```powershell
#Requires -Version 3.0
<#
    ================================================================================
    (C)2006-2025 KoXo Developpement
    x86 & x64
    KoXo Profiles Downloader Script  (PowerShell version)
    --------------------------------------------------------------------------------
    This script uses the domain controller holding the "PDC Emulator" role
    to run "KoXo Profiles Downloader".

    The architecture-specific executable is preferred (ProfilesDownloader64.exe
    or ProfilesDownloader32.exe). If it is missing, the script falls back to the
    generic "KoXoDownloader.exe" located in the same share.
    --------------------------------------------------------------------------------
    Command-line arguments (case-insensitive; prefix "-" or "/", separator
    "=", ":" or a space). They override the defaults set below:
        -site   / /site      Share prefix (CompanyShortName)
        -server / /server    File server to use instead of the PDC Emulator
        -silent / /silent    Flag: no error dialog. Optional value:
                             -silent  or  -silent=true   -> enabled
                             -silent=false               -> disabled
    When site/server are omitted: the PDC Emulator and the "KoXoProfiles$"
    share are used.

    GPO usage: put only the arguments in the "Script Parameters" field, e.g.
        -site=PARIS -server=DC01 -silent
    ================================================================================
#>

# ================================================================================
#  Default settings (the command line may override them)
# ================================================================================
# When $true, no error dialog is shown (recommended for COMPUTER / startup
# scripts in session 0, where a dialog would never be seen).
$Silent      = $false
# Auto-close delay (seconds) for the non-blocking error dialog. The box closes
# by itself after this delay if nobody clicks, so the script never hangs.
# Use 0 to wait indefinitely for a click (NOT recommended for GPO).
$ErrorTimeout = 10
# Fallback executable used when the architecture-specific one is missing.
$FallbackExe = 'ProfilesDownloader.exe'
# When $true, the script hides its OWN console window at startup, as a safety
# net (e.g. if the GPO is set to show scripts). Best effort: the console may
# still flash briefly before it is hidden. Set to $false to keep it visible.
$HideConsole = $true
# ================================================================================

$Title = 'KoXo Profiles'

# Values supplied on the command line (empty = use defaults: PDC / no prefix).
$CompanyShortName       = ''
$KoXoProfilesFileServer = ''
```

```
# ------------------------------------------------------------------------------
#  Native Win32 helpers, compiled ONCE (used for console hiding AND for the
#  process launch further down).
# ------------------------------------------------------------------------------
Add-Type -Name Native -Namespace KoXo -MemberDefinition @'
[StructLayout(LayoutKind.Sequential, CharSet=CharSet.Unicode)]
public struct STARTUPINFO {
    public int cb;
    public string lpReserved;
    public string lpDesktop;
    public string lpTitle;
    public int dwX; public int dwY; public int dwXSize; public int dwYSize;
    public int dwXCountChars; public int dwYCountChars; public int dwFillAttribute;
    public int dwFlags; public short wShowWindow; public short cbReserved2;
    public System.IntPtr lpReserved2;
    public System.IntPtr hStdInput; public System.IntPtr hStdOutput; public System.IntPtr hStdError;
}
[StructLayout(LayoutKind.Sequential)]
public struct PROCESS_INFORMATION {
    public System.IntPtr hProcess; public System.IntPtr hThread;
    public int dwProcessId; public int dwThreadId;
}
[DllImport("kernel32.dll")] public static extern System.IntPtr GetConsoleWindow();
[DllImport("user32.dll")]   public static extern bool ShowWindow(System.IntPtr hWnd, int nCmdShow);
[DllImport("kernel32.dll", CharSet=CharSet.Unicode, SetLastError=true)]
public static extern bool CreateProcess(
    string lpApplicationName, string lpCommandLine,
    System.IntPtr lpProcessAttributes, System.IntPtr lpThreadAttributes,
    bool bInheritHandles, uint dwCreationFlags, System.IntPtr lpEnvironment,
    string lpCurrentDirectory, ref STARTUPINFO lpStartupInfo,
    out PROCESS_INFORMATION lpProcessInformation);
[DllImport("kernel32.dll", SetLastError=true)]
public static extern uint WaitForSingleObject(System.IntPtr hHandle, uint dwMilliseconds);
[DllImport("kernel32.dll", SetLastError=true)]
public static extern bool GetExitCodeProcess(System.IntPtr hProcess, out uint lpExitCode);
[DllImport("kernel32.dll", SetLastError=true)]
public static extern bool CloseHandle(System.IntPtr hObject);
'@

# Hide our own console window as early as possible (no effect in the ISE, which
# has no console). This is a safety net when the GPO shows scripts; the proper
# flash-free way is still to launch hidden (-WindowStyle Hidden / GPO not visible).
if ($HideConsole) {
    $__console = [KoXo.Native]::GetConsoleWindow()
    if ($__console -ne [System.IntPtr]::Zero) {
        [void][KoXo.Native]::ShowWindow($__console, 0)   # 0 = SW_HIDE
    }
}

# Non-blocking error notification.
#   $Silent          -> nothing shown (message sent to the error stream only).
#   otherwise        -> a dialog that AUTO-CLOSES after $ErrorTimeout seconds,
#                       so it never waits for a click and never blocks logon.
```

```powershell
function Show-Error {
    param([string]$Message)
