbuBN35PHpOBwQYi2h1QhMaXlZKpk7Y9q4Z5GVR9
# DhYETMSIlyzGoahfFTrSZCvMPgx66KRWsR67z4TOjTU6NJxsLcB3tTCpH2fmOglE
# OkNyQaKRw0aaH7a5pw+vHHUbZCXnCGwUR/VHGt6O6qJjlX31qK1VomSbcm+5AnM/
# OYo5XMT+j/sRnL0QGUlj0EMii9arkpl0FM8wB75Pvf2Kj55a3208zFqZUJC5rcKX
# Q8Jf7c0zPYfMwaBbqWI7eH1ko6xNHyvXAxFscVSKsKuxHNZ9I9tABzcm21CvOD2m
# B3VvlwIDAQABo4HpMIHmMA4GA1UdDwEB/wQEAwIBBjASBgNVHRMBAf8ECDAGAQH/
# AgEAMB0GA1UdDgQWBBTowvHEMtwzNTe8ZXb1nBcuF0Us/jBLBgNVHSAERDBCMEAG
# CSsGAQQBoDIBHjAzMDEGCCsGAQUFBwIBFiVodHRwOi8vd3d3Lmdsb2JhbHNpZ24u
# bmV0L3JlcG9zaXRvcnkvMDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwuZ2xv
# YmFsc2lnbi5uZXQvcm9vdC5jcmwwHwYDVR0jBBgwFoAUYHtmGkUNl8qJUC99BM00
# qP/8/UswDQYJKoZIhvcNAQEFBQADggEBAF32yysNAUCEn4V6Q3Bq4MXnqgYA12cT
# yQiRMWVPFKipBdw4nmqgMAq9jceAKO5CRcqU895YRamAMgT1WVxqcAA5J5RN9bRG
# NOgcUzGys1QW6cxCq9XZWTAc+0YnJbiHI7HodYgkgx7Idjd7AUlFSKTt4l3SfJyi
# 3C26EFoSYmWrrgDHEDQ7y3K9FCQM3MN2J7Sn/uFYKfIOFp+ROR2JpuYPHIeM4lis
# kn4kPqrsFOc6MzSLxjusg6sPFGJ6uhotTUsbxTDwC5J5fTx44Pjm0hWWWZk5KzBh
# 6Lj4wKHpIhQReH3E3Im+wLuU4XKu67VAQE/vFx5YXtCoiZaskijpur8wggQuMIID
# FqADAgECAgsBAAAAAAElsLTMATANBgkqhkiG9w0BAQUFADBUMRgwFgYDVQQLEw9U
# aW1lc3RhbXBpbmcgQ0ExEzARBgNVBAoTCkdsb2JhbFNpZ24xIzAhBgNVBAMTGkds
# b2JhbFNpZ24gVGltZXN0YW1waW5nIENBMB4XDTA5MTIyMTA5MzI1NloXDTIwMTIy
# MjA5MzI1NlowUjELMAkGA1UEBhMCQkUxFjAUBgNVBAoTDUdsb2JhbFNpZ24gTlYx
# KzApBgNVBAMTIkdsb2JhbFNpZ24gVGltZSBTdGFtcGluZyBBdXRob3JpdHkwggEi
# MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDNwj1ddyLQwn04MsMVgx9CajtT
# Zt1qNkQNac9ojYlFn34v7kI6M3w+ANOXatha1cNNkgpfBlD9v2zEA6KCYNjtUi4T
# dN6XxkUhe1X26rFkA/x0a7Jfx2xsQxSKJBA3SZWB0kgSpaJ2SVAhf8qFcwo8XbUu
```

```
# rZCqXk0yyxeT2X+WwMCJZVbZxbE/mBsn+knuHRvLBowwHDvFp3BbqKsYWv7I9o6/
# AV2PYZg0D1hR/98y6lRlHBQrbPwMkBln7ZvZ2mOb1loko3SOCCMAoZK1HgvRCKBm
# f5Ibo+2AZAJJj7aE79FVjl6pl1rFCAKIlFa/kusqLQY1krU3NjHsw/56O8KFAgMB
# AAGjggEBMIH+MB8GA1UdIwQYMBaAFOjC8cQy3DM1N7xldvWcFy4XRSz+MDwGA1Ud
# HwQ1MDMwMaAvoC2GK2h0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvVGltZXN0YW1w
