DeskUp

0.3

Generated by Doxygen 1.15.0

Chapter 1

Main Page

Chapter 2

DeskUp Internal Architecture

2.1 DeskUp Internal Architecture and Flow

DeskUp is organized into three main layers:

- **Frontend**: high-level orchestration of window and workspace operations.
- **Core**: backend initialization and management logic.
- **Backend**: platform-specific implementations for Windows

This page mirrors the GitHub README but adds Doxygen cross-references for symbols.

2.1.1 1. Initialization Choosing and bootstrapping a backend

When DeskUp starts, the application calls **DU_Init** (declared in `source/desk_up_backend_interface_backend/window_core.h` and implemented in `source/desk_up_backend_interface_backend/window_core.cc`).
DU_Init performs three key actions:

1. Iterates through a list of available **backend bootstraps** (currently only Windows).
    - The Windows backend defines its bootstrap (`winWindowDevice`) in `source/desk_up_backend_interface_backend/window_backends/desk_up_win/desk_up_win.h`.
2. For each backend, it checks availability via `DeskUpWindowBootStrap::isAvailable()` **WIN_isAvailable**.
3. Once it finds a working backend, it:
    - Creates the device with `DeskUpWindowBootStrap::createDevice()` **WIN_CreateDevice**.
    - Retrieves the base folder with `DeskUpWindowDevice::getDeskUpPath()` **WIN_getDeskUpPath**.
    - Sets the global variables:
        - **DESKUPDIR** base workspace directory.
        - **current_window_backend** active backend device.

If initialization succeeds, DU_Init logs the connected backend and returns `1`. If none is available, it returns `0`.

2.1.2 2. High-level operations DeskUpWindow faade

The structure DeskUpWindow is defined in source/desk_up_backend_interface/desk_up_backend_interface.h and implemented in source/desk_up_backend_interface/desk_up_backend_interface.cc.
It acts as the **frontend faade**, coordinating workspace-level operations with the backend.

2.1.2.1 `DeskUpWindow::saveAllWindowsLocal(std::string workspaceName)`

1. Builds <DESKUPDIR>/<workspaceName> using the global path set by DU_Init.
2. Ensures the directory exists (via `std::filesystem`).
3. Requests the active backend to enumerate all windows through `current_window_backend->getAllWindows(current_window_backend.get())`.
4. Receives a list of windowDesc records from the backend.
5. Saves each record to a text file using windowDesc::saveTo.

If any backend or I/O operation fails, the function catches the exception and returns `0`, avoiding crashes.

2.1.3 3. Window representation The `windowDesc` structure

Defined in source/desk_up_backend_interface_backend/window_desc/window_desc.h and implemented in source/desk_up_backend_interface_backend/window_desc/window_desc.cc.
Each windowDesc instance represents a window in an abstract, cross-platform way.
Fields:**

- `name` window or executable name.
- `x`, `y`, `w`, `h` position and size.
- `pathToExec` absolute path to the owning executable.

  Behavior:**
- windowDesc::saveTo writes the above fields as plain text.
- Returns `0` if the path is empty or cannot be opened.
- `operator!()` is a validity check (true if geometry is all zero).

2.1.4 4. Backend implementation Windows

The Windows backend lives in source/desk_up_backend_interface_backend/window_backends/desk_up_win/desk_up_win.h and source/desk_up_backend_interface_backend/window_backends/desk_up_win/desk_up_win.cc.
It defines:

```
DeskUpWindowBootStrap winWindowDevice = {
    "win",
    WIN_CreateDevice,
    WIN_isAvailable
};
```

2.1.4.1 Backend creation

- WIN_CreateDevice builds a new DeskUpWindowDevice and wires function pointers for:
  - `getAllWindows`
  - `getWindowHeight`, `getWindowWidth`, `getWindowXPos`, `getWindowYPos`
  - `getDeskUpPath`

2.1.4.2 Enumerating windows

WIN_getAllWindows calls `EnumDesktopWindows`, which triggers a callback to:

1. Skip invisible or zero-sized windows.
2. Populate a windowDesc using:
   - WIN_getWindowXPos, WIN_getWindowYPos, WIN_getWindowWidth, WIN_getWindowHeight `GetWindowInfo`.
   - WIN_getPathFromWindow process path via `GetWindowThreadProcessId` `OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION)` `QueryFullProcessImageNameW`.
3. Append each valid record to a `std::vector<windowDesc>`.

