Disable DNS Debug Logging

Generated from claude.ai

```
# GUI Script to disable DNS debug logging on multiple domain controllers
# ------------------------------------------------------------

Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

# Enable DPI awareness to improve scaling
[System.Windows.Forms.Application]::EnableVisualStyles()

# Use this for better DPI scaling
Add-Type -TypeDefinition @"
using System;
using System.Runtime.InteropServices;

public class DpiAwareness {
    [DllImport("user32.dll")]
    public static extern bool SetProcessDPIAware();
}
"@
[DpiAwareness]::SetProcessDPIAware()

# Create the main form with AutoScaleMode
$form = New-Object System.Windows.Forms.Form
$form.Text = "DNS Debug Logging Manager"
$form.ClientSize = New-Object System.Drawing.Size(600, 500)
$form.StartPosition = "CenterScreen"
$form.FormBorderStyle = "Sizable"
$form.MinimumSize = New-Object System.Drawing.Size(600, 500)
$form.MaximizeBox = $true
$form.MinimizeBox = $true
$form.Font = New-Object System.Drawing.Font("Segoe UI", 9)
$form.AutoScaleMode = [System.Windows.Forms.AutoScaleMode]::Dpi
```

```
# Create a TableLayoutPanel for better layout control
$tableLayoutPanel = New-Object System.Windows.Forms.TableLayoutPanel
$tableLayoutPanel.Dock = [System.Windows.Forms.DockStyle]::Fill
$tableLayoutPanel.RowCount = 6
$tableLayoutPanel.ColumnCount = 1
$tableLayoutPanel.Padding = New-Object System.Windows.Forms.Padding(10)
$tableLayoutPanel.ColumnStyles.Add((New-Object
System.Windows.Forms.ColumnStyle([System.Windows.Forms.SizeType]::Percent, 100)))

# Row styles - control proportions
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Absolute, 30))) # Title
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Absolute, 40))) # Instructions
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Percent, 35))) # DC Textbox
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Absolute, 30))) # Progress
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Percent, 45))) # Log
$tableLayoutPanel.RowStyles.Add((New-Object
System.Windows.Forms.RowStyle([System.Windows.Forms.SizeType]::Absolute, 50))) # Buttons

$form.Controls.Add($tableLayoutPanel)

# Create title label
$titleLabel = New-Object System.Windows.Forms.Label
$titleLabel.Text = "Disable DNS Debug Logging on Remote Domain Controllers"
$titleLabel.Font = New-Object System.Drawing.Font("Segoe UI", 12,
[System.Drawing.FontStyle]::Bold)
$titleLabel.Dock = [System.Windows.Forms.DockStyle]::Fill
$titleLabel.AutoSize = $false
$tableLayoutPanel.Controls.Add($titleLabel, 0, 0)

# Create instructions label
$instructionsLabel = New-Object System.Windows.Forms.Label
$instructionsLabel.Text = "Enter domain controller names (one per line) or paste a list:"
$instructionsLabel.Dock = [System.Windows.Forms.DockStyle]::Fill
$instructionsLabel.AutoSize = $false
$tableLayoutPanel.Controls.Add($instructionsLabel, 0, 1)
```

```
# Create textbox for domain controllers
$dcTextBox = New-Object System.Windows.Forms.TextBox
$dcTextBox.Multiline = $true
$dcTextBox.ScrollBars = "Vertical"
$dcTextBox.AcceptsReturn = $true
$dcTextBox.WordWrap = $false
$dcTextBox.Dock = [System.Windows.Forms.DockStyle]::Fill
$dcTextBox.Anchor = [System.Windows.Forms.AnchorStyles]::Top -bor
[System.Windows.Forms.AnchorStyles]::Bottom -bor [System.Windows.Forms.AnchorStyles]::Left -
bor [System.Windows.Forms.AnchorStyles]::Right
$tableLayoutPanel.Controls.Add($dcTextBox, 0, 2)

# Create progress panel
$progressPanel = New-Object System.Windows.Forms.Panel
$progressPanel.Dock = [System.Windows.Forms.DockStyle]::Fill
$tableLayoutPanel.Controls.Add($progressPanel, 0, 3)

# Create progress bar
$progressBar = New-Object System.Windows.Forms.ProgressBar
$progressBar.Dock = [System.Windows.Forms.DockStyle]::Fill
$progressBar.Style = "Continuous"
$progressPanel.Controls.Add($progressBar)

# Create a RichTextBox for logs
$logTextBox = New-Object System.Windows.Forms.RichTextBox
$logTextBox.ReadOnly = $true
$logTextBox.BackColor = [System.Drawing.Color]::White
$logTextBox.Font = New-Object System.Drawing.Font("Consolas", 9)
$logTextBox.Dock = [System.Windows.Forms.DockStyle]::Fill
$logTextBox.Anchor = [System.Windows.Forms.AnchorStyles]::Top -bor
[System.Windows.Forms.AnchorStyles]::Bottom -bor [System.Windows.Forms.AnchorStyles]::Left -
