A customer site I was at recently needed their XenDesktop 4 farm documented. Since I had already created PowerShell scripts to document the various versions of XenApp, I figured a XenDesktop script should be easy to create. This article and the script were written for “SR” at the customer site.

This article will focus only on XenDesktop 4. I am planning on writing articles and scripts for XenDesktop 5.x.

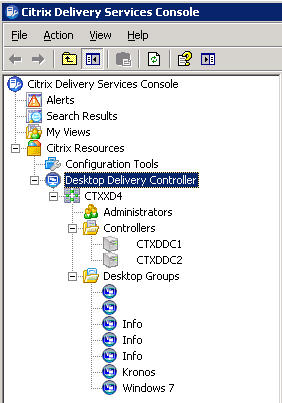
The prerequisites to follow along with this article are:

* A computer, physical or virtual, running Microsoft Windows Server 2003, Server 2008, Server 2008 R2 or Microsoft Windows XP, Vista or 7 for running the XenDesktop Desktop Delivery Controller SDK.
* Citrix XenDesktop 4 Desktop Delivery Controller (DDC) installed with at least one Desktop Group created.

In this article, we will be installing the Citrix Delivery Controller SDK. You can install the SDK from either the XenDesktop 4 installation media or download it from citrix.com. Since I am at a customer site creating this script on a production network, I do not have access to the installation media. Therefore, I will be downloading the SDK.

My initial goal was to see if I could walk down the nodes in the Delivery Services Console (DSC) (Figure 1) and see if I could document every nook and cranny.

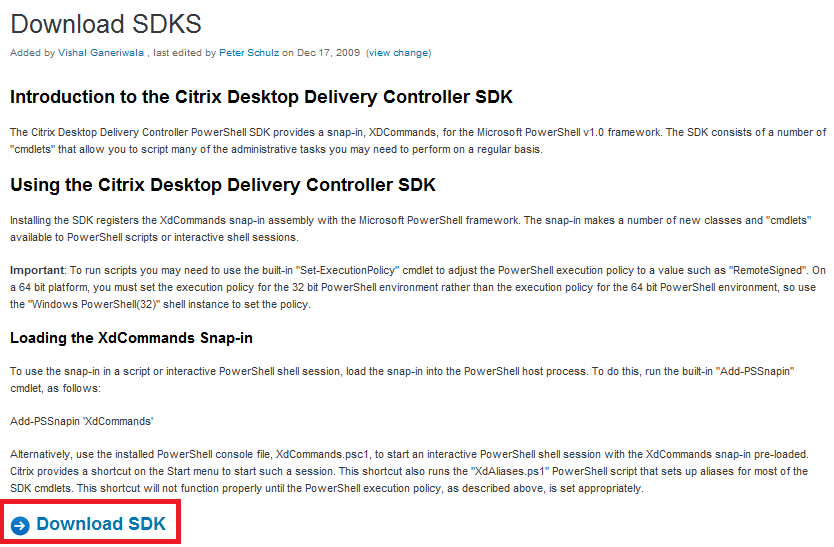
Figure



Before we can start using PowerShell to document anything in the XenDesktop 4 farm we first need to install the XenDesktop SDK. From either your XenDesktop 4 DDC or another computer, go to <http://community.citrix.com/display/xd/Download+SDKS> (Figure 2).

**Note:** For instructions on how to install the SDK from the XenDesktop 4 installation media, please see <http://blogs.citrix.com/2010/08/11/xendesktop-4-powershell-sdk-primer-part-1-getting-started/>.

Figure

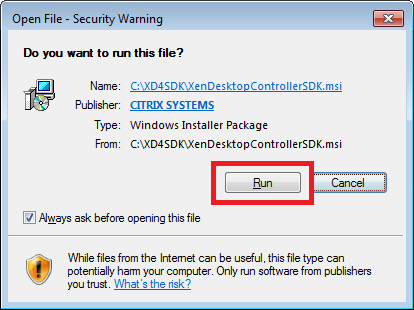


Click on *Download SDK* and Save the file to C:\XD4SDK. You can now close your Internet browser.

