FREE eBook

LEARNING

nsis

Free unaffiliated eBook created from
Stack Overflow contributors.

#nsis

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: nsis

It is an unofficial and free nsis ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official nsis.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with nsis

Remarks

NSIS (Nullsoft Scriptable Install System) is a professional open source system to create Windows installers. It is designed to be as small and flexible as possible and is therefore very suitable for internet distribution.

Being a user's first experience with your product, a stable and reliable installer is an important component of succesful software. With NSIS you can create such installers that are capable of doing everything that is needed to setup your software.

NSIS is script-based and allows you to create the logic to handle even the most complex installation tasks. Many plug-ins and scripts are already available: you can create web installers, communicate with Windows and other software components, install or update shared components and more.

--> Taken from: https://sourceforge.net/projects/nsis/

Versions

Version	Notes	Release Date
2.00.0		2004-02-07
2.46.0	Probably the most common version seen in the wild.	2009-12-06
2.51.0	Final 2.x release, contains important security fixes.	2016-04-01
3.00.0	First version with official Unicode support.	2016-07-24

Examples

Installation or Setup

NSIS installer can be downloaded from http://nsis.sourceforge.net/Download. An exe of 1.6 MB will be downloaded. You can install it using the wizard. While installing there are options to install

```
1. Full: Installs all the components
2. Lite: Basic and only essential components from the UI
3. Minimal: The NSIS core files only.
4. Custom: It's up to us to install whichever components that we need.
```

Hello World!

Code, to be saved in „helloworld.nsi“:

```
Name "Hello World"
OutFile "helloworld.exe"
Section "Hello World"
MessageBox MB_OK "Hello World!"
SectionEnd
```

Compile it with:

```
<Path to NSIS>\makensis.exe <Path to script>\helloworld.nsi
```

Read Getting started with nsis online: https://riptutorial.com/nsis/topic/5491/getting-started-with-nsis

Chapter 2: DotNET

Introduction

When the .NET Framework is being installed on a computer the .NET installer writes registry keys when installation is successful. You can test whether the .NET Framework 4.5 or later is installed by checking the registry key `HKLM\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full` for a `DWORD` value named *Release*. The existence of this key indicates that the .NET Framework 4.5 or later has been installed on that computer.

Remarks

.NET Values

Below is a chart that shows the versions number corresponding `DWORD` value in the *Release* key. See below this chart for an example on how to use this in action. I will do my best to try and keep this page up-to-date with the latest values but you can visit this page for an updated list if the version you need isn't listed here.

> Take special notice how there are two rows for `4.7`, `4.6.2`, `4.6.1`, and `4.6`. These versions require you to check for both values as the operating systems vary.

Version	Operating System	Value of Release
4.7	Windows 10 Creator's Update	460798
4.7	All except Windows 10 Creator's Update	460805
4.6.2	Windows 10 Anniversary Update	394802
4.6.2	All except Windows 10 Anniversary Update	394806
4.6.1	Windows 10 November Update	394254
4.6.1	All except Windows 10 November Update	394271
4.6	Windows 10	393295
4.6	All except Windows 10	393297
4.5.2	All	379893
4.5.1	Windows 8.1 or Windows Server 2012 R2	378675
4.5.1	Windows 8 and Wiindows 7	378758

Version	Operating System	Value of Release
4.5	All	378389

Conclusion

> I haven't fully tested these functions completely so if you run into problems with one of them let me know and I'll do my best to help. You can see this function working in action with SharpDevelop Portable.

Like I said above I will do my best to try and keep these up-to-date but if you need a version that these functions aren't checking for, let me know so I can revise them and update this page.

Examples

Sample Example

Here's an example on how to use the values in the registry to check for dotNET 4.5 or higher. I'd recommend putting this snippet of code somewhere like `.OnInit` as this will execute before anything else happens; this way it checks for .NET before any files get copied or registry changes take place.

```
;=#
;= Define the registry key we're looking for.
!define DOTNET  `SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full`

Section "Main"
    ClearErrors ;= Clear any errors we may have encountered before hand.
    ReadRegDWORD $0 HKLM `${DOTNET}` `Release` ;= Read the value of Release.
    IfErrors +3 ;= If there is an error than there is no .NET installed so we jump 3 lines
down which lands on MessageBox.
    IntCmp $0 394806 +5 0 +5    ;= Compares the value for v4.6.2 if it matches then we jump 5
lines and avoids the MessageBox
    IntCmp $0 394802 +4 0 +4    ;= Remember to check for Windows 10's value aswell as the
above line won't.
    MessageBox MB_ICONSTOP|MB_TOPMOST `You must have v4.6.2 or greater of the .NET Framework
installed. Launcher aborting!` ;= If the check failed then we alert the user the required
version wasn't found.
    Call Unload ;= We call the Unload function here because we failed the .NET check.
    Quit ;= Closes the Launcher
SectionEnd
```