bor [System.Windows.Forms.AnchorStyles]::Right
$tableLayoutPanel.Controls.Add($logTextBox, 0, 4)

# Create button panel and flowLayoutPanel for buttons
$buttonPanel = New-Object System.Windows.Forms.Panel
$buttonPanel.Dock = [System.Windows.Forms.DockStyle]::Fill
$tableLayoutPanel.Controls.Add($buttonPanel, 0, 5)

# Create status label
```

```
$statusLabel = New-Object System.Windows.Forms.Label
$statusLabel.Text = "Ready"
$statusLabel.AutoSize = $true
$statusLabel.Location = New-Object System.Drawing.Point(0, 15)
$buttonPanel.Controls.Add($statusLabel)

# Create a FlowLayoutPanel for buttons
$buttonFlowPanel = New-Object System.Windows.Forms.FlowLayoutPanel
$buttonFlowPanel.FlowDirection = [System.Windows.Forms.FlowDirection]::RightToLeft
$buttonFlowPanel.WrapContents = $false
$buttonFlowPanel.AutoSize = $true
$buttonFlowPanel.Anchor = [System.Windows.Forms.AnchorStyles]::Right -bor
[System.Windows.Forms.AnchorStyles]::Bottom
$buttonFlowPanel.Location = New-Object System.Drawing.Point(290, 10)
$buttonPanel.Controls.Add($buttonFlowPanel)

# Create buttons and add to flow panel
$executeButton = New-Object System.Windows.Forms.Button
$executeButton.Text = "Execute"
$executeButton.Size = New-Object System.Drawing.Size(100, 30)
$executeButton.BackColor = [System.Drawing.Color]::FromArgb(0, 120, 212)
$executeButton.ForeColor = [System.Drawing.Color]::White
$executeButton.FlatStyle = "Flat"
$buttonFlowPanel.Controls.Add($executeButton)

$importButton = New-Object System.Windows.Forms.Button
$importButton.Text = "Import List"
$importButton.Size = New-Object System.Drawing.Size(100, 30)
$importButton.BackColor = [System.Drawing.Color]::LightGray
$importButton.FlatStyle = "Flat"
$buttonFlowPanel.Controls.Add($importButton)

$exportButton = New-Object System.Windows.Forms.Button
$exportButton.Text = "Export Results"
$exportButton.Size = New-Object System.Drawing.Size(100, 30)
$exportButton.BackColor = [System.Drawing.Color]::LightGray
$exportButton.FlatStyle = "Flat"
$exportButton.Enabled = $false
$buttonFlowPanel.Controls.Add($exportButton)

# Update button positions on form resize
```

```
$form.Add_Resize({
    # Calculate position manually using integers to avoid subtraction method issues
    $xPos = $buttonPanel.Width
    $xPos -= $buttonFlowPanel.Width
    $xPos -= 10  # Add 10px margin
    $buttonFlowPanel.Location = New-Object System.Drawing.Point($xPos, 10)
})

# Function to log messages
function Log-Message {
    param (
        [string]$Message,
        [string]$Color = "Black"
    )

    # Convert string color to System.Drawing.Color
    $drawingColor = switch ($Color) {
        "Red"    { [System.Drawing.Color]::Red }
        "Green"  { [System.Drawing.Color]::Green }
        "Blue"   { [System.Drawing.Color]::Blue }
        "Black"  { [System.Drawing.Color]::Black }
        default  { [System.Drawing.Color]::Black }
    }

    # Append text with color
    $logTextBox.SelectionStart = $logTextBox.TextLength
    $logTextBox.SelectionLength = 0
    $logTextBox.SelectionColor = $drawingColor
    $logTextBox.AppendText("$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss') - $Message`r`n")
    $logTextBox.SelectionStart = $logTextBox.TextLength
    $logTextBox.ScrollToCaret()

    # Update status label
    $statusLabel.Text = $Message
    [System.Windows.Forms.Application]::DoEvents()
}

# Function to disable DNS debug logging
function Disable-DnsDebugLogging {
    param (
        [Parameter(Mandatory=$true)]
```

```
        [string]$ServerName
    )

    try {
        Log-Message "Connecting to $ServerName..." "Blue"

        # Check if the server is reachable
        if (-not (Test-Connection -ComputerName $ServerName -Count 1 -Quiet)) {
            Log-Message "Cannot reach $ServerName. Skipping..." "Red"
            return $false
        }

        # Connect to remote DNS server and disable debug logging only
        $result = Invoke-Command -ComputerName $ServerName -ScriptBlock {
            try {
                # Get DNS Server service
                $dnsServer = Get-Service -Name "DNS" -ErrorAction Stop

                if ($dnsServer.Status -ne "Running") {
                    return "DNS Server service is not running on this server."