Click *Start*, *Run*, type in **C:\XD4SDK\XenDesktopControllerSDK.msi** and press *Enter*.

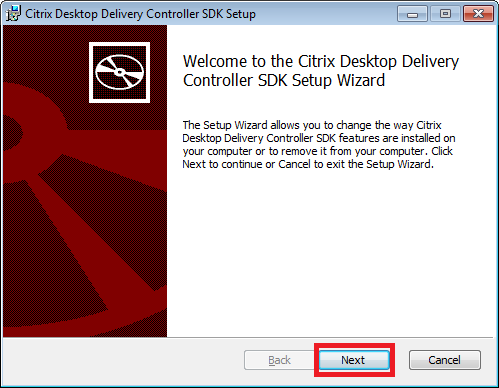
Click *Run* (Figure 3).

Figure



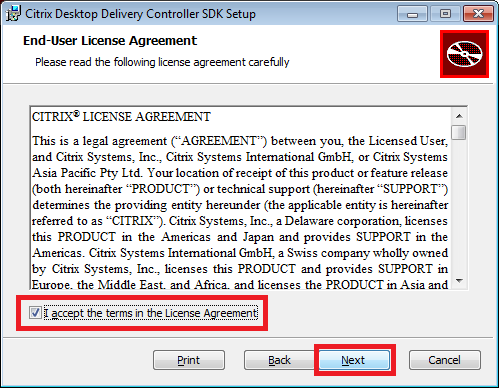
Click *Next* (Figure 4).

Figure



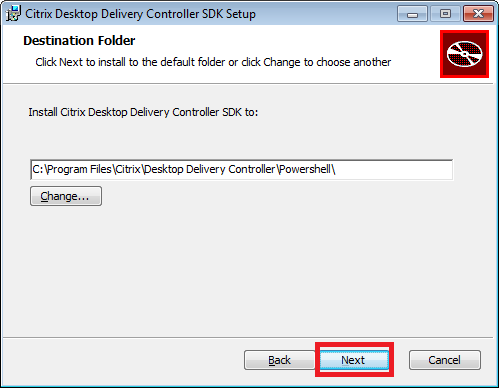
Select *I accept the terms in the License Agreement* and click *Next* (Figure 5).

Figure



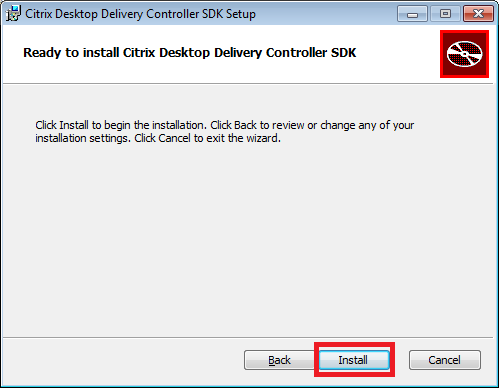
Click *Next* to accept the default installation location (Figure 6).

Figure



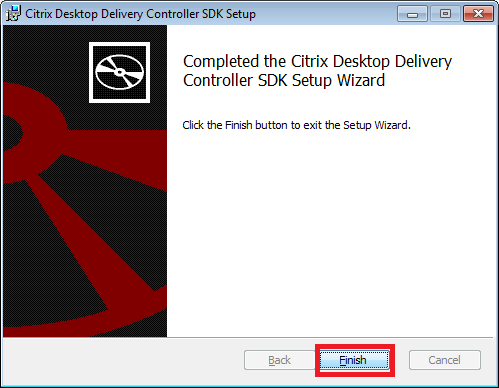
Click *Install* (Figure 7).

Figure



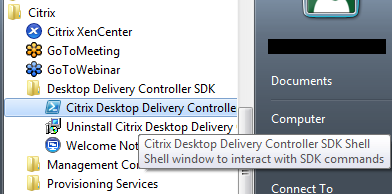
After a few seconds, the installation completes. Click *Finish* (Figure 8).