Function w/ Macro

Function dotNETCheck

Alternatively you can use the function I wrote below. This one makes use of *LogicLib.nsh*. It should work out-of-the-box without having to know the .NET versions value from the *Release* key in the registry. As it is written right now it only checks for versions between `4.5`—`4.7`.

```
##
# This one requires the use of LogicLib.nsh
# Copy and paste this code somewhere like .OnInit
Function dotNETCheck
    !define CheckDOTNET "!insertmacro _CheckDOTNET"
    !macro _CheckDOTNET _RESULT _VALUE
        Push `${_VALUE}`
        Call dotNETCheck
        Pop ${_RESULT}
    !macroend
    Exch $1
    Push $0
    Push $1

    ${If} $1 == "4.7"
        StrCpy $R1 460798
    ${ElseIf} $1 == "4.6.2"
        StrCpy $R1 394802
    ${ElseIf} $1 == "4.6.1"
        StrCpy $R1 394254
    ${ElseIf} $1 == "4.6"
        StrCpy $R1 393295
    ${ElseIf} $1 == "4.5.2"
        StrCpy $R1 379893
    ${ElseIf} $1 == "4.5.1"
        StrCpy $R1 378675
    ${ElseIf} $1 == "4.5"
        StrCpy $R1 378389
    ${Else}
        Goto dotNET_FALSE
    ${EndIf}

    ReadRegDWORD $R0 HKLM `SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full` `Release`
    IfErrors dotNET_FALSE

    IntCmp $R0 $R1 dotNET_TRUE dotNET_FALSE

    dotNET_TRUE:
    StrCpy $0 true
    Goto dotNET_END

    dotNET_FALSE:
    StrCpy $0 false
    SetErrors

    dotNET_END:
    Pop $1
    Exch $0
FunctionEnd

##
# USAGE
# ${CheckDOTNET} $0 "Version Number"
##
# $0 Will hold the version number of the installed .NET
# If $0 is empty ($0 == "") then the error flag is set.
${CheckDOTNET} $0 "4.5"
IfErrors 0 +4
MessageBox MB_ICONSTOP|MB_TOPMOST `You must have v4.5 or greater of the .NET Framework
installed. Launcher aborting!`
Call Unload
```

```
Quit
StrCmpS $0 true 0 -3
```

Read DotNET online: https://riptutorial.com/nsis/topic/10139/dotnet

Chapter 3: Registering Libraries (RegDLL)

Introduction

In short, a DLL is a collection of small executable code, which can be called upon when needed by a program that's running. The DLL lets the executable communicate with a specific device such as a printer or may contain code to do any number of particular functions. As there are several methods of implementations to do this, in this topic I'll be showing you how to register and unregister any DLL that your application calls for; and we'll be doing so using the `RegSrv32.exe` command line.

Syntax

- `${Register::DLL} "codec.ocx" "" $1 $0`
- `${UnRegister::DLL} "volumeCtrl.cpl" /DISABLEFSR $1 $0`
- `${RegisterDLL} "print.drv"`

Parameters

Switch	Discription
`/u`	Unregisters server.
`/s`	Specifies `regsvr32` to run silently and to not display any message boxes.
`/n`	Specifies not to call *DllRegisterServer*. You must use this option with `/i`.
`/i` **:cmdline**	Calls *DllInstall* passing it an optional **[cmdline]**. When used with `/u`, it calls dll uninstall.
dllname	Specifies the name of the dll file that will be registered.
`/?`	Displays help at the command prompt.

Remarks

RegSrv32.exe

Using the `RegSvr32.exe` command line is my preferred method on getting you libraries registered so let's start here first. Windows PCs coupled with Internet Explorer 3.0 or later come stock with `RegSvr32.exe`. So there's a good chance your PC comes standard with this utility. Now if you are running on a 64-bit machine, there are two variants you can consider. They can be found in either `$WINDIR\system32` or `$WINDIR\SysWow32`.

