# NSIS nsDialogs Plug-in

*Next generation of user interface design*

## Table of Contents

## Introduction

nsDialogs allows creation of custom pages in the installer. On top of the built-in pages, nsDialogs can create pages with any type of controls in any order and arrangement. It can create everything from a simple page with one label to form which reacts to user's actions. Modern UI 2, for example, uses nsDialogs to create the welcome and finish pages.

nsDialogs is a NSIS plug-in, introduced in version 2.29 as a replacement for InstallOptions. nsDialogs doesn't use INI files, so it's way faster than InstallOptions. Integration with the script is tighter and more natural - creating controls is done using plug-in functions and notification is done by directly calling a function in the script. Unlike InstallOptions, there isn't a predefined set of available control type and by providing a lower level access to Windows API, every type of control can be created and pages can be more customizable.

The same thing that makes nsDialogs more flexible can also make it more complicated for users with no knowledge of Win32 API. This is solved by creating a library of predefined functions, defined in script, that allow creation and handling of controls. This way, novices get easy access to the flexibility, while advanced users still get access to the core functionality by modifying the library or simply avoid using it.

## Step-by-Step Tutorial

### Custom Page

Before you can start adding controls you need to create a nsDialogs custom page. nsDialogs pages can only be created in a custom page's creator function, not in sections nor regular functions. Lets create a basic script with a empty nsDialogs page as our skeleton:

```
!include LogicLib.nsh

Name nsDialogs
OutFile nsDialogs.exe
XPStyle on


Var Dialog

Page custom nsDialogsPage
Page instfiles

Function nsDialogsPage
        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        nsDialogs::Show
FunctionEnd

Section
SectionEnd
```

The first call must always be to *nsDialogs::Create*. It will create a new dialog in the page and return its *HWND* on the stack. The result must be popped from the stack to prevent stack corruption. If the result is *error*, the dialog couldn't be created and the page must be aborted!

*nsDialogs::Create* accepts one parameter. It has a very specific function, but to keep things simple for this tutorial, it must always be 1018.

*HWND* is a number that uniquely identifies the dialog and can be used with *SendMessage*, *SetCtlColors* and Win32 API functions.

The dialog is not fully constructed at this point and you can make modifications to it before it is displayed. To finalize and display the dialog you must call *nsDialogs::Show*. This function will not return until the user clicks Next, Back or Cancel.

## Adding Controls

Compiling the last script and running it results in an empty page which is not very useful. So now we'll add some controls to it to. To do so, we'll use ${NSD_Create*} macros from nsDialogs.nsh. Each of those macros takes 5 parameters - x, y, width, height and text. Each macro also returns one value on the stack, which is the new control's *HWND*. Like the dialogs *HWND*, it must be popped from the stack and saved.

Each of the measurements that the macros take can use one of three unit types - pixels, dialog units or percentage of the dialog's size. It can also be negative to indicate it should be measured from the end. To use dialog units, the measurement must be suffixed with the letter *u*. To use percentage, the measurement must be suffixed with the percentage sign - *%*. Any other suffix, or no suffix, means pixels.

Dialog units allow creation of dialogs that scale well when different fonts or DPI is used. Its size in pixels is determined at runtime based on the font and the DPI. For example, standard pages in the classic NSIS user interface are 266 dialog units wide and 130 dialog units high. Pages in Modern UI are 300 dialogs units wide and 140 dialog units high. In different resolutions, using different fonts or DPI settings, the dialogs will always have the same size in dialog units, but different size in pixels.

```
!include nsDialogs.nsh
!include LogicLib.nsh

Name nsDialogs
OutFile nsDialogs.exe
XPStyle on

Var Dialog
Var Label
Var Text

Page custom nsDialogsPage
Page instfiles

Function nsDialogsPage

        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        ${NSD_CreateLabel} 0 0 100% 12u "Hello, welcome to nsDialogs!"
        Pop $Label

        ${NSD_CreateText} 0 13u 100% -13u "Type something here..."
        Pop $Text

        nsDialogs::Show

FunctionEnd

Section
SectionEnd
```

Available control types that can be created with ${NSD_Create*} are: HLine, VLine, Label, Icon, Bitmap, BrowseButton, Link, Button, GroupBox, CheckBox, RadioButton (FirstRadioButton and AdditionalRadioButton), Text, MLText, Password, Number, FileRequest, DirRequest, RichEdit, ComboBox, DropList, ListBox, SortedListBox, ProgressBar, Animation, HTrackBar, VTrackBar, UpDown, HotKey, DatePicker, IPAddress, NetAddress.