2.1.4.3 Workspace path resolution

WIN_getDeskUpPath determines the folder:

- Uses `SHGetKnownFolderPath(FOLDERID_RoamingAppData)` when available.
- Falls back to `APPDATA%` or the executable directory.
- Ensures a `DeskUp/` folder exists.

2.1.5 5. Flow summary

```
DU_Init()
  |--> winWindowDevice.isAvailable() -> WIN_isAvailable()
  |--> winWindowDevice.createDevice() -> WIN_CreateDevice()
  |--> dev.getDeskUpPath() -> WIN_getDeskUpPath()
  \--> sets DESKUPDIR and current_window_backend

DeskUpWindow::saveAllWindowsLocal("WorkspaceName")
  |--> builds <DESKUPDIR>\WorkspaceName
  |--> current_window_backend->getAllWindows(...) -> WIN_getAllWindows()
  |     |--> EnumDesktopWindows -> WIN_createAndSaveWindow()
  |     \--> fills std::vector<windowDesc>
  |--> iterates vector
  |     \--> windowDesc::saveTo(<workspace path>)
  \--> returns success (1) or failure (0)
```

2.1.6 6. File map (as in the repository)

- `source/desk_up_backend_interface/desk_up_backend_interface.h` / `.cc` frontend orchestration (uses backend).
- `source/desk_up_backend_interface_backend/window_core.h` / `.cc` backend initialization (DU_Init) and global state.
- `source/desk_up_backend_interface_backend/window_backends/desk_up_win/desk_up_win.h` / `.cc` Windows-specific backend.
- `source/desk_up_backend_interface_backend/window_desc/window_desc.h` / `.cc` window record + persistence.
- `source/desk_up_backend_interface_backend/backend_utils.cc` backend helpers.
- `source/desk_up_backend_interface_backend/desk_up_backend_interface_device.h`, `desk_up_backend_interface_bootstrap.h` device/bootstrap interfaces.
- `source/desk_up/`, `source/desk_up/` additional higher-level modules.
- `source/main.cc` entry point.

2.1.7 7. Extensibility

The folder source/desk_up_backend_interface_backend/window_backends/desk_up_x11 scaffolds a future **X11/Linux backend**. Adding a new backend requires a new DeskUpWindowBootStrap with↩ :

- `isAvailable()` to detect platform support.
- `createDevice()` to provide the correct function pointers.

Chapter 3

Directory Hierarchy

3.1 Directories

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

File Index

6.1 File List

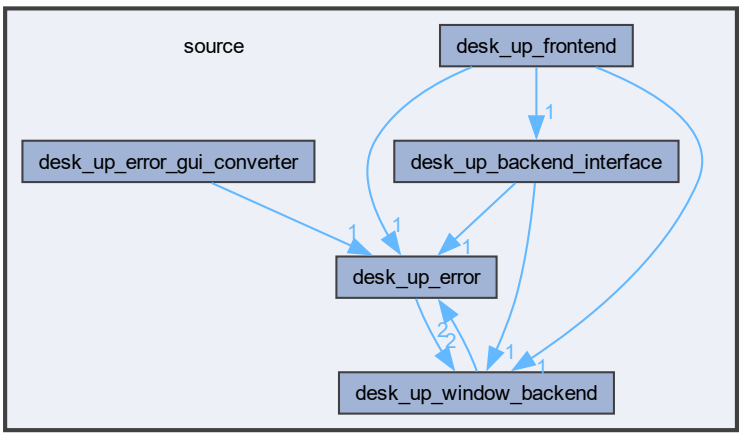
Here is a list of all documented files with brief descriptions:

Chapter 7

Directory Documentation

7.1 source Directory Reference

Directory dependency graph for source:



Directories

- directory desk_up_backend_interface
- directory desk_up_error
- directory desk_up_error_gui_converter
- directory desk_up_frontend
- directory desk_up_window_backend

7.1.1 Detailed Description

7.1.2 DeskUp internal functioning

DeskUp is organized into three main layers:

- **Frontend**: GUI of window and workspace operations.
- **Core**: backend initialization and management logic.
- **Backend**: platform-specific implementations for Windows

This document is here for anyone to have a clear understanding of how DeskUp operates, or at least to know **where** to look for a specific thing

7.1.3 1. Initialization - Choosing and bootstrapping a backend

When DeskUp starts, the application calls `DU_Init` (declared in `source/desk_up_window_backend/window_core.h` and implemented in `source/desk_up_window_backend/window_core.cc`).
`DU_Init()` performs three key actions:

