NTLua Scripting Windows NT Administration and Logon
Version 2

Overview

NTLua is a console application to create Windows NT administration and logon scripts.

It can be used as a Logon scripting tool. Copy the "ntlua.exe" file to the NTLOGON share, edit a file called "ntlua.lua" in the same place, and configure the user logon script to "ntlua.exe". The "ntlua.lua" script will be automatically loaded at logon time.

NTLua runs only in Windows NT systems (4.0, 2000 and XP). It does not run in Windows 9x/Me.

The NTLua script log has a blue background. Normal text is printed in white, errors are printed in red, and warnings are printed in yellow. If the text is greater than the buffer will automatically pause (an automatic "more").

The interactive console has some special features inherited from LuaCmd (http://www.tecgraf.puc-rio.br/luacmd).

Usage

Usually you will execute:

```
    ntlua somescript.lua arg1 arg2 arg3 ...
```

Arguments are passed to the Lua environment as a table "arg[1]", "arg[2]", "arg[3]" and so on, "arg.c" contains the number of arguments. Also the script file name can be used inside the script accessing the variables `SCRIPT_FILE_NAME`, `SCRIPT_NAME`, `SCRIPT_PATH` (filename = path / name).

If executed without a script it will try to load a script called "ntlua.lua" if it fails to find it , it starts the console.

Samples

Logon Script - ntlua.lua

Domain User Creation - mkuser.lua

Domain User Removal - rmuser.lua

History

26 Apr 2002 - Version 2.1.2 - Changed the behavior of the functions: **version** and **os**. They also now supports XP.
19 Apr 2002 - Version 2.1.1 - New funtions: **readinisection**, **readinikeys**, **readinisections**. (Thanks to Juan Duarte for the suggestion.)
05 Oct 2001 - Version 2.1- Now only invalid parameter errors will abort scripts. Some startup code were improved. The **wait** param in **start** function is now optional. **df** now returns more precise values and accepts a unit division.
25 Oct 2000 - Version 2.0
22 Feb 1999 - Version 1.0 (only a logon script tool, this version runs also under Windows 9x)

To Do's

Windows API - Services, Process (List and Kill), Shell DDE/Shell Link, Shell Special Folders

AppActivate / SendKeys - to control an application.

LogToFile - log output to a file.

Support

If you interested in help, send comments, critics, suggestions, etc to me. Please, specify platform, compiler, version you are using in your message.

Looking for Lua? http://www.tecgraf.puc-rio.br/lua.

This program is free for every usage. The source code is public available. The author does not offer any guaranties, nor support, etc...

Author

Antonio E. Scuri (messages to scuri@tecgraf.puc-rio.br, home page in Portuguese at http://www.tecgraf.puc-rio.br/~scuri).

The code for changing access control lists was used from the Platform SDK tool cacls developed by Dave Mont and they are copyright of Microsoft.

Copyright

See the Copyright Notice, is the same copyright used by Lua.

Download

The program source code, HTML pages, samples, and pre-compiled binaries:

ntlua212src.zip (source code, html, makefiles) - 120Kb
ntlua212.zip (binary only) - 80Kb

ntlua10.zip (binary only) - 100Kb

About NTLua Online

This user guide is available at http://www.tecgraf.puc-rio.br/~scuri/ntlua.

The manual is also available in Windows HTML Help format (NTLUA.CHM 20Kb).

This manual was created using the manual creation toolkit WebBook, which can be found at http://www.tecgraf.puc-rio.br/webbook.

.. *"Make it Reusable, Make it Simple, Make it Small"* ...

Functions

Interaction

`echo/print(text)` - Prints the **`text`** string on the console. Always include a line break.

`cls()` - Clears he console string.

`beep()` - Sounds a beep.

`wait(time)` - Delay processing for **time** milliseconds.