## Control State

Now that we have some controls that the user can interact with, it's time to see what the user actually does with them. For that, we'll first add a "leave callback function" to our page. In that function, we'll query the state of the text control we've created and display it to the user. To do so, we'll use the ${NSD_GetText} macro. Use the ${NSD_GetState} macro for RadioButton and CheckBox controls.

Note that not all controls support ${NSD_GetText} and some require special handling with specific messages defined in WinMessages.nsh. For example, the ListBox control requires usage of *LB_GETCURSEL* and *LB_GETTEXT*. With time, the library of macros in nsDialogs.nsh will fill with more and more macros that'll handle more cases like this.

```
!include nsDialogs.nsh
!include LogicLib.nsh

Name nsDialogs
OutFile nsDialogs.exe
XPStyle on

Var Dialog
Var Label
Var Text

Page custom nsDialogsPage nsDialogsPageLeave
Page instfiles

Function nsDialogsPage

        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        ${NSD_CreateLabel} 0 0 100% 12u "Hello, welcome to nsDialogs!"
        Pop $Label

        ${NSD_CreateText} 0 13u 100% -13u "Type something here..."
        Pop $Text

        nsDialogs::Show

FunctionEnd

Function nsDialogsPageLeave

        ${NSD_GetText} $Text $0
        MessageBox MB_OK "You typed:$\n$\n$0"

FunctionEnd

Section
SectionEnd
```

## Real-time Notification

One of the more exciting new features of nsDialogs is callback function notification of changes to the dialog. nsDialogs can call a function defined in a script in response to a user action such as changing of a text field or click of a button. To make nsDialogs notify us of events, we'll use ${NSD_OnClick} and ${NSD_OnChange}. Not every control supports both of the events. For example, there is nothing to notify about label changes, only clicks.

When the callback function is called, the control's *HWND* will be waiting on the stack and must be popped to prevent stack corruption. In this simple example, this is not so useful. But in case of a bigger script where several controls are associated with the same callback function, the *HWND* can shed some light on which control originated the event.

The new example will respond to the user typing "hello" in the text box.

```
!include nsDialogs.nsh
!include LogicLib.nsh

Name nsDialogs
OutFile nsDialogs.exe
XPStyle on

Var Dialog
Var Label
Var Text

Page custom nsDialogsPage nsDialogsPageLeave
Page instfiles

Function nsDialogsPage

        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        ${NSD_CreateLabel} 0 0 100% 12u "Hello, welcome to nsDialogs!"
        Pop $Label

        ${NSD_CreateText} 0 13u 100% -13u "Type something here..."
        Pop $Text
        ${NSD_OnChange} $Text nsDialogsPageTextChange

        nsDialogs::Show

FunctionEnd

Function nsDialogsPageLeave

        ${NSD_GetText} $Text $0
        MessageBox MB_OK "You typed:$\n$\n$0"

FunctionEnd

Function nsDialogsPageTextChange

        Pop $1 # $1 == $ Text

        ${NSD_GetText} $Text $0

        ${If} $0 == "hello"

                MessageBox MB_OK "right back at ya!"

        ${EndIf}

FunctionEnd

Section
SectionEnd
```

## Memory

So far we have a page that has some basic input controls. But what happens when the user goes to the next page and comes back? With the current code, the user's input will not be remembered. To remember, we'll use the already present leave callback function to store the user's choice in variables and pass these values when creating the controls the next time. For a better example, we'll also add a checkbox to the page and use ${NSD_GetState} and ${NSD_SetState} to get and set its state.

For clarity, we'll remove some of the notifications from the previous step.

```
!include nsDialogs.nsh
!include LogicLib.nsh

Name nsDialogs
OutFile nsDialogs.exe
XPStyle on

Var Dialog
Var Label
Var Text
Var Text_State
Var Checkbox
Var Checkbox_State

Page custom nsDialogsPage nsDialogsPageLeave
Page license
Page instfiles

Function .onInit

        StrCpy $Text_State "Type something here..."

FunctionEnd

Function nsDialogsPage

        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        ${NSD_CreateLabel} 0 0 100% 12u "Hello, welcome to nsDialogs!"
        Pop $Label

        ${NSD_CreateText} 0 13u 100% 12u $Text_State
        Pop $Text

        ${NSD_CreateCheckbox} 0 30u 100% 10u "&Something"
        Pop $Checkbox

        ${If} $Checkbox_State == ${BST_CHECKED}
                ${NSD_Check} $Checkbox
        ${EndIf}

        # alternative for the above ${If}:
        #${NSD_SetState} $Checkbox_State

        nsDialogs::Show

FunctionEnd

Function nsDialogsPageLeave

        ${NSD_GetText} $Text $Text_State
        ${NSD_GetState} $Checkbox $Checkbox_State

FunctionEnd

Section
SectionEnd
```