The parameters you can use with RegSrv32 are `/u /s /i /n`. The `/u` command switch will unregister the file. The `/i` switch can be used with `/u` to call for DLL uninstallation. The `/n` parameter will not call *DllRegisterServer*; it's used with `/i` which is the install switch. If you use `/s`, which means silent, no message boxes will be displayed on Windows XP or later.

When using `RegSvr32.exe` from the command line you'll get message boxes after calling it. The *DLLSelfRegister* function will be invoked unless using the aforementioned switch of course; if successful an alert box will be shown denoting its success—as the same for failure which throws an error message.

> It's been my experience that the x64 `RegSvr32.exe` registers x86 DLL's properly on Windows Vista and above (excludes XP; visit this article for more) and above so I use it when installing on x64 systems even when registering a x86 file. Besides, Windows XP is a dying art; bless it's heart. =)

Examples

RegSrv32.exe Simple Usage

```
#= Regardless of architecture we're using just the following

!define REGSVR `$SYSDIR\regsvr32.exe` #= define where RegSrv32 is
!define DLL `$AppDirectory\App\MyLegalProgram\myLegit.dll` #= define the file to register

##=
#= Command line usage is the same for both variants of RegSrv32 as follows
#= regsvr32 [/u] [/s] [/n] [/i[:cmdline]] DLL
#=
##= So in our .nsi file it would be similar to the following:

Exec `"${REGSVR}" /s "${DLL}"`

#= Moreover, you may also use the following

ExecWait `"${REGSVR}" /s "${DLL}"` $0 #= The $0 will contain the error code if any

#= The above will wait for exe to quit it's process before continuing
```

Macro: Register::DLL

```
##=
# The variable $Bit will hold either 64 or 32 depending on system architecture
# This is used with ${Register::DLL} and ${UnRegister::DLL}
# Use this in the beginning of your code.
# This snippet should only be used once.
#
Var Bit
System::Call "kernel32::GetCurrentProcess()i.s"
System::Call "kernel32::IsWow64Process(is,*i.r0)"
StrCmpS $0 0 +3
StrCpy $Bit 64
Goto +2
StrCpy $Bit 32
```

```
##=
#= Register::DLL
#
# USAGE:
# ${Register::DLL} "DLL Filename" /DISABLEFSR $0 $1
#
#        ::DLL       = Registers a DLL file.
#        /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#        $0          = Return after call
#        $1          =  ''      ''    ''
#
!define Register::DLL `!insertmacro _Register::DLL`
!macro _Register::DLL _DLL _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +4
    StrCmp ${_FSR} /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"${REGSVR}" /s "${_DLL}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${REGSVR}" /s "${_DLL}"`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend

##=
# Alternatively you can use this macro found in my travels
# but you should include x64.nsh as this macro makes use
# of it's function.
#
#= USAGE:
# ${RegisterDLL} "SomeLibrary.dll"
#
!define RegisterDLL "!insertmacro _RegisterDLL"
!macro _RegisterDLL _DLL
    ${If} ${RunningX64}
        ${DisableX64FSRedirection}
        ExecWait '"$SYSDIR\regsvr32.exe" /s "${_DLL}"'
        ${EnableX64FSRedirection}
    ${Else}
        RegDLL "${DLL}"
    ${EndIf}
!macroend
```

The /DISABLEFSR parameter should only be used on x64 machines. However, if you mess up there's a failsafe in the macros that will dodge this bullet for you.

Macro: UnRegister::DLL

```
##=
# The variable $Bit will hold either 64 or 32 depending on system architecture
# This is used with ${Register::DLL} and ${UnRegister::DLL}
# Use this in the beginning of your code.
# This snippet should only be used once.
#
Var Bit
System::Call "kernel32::GetCurrentProcess()i.s"
System::Call "kernel32::IsWow64Process(is,*i.r0)"
StrCmpS $0 0 +3
StrCpy $Bit 64
Goto +2
```

```
StrCpy $Bit 32

#=#
#= UnRegister::DLL
#
# USAGE:
# ${UnRegister::DLL} "DLL Filename" /DISABLEFSR $0 $1
#
#        ::DLL       = Unregisters a DLL file.
#        /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#        $0          = Return after call
#        $1          =  ''      ''    ''
#
!define UnRegister::DLL `!insertmacro _UnRegister::DLL`
!macro _UnRegister::DLL _DLL _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +4
    StrCmp ${_FSR} /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"${REGSVR}" /s /u "${_DLL}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${REGSVR}" /s /u "${_DLL}"`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend

##=
# Alternatively you can use this macro I found in my travels
# but be sure to include the x64 plugin as this macro makes
# use of it's function.
#
#= USAGE:
# ${UnregisterDLL} "SomeLibrary.dll"
#
!define UnregisterDLL "!insertmacro _UnregisterDLL"
!macro _UnregisterDLL _DLL
    ${If} ${RunningX64}
        ${DisableX64FSRedirection}
        ExecWait '"$SYSDIR\regsvr32.exe" /s /u "${_DLL}"'
        ${EnableX64FSRedirection}
    ${Else}
        UnRegDLL "${DLL}"
    ${EndIf}
!macroend
```