`msgbox(text, title, type, icon)` - Displays the standard message box dialog. **type** can be `MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL, MB_ABORTRETRYIGNORE`. **icon** can be `MB_ICONEXCLAMATION, MB_ICONINFORMATION, MB_ICONQUESTION, MB_ICONSTOP`.

`pause()` - Displays the message "Press any key to continue..." and stops the script execution.

`getkey()` - Waits until the user press a key and returns that key as a string.

`getline()` - Get keys until the user press Enter, then returns the keys in a string.

`ok()` - Returns a non nil value if the script execution is ok.

System Information

`computername()` - Returns the current computer name.

`username/whoami()` - Returns the current user name.

`version()` - Returns a detailed Operating System string description. Ex:

```
Microsoft Windows 2000 Workstation [Version 5.0.2195] Service Pa
```

`os()` - Returns the same informatio above but in a Lua table.

```
{
  system = "2000",
  type = "Workstation",
  major = 5,
  minor = 0,
  build = 2195,
  service = "Service Pack 1"
}
```

`datetime()` - Returns a string describing the current date and time (ex. "Thu Oct 26 11:01:51 2000").

`ipaddress()` - Returns the computer IP address if any configured. Return nil if not.

Environment

`setenv(name, value)` - Sets an environment variable in the user environment.

`getsysenv(name)` - Returns an environment variable if its set in the system environment.

`getusrenv(name)` - Returns an environment variable if its set in the user environment.

`expandenv(name)` - Expands an environment variable that contains references to other environment variables. Returns the result.

Registry

`writeregkey(bkey, key_name, value_name, value)` - Writes a string in the registry. **key_name** is the path. **bkey** can be: `KEY_CLASSES_ROOT`, `KEY_CURRENT_USER`, `KEY_LOCAL_MACHINE`, `KEY_USERS`.

`readregkey(bkey, key_name, value_name)` - Reads a string from the registry and returns it. **key_name** is the path. **bkey** can be: `KEY_CLASSES_ROOT`, `KEY_CURRENT_USER`, `KEY_LOCAL_MACHINE`, `KEY_USERS`.

Utilities

`writeinikey(filename, section, value_name, value)` - Writes a string in a ".INI" file.

`readinikey(filename, section, value_name)` - Reads a string from a ".INI" file and returns it.

`readinikeys(filename, section)` - Reads all the keys from a ".INI" file section and returns them as a table.

`readinisection(filename, section)` - Reads all the keys and values from a ".INI" file section and returns them as a table indexed by the key names.

`readinisections(filename)` - Reads all the sections names from a ".INI" file and returns them as a table.

`logevent(servername, type, msg, datastr)` - Sends an event to the application log of the specified computer. If **servername** is nil it uses the local computer. **type** can be: `EVENTLOG_ERROR_TYPE`, `EVENTLOG_WARNING_TYPE`, `EVENTLOG_INFORMATION_TYPE`, `EVENTLOG_AUDIT_SUCCESS`, `EVENTLOG_AUDIT_FAILURE`.

`netsend(servername, dstname, msg)` - Sends a message to s specified user or computer. If **servername** is nil it uses the local computer.

`shutdown(servername, message, timeout, force, reboot)` - Initiates a shutdown and optional restart of the specified computer. **message** is optional and **timeout** is in seconds. If **servername** is nil it uses the local computer.

`winexit(flag, force)` - Either logs off the current user, shuts down the system, or shuts down and restarts the system.. **flag** can be `EWX_LOGOFF`, `EWX_POWEROFF`, `EWX_REBOOT`, `EWX_SHUTDOWN`. **force** will force processes to terminate.

Process

`execute(command_line)` - Starts the command line as the Explorer "open" command. So you can execute folders, documents and executables.

`start(command_line, wait)` - Starts the command line creating a new process. **wait** forces the execution to wait for the process to terminate, can be omitted, default is no wait.

Net Drive Map

`netuse(driveletter, path)` - Connects a drive letter to a network path.

