Toolbar Class v1.0.0 - Documentation

Overview

Production-ready toolbar class for AutoHotkey v2 that provides an easy-to-use interface for creating professional toolbars with icons, tooltips, and comprehensive error handling.

Features

- **Button Management**: Add, enable, disable, show, and hide buttons
- **Icons**: Support for system icon libraries and custom icon files
- **Enhanced Tooltips**: Custom tooltip text with keyboard shortcuts and descriptions
- **Separators**: Visual grouping of related buttons
- **Validation**: Comprehensive error checking for GUI objects, callbacks, and icon files
- **Debug Logging**: Optional logging to file for troubleshooting
- **Resizable**: Automatic sizing and manual resize support

Installation

1. Save `Toolbar Class.ahk` to your AutoHotkey Lib folder:
   - `%A_MyDocuments%\AutoHotkey\Lib\`
2. Include in your script:
3. `#Include <Toolbar Class>`

Quick Start

```
#Requires AutoHotkey v2.0.2+
#Include <Toolbar Class>

; Create GUI
objMyGui := Gui("+Resize", "My Application")

; Create toolbar
objMyToolbar := Toolbar(objMyGui, "x0 y0 w800 h35")

; Add buttons
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1,
"Create new document (Ctrl+N)")
objMyToolbar.Add("Open", handleOpen, "C:\Windows\System32\shell32.dll", 4,
"Open file (Ctrl+O)")
objMyToolbar.AddSeparator()
objMyToolbar.Add("Exit", handleExit, "C:\Windows\System32\shell32.dll", 238,
"Exit application")
```

```
; Finalize and show
objMyToolbar.AutoSize()
objMyGui.Show("w800 h600")

; Callback functions
handleNew(*) {
    MsgBox("New clicked")
}

handleOpen(*) {
    MsgBox("Open clicked")
}

handleExit(*) {
    ExitApp()
}
```

Constructor

`Toolbar(objGui, strOptions := "x0 y0 w800 h35")`

Creates a new toolbar attached to a GUI window.

Parameters:

- `objGui` (Gui object, required) - Parent GUI window object
- `strOptions` (string, optional) - Positioning options (default: "x0 y0 w800 h35")

Returns: Toolbar object

Validation:

- Throws `TypeError` if objGui is not a valid Gui object
- Throws `TypeError` if objGui lacks hwnd property
- Throws `ValueError` if GUI hwnd is invalid or zero

Example:

```
objMyGui := Gui()
objMyToolbar := Toolbar(objMyGui)  ; Uses default position
objMyToolbar2 := Toolbar(objMyGui, "x0 y0 w1200 h40")  ; Custom size
```

Methods

`Add(strText, funcCallback, strIconFile := "", intIconIndex := 1, strTooltip := "")`

Adds a button to the toolbar.

Parameters:

- `strText` (string, required) - Button label text (can be empty for icon-only)
- `funcCallback` (function, required) - Function to call when button is clicked
- `strIconFile` (string, optional) - Path to icon file (e.g., DLL, ICO, EXE)
- `intIconIndex` (integer, optional) - Icon index within file (default: 1)
- `strTooltip` (string, optional) - Custom tooltip text (defaults to strText)

Returns: Command ID (integer) - Use to enable/disable/show/hide the button

Validation:

- Throws `TypeError` if funcCallback is not a function object
- Shows warning if both strText and strIconFile are empty
- Shows warning if strIconFile doesn't exist (creates text-only button)

Examples:

```
; Standard button with icon and text
intID := objMyToolbar.Add("Save", handleSave,
"C:\Windows\System32\shell32.dll", 259)

; Button with enhanced tooltip
objMyToolbar.Add("Copy", handleCopy, "C:\Windows\System32\shell32.dll", 261,
"Copy selected text (Ctrl+C)")

; Text-only button (no icon)
objMyToolbar.Add("Help", handleHelp, "", 0, "Show help information")

; Icon-only button (no text)
objMyToolbar.Add("", handleInfo, "C:\Windows\System32\shell32.dll", 222,
"Application information")
```

`AddSeparator()`

Adds a visual separator between buttons for grouping.