> The /DISABLEFSR parameter should only be used on x64 machines. However, if you mess up there's a failsafe in the macros that will dodge this bullet for you.

Read Registering Libraries (RegDLL) online: https://riptutorial.com/nsis/topic/10142/registering-libraries--regdll-

Chapter 4: Services

Introduction

When installing a new program or updating an installation, it's good practice for you to stop an installed application and anything related with it before overwriting any of its files. The same goes for services. We need to be sure that the locally run service (if any) is stopped before we can install or upgrade our program. In this topic I'll share a few macros ranging from creating a service to querying a service to removing one all together.

Syntax

- ${Service::Create} "NAME" "PATH" "TYPE" "START" "DEPEND" /DISABLEFSR $0 $1
- ${Service::Remove} "NAME" "" $0 $1
- ${Service::QueryConfig} "NAME" /DISABLEFSR $0 $1

Parameters

${Service::Create}	Description
NAME	The service name
PATH	BinaryPathName to the .exe file
TYPE	own, share, interact, kernel, filesys, rec
START	boot, system, auto, demand, disabled, delayed-auto
DEPEND	Dependencies(separated by / (forward slash))
/DISABLEFSR	Disables redirection if x64. Use "" to skip.
$0	Return after call
$1	Return after call

Examples

Service::Create

```
##=
# The variable $Bit will hold either 64 or 32 depending on system architecture
# This is used with all the ${Service::} macros
# Use this in the beginning of your code.
# This snippet should only be used once.
```

```
#
Var Bit
System::Call "kernel32::GetCurrentProcess()i.s"
System::Call "kernel32::IsWow64Process(is,*i.r0)"
StrCmpS $0 0 +3
StrCpy $Bit 64
Goto +2
StrCpy $Bit 32

##=
#= Service::Create
#
# USAGE:
# ${Service::Create} "NAME" "PATH" "TYPE" "START" "DEPEND" /DISABLEFSR $0 $1
#
#    ::Create    = Creates a service entry in the registry and Service Database
#    NAME        = The Service name
#    PATH        = BinaryPathName to the .exe file
#    TYPE        = own|share|interact|kernel|filesys|rec
#    START       = boot|system|auto|demand|disabled|delayed-auto
#    DEPEND      = Dependencies(separated by / (forward slash))
#    /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#    $0          = Return after call
#    $1          =   ''    ''    ''
#
!define Service::Create `!insertmacro _Service::Create`
!macro _Service::Create _SVC _PATH _TYPE _START _DEPEND _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +7
    StrCmp "${_FSR}" /DISABLEFSR 0 +6
        StrCmp "${_DEPEND}" "" 0 +3
        ExecDos::Exec /TOSTACK /DISABLEFSR `"${SC}" create "${_SVC}" DisplayName=
"${FULLNAME}" binpath= "${_PATH}" type= "${_TYPE}" start= "${_START}"`
        Goto +7
        ExecDos::Exec /TOSTACK /DISABLEFSR `"${SC}" create "${_SVC}" DisplayName=
"${FULLNAME}" binpath= "${_PATH}" type= "${_TYPE}" start= "${_START}" depend= ""${_DEPEND}""`
        Goto +5
    StrCmp "${_DEPEND}" "" 0 +3
    ExecDos::Exec /TOSTACK `"${SC}" create "${_SVC}" DisplayName= "${FULLNAME}" binpath=
"${_PATH}" type= "${_TYPE}" start= "${_START}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${SC}" create "${_SVC}" DisplayName= "${FULLNAME}" binpath=
"${_PATH}" type= "${_TYPE}" start= "${_START}" depend= ""${_DEPEND}""`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

The /DISABLEFSR parameter should only be used on x64 machines. However, if you mess up there's a failsafe in the macros that will dodge this bullet for you. This applies to al the service macros listed here.

Service::QueryConfig