`netdel(driveletter)` - Disconnects a mapped drive letter.

Share

`netshareadd(servername, sharename, path, comment)` - Adds a share to the specified computer. **comment** is optional. If **servername** is nil it uses the local computer.

`netsharedel(servername, sharename)` - Removes a share from the specified computer. If **servername** is nil it uses the local computer.

`netshareenum(servername, doshare_func)` - Calls the specified function for each share in the specified computer. If **servername** is nil it uses the local computer. **doshare_func** must be a function like:

```
function do_share(sharename, path, type)
{
  if (abort) then
    return 0
  else
    return 1;
  end
}
```

type can be: `"FOLDER", "PRINTER", "DEVICE", "IPC", "SPECIAL".`
If **shareenum_func** is nil it will print all the shares.

User

For all the functions: **servername** is the computer where the function actually executes. If nil executes at the local computer. **local** (1 or 0) specifies that the group is a local group.

`netuseradd(servername, username, full_name, comment, password, profile_path, script_path, home_dir_drive, home_dir_path)` - Creates a new user. **`full_name, comment, password, profile_path, script_path, home_dir_drive, home_dir_path`** are optional and can be nil.

`netuserrename(servername, oldusername, newusername)` - Renames the user.

`netusersetinfo(servername, username, param_name, param)` - Changes user parameters. **param_name** can be: "full_name", "comment", "profile_path", "script_path", "home_dir_drive", home_dir_path.

`netusergetinfo(servername, username)` - Returns the user full name, comment, profile path, script path, home dir drive and home dir path.

`netuserdel(servername, username)` - Removes the user.

`netuserenum(servername, douser_func)` - Calls the specified function for each user. **douser_func** must be a function like:

```
function do_user(username)
{
  if (abort) then
    return 0
  else
    return 1;
  end
}
```

`netusergroupsenum(servername, username, local, dousergroup_func)` - Calls the specified function for each group the user belongs to. **dousergroup_func** must be a function like:

```
function do_usergroup(groupname)
{
  if (abort) then
    return 0
  else
    return 1;
    end
  }
```

`ifmember(servername, username, local, groupname)` - Checks if the user belongs to a group. Returns nil otherwise.

`netuserchangepassword(domainname, username, oldpassword, newpassword)` - Change the user password. **domainname** can be nil to specify the current domain. **username** can be nil to specify the current user.

Groups

For all the functions: **servername** is the computer where the function actually executes. If nil executes at the local computer. **local** (1 or 0) specifies that the group is a local group.

`netgroupadduser(servername, local, groupname, username)` - Adds a user to a group.

`netgroupdeluser(servername, local, groupname, username)` - Removes a user from a group.

`netgroupadd(servername, local, groupname, comment)` - Creates a new group. **comment** is optional, can be nil.

`netgroupdel(servername, local, groupname)` - Removes a group.

`netgrouprename(servername, local, oldgroupname, newgroupname)` - Renames a group (this is not available in the UserManager, dont' know why).

`netgroupsetinfo(servername, local, groupname, param_name, param)` - Changes group comment. **param_name** can only be "comment".

`netgroupgetinfo(servername, local, groupname)` - Returns the group comment.

`netgroupenum(servername, local, dogroup_func)` - Calls the specified function for each group. **dogroup_func** must be a function like:

```
  function do_group(groupname)
  {
    if (abort) then
      return 0
    else
      return 1;
    end
  }
```

Domain

`netgetdomain()` - Returns the current user domain.

`netgetpdc(servername, domainname)` - Returns the name of the Primary Domain Controller of the specified domain. If **servername** is nil it uses the local computer.

File

`getdir(dir, file_mask, subdir)` - Returns a table with all the folder structure of the specified folder. Subfolder are treated as new sub tables, and files are values. **file_mask** can restrict the file selection, if nil the default is "*.*". **subdir** allows recursion of subfolders.

