LCCPowerShell GUI Administrator's Guide

Description

This document describes the **LCCPowerShellGUI** application.

The **LCCPowerShellGUI** program (application) was designed to allow an ***Information Services*** department to launch ***In-House PowerShell scripts*** from a GUI (window). The scripts are referenced in a ***Centralized Network Location (CNL)***. Each time the application is launched, the latest list of scripts are loaded. Each time a script is requested, it retrieves the latest copy. If a technician changes a script while someone is running the application, the user will be using the newest changes, as long as they are not currently selected on the same Function (script). If so, switching to another Function and back will refresh the script.

The scripts provided with this application were tailored for local and remote machine queries.

INDEX

A Centralized Network Location (CNL) for PowerShell scripts

To allow multiple users to use the same scripts (get updated ones), a network location (share) will need to be created. We recommend making a '***hidden***' share on a server that is accessible by the users the will use this application (i.e. **LCCPowerShellGUI**). Hidden shares end with a dollar sign '***$***'. ***Read-only*** permission will be the only requirement, as the application does not modify files. Both the share and folder/file permissions should only be ***Read-Only*** for the users. All information generated by the application will be *on-the-fly* within the window the user is viewing.

Creating the mandatory Logic file

The application requires a ***Logic file***. The ***Logic file*** is a plain text file (best edited with **Notepad**). This file should be placed in the **CNL**. Each section of the ***Logic file*** is separated by section labels.

The filename of the Logic file does not matter. We recommend using a 'txt' extension.
Example: ourLogic.txt

Any lines beginning with '**#**' will be considered a remark line and ignored. You can give different logic files to different users. This would allow you to provide different **Functions** (scripts) depending on the users needs.

The order of the sections in the **Logic file** do not matter.

Logic file Section Lables

[Logic:AboutFilename] (optional)

The application will display a ***LCC generic About message*** when the user clicks on the About option in the toolbar. If you would like to change this message, create a text file in the **CNL** and add the ***[Logic:AboutFilename]*** section to the logic file. Then enter the filename of the ***About file*** you wish them to see.

Example: If we have an About file called ***about.txt***, we would have the following section in our ***Logic file***...

```
[Logic:AboutFilename]
about.txt
```

[Logic:ScriptPrepend]

What text will be **prePended** to ***script filenames***. This allows you start all your scripts with the same naming convention to better organize.

Example, you may have scripts with 'ourScript' prepended. If you had two scripts, they could end up being named:

ourScriptMeltScreen.ps1
ourScriptFreezeKeyboard.ps1

[Logic:Functions]

What **Functions** will be available to the user. The application will only show the **Functions** (i.e. scripts) that you designate. You may have other scripts in the network location, but, not available to the user. If you desire parameters to be passed to your script, you can designate the name of the parameter to the user after the **Function** name. Place a [tab] character after the **Function** name, followed by the Parameter name (a.k.a. **Input Type**). If you have multiple parameters desired, separate them with a semicolor (no spaces).

It is recommended that **Function** names do not contain spaces.

Example: If you have two **Functions** (scripts), one called ***MeltScreen***, and the other called ***FreezeKeyboard***, and you wanted a parameter of ***thePC*** passed on both, plus a second parameter passed on the ***FreezeKeyboard*** script of theUser, you would have the following two lines:

```
MeltScreen     thePC
FreezeKeyboard       thePC;theUser
```

The Logic File Example

```
# Our Logic File
# Created By: Some User, 1/1/1234
[Logic:ScriptPrepend]
ourScript
[Logic:Functions]
MeltScreen    thePC
FreezeKeyboard       thePC;theUser
```

Creating certificates for signing

To create a **certificate** for use in signing scripts, you will need to run the following command. You will need to be an administrator on the machine.

```
makecert -n "CN=PowerShell Local Certificate Root" -a sha1 -eku 1.3.6.1.5.5.7.3.3 -r -sv root.pvk root.cer -ss Root -sr localMachine
```