Figure



You now have new Start Menu items under *All Programs*, *Citrix*. Windows 7 is shown in Figure 9.

Figure



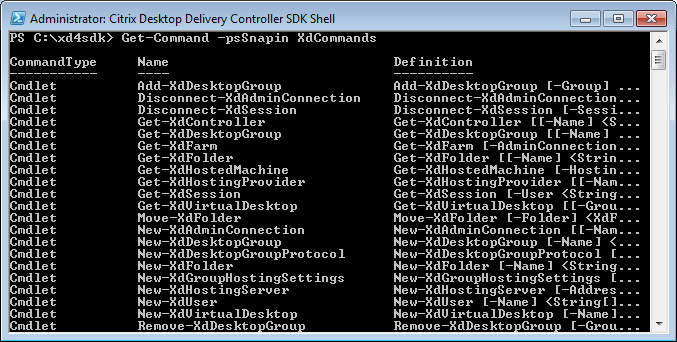
Click *Start, All Programs, Citrix, Desktop Delivery Controller SDK, Citrix Desktop Delivery Controller SDK Shell*. A PowerShell session starts with the Citrix XenDesktop 4 PowerShell modules already loaded.

Everything is now setup for us to get started. The download page for the SDK says to get a listing of the available XenDesktop commands to enter the following command:

**Get-Command -psSnapin XdCommands**

Typing that line into the PowerShell session returns a list of Citrix XenDesktop PowerShell commands. A sample is shown in Figure 10.

Figure

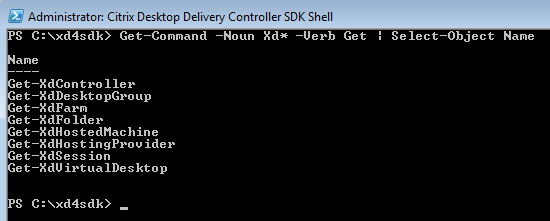


PowerShell commands are done in a Verb-Noun format. Get-Something, New-Something, Copy-Something, etc. I don’t want to change anything in my script, I just want to get, or retrieve, all the farm information possible. So how can I get a list of just the “Get” Citrix XenDesktop PowerShell commands, showing just the name, in an alphabetical order? Being a good PowerShell citizen, Citrix puts “Xd” at the beginning of their Get commands.

To get a list of the Get commands, showing just the Name, where the noun starts with “Xd”, type the following in the PowerShell session (results are shown in Figure 11):

**Get-Command –Noun Xd\* -Verb Get | Select-Object Name**

Figure



The XenDesktop 4 cmdlets are very limited. There are no cmdlets to Get, or retrieve, information about administrators or Citrix Policies. Also, even though Controllers can be segregated into Folders, there is no way to retrieve the Folders that Controllers are placed in. When retrieving the Farm properties, very few of the Farm properties seen in the DSC are available.

Of the eight Get-Xd\* commands listed, only three are usable for documenting the Farm:

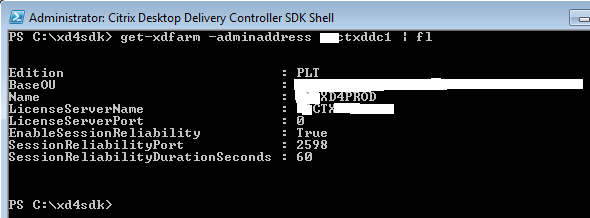
* Get-XdFarm
* Get-XdController
* Get-XdDesktopGroup

One feature I like of the XenDesktop PowerShell commands is that you are not required to run them directly on a Controller. You can use the –AdminAddress parameter to give the name or IP address of a Controller to run the command against. Since I am developing this script on a production Farm, I am being safe and staying off the customer’s production Controllers.