`forfiles(dir, file_mask, subdir, dofile_func)` - Calls the specified function for each file selected by the **file_mask** inside the **dir** folder structure. **subdir** allows recursion of subfolders. **dofile_func** must be a function like:

```
function do_file(filename, status)
{
  if (abort) then
    return 0
  else
    return 1;
  end
}
```

status is 1 when entering the specified folder, -1 leaving the specified folder, 0 is a file.

`cacls(filefilter, subdir, continue, clear, grant, replace, revoke, deny)` - Changes access control lists of the specified files. **subdir**

allows recurtion of subfolders. the function stops if an error occurs, **continue** allows to ignore errors. **clear** will clear all the actual acls before doing any operations, so **replace** and **deny** can not be used, and must be nil. **grant** and **replace** are tables with the list of users and permissions to grant or replace respectively in the acls (ex: {"user1", "F", "test2", "R"}). **revoke** and **deny** are tables with the list of users to revoke from the list or deny access respectively. Permissions can be: "F", "C", "R", "N" (Full, Change, Read, None). The table can also be nil, but not all of them at the same time.

`dumpacls(filefilter, subdir)` - Dumps (print) access control lists of the specified files. **subdir** allows recurtion of subfolders.

`exist(filename)` - Returns a non nil value if the file exists.

`windir()` - Returns the Windows folder.

`sysdir()` - Returns the Windows System folder.

`getcwd/pwd()` - Returns the current folder.

`mkdir/md(dirname)` - Creates a folder.

`rmdir/rd(dirname)` - Removes an empty folder.

`chdir/cd(dirname)` - Changes the current folder. If **dir** is not specified returns the current folder.

`copy/cp(srcfilename, dstfilename)` - Copies a file. Overwrite destination if exists.

`move/mv/rename(srcfilename, dstfilename)` - Moves or renames a file. Overwrite destination if exists.

`copydir(srcdirname, dstdirname)` - Copies the contents of a folder to another.

`deltree/prune(dirname)` - Removes a folder and all its contents including subfolders.

`del/remove/rm/erase(filename)` - Removes the file specified.

`access(filename, mode)` - Determine file-access permission. **mode** can be: "x", "w", "r", "f" (existence, can write, can read, can read or write). Returns a non nil value if mode is allowed.

`attrib(filename, mask)` - Changes the attributes of a file. **mask** can be a combination of the following: "a", "h", "o", "r", "s" ,"t" (archive, hidden, offline, read-only, system, temporary). If **mask** is nil returns the current attributes mask.

`filesize(filename)` - Returns the size of the file in bytes.

`filetitle(filename)` - Returns the name of the file without any path. (filename = path / title)

`filepath(filename)` - Returns the path of the file.

`filefullpath()` - Returns the complete filename of a file. You can specify just the title and you get the current folder and add to the file tile.

`filetime(filename)` - Returns 3 strings regarding to file time: *creationtime*, *lastaccesstime*, *lastwritetime*. Uses the same format as the **datetime** function.

`fileshortpath(filename)` - Returns the short name of the file (8.3 characters).

`bintype(filename)` - Returns the binary type of the executable. It can be: "Win32", "DOS", "OS216", "PIF", "POSIX", "Win16" or nil if unknown.

`where(filename)` - Searches for the file in the PATH and returns the complete file name if found. Returns nil otherwise.

`df/diskfree(filename, div)` - Returns 3 values regarding to disk space: *available*, *total* and *free*. If per-user quotas are in use, the first value may be less than the total number of free bytes on the disk. If **filename** is nil uses the current folder. **div** can be "b", "Mb", "Kb" and "Gb". **div** is optional, and the default is "Mb".

LuaCmd COPYRIGHT NOTICE

The **LuaCmd** implementation have been entirely designed and written by **Antonio Escaño Scuri** at TeCGraf, PUC-Rio.

This implementation contains no third-party code.

==