1. Iterates through a list of available **backend bootstraps** (currently only Windows).
    - The Windows backend defines its bootstrap (`winWindowDevice`) in `source/desk_up_window_backend/window_backends/desk_up_win/desk_up_win.h`.
2. For each backend, it checks availability via `DeskUpWindowBootStrap::isAvailable()` (Windows maps to `WIN_isAvailable`).
3. Once it finds a working backend, it:
    - Creates the device with `DeskUpWindowBootStrap::createDevice()` (Windows maps to `WIN_CreateDevice`).
    - Retrieves the base folder with `DeskUpWindowDevice::getDeskUpPath()` (Windows maps to `WIN_getDeskUpPath`).
    - Sets the global variables:
        - **DESKUPDIR** base workspace directory.
        - **current_window_backend** active backend device.

If initialization succeeds, `DU_Init()` logs the connected backend and returns `1`. If none is available, it returns `0`.

7.1.3.1 2. High-level operations DeskUpBackendInterface facade

`DeskUpBackendInterface` is defined in `source/desk_up_backend_interface/desk_up_backend_interface.h` and implemented in `source/desk_up_backend_interface/desk_up_backend_interface.cc`.
This class acts as the **frontend facade**, coordinating workspace-level operations with the backend.

7.1.3.1.1 `DeskUpBackendInterface::saveAllWindowsLocal(std::string workspaceName)`

1. Builds `<DESKUPDIR>/<workspaceName>` using the global path set by `DU_Init()`.
2. Ensures the directory exists (via `std::filesystem`).
3. Requests the active backend to enumerate all windows through `current_window_backend->getAllWindows(current_window_backend.get())`.
4. Receives a list of `windowDesc` records from the backend.
5. Saves each record to a text file using `windowDesc::saveTo()`.

If any backend or I/O operation fails, the function catches the exception and returns `0`, avoiding crashes.

7.1.3.2 3. Window representation The `windowDesc` structure

Defined in `source/desk_up_window_backend/window_desc/window_desc.h` and implemented in `source/desk_up_window_backend/window_desc/window_desc.cc`.
Each `windowDesc` instance represents a window in an abstract, cross-platform way.
Fields:

- `name` window or executable name.
- `x`, `y`, `w`, `h` position and size.

- `pathToExec` absolute path to the owning executable.

Behavior:

- `saveTo(path)` writes the above fields as plain text.
- Returns `0` if the path is empty or cannot be opened.
- `operator!()` is a validity check (true if geometry is all zero).

7.1.3.3 4. Backend implementation Windows

The Windows backend lives in source/desk_up_window_backend/window_backends/desk_up_win/desk_up_win.h and source/desk_up_window_backend/window_backends/desk_up_win/desk_up_win.cc.
It defines:

```
DeskUpWindowBootStrap winWindowDevice = {
    "win",
    WIN_CreateDevice,
    WIN_isAvailable
};
```

7.1.3.3.1 Backend creation

- `WIN_CreateDevice()` builds a new `DeskUpWindowDevice` and wires function pointers for:
  - `getAllWindows`
  - `getWindowHeight`, `getWindowWidth`, `getWindowXPos`, `getWindowYPos`
  - `getDeskUpPath`
  - ...

All the common functions necessary for any platform to make DeskUp work. If there are platform-specific calls, they won't appear in the API.

7.1.3.3.2 Enumerating windows

`WIN_getAllWindows()` calls `EnumDesktopWindows`, which triggers a callback to:

1. Skip invisible or zero-sized windows.
2. Populate a `windowDesc` using:
   - `WIN_getWindowXPos`, `WIN_getWindowYPos`, `WIN_getWindowWidth`, `WIN_getWindowHeight` `GetWindowInfo`.
   - `WIN_getPathFromWindow` process path via `GetWindowThreadProcessId` `OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION)` `QueryFullProcessImageNameW`.
3. Append each valid record to a `std::vector<windowDesc>`.

7.1.3.3.3 Workspace path resolution

`WIN_getDeskUpPath()` determines the folder:

- Uses `SHGetKnownFolderPath(FOLDERID_RoamingAppData)` when available.
- Falls back to `APPDATA%` or the executable directory.
- Ensures a `DeskUp/` folder exists.