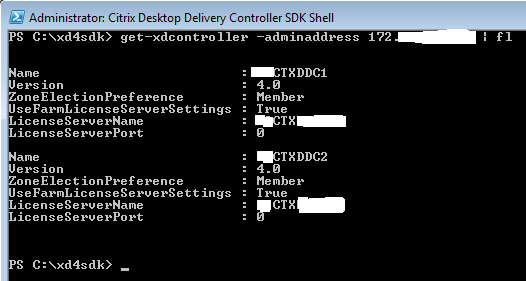
Typing each of the commands into the PowerShell session gave me the results shown in Figure 12, Figure 13 and Figure 14.

**Note:** I am using a Formatted List (| fl) to see all the properties returned by each command.

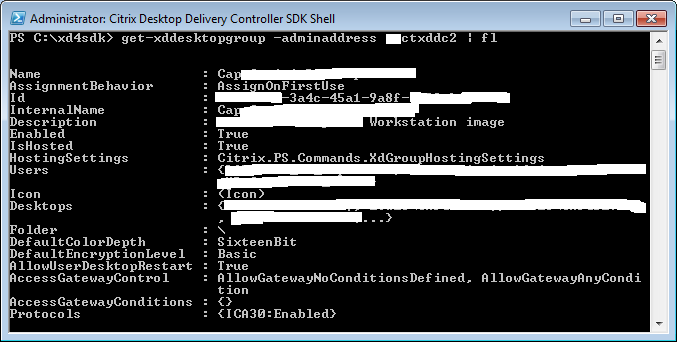
Figure



Figure



Figure



My goal is to use the same wording as what is seen in the DSC for headings, captions and text. In order to do that, I needed a way to format the output text. Michael B. Smith (MBS) developed a function for me to use called ***Line***.

Function line

#function created by Michael B. Smith, Exchange MVP

#@essentialexchange on Twitter

#http://TheEssentialExchange.com

{

Param( [int]$tabs = 0, [string]$name = ’’, `

[string]$value = ’’, `

[string]$newline = “`n”, `

[switch]$nonewline )

While( $tabs –gt 0 ) { $global:output += “`t”; $tabs--; }

If( $nonewline )

{

$global:output += $name + $value

}

Else

{

$global:output += $name + $value + $newline

}

}

Another lesson MBS taught me is to check to see if each cmdlet used returned an error and how to tell the cmdlet how I wanted to proceed if there was an error. This is done by using –ErrorAction, or –EA. ErrorAction has four values (Table 1):

Table

|  |  |  |
| --- | --- | --- |
| Enumeration | Value | Description |
| SilentlyContinue | 0 | The Windows PowerShell runtime will continue processing without notifying the user that an action has occurred. |
| Stop | 1 | The Windows PowerShell runtime will stop processing when an action occurs. |
| Continue | 2 | The Windows PowerShell runtime will continue processing and notify the user that an action has occurred. |
| Inquire | 3 | The Windows PowerShell runtime will stop processing and ask the user how it should proceed. |

For this documentation script, I always use 0. If an error occurs, I want the rest of the script to continue.

Next, I needed to know how to test to see if an action, like Get-XdFarm, succeeded or had an error. MBS said to use $? to test if the most recent action succeeded (True) or had an error (False). For example:

$farm = Get-XdFarm -EA 0

If( $? )

{

#success

}

Else

{

#error

}

Because this script does not have to run on a Controller, a parameter is created to allow a Controller name or IP address to be used.

Let’s get started. We will build the script node by node. Some of the lines may wrap.

The beginning of the script:

#Carl Webster, CTP and independent consultant

#webster@carlwebster.com

#@carlwebster on Twitter

#http://www.CarlWebster.com

#This script written for "SR", March 9, 2012

#Thanks to Michael B. Smith, Joe Shock, Jarian Gibson and James Rankin

#for testing and fine-tuning tips

Param(

[string]$DDCAddress = ""

)

Function line

#function created by Michael B. Smith, Exchange MVP

#@essentialexchange on Twitter

#http://TheEssentialExchange.com

{

Param( [int]$tabs = 0, [string]$name = ’’, [string]$value = ’’, [string]$newline = “`n”, [switch]$nonewline )

While( $tabs –gt 0 ) { $global:output += “`t”; $tabs--; }

If( $nonewline )

{

$global:output += $name + $value

}

Else

{

$global:output += $name + $value + $newline

}

}

#script begins

$global:output = ""

The first node in the DSC is the Farm itself.