# aW5nMS5jcmwwHQYDVR0OBBYEFKqqporvpGRz1pXieciP6s+lYCnKMAkGA1UdEwQC
# MAAwDgYDVR0PAQH/BAQDAgeAMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMIMEsGA1Ud
# IAREMEIwQAYJKwYBBAGgMgEeMDMwMQYIKwYBBQUHAgEWJWh0dHA6Ly93d3cuZ2xv
# YmFsc2lnbi5uZXQvcmVwb3NpdG9yeS8wDQYJKoZIhvcNAQEFBQADggEBALyJ7P7m
# NlWTXHnUEXqGgI8XtpOybZuRoVYYEcZV6vYI7a2bnvUrgci73WB7G0eZHm1APh2A
# whPVjgQFL9vnrlKeaIRyoeVKYDz4m9UvRtjDsreTU6ybbEMkJNHx/OlWLjQRWBhD
# 6u//NHRsoMBsf60DGWmIHpVgyru9DLt278cksIHGODHPNq0MOLiQIISbLo8ouZ/2
# ypQnzaw5YVfg45VanHaSMPXeppc9chwqYDKoM02GNTOKXPOk/fcGLOFrSzD1y9ND
# YvhBud59IMsFjI4s9l81/TONQollCDYso4n0WoWLsLl722zLofjSDhu7l3zRJ3m+
# nXw75qdWNNjJkakwggTTMIIDu6ADAgECAgsEAAAAAEjng+vJDANBgkqhkiG9w0B
# AQUFADCBgTELMAkGA1UEBhMCQkUxGTAXBgNVBAoTEEdsb2JhbFNpZ24gbnYtc2Ex
# JTAjBgNVBAsTHFByaW1hcnkgT2JqZWN0IFB1Ymxpc2hpbmcgQ0ExMDAuBgNVBAMT
# J0dsb2JhbFNpZ24gUHJpbWFyeSBPYmplY3QgUHVibGlzaGluZyBDQTAeFw0wNDAx
# MjIxMDAwMDBaFw0xNzAxMjcxMDAwMDBaMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQK
# ExBHbG9iYWxTaWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYD
# VQQDExhHbG9iYWxTaWduIE9iamVjdFNpZ24gQ0EwggEiMA0GCSqGSIb3DQEBAQUA
# A4IBDwAwggEKAoIBAQCwsfKAAHDO7MOMtJftxgmMJm+J32dZgc/eFBNMwrFF4lN1
# QfoHNm+6EXAolHxtcr0HFSVlOgn/hdz6e143hzjkx0sIgJieis1YCQLAwwFJlliI
# iSZZ9W3GucH7GCXt2GJOygpsXXDvztObKQsJxvbuthbUPFSOzF3gr9vdIwkyezKB
# FmIKBst6zzQhtm82trHOy5opNUA+nVh8/62CmPq41YnKNd3LzVcGy5vkv5SogJhf
# d5bwtuerdHlAIaZj6dAHkb2FOLSulqyh/xRz2qVFuE2Gzio879TfKA51qaiIE8Lk
# fGCT8iXMA4SX5k62ny3WtYs0PKvVODrIPcSx+ZTNAgMBAAGjggFnMIIBYzAOBgNV
# HQ8BAf8EBAMCAQYwEgYDVR0TAQH/BAgwBgEB/wIBADAdBgNVHQ4EFgQU0lvzSyZL
# pbDnXf1Wf/bxLjhOU6AwSgYDVR0gBEMwQTA/BgkrBgEEAaAyATIwMjAwBggrBgEF
# BQcCARYkaHR0cDovL3d3dy5nbG9iYWxzaWduLm5ldC9yZXBvc2l0b3J5MDkGA1Ud
# HwQyMDAwLqAsoCqGKGh0dHA6Ly9jcmwuZ2xvYmFsc2lnbi5uZXQvcHJpbW9iamVj
# dC5jcmwwTgYIKwYBBQUHAQEEQjBAMD4GCCsGAQUFBzAChjJodHRwOi8vc2VjdXJl
# Lmdsb2JhbHNpZ24ubmV0L2NhY2VydC9QcmltT2JqZWN0LmNydDARBglghkgBhvhC
# AQEEBAMCAAEwEwYDVR0lBAwwCgYIKwYBBQUHAwMwHwYDVR0jBBgwFoAUFVF5GnwM