## Function Reference

## Create

```
nsDialogs::Create rect
```

Creates a new dialog. *rect* specific the identifier of the control whose location will be mimiced. This should usually be 1018, which is control mimiced for creation of built-in pages. The Modern UI also has control 1044 for the welcome and the finish page.

Returns the new dialog's HWND on the stack or *error*.

## CreateControl

```
nsDialogs::CreateControl class style extended_style x y width height text
```

Create a new control in the current dialog. A dialog must exist for this to work, so nsDialogs::Create must be called prior to this function.

Returns the new control's HWND on the stack or *error*.

## Show

```
nsDialogs::Show
```

Displays the page. Call this once finished with nsDialogs::Create, nsDialogs::CreateControl and the rest.

Returns nothing.

## SelectFileDialog

```
nsDialogs::SelectFileDialog mode initial_selection filter
```

Displays a file selection dialog to the user. If *mode* is set to *save*, displays a file save dialog. If *mode* is set to *open*, displays a file open dialog.

*initial_selection* can be used to provide the user with a default file to look for and/or a default folder to look in. If *initial_selection* is empty no default filename will be provided for the user and the dialog will start in the current working directory. If *initial_selection* specifies just a filename, for example "test.exe", the dialog will be set up to look for a file called test.exe in the current working directory. If *initial_selection* specifies just a directory, for example "C:\Program Files", the dialog starts in the provided directory with no file name provided. If *initial_selection* specifies a directory and a filename, for example "C:\Windows\System32\calc.exe", the dialog will be set up to look for a file called calc.exe in the directory C:\Windows\System32.

*filter* is a list of available file filter pairs separated by pipes. A filter pair consists of a display string and a DOS-style wildcard pattern. If an empty string is passed, the default is used (*"All Files|*.*"*).

Returns the selected file on the stack or an empty string if the user canceled the operation.

```
nsDialogs::SelectFileDialog open "$DOCUMENTS\Config.ini" ".ini files|*.ini|All File
Pop $0
StrCmp $0 "" ...
```

## SelectFolderDialog

```
nsDialogs::SelectFolderDialog title initial_selection
```

Displays a directory selection dialog to the user.

Returns the selected directory on the stack or "error" in case the user canceled the operation or an error occurred.

### SetRTL

```
nsDialogs::SetRTL rtl_setting
```

Sets right-to-left mode on or off. If *rtl_setting* is 0, it's set to off. If *rtl_setting* is 1, it's set to on. This function must be called before any calls to nsDialogs::CreateControl.

Returns nothing.

### GetUserData

```
nsDialogs::GetUserData control_HWND
```

Returns user data associated with the control on the stack. Use nsDialogs::SetUserData to set this data.

### SetUserData

```
nsDialogs::SetUserData control_HWND data
```

Associates *data* with the control. Use nsDialogs::GetUserData to get this data.

Returns nothing.

### OnBack

```
nsDialogs::OnBack function_address
```

Sets the callback function for the Back button. This function will be called when the user clicks the back button. Call Abort in this function to prevent the user from going back to the last page.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

### OnChange

```
nsDialogs::OnChange control_HWND function_address
```

Sets a change notification callback function for the given control. Whenever the control changes, the function will be called and the control's HWND will be waiting on its stack.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

### OnClick

```
nsDialogs::OnClick control_HWND function_address
```

Sets a click notification callback function for the given control. Whenever the control is clicked, the function will be called and the control's HWND will be waiting on its stack.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

### OnNotify

```
nsDialogs::OnNotify control_HWND function_address
```

Sets a notification callback function for the given control. Whenever the control receives the WM_NOTIFY message, the function will be called and the control's HWND, notification code and a pointer to the NMHDR structure will be waiting on its stack. Use ${NSD_Return} to return a value.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

### CreateTimer

```
nsDialogs::CreateTimer function_address timer_interval
```

Sets a timer that'd call the callback function for the given control every in a constant interval. Interval times are specified in milliseconds.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

### KillTimer

```
nsDialogs::KillTimer function_address
```

Kills a previously set timer.