#get farm information

$global:output = ""

If($DDCAddress)

{

$farm = Get-XdFarm -adminaddress $DDCAddress -EA 0

}

Else

{

$farm = Get-XdFarm -EA 0

}

If( $? )

{

line 0 "XenDesktop Farm Name: " $farm.Name

line 1 "XenDesktop Edition: " -nonewline

switch ($farm.edition)

{

"PLT" {line 0 "Platinum" }

"STD" {line 0 "VDI" }

"ADV" {line 0 "Advanced" }

"ENT" {line 0 "Enterprise"}

default {line 0 "Farm Edition could not be determined: $($farm.edition)"}

}

line 1 "Base OU: " $farm.BaseOU

line 1 "License server"

line 2 "Name: " $farm.LicenseServerName

line 2 "Port number: " $farm.LicenseServerPort

line 1 "Session reliability"

line 2 "Allow users to view sessions during broken connections: " $farm.EnableSessionReliability

If($farm.EnableSessionReliability)

{

line 3 "Port number: " $farm.SessionReliabilityPort

line 3 "Seconds to keep sessions active: " $farm.SessionReliabilityDurationSeconds

}

}

Else

{

line 0 "XenDesktop Farm information could not be retrieved"

}

write-output $global:output

$farm = $null

$global:output = $null

Sample script output:

XenDesktop Farm Name: SAMPLEXD4

XenDesktop Edition: Platinum

Base OU: XD4BaseOU

License server

Name: CTXLICENSE

Port number: 0

Session reliability

Allow users to view sessions during broken connections: True

Port number: 2598

Seconds to keep sessions active: 60

Even though the license server port number is displayed as 27000 in the console, the Citrix cmdlet returns 0.

The next node in the DSC is Administrators but Citrix does not provide a way to retrieve any information about the Administrators.

The next node is Controllers. Even though Controllers can be in folders, Citrix does not provide a way to retrieve Folder information.

#get controller information

If($DDCAddress)

{

$XDControllers = Get-XdController -adminaddress $DDCAddress -EA 0

}

Else

{

$XDControllers = Get-XdController -EA 0

}

If( $? )

{

line 0 "Desktop Delivery Controllers:"

ForEach($XDController in $XDControllers)

{

line 1 "Controller: " $XDController.Name

line 1 "Version: " $XDController.Version

line 1 "Zone Election Preference: " $XDController.ZoneElectionPreference

line 1 "License Server"

If($XDController.UseFarmLicenseServerSettings)

{

line 2 "Using Farm Setting"

}

Else

{

Line 2 "License server"

line 3 "Name: " $XDController.LicenseServerName

line 3 "Port number: " $XDController.LicenseServerPort

}

line 1 ""

}

}

Else

{

line 0 "Desktop Delivery Controller information could not be retrieved"

}

write-output $global:output

$XDControllers = $null

$global:output = $null

Sample script output:

Desktop Delivery Controllers:

Controller: DDC1

Version: 4.0

Zone Election Preference: Member

License Server

Using Farm Setting

Controller: DDC2

Version: 4.0

Zone Election Preference: Member

License Server

Using Farm Setting

Last node is Desktop Groups. There are different options available depending on if the Desktop Group is Pooled or Assigned. Instead or returned the AD Computer Account name as shown in the console, the cmdlet returns the Machine SID. Also, not every user account returned a name but the User SID was always returned. MBS showed me how to covert a Machine or User Account SID to the Machine or User Name.