Parameters: None

Returns: None

Example:

```
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
objMyToolbar.Add("Open", handleOpen, "C:\Windows\System32\shell32.dll", 4)
objMyToolbar.AddSeparator()  ; Visual divider
objMyToolbar.Add("Cut", handleCut, "C:\Windows\System32\shell32.dll", 260)
```

`Enable(intCmdID)`

Enables a previously disabled button.

Parameters:

- `intCmdID` (integer, required) - Command ID returned from Add()

Returns: None

Example:

```
intSaveID := objMyToolbar.Add("Save", handleSave,
"C:\Windows\System32\shell32.dll", 259)
objMyToolbar.Disable(intSaveID)  ; Initially disabled
; ... later ...
objMyToolbar.Enable(intSaveID)   ; Now enabled
```

`Disable(intCmdID)`

Disables a button (grayed out, not clickable).

Parameters:

- `intCmdID` (integer, required) - Command ID returned from Add()

Returns: None

Example:

```
intSaveID := objMyToolbar.Add("Save", handleSave,
"C:\Windows\System32\shell32.dll", 259)
objMyToolbar.Disable(intSaveID)  ; Disable the Save button
```

`Show(intCmdID)`

Shows a previously hidden button.

Parameters:

- `intCmdID` (integer, required) - Command ID returned from Add()

Returns: None

Example:

```
objMyToolbar.Show(intButtonID)
```

`Hide(intCmdID)`

Hides a button from the toolbar.

Parameters:

- `intCmdID` (integer, required) - Command ID returned from Add()

Returns: None

Example:

```
objMyToolbar.Hide(intButtonID)
```

`AutoSize()`

Automatically sizes the toolbar to fit all buttons.

Parameters: None

Returns: None

Example:

```
; Add all buttons first
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
objMyToolbar.Add("Open", handleOpen, "C:\Windows\System32\shell32.dll", 4)
; Then auto-size
objMyToolbar.AutoSize()
```

`Resize(intWidth, intHeight)`

Manually resizes the toolbar.

Parameters:

- `intWidth` (integer, required) - New width in pixels
- `intHeight` (integer, required) - New height in pixels

Returns: None

Example:

```
resizeWindow(objGui, intMinMax, intWidth, intHeight) {
    if (intMinMax = -1)
        return
    objMyToolbar.Resize(intWidth, 35)
}

objMyGui.OnEvent("Size", resizeWindow)
```

`GetButtonCount()`

Returns the number of buttons in the toolbar.

Parameters: None

Returns: Integer - Number of buttons

Example:

```
intCount := objMyToolbar.GetButtonCount()
MsgBox("Toolbar has " intCount " buttons")
```

`SetDebug(boolEnable)`

Enables or disables debug logging to file.

Parameters:

- `boolEnable` (boolean, required) - True to enable, False to disable

Returns: None

Notes:

- Creates `ToolbarDebug.log` in script directory
- Logs all notifications, warnings, and errors
- Useful for troubleshooting tooltip and callback issues

Example:

```
objMyToolbar.SetDebug(true)   ; Enable logging
; ... test your toolbar ...
objMyToolbar.SetDebug(false)  ; Disable logging
```

Common Icon Sources

Windows System Icons (shell32.dll)

Located at: `C:\Windows\System32\shell32.dll`

Common icon indices:

- 1: New document
- 4: Open folder
- 16: Folder
- 21: Help
- 22: Search
- 238: Power/Exit
- 259: Save
- 260: Cut
- 261: Copy
- 262: Paste

Example:

```
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
```

Custom Icons

You can use:

- `.ico` files
- `.exe` files (program icons)
- `.dll` files (icon libraries)

Example:

```
objMyToolbar.Add("Custom", handleCustom, "C:\MyApp\Icons\custom.ico", 1)
objMyToolbar.Add("App", handleApp, "C:\Program Files\MyApp\app.exe", 1)
```

Error Handling

The Toolbar class includes comprehensive validation:

Constructor Errors

```
; Invalid GUI parameter
try {
    objBadToolbar := Toolbar("notAGui")  ; Throws TypeError
} catch as objError {
    MsgBox("Error: " objError.Message)
}

; Destroyed GUI
try {
    objGui := Gui()
    objGui.Destroy()
    objBadToolbar := Toolbar(objGui)  ; Throws ValueError
} catch as objError {
    MsgBox("Error: " objError.Message)
}
```