Note: This will place a certificate in the store - ***Certificates-->Current User-->Trusted Root Certification Authorities-->Certificates***

```
makecert -pe -n "CN=PowerShell User" -ss MY -a sha1 -eku 1.3.6.1.5.5.7.3.3 -iv root.pvk -ic root.cer;
```

Note: This will place a certificate in the store - ***Certificates-->Current User-->Personal-->Certificates***

Changing/adding scripts

Each time you change/add a script, you will need to update the ***Digital Certificate*** embedded at the end of the script. To sign a script, run the following command at **PowerShell Command Prompt**:

```
Set-AuthenticodeSignature [full path of script] -Certificate @(Get-ChildItem cert:\CurrentUser\My -codesigning)[0];
```

Note: This assumes you are running the **PowerShell Command Prompt** as the user who has a digital certificate for signing installed and that the certificate is referenced by the Id '**My**'. Running the command above will replace any previously placed signature on the script.

Scripts are named according to the following naming standard:

[CNL]\[scriptPrepend][FunctionName].ps1

Example: If your **CNL** was on a server called ***ourServer*** within a share called ***ourScripts$***, you had set your **scriptPrepend** to ***ourScript*** and had a **Function** named ***MeltScreen***, the script file would be named:

\\\\outServer\\ourScripts$\\ourScriptMeltScreen.ps1

Reminder: each time you modify a script, you will need to re-sign them, as the contents have changed and the signature will no longer be valid. To save time, we created a PowerShell script that re-signs each of the scripts in our **CNL**. When one is modified, we re-run the signing script (usually takes around 10 seconds).

Changing/adding help files

The application will look for help files in the **CNL**. It will try to locate a help file with the same filename as the Function (script) filename, except with an appended -help.

Example: If your **CNL** was on a server called ***ourServer*** within a share called ***ourScripts$***, you had set your **scriptPrepend** to ***ourScript*** and had a **Function** named ***MeltScreen***, the script file would be named:

\\\\outServer\\ourScripts$\\ourScriptMeltScreen.ps1

while the help file would be named:

\\\\outServer\\ourScripts$\\ourScriptMeltScreen-help.ps1

Help files are optional. However, they are helpful to the users. When a user switches **Functions**, the application will look for a help file on that **Function** and display it. Click on some of the **Functions** provided with the application to see examples of how we structured our ***help files***.

Information that may benefit the user could be:

+ Created/modified person/date

+ Parameters allowed (optional/mandatory)

+ Description

+ Output fields that will show when the script is executed

Computer that runs LCCPowerShellGUI

To utilize powershell abilities, a computer must have the **PowerShell foundation** installed. ***Windows Server 2008*** and ***Exchange 2007*** already have the **PowerShell foundation**. All other/prior operating systems will need to install PowerShell. See below for installation instructions.

This program was tested utilizing **PowerShell 2.0 CTP** and should function with that version or newer.

The following will be required on the system (see below for more information) for **LCCPowerShellGUI** to function properly:

+ Windows Management Ver 1.1 or newer
+ PowerShell Foundation Ver 2.0 CTP or newer
+ Certificate of the signed scripts
+ Execution policy set to AllSigned
+ Run one script with your digital signature manually first

Windows Management Ver 1.1 or newer
A prerequisite of the **PowerShell foundation** is the installation of ***Windows Management Ver 1.1*** or newer. See **KB936059** for installation instructions/files. The installation is a single exe and should take less than a minute.

PowerShell Foundation Ver 2.0 CTP or newer
The **PowerShell foundation** can be installed by downloading the **MSI** file from ***Microsoft.*** As of this writing, we used PowerShell_2CTP_Setup_x86.msi.