7.1.3.4 5. How everything connects Flow summary

```
DU_Init()
  |--> winWindowDevice.isAvailable() -> WIN_isAvailable()
  |--> winWindowDevice.createDevice() -> WIN_CreateDevice()
  |--> dev.getDeskUpPath() -> WIN_getDeskUpPath()
  \--> sets DESKUPDIR and current_window_backend

DeskUpBackendInterface::saveAllWindowsLocal("WorkspaceName")
  |--> builds <DESKUPDIR>\WorkspaceName
  |--> current_window_backend->getAllWindows(...) -> WIN_getAllWindows()
  |     |--> EnumDesktopWindows -> WIN_createAndSaveWindow()
  |     \--> fills std::vector<windowDesc>
  |--> iterates vector
  |     \--> windowDesc::saveTo(<workspace path>)
  \--> returns success (1) or failure (0)
```

7.1.3.5 6. File map

Layer	Path	Description
Frontend	source/desk_up/main↩ Window.h / .cpp	Qt GUI layer orchestrating workspace operations.
Core (Backend interface)	source/desk_up_backend_interface/desk_up_backend_interface.h / .cc	Backend communication facade (DeskUpBackendInterface).
Core (Initialization)	source/desk_up_window↩ _backend/window_core.h / .cc	Backend initialization (DU_Init) and global state.
Backend (Windows)	source/desk_up_window_backend/desk_up_win/desk_up_w / .cc	Implements Windows-specific logic.
Window record	source/desk_up_window_backend/desk_up_win/desk_up_window.h / .cc	Data structure representing windows.
Backend utilities	source/desk_up_window↩ _backend/backend_↩ utils/backend_utils.cc	Shared helper functions for backends.
Interfaces	source/desk_up_window_backend/desk_up_win/desk_up_device.h, desk_up_window_bootstrap.h	Device and bootstrap definitions.
Error system	source/desk_up_error/ and source/desk_up_error_gui↩ _converter/	Error logic and GUI integration.
Entry point	source/desk_up/main.cpp	Program start (Qt).

7.1.3.6 7. Extensibility

DeskUps modular design allows adding new platforms easily.
Adding a new backend only requires implementing a DeskUpWindowBootStrap with:

- isAvailable() to detect platform support.
- createDevice() to provide the correct function pointers.

It also facilitates creating tests for the backend, as one can create a testing device, which includes predefined values for each function. Multiple devices can be connected, so as to test multiple functionalities.

Chapter 8

Class Documentation

8.1 DeskUpBackendInterface Struct Reference

Convenience façade for workspace-level window operations.
`#include <desk_up_backend_interface.h>`

Static Public Member Functions

- static DeskUp::Status saveAllWindowsLocal (std::string workspaceName)

  Saves all currently enumerated windows to a local workspace folder.
- static DeskUp::Status restoreWindows (std::string workspaceName)

  Restores all tabs saved previously in the workspace name specified by the parameter.
- static bool isWorkspaceValid (const std::string &workspaceName)

  This function checks whether if a string is a valid name for a workspace folder.
- static bool existsWorkspace (const std::string &workspaceName)

  This function checks whether if a given workspace with the name `workspaceName` already exists.
- static int removeWorkspace (const std::string &workspaceName)

  This function deletes a workspace.
- static bool existsFile (const fs::path &filePath)

  Checks whether a given file path exists on disk.

8.1.1 Detailed Description

Convenience façade for workspace-level window operations.
This lightweight façade exposes high-level operations that orchestrate the active backend device (see current_↩ window_backend) to work with window snapshots and workspaces.
Error handling:

- All functions now return `DeskUp::Status` or `DeskUp::Result<T>` values instead of throwing exceptions.
- Each function propagates `DeskUp::Error` instances describing severity (`Level`) and category (`Err↩ Type`).
- Fatal errors stop the current operation immediately.
- Retry and Warning levels are non-fatal and allow continuation.

See also

DU_Init

DESKUPDIR

current_window_backend

DeskUpWindowDevice::getAllOpenWindows

windowDesc

windowDesc::saveTo

Version

0.1.1

Date

2025

8.1.2 Member Function Documentation

8.1.2.1 existsFile()

```
bool DeskUpBackendInterface::existsFile (
            const fs::path & filePath)  [static]
```

Checks whether a given file path exists on disk.
Uses the C++17 <filesystem> library to test whether the file or directory specified by `filePath` exists in the current filesystem. Returns `true` if it exists, `false` otherwise.
Reads:

- Filesystem state at the given `filePath`.