Button Addition Errors

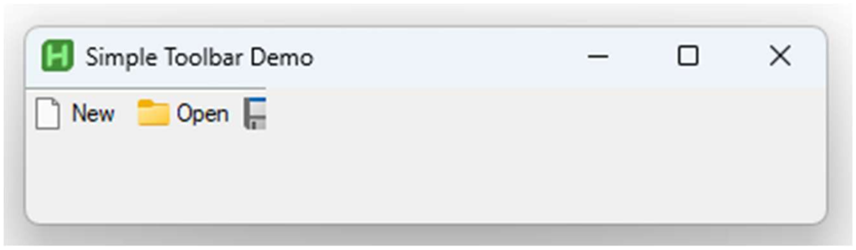
```
; Invalid callback
try {
    objMyToolbar.Add("Bad", "notAFunction",
"C:\Windows\System32\shell32.dll", 1)  ; Throws TypeError
} catch as objError {
    MsgBox("Error: " objError.Message)
}

; Missing icon file (warning, not error)
objMyToolbar.Add("Missing", handleTest, "C:\DoesNotExist.ico", 1)  ; Shows
warning, creates text-only button

; Empty button (warning, not error)
objMyToolbar.Add("", handleTest, "", 1)  ; Shows warning once
```

Complete Examples

Example 1: Simple Toolbar



```
#Requires AutoHotkey v2.0.2+
#SingleInstance
#Include <Toolbar Class>
```

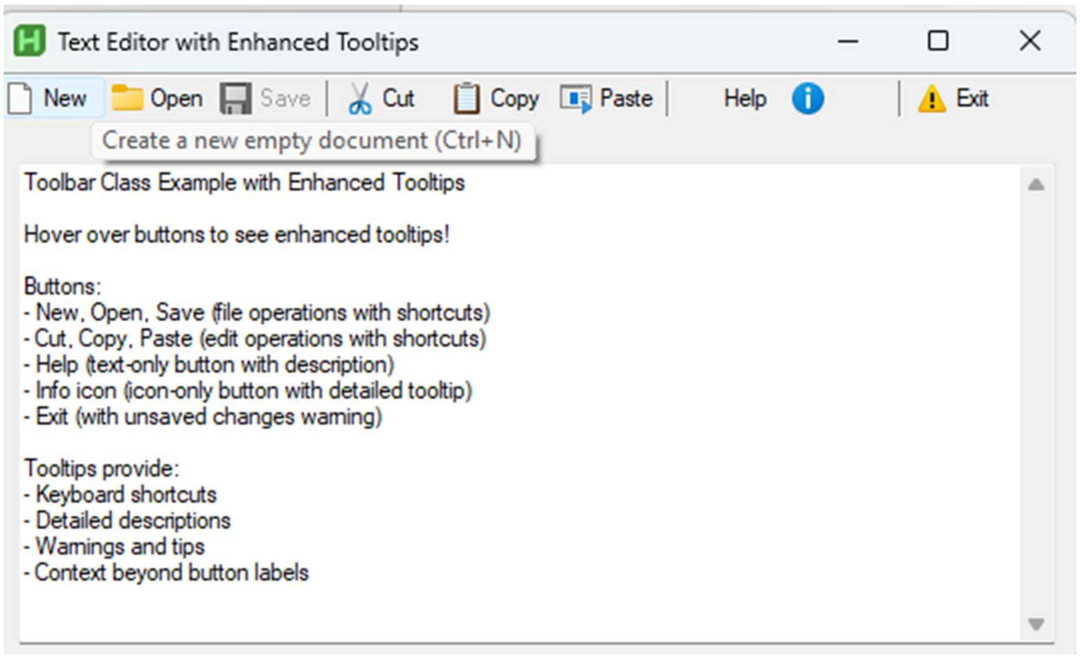
```
global objMyGui := Gui("+Resize", "Simple Toolbar Demo")
global objMyToolbar := Toolbar(objMyGui)

objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
objMyToolbar.Add("Open", handleOpen, "C:\Windows\System32\shell32.dll", 4)
objMyToolbar.Add("Save", handleSave, "C:\Windows\System32\shell32.dll", 259)
objMyToolbar.AddSeparator()
objMyToolbar.Add("Exit", handleExit, "C:\Windows\System32\shell32.dll", 238)

objMyToolbar.AutoSize()
objMyGui.Show("w400 h300")

handleNew(*) => MsgBox("New")
handleOpen(*) => MsgBox("Open")
handleSave(*) => MsgBox("Save")
handleExit(*) => ExitApp()
```

Example 2: Text Editor with Full Functionality


Text Editor with Enhanced Tooltips
New
Open
Save
Cut
Copy
Paste
Help
Exit
Create a new empty document (Ctrl+N)

Toolbar Class Example with Enhanced Tooltips

Hover over buttons to see enhanced tooltips!