```
##=
#= Service::QueryConfig
#
# USAGE:
# ${Service::QueryConfig} "NAME" /DISABLEFSR $0 $1
#
```

```
#    ::QueryConfig = The service's binary path is returned.
#    NAME          = The Service name
#    /DISABLEFSR   = Disables redirection if x64. Use "" to skip.
#    $0            = Return after call | 1 = success
#    $1            =   ''    ''    ''  | Should be the file path
#
# $1 will now hold the path to it's binary executable or an error
#
!define Service::QueryConfig `!insertmacro _Service::QueryConfig`
!macro _Service::QueryConfig _SVC _FSR _ERR1 _ERR2
    ReadEnvStr $R0 COMSPEC
    StrCmpS $Bit 64 0 +4
    StrCmp "${_FSR}" /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"$R0" /c "${SC} qc "${_SVC}" | FIND
"BINARY_PATH_NAME""`
    Goto +2
    ExecDos::Exec /TOSTACK `"$R0" /c "${SC} qc "${_SVC}" | FIND "BINARY_PATH_NAME""`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

Service::State

```
##=
#= Service::State
#
# USAGE:
# ${Service::State} "NAME" /DISABLEFSR $0 $1
#
#    ::State     = The service's status is returned.
#    NAME        = The Service name
#    /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#    $0          = Return after call | 1 = success
#    $1          =   ''    ''    ''  | 1 = running
#
# $1 will now hold "1" if running or "0" if not
#
!define Service::State `!insertmacro _Service::State`
!macro _Service::State _SVC _FSR _ERR1 _ERR2
    ReadEnvStr $R0 COMSPEC
    StrCmpS $Bit 64 0 +4
    StrCmp "${_FSR}" /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"$R0" /c "${SC} query "${_SVC}" | find /C "RUNNING""`
    Goto +2
    ExecDos::Exec /TOSTACK `"$R0" /c "${SC} query "${_SVC}" | find /C "RUNNING""`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

Service::Start

```
##=
#= Service::Start
#
# USAGE:
# ${Service::Start} "NAME" /DISABLEFSR $0 $1
#
```

```
#    ::Start     = Start a service.
#    NAME        = The Service name
#    /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#    $0          = Return after call
#    $1          =   ''    ''    ''
#
# $1 will now hold "1" if running or "0" if not
#
!define Service::Start `!insertmacro _Service::Start`
!macro _Service::Start _SVC _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +4
    StrCmp "${_FSR}" /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"${SC}" start "${_SVC}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${SC}" start "${_SVC}"`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

Service::Stop

```
##=
#= Service::Stop
#
# USAGE:
# ${Service::Stop} "NAME" /DISABLEFSR $0 $1
#
#    ::Stop      = Sends a STOP control request to a service.
#    NAME        = The Service name
#    /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#    $0          = Return after call
#    $1          =   ''    ''    ''
#
!define Service::Stop `!insertmacro _Service::Stop`
!macro _Service::Stop _SVC _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +4
    StrCmp "${_FSR}" /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"${SC}" stop "${_SVC}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${SC}" stop "${_SVC}"`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

Service::Remove

```
##=
#= Service::Remove
#
# USAGE:
# ${Service::Remove} "NAME" /DISABLEFSR $0 $1
#
#    ::Remove    = Deletes a service entry from the registry.
#    NAME        = The Service name
#    /DISABLEFSR = Disables redirection if x64. Use "" to skip.
#    $0          = Return after call
#    $1          =   ''    ''    ''
```

```
#
# Be sure to stop a service first if it's running.
#
!define Service::Remove `!insertmacro _Service::Remove`
!macro _Service::Remove _SVC _FSR _ERR1 _ERR2
    StrCmpS $Bit 64 0 +4
    StrCmp "${_FSR}" /DISABLEFSR 0 +3
    ExecDos::Exec /TOSTACK /DISABLEFSR `"${SC}" delete "${_SVC}"`
    Goto +2
    ExecDos::Exec /TOSTACK `"${SC}" delete "${_SVC}"`
    Pop ${_ERR1}
    Pop ${_ERR2}
!macroend
```

Read Services online: https://riptutorial.com/nsis/topic/10184/services

Credits

S. No	Chapters	Contributors
1	Getting started with nsis	Anders, Community, knipp, Roland Bär, wintersolider
2	DotNET	demon.devin
3	Registering Libraries (RegDLL)	demon.devin
4	Services	demon.devin