Parameters

filePath	The absolute or relative filesystem path to check.

Returns

`true` if the file or directory exists, `false` otherwise.

Note

This function is typically used when generating unique file names for saving window descriptors (to avoid overwriting files when multiple windows share the same name).

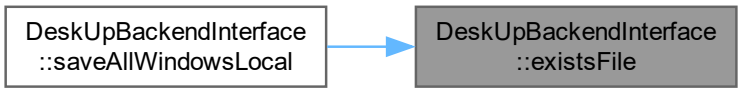
Version

0.1.1

Date

2025

Here is the caller graph for this function:



8.1.2.2 existsWorkspace()

```
bool DeskUpBackendInterface::existsWorkspace (
            const std::string & workspaceName)  [static]
```

This function checks whether if a given workspace with the name `workspaceName` already exists.
Checks whether `<DESKUPDIR>/<workspaceName>` exists and is a directory.
Reads:

- Filesystem state under DESKUPDIR.

Parameters

workspaceName	Name of the workspace folder to use under DESKUPDIR.

Returns

`true` if the workspace exists, `false` otherwise.

Note

Ensure DU_Init has been called successfully before invoking this method so that DESKUPDIR and current↩
_window_backend are properly initialized.

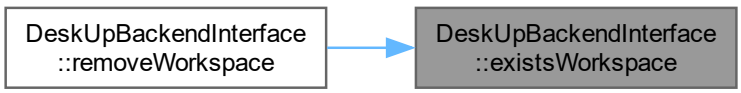
Version

0.1.1

Date

2025

Here is the caller graph for this function:



8.1.2.3 isWorkspaceValid()

```
bool DeskUpBackendInterface::isWorkspaceValid (
            const std::string & workspaceName)  [static]
```

This function checks whether if a string is a valid name for a workspace folder.
A workspace name is invalid if it is empty or contains any of the following forbidden characters: \\ / : ? * " < > |.
Reads:

- Pure string validation, does not access filesystem.

Parameters

workspaceName	Name of the workspace to check.

Returns

`true` if the workspace name is valid, `false` otherwise.

Note

Ensure DU_Init has been called successfully before invoking this method so that DESKUPDIR and current↩ _window_backend are properly initialized.

Version

0.1.1

Date

2025

8.1.2.4 removeWorkspace()

```
int DeskUpBackendInterface::removeWorkspace (
            const std::string & workspaceName)  [static]
```

This function deletes a workspace.
Removes <DESKUPDIR>/<workspaceName> recursively from disk. Returns `1` on success, `0` if the workspace does not exist or deletion failed.
Reads:

- Filesystem state under DESKUPDIR.

Parameters

workspaceName	Name of the workspace folder to use under DESKUPDIR.

Returns

`1` if deleted, `0` otherwise.

Note

Ensure DU_Init has been called successfully before invoking this method so that DESKUPDIR and current↩ _window_backend are properly initialized.

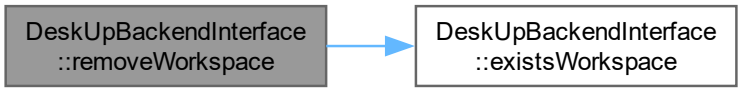
Version

0.1.1

Date

2025

Here is the call graph for this function:



8.1.2.5 restoreWindows()

```
DeskUp::Status DeskUpBackendInterface::restoreWindows (
            std::string workspaceName)  [static]
```

Restores all tabs saved previously in the workspace name specified by the parameter.
Iterates over all files in <DESKUPDIR>/<workspaceName> and, for each saved window:

1. Loads the window description from file (recoverSavedWindow).
2. Closes existing process instances of that executable (closeProcessFromPath).
3. Launches a new process (loadWindowFromPath).
4. Resizes the new window to the stored geometry (resizeWindow).

Non-fatal backend errors (Retry or Warning) are logged to console but do not abort the overall restore cycle. Fatal errors propagate as a failed DeskUp::Status.
Calls (indirectly through the backend):

- DeskUpWindowDevice::recoverSavedWindow
- DeskUpWindowDevice::closeProcessFromPath
- DeskUpWindowDevice::loadWindowFromPath
- DeskUpWindowDevice::resizeWindow

Reads:

- DESKUPDIR (workspace base directory).

Parameters

workspaceName	Name of the workspace folder to use under DESKUPDIR.