                }

                # Disable debug log settings via registry while preserving standard logging
                Set-ItemProperty "HKLM:\SYSTEM\CurrentControlSet\Services\DNS\Parameters" -
Name "LogLevel" -Value 0 -ErrorAction Stop

                # Use dnscmd.exe if available to disable debug logging
                if (Get-Command dnscmd.exe -ErrorAction SilentlyContinue) {
                    & dnscmd.exe /config /LogLevel 0
                }

                                    # Check if DNS Server PowerShell module is available
                if (Get-Module -ListAvailable -Name DnsServer) {
                    # Use DNS Server cmdlets to disable debug logging
                    Import-Module DnsServer

                    # Use Set-DnsServerDiagnostics directly with parameters instead of
InputObject
                    # This is more compatible across different PowerShell versions
                    try {
                        Set-DnsServerDiagnostics -Queries $false `
```

```
                                        -Answers $false `
                                        -Notifications $false `
                                        -Update $false `
                                        -QuestionTransactions $false `
                                        -UnmatchedResponse $false `
                                        -SendPackets $false `
                                        -ReceivePackets $false `
                                        -TcpPackets $false `
                                        -UdpPackets $false `
                                        -FullPackets $false `
                                        -FilterIPAddressList @() `
                                        -EnableLoggingToFile $false `
                                        -ErrorAction Stop
                    }
                    catch {
                        # Fallback for older versions with different parameter names
                        $diagnosticParams = @{
                            ErrorAction = 'SilentlyContinue'
                        }

                        # Try each parameter separately to handle different PowerShell
versions
                        Set-DnsServerDiagnostics -Queries $false @diagnosticParams
                        Set-DnsServerDiagnostics -Answers $false @diagnosticParams
                        Set-DnsServerDiagnostics -Notifications $false @diagnosticParams
                        Set-DnsServerDiagnostics -Update $false @diagnosticParams
                        Set-DnsServerDiagnostics -QuestionTransactions $false
@diagnosticParams
                        Set-DnsServerDiagnostics -UnmatchedResponse $false @diagnosticParams
                        Set-DnsServerDiagnostics -SendPackets $false @diagnosticParams
                        Set-DnsServerDiagnostics -ReceivePackets $false @diagnosticParams
                        Set-DnsServerDiagnostics -TcpPackets $false @diagnosticParams
                        Set-DnsServerDiagnostics -UdpPackets $false @diagnosticParams
                        Set-DnsServerDiagnostics -FullPackets $false @diagnosticParams
                        Set-DnsServerDiagnostics -EnableLoggingToFile $false @diagnosticParams
                    }
                }

                return "DNS debug logging successfully disabled while preserving standard
event logging."