    if ($Silent) {
        Write-Error $Message
        return
    }
    # WScript.Shell.Popup: last arg 16 = "Stop" icon; returns at once after the
    # timeout even if the user does not click. This does not block the script.
    $wshell = New-Object -ComObject WScript.Shell
    [void]$wshell.Popup($Message, $ErrorTimeout, $Title, 16)
}

# ------------------------------------------------------------------------
#  0) Parse command-line arguments
#     Accepted forms (prefix - or /, separator =, : or space):
#         -site VALUE   -site=VALUE   -site:VALUE   /site=VALUE   -silent  ...
# ------------------------------------------------------------------------
# Options that are flags (never consume a following token as their value).
$FlagNames = @('silent')

for ($i = 0; $i -lt $args.Count; $i++) {
    $token = [string]$args[$i]

    # Only tokens starting with - or / are treated as option names.
    if ($token -notmatch '^[-/]') { continue }

    # Drop the leading - or /.
    $token = $token.Substring(1)

    # Split "name<sep>value" on the first = or :, else take the next token
    # (except for flags, which never take a value from the next token).
    $name  = $token
    $value = $null
    $sep   = $token.IndexOfAny([char[]]('=', ':'))
    if ($sep -ge 0) {
        $name  = $token.Substring(0, $sep)
        $value = $token.Substring($sep + 1)
    }
    elseif ($FlagNames -notcontains $name.ToLowerInvariant() -and
            ($i + 1) -lt $args.Count -and
            ([string]$args[$i + 1]) -notmatch '^[-/]') {
        $i++
        $value = [string]$args[$i]
    }

    # Strip any surviving surrounding quotes.
    if ($null -ne $value) { $value = $value.Trim('"', "'") }

    switch ($name.ToLowerInvariant()) {
        'site'   { $CompanyShortName       = $value }
        'server' { $KoXoProfilesFileServer = $value }
        'silent' {
            if ($null -eq $value) {
```

```
                $Silent = $true
            }
            else {
                $Silent = @('1', 'true', 'yes', 'on', 'y') -contains $value.ToLowerInvariant()
            }
        }
        default  { }   # unknown option: silently ignored
    }
}

# ------------------------------------------------------------------------
#  1) Determine the server: PDC Emulator unless supplied on the command line
# ------------------------------------------------------------------------
if ([string]::IsNullOrEmpty($KoXoProfilesFileServer)) {
    try {
        # Robust query of the current domain's PDC Emulator role owner (FQDN).
        $Server =
([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()).PdcRoleOwner.Name
        if ([string]::IsNullOrEmpty($Server)) {
            throw 'PDC Emulator not found.'
        }
    }
    catch {
        Show-Error ("Directory access error!`r`n" + $_.Exception.Message)
        exit 1
    }
}
else {
    $Server = $KoXoProfilesFileServer
}

# ------------------------------------------------------------------------
#  2) Build the base UNC share path
# ------------------------------------------------------------------------
$SharePath = "\\$Server\"

if (-not [string]::IsNullOrEmpty($CompanyShortName)) {
    $SharePath += $CompanyShortName + '.KoXoProfiles$\'
}
else {
    $SharePath += 'KoXoProfiles$\'
}

# Architecture-specific executable (preferred candidate).
if ([Environment]::Is64BitOperatingSystem) {
    $PreferredExe = 'ProfilesDownloader64.exe'
}
else {
    $PreferredExe = 'ProfilesDownloader32.exe'
}

# ------------------------------------------------------------------------
#  3) Pick the first existing candidate (preferred, then fallback), then run it
```

```powershell
# ------------------------------------------------------------------------
$Candidates = @($PreferredExe, $FallbackExe)
$Execute    = $null

foreach ($exe in $Candidates) {
    $candidatePath = $SharePath + $exe
    if (Test-Path -LiteralPath $candidatePath -PathType Leaf) {
        $Execute = $candidatePath
        break
    }
}

if ($null -eq $Execute) {
    Show-Error ("No downloader executable found in:`r`n$SharePath`r`n`r`nLooked for: " +
                ($Candidates -join ', '))
    exit 1
}

# ------------------------------------------------------------------------
#  Launch the downloader with an EXPLICIT "show window" flag (SW_SHOWNORMAL).
#  Via CreateProcess so that:
#    - the downloader window is visible even when this script's console is
#      hidden (a child would otherwise inherit the parent's hidden state);
#    - no "Open File - Security Warning" prompt appears (no shell / Attachment
#      Manager involved, unlike ShellExecute).
#  NOTE: the window is only seen if this script runs in the user's interactive
#  session (logon script). A computer/startup script runs in the isolated
#  session 0 and its windows are never shown to the logged-on user.
# ------------------------------------------------------------------------
$si = New-Object 'KoXo.Native+STARTUPINFO'
$si.cb          = [System.Runtime.InteropServices.Marshal]::SizeOf($si)
$si.dwFlags     = 0x00000001    # STARTF_USESHOWWINDOW
$si.wShowWindow = 1             # SW_SHOWNORMAL -> force a visible window
$pi = New-Object 'KoXo.Native+PROCESS_INFORMATION'

$started = [KoXo.Native]::CreateProcess(
    $Execute, $null, [System.IntPtr]::Zero, [System.IntPtr]::Zero,
    $false, 0, [System.IntPtr]::Zero, $SharePath, [ref]$si, [ref]$pi)

if (-not $started) {
    Show-Error ("Cannot start:`r`n$Execute")
    exit 1
}

[void][KoXo.Native]::WaitForSingleObject($pi.hProcess, [uint32]::MaxValue)   # INFINITE: wait for exit
$exitCode = 0
[void][KoXo.Native]::GetExitCodeProcess($pi.hProcess, [ref]$exitCode)
[void][KoXo.Native]::CloseHandle($pi.hThread)
[void][KoXo.Native]::CloseHandle($pi.hProcess)
exit $exitCode
```