Returns

`DeskUp::Status` — empty on success, or `std::unexpected(DeskUp::Error)` on failure.

@errors

- Level::Fatal, ErrType::NotFound → Workspace directory missing.
- Level::Fatal, ErrType::InvalidInput → Corrupted or incomplete window file.
- Level::Retry, ErrType::NotFound → Process launched but no main HWND found.
- Level::Warning → Individual window restore failed but continued.

Note

Ensure DU_Init has been called successfully before invoking this method so that DESKUPDIR and current↩ _window_backend are properly initialized.

Version

0.1.1

Date

2025

8.1.2.6 saveAllWindowsLocal()

```
DeskUp::Status DeskUpBackendInterface::saveAllWindowsLocal (
            std::string workspaceName)  [static]
```

Saves all currently enumerated windows to a local workspace folder.
Builds `<DESKUPDIR>/<workspaceName>` and ensures the directory exists. Then asks the active backend device to enumerate all open windows and writes each window's description to an individual file using `windowDesc::saveTo()`. Non-fatal save errors are skipped; fatal ones abort the operation.
Calls (indirectly through the backend):

- `DeskUpWindowDevice::getAllOpenWindows(DeskUpWindowDevice*)`
- `windowDesc::saveTo(const std::string&)`

Reads:

- DESKUPDIR (must have been set by a prior DU_Init call).

Parameters

workspaceName	Name of the workspace folder to create/use under DESKUPDIR.

Returns

`DeskUp::Status` — empty on success, or `std::unexpected(DeskUp::Error)` on failure.

@errors

- Level::Fatal, ErrType::Os or ErrType::InvalidInput → Enumeration or write failure.
- Level::Warning → One or more windows skipped (non-fatal save errors).
- Level::Retry → Temporary filesystem access problems.

Note

Ensure DU_Init has been called successfully before invoking this method so that DESKUPDIR and current_window_backend are properly initialized.

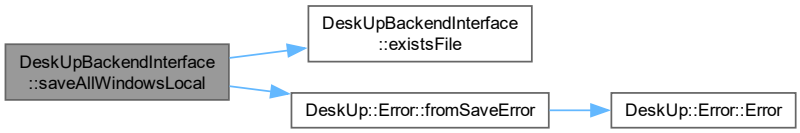
Version

0.1.1

Date

2025

Here is the call graph for this function:

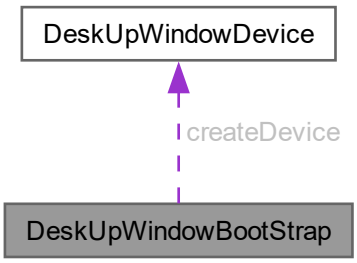


The documentation for this struct was generated from the following files:

- desk_up_backend_interface.h
- desk_up_backend_interface.cc

8.2 DeskUpWindowBootStrap Struct Reference

This struct is a wrapper that holds a call to create a window device.
`#include <desk_up_window_bootstrap.h>`
Collaboration diagram for DeskUpWindowBootStrap:



Public Attributes

- const char * name

  The name of the backend associated with the device call.
- DeskUpWindowDevice(* createDevice)()

  A pointer to function that creates a Device.
- bool(* isAvailable)(void)

  A pointer to function that is used to check whether if a given backend is available in the current device.

8.2.1 Detailed Description

This struct is a wrapper that holds a call to create a window device.
Each backend packs all of it's functions in a device, which should be the only way to access it's calls. It fills backend-specific info in the device, as well as connecting all the implementations of the generic backend calls.

See also

DeskUpWindowDevice

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Version

0.1.0

Date

2025

8.2.2 Member Data Documentation

8.2.2.1 createDevice

```
DeskUpWindowDevice(* DeskUpWindowBootStrap::createDevice) ()
```

A pointer to function that creates a Device.

Returns

A pointer to a heap allocated windowDevice.

Version

0.1.0

Date

2025

8.2.2.2 isAvailable

```
bool(* DeskUpWindowBootStrap::isAvailable) (void)
```

A pointer to function that is used to check whether if a given backend is available in the current device.

Returns

`true` if the backend is available in this device, `false` otherwise

Version

0.1.0

Date

2025

8.2.2.3 name

```
const char* DeskUpWindowBootStrap::name
```

The name of the backend associated with the device call.

Warning

This is a raw C-string pointer. The referenced literal becomes invalid once the struct goes out of scope.