            }
```

```
            catch {
                return "Error: $_"
            }
        }

        # Output results
        $statusMessage = "$($ServerName): $result"
        if ($result -like "Error:*" -or $result -like "DNS Server service is not*") {
            Log-Message $statusMessage "Red"
            return $false
        } else {
            Log-Message $statusMessage "Green"
            return $true
        }
    }
    catch {
        Log-Message "Failed to connect to $ServerName. Error: $_" "Red"
        return $false
    }
}

# Results array to store outcomes
$script:results = @()

# Import button click event
$importButton.Add_Click({
    $openFileDialog = New-Object System.Windows.Forms.OpenFileDialog
    $openFileDialog.Filter = "Text files (*.txt)|*.txt|CSV files (*.csv)|*.csv|All files
(*.*)|*.*"
    $openFileDialog.Title = "Select a file containing domain controller names"

    if ($openFileDialog.ShowDialog() -eq "OK") {
        try {
            $fileContent = Get-Content $openFileDialog.FileName
            $dcTextBox.Text = $fileContent -join "`r`n"
            Log-Message "Imported $($fileContent.Count) domain controllers from
$($openFileDialog.FileName)" "Blue"
        }
        catch {
            Log-Message "Error importing file: $_" "Red"
        }
```

```
    }
})

# Export button click event
$exportButton.Add_Click({
    $saveFileDialog = New-Object System.Windows.Forms.SaveFileDialog
    $saveFileDialog.Filter = "CSV files (*.csv)|*.csv|All files (*.*)|*.*"
    $saveFileDialog.Title = "Save results to a CSV file"
    $saveFileDialog.FileName = "DNSDebugLoggingResults_$(Get-Date -Format
'yyyyMMdd_HHmmss').csv"

    if ($saveFileDialog.ShowDialog() -eq "OK") {
        try {
            $script:results | Export-Csv -Path $saveFileDialog.FileName -NoTypeInformation
            Log-Message "Results exported to $($saveFileDialog.FileName)" "Green"
        }
        catch {
            Log-Message "Error exporting results: $_" "Red"
        }
    }
})

# Execute button click event
$executeButton.Add_Click({
    # Disable controls during execution
    $executeButton.Enabled = $false
    $importButton.Enabled = $false
    $dcTextBox.Enabled = $false

    # Clear previous results
    $script:results = @()

    # Get domain controllers from textbox
    $domainControllers = $dcTextBox.Text -split "`r`n" | Where-Object { $_ -ne "" }

    if ($domainControllers.Count -eq 0) {
        Log-Message "No domain controllers specified." "Red"
        $executeButton.Enabled = $true
        $importButton.Enabled = $true
        $dcTextBox.Enabled = $true
        return
```

```
    }

    Log-Message "Starting DNS debug logging disable process on $($domainControllers.Count)
domain controllers..." "Blue"

    # Initialize counters
    $successful = 0
    $failed = 0
    $current = 0

    # Update progress bar maximum
    $progressBar.Maximum = $domainControllers.Count
    $progressBar.Value = 0

    # Process each domain controller
    foreach ($dc in $domainControllers) {
        $current++
        $progressBar.Value = $current
        $statusLabel.Text = "Processing $current of $($domainControllers.Count): $dc"
        [System.Windows.Forms.Application]::DoEvents()

        # Skip empty entries
        if ([string]::IsNullOrWhiteSpace($dc)) {
            continue
        }

        # Process the domain controller
        $success = Disable-DnsDebugLogging -ServerName $dc.Trim()

        # Track results
        if ($success) {
            $successful++
            $script:results += [PSCustomObject]@{
                DomainController = $dc.Trim()
                Status = "Success"
                Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
            }
        }
        else {
            $failed++
            $script:results += [PSCustomObject]@{
```

```
                DomainController = $dc.Trim()
                Status = "Failed"
                Timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
            }
        }
    }

    # Summary
    Log-Message "-------------------------------------------------" "Blue"
    Log-Message "Summary:" "Blue"
    Log-Message "Total Domain Controllers: $($domainControllers.Count)" "Black"
    Log-Message "Successfully disabled debug logging: $successful" "Green"

    if ($failed -gt 0) {
        Log-Message "Failed to disable debug logging: $failed" "Red"
    } else {
        Log-Message "Failed to disable debug logging: $failed" "Green"
    }

    Log-Message "-------------------------------------------------" "Blue"
    Log-Message "Process completed!" "Blue"

    # Enable export button if there are results
    if ($script:results.Count -gt 0) {
        $exportButton.Enabled = $true
    }

    # Re-enable controls
    $executeButton.Enabled = $true
    $importButton.Enabled = $true
    $dcTextBox.Enabled = $true
    $statusLabel.Text = "Ready"
})

# Set initial flow panel position after form loads
$form.Add_Shown({
    $form.Activate()
    # Calculate position manually using integers to avoid subtraction method issues
    $xPos = $buttonPanel.Width
    $xPos -= $buttonFlowPanel.Width
    $xPos -= 10  # Add 10px margin
```

```
    $buttonFlowPanel.Location = New-Object System.Drawing.Point($xPos, 10)
})

# Show the form
[void] $form.ShowDialog()
```

Revision #5
Created 2025-02-26 12:17:25 UTC by Ryan
Updated 2025-03-13 00:56:28 UTC by Ryan