Use GetFunctionAddress to get the address of the desired callback function.

Returns nothing.

## Macro Reference

nsDialogs.nsh contains a lot of macros that can make nsDialogs usage a lot easier. Below is a description of each of those macros including purpose, syntax, input and output.

### NSD_Create*

```
${NSD_Create*} x y width height text
```

Create a new control in the current dialog. A dialog must exist for this to work, so nsDialogs::Create must be called prior to this function.

Available variants:

- ${NSD_CreateHLine}
- ${NSD_CreateVLine}
- ${NSD_CreateLabel}
- ${NSD_CreateIcon}
- ${NSD_CreateBitmap}
- ${NSD_CreateBrowseButton}
- ${NSD_CreateLink}
- ${NSD_CreateButton}
- ${NSD_CreateGroupBox}
- ${NSD_CreateCheckBox}
- ${NSD_CreateRadioButton} (${NSD_CreateFirstRadioButton} and ${NSD_CreateAdditionalRadioButton})
- ${NSD_CreateText}

- ${NSD_CreateMLText}
- ${NSD_CreatePassword}
- ${NSD_CreateNumber}
- ${NSD_CreateFileRequest}
- ${NSD_CreateDirRequest}
- ${NSD_CreateRichEdit}
- ${NSD_CreateComboBox}
- ${NSD_CreateDropList}
- ${NSD_CreateListBox}
- ${NSD_CreateSortedListBox}
- ${NSD_CreateProgressBar}
- ${NSD_CreateAnimation}
- ${NSD_CreateHTrackBar}
- ${NSD_CreateVTrackBar}
- ${NSD_CreateUpDown}
- ${NSD_CreateHotKey}
- ${NSD_CreateDatePicker} (Requires IE 3.1 and *${NSD_InitCommonControlsEx}* )
- ${NSD_CreateIPAddress} (Requires IE 4 and *${NSD_InitCommonControl_IPAddress}* )
- ${NSD_CreateNetAddress} (Requires Vista and *${NSD_InitCommonControl_NetAddress}* )

Returns the new control's HWND on the stack or *error*

## NSD_OnBack

```
${NSD_OnBack} function_name
```

See OnBack for more details.

## NSD_OnChange

```
${NSD_OnChange} control_HWND function_name
```

See OnChange for more details.

See Real-time Notification for usage example.

## NSD_OnClick

```
${NSD_OnClick} control_HWND function_name
```

See OnClick for more details.

## NSD_OnNotify

```
${NSD_OnNotify} control_HWND function_name
```

See OnNotify for more details.

## NSD_Return

```
${NSD_Return} value
```

Used to return a value back to Windows from a OnNotify callback.

## NSD_SetFocus

```
${NSD_SetFocus} control_HWND
```

Sets focus to a control.

## NSD_CreateTimer

```
${NSD_CreateTimer} function_name timer_interval
```

See CreateTimer for more details.

## NSD_KillTimer

```
${NSD_KillTimer} function_name
```

See KillTimer for more details.

## NSD_AddStyle

```
${NSD_AddStyle} control_HWND style
```

Adds one or more window styles to a control. Multiple styles should be separated with pipes `|'.

See MSDN for shared and per-control style descriptions.

## NSD_AddExStyle

```
${NSD_AddExStyle} control_HWND style
```

Adds one or more extended window styles to a control. Multiple styles should be separated with pipes `|'.

See MSDN for style descriptions.

## NSD_RemoveStyle

```
${NSD_RemoveStyle} control_HWND style
```

Removes one or more window styles from a control.

## NSD_RemoveExStyle

```
${NSD_RemoveExStyle} control_HWND style
```

Removes one or more extended window styles from a control.

## NSD_GetText

```
${NSD_GetText} control_HWND $output_variable
```

Retrieves the text of a control and stores it into *output_variable*. Especially useful for textual controls.

See Control State for usage example.

## NSD_SetText

```
${NSD_SetText} control_HWND text
```

Sets the text of a control.

## NSD_Edit_SetTextLimit

```
${NSD_Edit_SetTextLimit} control_HWND limit
```

Sets the input length limit for a text control.