Buttons:
- New, Open, Save (file operations with shortcuts)
- Cut, Copy, Paste (edit operations with shortcuts)
- Help (text-only button with description)
- Info icon (icon-only button with detailed tooltip)
- Exit (with unsaved changes warning)

Tooltips provide:
- Keyboard shortcuts
- Detailed descriptions
- Warnings and tips
- Context beyond button labels

Example 3: Error Testing

- Constructor validation
- Callback validation
- Icon file validation
- Button configuration validation

```
#Requires AutoHotkey v2.0.2+

#SingleInstance


;================= Toolbar Error Testing Script v1.2.0 =================

; Description: Tests error handling and validation in Toolbar class

;


;=================


#Include <Toolbar Class>


; ===== TEST GROUP A: Constructor Validation (before creating GUI) =====

MsgBox("TEST GROUP A: Constructor (GUI) Validation`n`nTesting various invalid GUI
parameters.", "Test Group", "Icon! 64")


; ===== TEST A1: String instead of GUI object =====

intResult := MsgBox("TEST A1: Creating toolbar with STRING instead of GUI object`n`nThis
should THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objBadToolbar := Toolbar("myGui")

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {

        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")
```

```
    }

}



; ===== TEST A2: Number instead of GUI object =====

intResult := MsgBox("TEST A2: Creating toolbar with NUMBER instead of GUI object`n`nThis
should THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objBadToolbar := Toolbar(12345)

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {

        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

    }

}



; ===== TEST A3: Wrong object type (no hwnd property) =====

intResult := MsgBox("TEST A3: Creating toolbar with WRONG OBJECT TYPE`n`nThis
should THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objWrongType := {test: "value", data: 123}

        objBadToolbar := Toolbar(objWrongType)

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {
        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

    }

}



; ===== TEST A4: Destroyed GUI (hwnd = 0) =====

intResult := MsgBox("TEST A4: Creating toolbar with DESTROYED GUI`n`nThis should
THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objDeadGui := Gui()

        objDeadGui.Destroy()

        objBadToolbar := Toolbar(objDeadGui)

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {

        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

    }

}



; ===== Now create valid GUI for remaining tests =====

MsgBox("Constructor validation complete!`n`nNow creating VALID GUI for remaining tests.",
"Test Info", "Icon!")



; Global variables
```

```
global objMyGui := ""

global objMyToolbar := ""


; Valid callback functions

handleTest1(*) {

    MsgBox("Test 1 clicked - Valid callback")

}


handleTest2(*) {

    MsgBox("Test 2 clicked - Valid callback")

}


handleTest3(*) {

    MsgBox("Test 3 clicked - Valid callback")

}


handleExit(*) {

    ExitApp()

}


; Create GUI and toolbar

objMyGui := Gui("+Resize", "Toolbar Error Testing")

objMyToolbar := Toolbar(objMyGui, "x0 y0 w600 h35")
```

```
; Enable debug mode to see all warnings/errors logged

objMyToolbar.SetDebug(true)



MsgBox("TEST GROUP B: Button Validation`n`nTesting various button configurations.", "Test Group", "Icon! 64")



; ===== TEST 1: Valid buttons (should work fine) =====

MsgBox("TEST 1: Adding valid buttons`n`nThese should all work correctly.", "Test Info", "Icon! 64")



objMyToolbar.Add("Valid 1", handleTest1, "shell32.dll", 1, "This is a valid button with text and icon")

objMyToolbar.Add("Valid 2", handleTest2, "shell32.dll", 4, "Another valid button")

objMyToolbar.AddSeparator()



; ===== TEST 2: Button with no text or icon (should warn) =====

intResult := MsgBox("TEST 2: Adding button with no text AND no icon`n`nThis should show a validation warning.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    objMyToolbar.Add("", handleTest3, "", 1)

}



; ===== TEST 3: Text-only button (should work, no warning) =====

intResult := MsgBox("TEST 3: Adding text-only button (no icon)`n`nThis is valid - no warning expected.", "Test Info", "OKCancel Icon! 64")
```

```
if (intResult = "OK") {

    objMyToolbar.Add("Text Only", handleTest3)

}



; ===== TEST 4: Icon-only button (should work, no warning) =====