# WfnazdjEOhOayXgtf00wDQYJKoZIhvcNAQEFBQADggEBAB5q8230jqki/nAIZS6h
# XaszMN1sePpL6q3FjewQemrFWJc5a5LzkeIMpygc0V12josHfBNvrcQ2Q7PBvDFZ
# zxg42KM7zv/KZ1i/4PGsYT6iOx68AltBrERr9Sbz7V6oZfbKZaY/yvV366WGKlgp
# Vvi+FhBA6dL8VyxjYTdmJTkgLgcDoDYDJZS9fOt+06PCxXYWdTCSuf92QTUhaNEO
# XlyOwwNg5oBA/MBdolRubpJnp4ESh6KjK9u3Tf/k1cflBebV8a78zWYYIfM+R8nl
# lUJhLJ0mgLIPqD0Oyad43250jCxG9nLpPGRrKFXES2Qzy3hUEzjw1XEG1D4NCjUO
# 4LMxggQtMIIEKQIBATByMGMxCzAJBgNVBAYTAkJFMRkwFwYDVQQKExBHbG9iYWxT
# aWduIG52LXNhMRYwFAYDVQQLEw1PYmplY3RTaWduIENBMSEwHwYDVQQDExhHbG9i
# YWxTaWduIE9iamVjdFNpZ24gQ0ECCwEAAAAAAR5GQJ02MAkGBSsOAwIaBQCgeDAY
# BgorBgEEAYI3AgEMMQowCKACgAChAoAAMBkGCSqGSIb3DQEJAzEMBgorBgEEAYI3
# AgEEMBwGCisGAQQBgjcCAQsxDjAMBgorBgEEAYI3AgEVMCMGCSqGSIb3DQEJBDEW
# BBSiIiXSxsiRuvScSMMv73NR/yShjTANBgkqhkiG9w0BAQEFAASBgHW1N78Oem4s
# f3z+dzjdv0FdDkXuMwVlcxioh7e38BJM3XoWW4WXolhX9i2Ul9j7Fe/xYCXB28v8
# hMt8vn2/XMIgzjSWVBSnVJjp3mk2fwTbx6nkLu4+gPpZum031DseHMuQ5etw7qs/
# rXGZoyjigI8gN/ueAhx1p9DEolF5o2kioYIClzCCApMGCSqGSIb3DQEJBjGCAoQw
# ggKAAgEBMGMwVDEYMBYGA1UECxMPVGltZXN0YW1waW5nIENBMRMwEQYDVQQKEwpH
# bG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxTaWduIFRpbWVzdGFtcGluZyBDQQIL
# AQAAAAABJbC0zAEwCQYFKw4DAhoFAKCB9zAYBgkqhkiG9w0BCQMxCwYJKoZIhvcN
# AQcBMBwGCSqGSIb3DQEJBTEPFw0xMTA0MTcyMjE5MDVaMCMGCSqGSIb3DQEJBDEW
# BBQMCdj+RQ8sBxtlq/wC2c6lg6j0LDCBlwYLKoZIhvcNAQkQAgwxgYcwgYQwgYEw
# fwQUrt9992u6JBDWfbrxj1uhW0F+SWwwZzBYpFYwVDEYMBYGA1UECxMPVGltZXN0
# YW1waW5nIENBMRMwEQYDVQQKEwpHbG9iYWxTaWduMSMwIQYDVQQDExpHbG9iYWxT
# aWduIFRpbWVzdGFtcGluZyBDQQILAQAAAAABJbC0zAEwDQYJKoZIhvcNAQEBBQAE
# ggEAf/Dk5n/4O3YZxgY/pUPaIMhj6kNvfKdY8MvmEHdbZtSYoQA0ckFntcVbUPr6
# 2uIy3en8deDIFsUmKZYS7ETVcclthwJyuc4X9q0V4Q8rACbbHaNwkCmpy+ka5QKR
# uktL0JTEC9DCWjEDIMj4sdDhrArhB7WX9YNm/qh8iq+okmeiDsM1UTfT5odavkvh
# F0znKmmtgB45TWHnxSPxYlz6ORrqe5GkFHPF+E4qxqHfyp+WX6fgSeekC+HILdJA
# FnnQ+EMMI32vW6h/um542lajx3CFkpcsMBxKNNLlAa4uW5+9TJQMcES3i/zKgqzT
# hrJ1SwRZgiAWSDAPpcOf1VJ1Kw==
# SIG # End signature block
```