$objSID = New-Object System.Security.Principal.SecurityIdentifier ($Desktop.MachineSid.Value)

$objComputer = $objSID.Translate([System.Security.Principal.NTAccount])

"AD Computer Account: " $objComputer.Value

And

$objSID = New-Object System.Security.Principal.SecurityIdentifier($Desktop.AssignedUserSid.Value)

$objUser = $objSID.Translate([System.Security.Principal.NTAccount])

“Assigned User: “ $objUser.Value

#get desktop group information

$global:output = ""

If($DDCAddress)

{

$XDGroups = Get-XdDesktopGroup -adminaddress $DDCAddress -EA 0

}

Else

{

$XDGroups = Get-XdDesktopGroup -EA 0

}

If( $? )

{

line 0 "Desktop Groups:"

ForEach($XDGroup in $XDGroups)

{

line 1 "Basic"

line 2 "Desktop Group Name"

line 3 "Display name: " $XDGroup.Name

line 3 "Description: " $XDGroup.Description

line 3 "Desktop Group name: " $XDGroup.InternalName

line 3 "Disable desktop group: " -nonewline

If($XDGroup.Enabled)

{

line 0 "group is enabled"

}

Else

{

line 0 "group is disabled"

}

line 2 "Assignment Type"

line 3 "Assignment Behavior: " $XDGroup.AssignmentBehavior

If($XDGroup.IsHosted)

{

line 2 "Hosting infrastructure: " $XDGroup.HostingSettings.HostingServer

}

line 2 "Users"

line 3 "Configured users:"

ForEach($User in $XDGroup.Users)

{

line 4 $User

#line 4 "SID: " $User.Sid

line 4 "Group or User: " -nonewline

If($User.IsSecurityGroup)

{

line 0 "Group"

}

Else

{

line 0 "User"

}

}

line 2 "Virtual Desktops"

line 3 "Virtual desktops:"

ForEach($Desktop in $XDGroup.Desktops)

{

line 4 "Folder: " $XDGroup.Folder

line 4 "Virtual Machine: " $Desktop

$objSID = New-Object System.Security.Principal.SecurityIdentifier ($Desktop.MachineSid.Value)

$objComputer = $objSID.Translate([System.Security.Principal.NTAccount])

line 4 "AD Computer Account: " $objComputer.Value

line 4 "Desktop State: " $Desktop.State

line 4 "Assigned User: " -nonewline

If($Desktop.AssignUserName)

{

line 0 $Desktop.AssignUserName

}

ElseIf($Desktop.AssignedUserSid)

{

$objSID = New-Object System.Security.Principal.SecurityIdentifier ($Desktop.AssignedUserSid.Value)

$objUser = $objSID.Translate([System.Security.Principal.NTAccount])

line 0 $objUser.Value

}

Else

{

line 0 ""

}

line 4 "Maintenance Mode: " $Desktop.MaintenanceMode

line 4 "Machine State: " $Desktop.PowerState

line 4 "Controller: " $Desktop.Controller

line 4 "Agent Version: " $Desktop.AgentVersion

line 1 ""

}

line 1 "Advanced"

line 2 "Access Control"

$test = $XDGroup.AccessGatewayControl.ToString()

$test1 = $test.replace(", ","`n`t`t")

line 3 $test1

line 2 "Access Gateway Conditions: "

ForEach($Condition in $XDGroup.AccessGatewayConditions)

{

line 3 $Condition

}

line 2 "Client Options"

line 3 "Appearance"

line 4 "Colors: " -nonewline

switch ($XDGroup.DefaultColorDepth)

{

"FourBit" {line 0 "16 colors" }

"EightBit" {line 0 "256 colors" }

"SixteenBit" {line 0 "High color (16-bit)"}

"TwentyFourBit" {line 0 "True color (24-bit)"}

default {line 0 "Color depth could not be determined: $($XDGroup.DefaultColorDepth)"}

}

line 3 "Connection"

line 4 "Encryption: " -nonewline

switch ($XDGroup.DefaultEncryptionLevel)