### NSD_Edit_SetReadOnly

```
${NSD_Edit_SetReadOnly} control_HWND readonly
```

1 to make the text control read-only or 0 to allow the user to input data.

### NSD_Edit_SetCueBannerText

```
${NSD_Edit_SetCueBannerText} control_HWND displaywhenfocused text
```

Set a text hint displayed when the control is empty. The text is only visible on WinXP and later. Requires *XPStyle on*.

### NSD_RichEd_SetTextLimit

```
${NSD_RichEd_SetTextLimit} control_HWND limit
```

Sets the input length limit.

### NSD_RichEd_SetEventMask

```
${NSD_RichEd_SetEventMask} control_HWND eventmask
```

### NSD_RichEd_SetCustomBackgroundColor

```
${NSD_RichEd_SetCustomBackgroundColor} control_HWND COLORREF
```

### NSD_GetState

```
${NSD_GetState} control_HWND $output_variable
```

Retrieves the state of a check box or a radio button control. Possible outputs are ${BST_CHECKED} and ${BST_UNCHECKED}.

See Memory for usage example.

### NSD_SetState

```
${NSD_SetState} control_HWND state
```

Sets the state of a check box or a radio button control. Possible values for *state* are ${BST_CHECKED} and ${BST_UNCHECKED}.

See Memory for usage example.

### NSD_Check

```
${NSD_Check} control_HWND
```

Checks a check box or a radio button control. Same as calling ${NSD_SetState} with ${BST_CHECKED}.

### NSD_Uncheck

```
${NSD_Uncheck} control_HWND
```

Unchecks a check box or a radio button control. Same as calling ${NSD_SetState} with ${BST_UNCHECKED}.

See Memory for usage example.

## NSD_CB_AddString

```
${NSD_CB_AddString} combo_HWND string
```

Adds a string to a combo box.

## NSD_CB_InsertString

```
${NSD_CB_InsertString} combo_HWND index string
```

```
${NSD_CB_PrependString} combo_HWND string
```

```
${NSD_CB_AppendString} combo_HWND string
```

Insert a string in a specified position in a combo box.

## NSD_CB_SelectString

```
${NSD_CB_SelectString} combo_HWND string
```

Selects a string in a combo box.

## NSD_CB_GetCount

```
${NSD_CB_GetCount} combo_HWND $output_variable
```

## NSD_LB_AddString

```
${NSD_LB_AddString} listbox_HWND string
```

Adds a string to a list box.

## NSD_LB_InsertString

```
${NSD_LB_InsertString} listbox_HWND index string
```

```
${NSD_LB_PrependString} listbox_HWND string
```

```
${NSD_LB_AppendString} listbox_HWND string
```

Insert a string in a specified position in a list box.

## NSD_LB_DelString

```
${NSD_LB_DelString} listbox_HWND string
```

Deletes a string from a list box.

### NSD_LB_DelItem

```
${NSD_LB_DelItem} listbox_HWND itemindex
```

Deletes a string from a list box.

### NSD_LB_Clear

```
${NSD_LB_Clear} listbox_HWND
```

Deletes all strings from a list box.

### NSD_LB_GetCount

```
${NSD_LB_GetCount} listbox_HWND $output_variable
```

Retrieves the number of strings from a list box.

### NSD_LB_SelectString

```
${NSD_LB_SelectString} listbox_HWND string
```

Selects a string in a list box.

### NSD_LB_GetSelection

```
${NSD_LB_GetSelection} listbox_HWND $output_variable
```

Retrieves the selected string from a list box. Returns an empty string if no string is selected.

### NSD_Anim_OpenFile

```
${NSD_Anim_OpenFile} anim_HWND avi_path
```

Opens the specified (silent) .AVI movie clip.

### NSD_Anim_Play

```
${NSD_Anim_Play} anim_HWND
```

Plays the movie clip repeatedly.

### NSD_Anim_Stop

```
${NSD_Anim_Stop} anim_HWND
```

Stops playback.

### NSD_TrackBar_GetPos

```
${NSD_TrackBar_GetPos} track_HWND $output
```

### NSD_TrackBar_SetPos

```
${NSD_TrackBar_SetPos} track_HWND pos
```

## NSD_TrackBar_SetRangeMin

```
${NSD_TrackBar_SetRangeMin} track_HWND minpos
```

## NSD_TrackBar_SetRangeMax

```
${NSD_TrackBar_SetRangeMax} track_HWND maxpos
```

## NSD_TrackBar_SetTicFreq

```
${NSD_TrackBar_SetTicFreq} track_HWND frequency
```

Sets the interval frequency for tick marks.