intResult := MsgBox("TEST 4: Adding icon-only button (no text)`n`nThis is valid - no warning
expected.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    objMyToolbar.Add("", handleTest3, "shell32.dll", 222)

}



objMyToolbar.AddSeparator()



; ===== TEST 5: Invalid callback - String instead of function =====

intResult := MsgBox("TEST 5: Adding button with STRING callback`n`nThis should THROW
AN ERROR and stop the script.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objMyToolbar.Add("Bad Callback", "handleTest1", "shell32.dll", 5)

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {

        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

    }

}
```

```
; ===== TEST 6: Invalid callback - Undefined function =====

intResult := MsgBox("TEST 6: Adding button with UNDEFINED function`n`nThis should
THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

   try {

      objMyToolbar.Add("Undefined", handleUndefinedFunction, "shell32.dll", 6)

      MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

   } catch as objError {

      MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

   }

}



; ===== TEST 7: Invalid callback - Wrong object type =====

intResult := MsgBox("TEST 7: Adding button with OBJECT callback (not Func)`n`nThis should
THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

   try {

      objBadCallback := {test: "value"}

      objMyToolbar.Add("Wrong Object", objBadCallback, "shell32.dll", 7)

      MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

   } catch as objError {

      MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")
```

```
    }

}



; ===== TEST 8: Invalid callback - Number =====

intResult := MsgBox("TEST 8: Adding button with NUMBER callback`n`nThis should
THROW AN ERROR.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    try {

        objMyToolbar.Add("Number", 12345, "shell32.dll", 8)

        MsgBox("ERROR: Should have thrown an exception!", "Test Failed", "IconX")

    } catch as objError {

        MsgBox("SUCCESS: Caught error as expected!`n`nError Type: " Type(objError)
"`nMessage: " objError.Message, "Test Passed", "Icon!")

    }

}



; ===== TEST 9: Invalid icon file - File doesn't exist =====

intResult := MsgBox("TEST 9: Adding button with NON-EXISTENT icon file`n`nThis should
show a warning and create button without icon.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    objMyToolbar.Add("Bad Icon", handleTest3, "C:\DoesNotExist.ico", 1, "Icon file doesn't
exist")

}



; ===== TEST 10: Invalid icon file - Typo in filename =====
intResult := MsgBox("TEST 10: Adding button with TYPO in icon filename`n`nThis should show a warning (if first invalid icon) and create button without icon.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    objMyToolbar.Add("Typo Icon", handleTest3, "shel32.dll", 1, "Typo in shell32.dll")

}



; ===== TEST 11: Valid icon file after invalid ones =====

intResult := MsgBox("TEST 11: Adding button with VALID icon file`n`nThis should work normally after previous icon errors.", "Test Info", "OKCancel Icon! 64")

if (intResult = "OK") {

    objMyToolbar.Add("Valid Again", handleTest3, "shell32.dll", 3, "Valid icon after errors")

}



; Add exit button

objMyToolbar.AddSeparator()

objMyToolbar.Add("Exit", handleExit, "shell32.dll", 238, "Close this test window")



; Finalize toolbar

objMyToolbar.AutoSize()



; Add info text

objMyGui.AddText("x10 y45 w580 h300",

    "Toolbar Error Testing Complete!`n`n"

    "Check the ToolbarDebug.log file in the script directory for detailed logging.`n`n"
    "Tests performed:`n`n"

    "GROUP A - Constructor Validation:`n"

    "✓ String instead of GUI (error caught)`n"

    "✓ Number instead of GUI (error caught)`n"

    "✓ Wrong object type (error caught)`n"

    "✓ Destroyed GUI (error caught)`n`n"

    "GROUP B - Button Validation:`n"

    "✓ Valid buttons with text and icon`n"

    "✓ Button with no text or icon (warning)`n"

    "✓ Text-only button (valid)`n"

    "✓ Icon-only button (valid)`n"

    "✓ Invalid callbacks (errors caught)`n"

    "✓ Non-existent icon file (warning)`n"

    "✓ Typo in icon filename (warning)`n"

    "✓ Valid icon after errors (works)`n`n"

    "Click the toolbar buttons to test functionality.")


; Show GUI

objMyGui.Show("w600 h400")



MsgBox("All tests complete!`n`nThe toolbar is now ready to use.`nCheck ToolbarDebug.log for
details.", "Testing Complete", "Icon!")
```