{

"Basic" {line 0 "Basic" }

"LogOnRC5Using128Bit" {line 0 "128-Bit Login Only (RC-5)"}

"RC5Using40Bit" {line 0 "40-Bit (RC-5)" }

"RC5Using56Bit" {line 0 "56-Bit (RC-5)" }

"RC5Using128Bit" {line 0 "128-Bit (RC-5)" }

default {line 0 "Encryption level could not be determined: $($XDGroup.DefaultEncryptionLevel)"}

}

line 3 "Connection Protocols: "

ForEach($Protocol in $XDGroup.Protocols)

{

line 4 "Name: " $Protocol.Protocol

line 4 "Enabled: " $Protocol.Enabled

}

#only show the next section if the Desktop Group is Pooled

If($XDGroup.AssignmentBehavior -eq "Pooled")

{

line 2 "Idle Pool Settings"

line 3 "Business Hours"

line 4 "Business days "

ForEach($Day in $XDGroup.HostingSettings.BusinessDays)

{

line 5 $Day

}

line 4 "Time zone " $XDGroup.HostingSettings.IdleTimesTimeZone

IF($XDGroup.HostingSettings.PeakHoursStart)

{

line 4 "Day start " $XDGroup.HostingSettings.PeakHoursStart.ToString()

}

If($XDGroup.HostingSettings.PeakHoursEnd)

{

line 4 "Peak end " $XDGroup.HostingSettings.PeakHoursEnd.ToString()

}

If($XDGroup.HostingSettings.BusinessHoursEnd)

{

line 4 "Day end " $XDGroup.HostingSettings.BusinessHoursEnd.ToString()

}

line 3 "Idle Desktop Count"

line 2 "Business hours " $XDGroup.HostingSettings.BusinessHoursIdleCount

line 2 "Peak time " $XDGroup.HostingSettings.PeakHoursIdleCount

line 2 "Out of hours " $XDGroup.HostingSettings.OutOfHoursIdleCount

}

# I can't find these settings in the console

line 1 "Other settings"

line 2 "Allow User Desktop Restart: " $XDGroup.AllowUserDesktopRestart

line 2 "Tainted Machine Action: " $XDGroup.HostingSettings.TaintedMachineAction

line 3 "Actions: "

ForEach($Action in $XDGroup.HostingSettings.Actions)

{

line 4 "Action point: " $Action.ActionPoint

line 4 "Action: " $Action.Action

line 4 "Delay: " $Action.Delay

line 4 ""

}

line 1 ""

}

}

Else

{

line 0 "Desktop Group information could not be retrieved"

}

write-output $global:output

$XDGroups = $null

$test = $null

$test1 = $null

$global:output = $null

Sample script output:

Desktop Groups:

Basic

Desktop Group Name

Display name: WebstersLab

Description: Used in Webster’s Lab for writing purposes

Desktop Group name: WebstersLab

Disable desktop group: group is enabled

Assignment Type

Assignment Behavior: Pooled

Hosting infrastructure: http://msvirtualcenter.carls.com/sdk

Users

Configured users:

CARLS\TS\_XD\_Lab

Group or User: Group

CARLS\TS\_XD\_Lab\_Admin

Group or User: Group

Virtual Desktops

Virtual desktops:

Folder: \

Virtual Machine: CARLS\COMPUTER01$

AD Computer Account: CARLS\COMPUTER01$

Desktop State: Available

Assigned User:

Maintenance Mode: False

Machine State: On

Controller: DDC1

Agent Version: 4.0.4522

Folder: \

Virtual Machine: CARLS\COMPUTER06$

AD Computer Account: CARLS\COMPUTER06$

Desktop State: NotRegistered

Assigned User:

Maintenance Mode: False

Machine State: On

Controller:

Agent Version:

Advanced

Access Control

AllowGatewayNoConditionsDefined

AllowGatewayAnyCondition

Access Gateway Conditions:

Client Options

Appearance

Colors: High color (16-bit)