## NSD_UD_SetBuddy

```
${NSD_UD_SetBuddy} ud_HWND buddy_HWND
```

## NSD_UD_GetPos

```
${NSD_UD_GetPos} ud_HWND $output
```

## NSD_UD_SetPos

```
${NSD_UD_SetPos} ud_HWND pos
```

## NSD_UD_SetPackedRange

```
${NSD_UD_SetPackedRange} ud_HWND packedrange
```

Sets the min-max range. Two signed 16-bit numbers packed into 32-bits.

## NSD_HK_GetHotKey

```
${NSD_HK_GetHotKey} hk_HWND $output
```

Bits 0..7 specify the virtual key code and bits 8..15 specify the HOTKEYF modifier flags.

## NSD_HK_SetHotKey

```
${NSD_HK_SetHotKey} hk_HWND packedhotkey
```

## NSD_Date_GetDateFields

```
${NSD_Date_GetDateFields} HWND
```

Returns the month, day and year on the stack.

## NSD_SetBitmap

```
${NSD_SetBitmap} control_HWND image_path $output_variable
```

Loads a bitmap from *image_path* and displays it on *control_HWND* created by ${NSD_CreateBitmap}. The image handle is stored in *output_variable* and should be freed using ${NSD_FreeBitmap} once no longer necessary.

The image must be extracted to the user's computer prior to calling this macro. A good place to extract images is $PLUGINSDIR.

```
!include nsDialogs.nsh

Name nsDialogs
OutFile nsDialogs.exe

Page custom nsDialogsImage
Page instfiles

Var Dialog
Var ImageCtrl
Var BmpHandle

Function .onInit
        InitPluginsDir
        File /oname=$PLUGINSDIR\image.bmp "${NSISDIR}\Contrib\Graphics\Header\nsis-
FunctionEnd

Function nsDialogsImage
        nsDialogs::Create 1018
        Pop $Dialog

        ${If} $Dialog == error
                Abort
        ${EndIf}

        ${NSD_CreateBitmap} 0 0 100% 100% ""
        Pop $ImageCtrl
        ${NSD_SetBitmap} $ImageCtrl $PLUGINSDIR\image.bmp $BmpHandle

        nsDialogs::Show

        ${NSD_FreeBitmap} $BmpHandle
FunctionEnd

Section
SectionEnd
```

## NSD_SetStretchedBitmap

`${NSD_SetStretchedBitmap}` *`control_HWND image_path $output_variable`*

Loads and displays a bitmap just like ${NSD_SetImage} but stretches the image to fit the control.

## NSD_ClearBitmap

`${NSD_ClearBitmap}` *`control_HWND`*

Clears a bitmap from a image control.

## NSD_FreeBitmap

`${NSD_FreeImage}` *`image_handle`*

Frees an image handle previously loaded with ${NSD_SetImage} or ${NSD_SetStretchedBitmap}.

## NSD_SetIcon

`${NSD_SetIcon}` *`control_HWND image_path $output_variable`*

Same as ${NSD_SetImage}, but used for loading and setting an icon in a control created by ${NSD_CreateIcon}. The image handle is stored in *output_variable* and should be freed using ${NSD_FreeIcon} once no longer necessary.

### NSD_SetIconFromInstaller

```
${NSD_SetIconFromInstaller} control_HWND $output_variable
```

Loads the icon used in the installer and displays it on *control_HWND* created by ${NSD_CreateIcon}. The image handle is stored in *output_variable* and should be freed using ${NSD_FreeIcon} once no longer necessary.

### NSD_ClearIcon

```
${NSD_ClearIcon} control_HWND
```

Clears an icon from a control.

### NSD_FreeIcon

```
${NSD_FreeIcon} icon_handle
```

Frees an icon handle previously loaded with ${NSD_SetIcon} or ${NSD_SetIconFromInstaller}.

## FAQ

- **Q:** Can nsDialogs handle InstallOptions INI files?

  **A:** nsDialogs.nsh contains a function called *CreateDialogFromINI* that can create nsDialogs' dialog from an INI file. It can handle every type of control InstallOptions supports, but doesn't handle the flags or notifications. *Examples\nsDialogs\InstallOptions.nsi* shows a usage example of this function.