Certificate of the signed scripts
Each script loaded used by **LCCPowerShellGUI** needs to be *digitally signed*. The signature of the script will need to match a valid installed ***Certificate*** in the **Certificates MMC**. This certificate needs to be installed in the following locations:

+ ***Certificates-->Current User-->Personal-->Certificates***

+ ***Certificates-->Current User-->Trusted Root Certification Authorities-->Certificates***

To import the certificate (you will need to be an administrator on the machine):

+ click START-->Run (on **Vista/2008**, click START-->click in the run box
+ type MMC and press ENTER
+ click File
+ click Add/Remove Snap-In
+ click Add
+ click Certificates
+ click Add
+ choose My user account
+ click Finish
+ click Close
+ click OK

[importing into Personal]

+ expand ***Certificates - Current User***

+ click on Personal
+ click Action
+ click All Tasks
+ click Import
+ follow the dialog to locate the certificate
+ choose ***Place all certificates in the following store*** (should say Personal in the box)
+ click Finish
+ click OK on the ***confirmation message***

[importing into Trusted Root Certification Authorities]

+ expand ***Certificates - Current User***
+ click on Trusted Root Certification Authorities
+ click Action
+ click All Tasks
+ click Import
+ follow the dialog to locate the certificate
+ choose Place all certificates in the following store (should say ***Trusted Root Certification Authorities*** in the box)
+ click Finish
+ click YES to the ***Security Warning message***
+ click OK on the ***confirmation message***

+ close the ***Certificates MMC***

Execution policy set to AllSigned
The **PowerShell** user environment defaults to ***Restricted*** mode when running script files. To allow signed scripts to run, you need to change the policy to AllSigned. With this new setting, you can launch scripts that are digitally signed, as long as the signature is from a trusted certificate authority (***Certificate of the signed scripts***).

To change the policy, run the following command from a **PowerShell Command Prompt**:

```
Set-ExecutionPolicy -executionPolicy AllSigned;
```

Run one script with your digital signature manually first
Each time you launch scripts with a new digital signature, **PowerShell** will ask how it should handle scripts with that specific signature. The question will have a format like:

```
Do you want to run software from this untrusted publisher?
The file [script path] is published by CN=<publisher-name>. This
publisher is not trusted on your system. Only run scripts from trusted
publishers.

[V] Never run  [D] Do not run  [R] Run once  [A] Always run
[?] Help (default is "D"):
```

For **Microsoft** scripts or scripts you have signed with an in-house certificate, run one of the scripts once and choose 'A' for Always. Otherwise, the **LCCPowerShellGUI** application will not be able to execute the scripts.

This step will only need to be performed once per certificate.

Remote network computers

Depending on your Group Policies, client computers may not allow RPC calls from PowerShell. To enable this, do the following.

In GPO, edit the policy which defines:
Computer Configuration
Administrative Templates
Network
Network Connections
Windows Firewall
Domain Profile (Standard Profile may need to be updated as well if configured)
Allow unsolicited incoming messages from
Enter the IP(s) as noted in the help information

Definitions

CNL - Centralized Network Location. A place on the network where multiple users can retrieve the PowerShell scripts.

GPO - Group Policy Object. A set of policies defined by a domain admin through Active Directory.

GUI - Graphical User Interface (a.k.a. window)

IP - Internet Protocol. The internet address of a computer. Example: 123.123.123.123

MMC - Microsoft Management Console. A window that allows you to load components of Microsoft tools, like Certificates, Computer Management, etc..

PowerShell - PowerShell is Windows new Command Prompt technology. With PowerShell, you can now do full programming, as opposed to batch files. PowerShell supports full programming logic, like If/Then, Switch, file/data manipulations, WMI objects (i.e. computer information, like memory, disks, etc..). Windows 2008/Exchange 2007 were built on top of PowerShell.

RPC - Remote Procedure Calls. The method which allows one computer to request information/perform actions on another computer.

Modifications

NAME	DATE	MODIFICATION
David Mielcarek	8/13/2008	Created
David Mielcarek	8/19/2008	Updated information

End of document