Connection

Encryption: Basic

Connection Protocols:

Name: ICA30

Enabled: True

Idle Pool Settings

Business Hours

Business days

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Time zone (UTC-05:00) Eastern Time (US & Canada)

Day start 01:00:00

Peak end 20:30:00

Day end 23:30:00

Idle Desktop Count

Business hours 7

Peak time 7

Out of hours 7

Other settings

Allow User Desktop Restart: False

Tainted Machine Action: Restart

Actions:

Action point: Disconnect

Action: DoNothing

Delay: 300

Action point: LogOff

Action: Shutdown

Delay: 420

How to use this script?

I saved the script as XD4\_Inventory.ps1 in the Z:\ folder. From the PowerShell prompt, change to the Z:\ folder, or the folder where you saved the script. From the PowerShell prompt, type in:

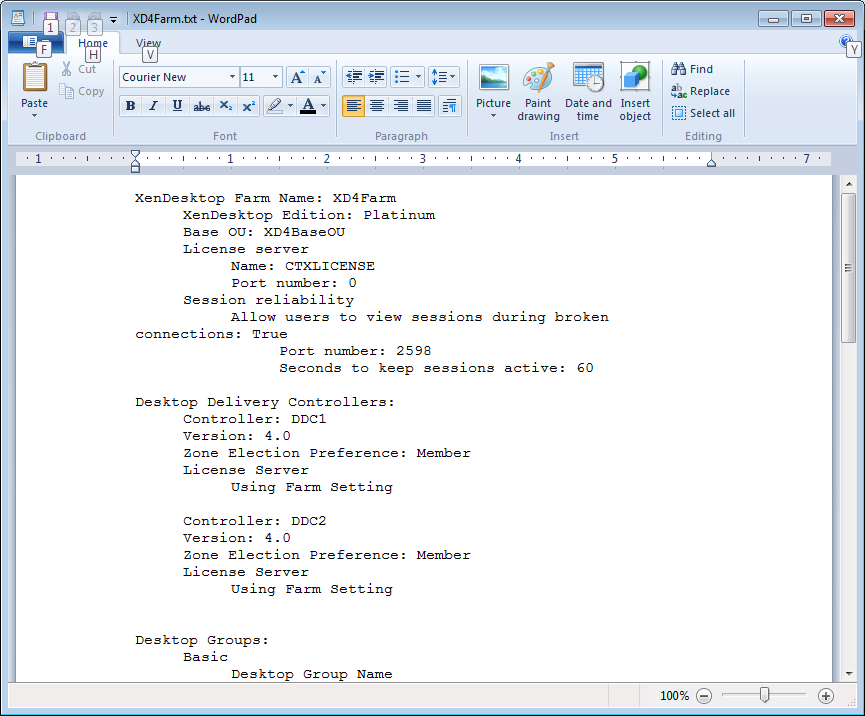
**.\XD4\_Inventory.ps1 |out-file Z:\XD4Farm.txt** and press *Enter*, or

**.\XD4\_Inventory.ps1 DDCName|out-file Z:\XD4Farm.txt** and press *Enter*, or

**.\XD4\_Inventory.ps1 DDCIPAddress|out-file Z:\XD4Farm.txt** and press *Enter*.

Open XD4Farm.txt in either WordPad or Microsoft Word (Figure 15).

Figure



If you have any suggestions for the script, please let me know. Send an e-mail to webster@carlwebster.com.

I have placed an unsigned copy of this script at http://dl.dropbox.com/u/43555945/XD4\_Inventory.ps1.

I have also placed a copy of the script with a TXT extension at http://dl.dropbox.com/u/43555945/XD4\_Inventory.txt.

I have placed a digitally signed copy of this script at http://dl.dropbox.com/u/43555945/XD4\_Inventory\_Signed.ps1.

I have also placed a copy of the digitally signed script with a TXT extension at http://dl.dropbox.com/u/43555945/XD4\_Inventory\_Signed.txt.