;================= End of Toolbar Error Testing Script =================

Best Practices

1. Store Command IDs for Dynamic Control

```
intSaveID := objMyToolbar.Add("Save", handleSave,
"C:\Windows\System32\shell32.dll", 259)
objMyToolbar.Disable(intSaveID)  ; Initially disabled

; Enable when text changes
objMyEdit.OnEvent("Change", (*) => objMyToolbar.Enable(intSaveID))
```

2. Use Separators for Visual Grouping

```
; File operations
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
objMyToolbar.Add("Open", handleOpen, "C:\Windows\System32\shell32.dll", 4)
objMyToolbar.AddSeparator()

; Edit operations
objMyToolbar.Add("Cut", handleCut, "C:\Windows\System32\shell32.dll", 260)
objMyToolbar.Add("Copy", handleCopy, "C:\Windows\System32\shell32.dll", 261)
```

3. Provide Enhanced Tooltips

```
; Include keyboard shortcuts
objMyToolbar.Add("Save", handleSave, "C:\Windows\System32\shell32.dll", 259,
"Save document (Ctrl+S)")

; Include helpful context
objMyToolbar.Add("Exit", handleExit, "C:\Windows\System32\shell32.dll", 238,
"Close application (unsaved changes will be lost)")
```

4. Handle Window Resizing

```
resizeWindow(objGui, intMinMax, intWidth, intHeight) {
    if (intMinMax = -1)
        return
    objMyToolbar.Resize(intWidth, 35)
    ; Adjust other controls...
}

objMyGui.OnEvent("Size", resizeWindow)
```

5. Use Full Paths for Icon Files

```
; Good - Full path
objMyToolbar.Add("New", handleNew, "C:\Windows\System32\shell32.dll", 1)
```

```
; Bad - Relative path (may cause warnings)
objMyToolbar.Add("New", handleNew, "shell32.dll", 1)  ; Will show warning
```

6. Enable Debug Mode During Development

```
objMyToolbar.SetDebug(true)  ; Enable during development
; Test your toolbar...
; Check ToolbarDebug.log for issues
objMyToolbar.SetDebug(false)  ; Disable for production
```

Troubleshooting

Tooltips Not Showing

1. Enable debug mode: `objMyToolbar.SetDebug(true)`
2. Check `ToolbarDebug.log` for TBN_GETINFOTIP notifications
3. Verify tooltip text is being set correctly
4. Ensure you're hovering long enough (~1 second)

Icons Not Appearing

1. Verify icon file path is correct and absolute
2. Check icon index is valid for that file
3. Enable debug mode to see icon file warnings
4. Use a known-good icon source like shell32.dll

Buttons Not Responding

1. Verify callback function is defined
2. Enable debug mode to see NM_CLICK notifications
3. Check that callback function signature matches: `handleFunction(*)`

Constructor Errors

1. Ensure GUI object is created before toolbar
2. Don't create toolbar on destroyed GUI
3. Pass actual Gui object, not string or number

Version History

v1.0.0 - Production Release

- Complete button management system
- Enhanced tooltip support
- Comprehensive error handling and validation
- Debug logging capability
- Full documentation and examples

Requirements

- AutoHotkey v2.0.2 or later
- Windows OS (uses Windows Common Controls)

License

This class is provided as-is for use in AutoHotkey v2 projects.

Support

For issues or questions:

1. Enable debug mode: `objMyToolbar.SetDebug(true)`
2. Check `ToolbarDebug.log` for detailed information
3. Review the error testing example for validation patterns
4. Consult the AutoHotkey v2 documentation at https://www.autohotkey.com/docs/v2/