Version

0.1.0

Date

2025

The documentation for this struct was generated from the following file:

- desk_up_window_bootstrap.h

8.3 DeskUpWindowDevice Struct Reference

This abstract struct represents all the common calls that any backend must have.
`#include <desk_up_window_device.h>`

Public Attributes

- DeskUp::Result< unsigned int >(* getWindowHeight)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get the height of a window.
- DeskUp::Result< unsigned int >(* getWindowWidth)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get the width of a window.
- DeskUp::Result< int >(* getWindowXPos)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get the X position of a the top left corner of a window.
- DeskUp::Result< int >(* getWindowYPos)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get the Y position of a the top left corner of a window.
- DeskUp::Result< fs::path >(* getPathFromWindow)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get the path to the executable that created the window.
- DeskUp::Result< std::string >(* getDeskUpPath)(void)

  A pointer to function that is used to get the generic DeskUp workspaces path.
- DeskUp::Result< std::vector< windowDesc > >(* getAllOpenWindows)(DeskUpWindowDevice *_this)

  A pointer to function that is used to get a list of abstract windows.
- DeskUp::Status(* loadWindowFromPath)(DeskUpWindowDevice *_this, const fs::path &path)

  A pointer to function that is used to open a window from a given path.
- DeskUp::Result< windowDesc >(* recoverSavedWindow)(DeskUpWindowDevice *_this, const fs::path &filePath)

  A pointer to function that is used to recover a window from a deskUp file, which shall be located inside appData\↩ DeskUp.
- DeskUp::Status(* resizeWindow)(DeskUpWindowDevice *_this, const windowDesc window)

  A pointer to function that is used to resize a given window.
- DeskUp::Result< unsigned int >(* closeProcessFromPath)(DeskUpWindowDevice *_this, const fs::path &path, bool allowForce)

  A pointer to function that is used to close all the windows associated with a given path.
- void * internalData

  A pointer that points to the specific information needed by each backend.
- void(* DestroyDevice)(DeskUpWindowDevice *_this)

  A pointer to function that is used to delete the device.

8.3.1 Detailed Description

This abstract struct represents all the common calls that any backend must have.
There are multiple pointers to functions inside this struct. Each one of them gets connected to a backend function whenever the backend gets created. When invoking the pointer, it implicitly calls the correct backend function, thus giving the expected result for that specific backend. This struct also carries specific backend information needed to get information from a backend successfully.

See also

windowData

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Version

0.1.0

Date

2025

8.3.2 Member Data Documentation

8.3.2.1 closeProcessFromPath

```
DeskUp::Result< unsigned int >(* DeskUpWindowDevice::closeProcessFromPath) (DeskUpWindowDevice
*_this, const fs::path &path, bool allowForce)
```

A pointer to function that is used to close all the windows associated with a given path.

Parameters

_this	The very same instance
path	a `std::string` instance that represents the path
allowForce	Whether if the call should force the program to close

Returns

The number of closed windows from a specific app

Version

0.2.0

Date

2025

8.3.2.2 DestroyDevice

```
void(* DeskUpWindowDevice::DestroyDevice) (DeskUpWindowDevice *_this)
```

A pointer to function that is used to delete the device.
Each device defines it's own deleter, which then gets called in the wrapper DU_destroy inside window_core.cc

Parameters

_this	The very same instance

Version

0.3.2

Date

2025

8.3.2.3 getAllOpenWindows

`DeskUp::Result< std::vector< windowDesc > >(* DeskUpWindowDevice::getAllOpenWindows) (DeskUpWindowDevice *_this)`

A pointer to function that is used to get a list of abstract windows.
For any backend to return the same thing, an abstract struct representing a windows is created. Regardless of the implementation, every device returns a vector of this type.

Parameters

_this	The very same instance

Returns

An `std::vector` representing all the visible and non-minimized windows currently open

Version

0.1.0

Date

2025

8.3.2.4 getDeskUpPath

`DeskUp::Result< std::string >(* DeskUpWindowDevice::getDeskUpPath) (void)`

A pointer to function that is used to get the generic DeskUp workspaces path.
The return path is the path to the top-level folder where all the user workspaces are saved. A workspace whose name is "foo" will have it's information saved in `getDeskUpPath() + "/foo"`

Returns

An `std::string` representing the path to the top-level Desk Up workspaces folder

Version

0.1.0

Date

2025

8.3.2.5 getPathFromWindow

`DeskUp::Result< fs::path >(* DeskUpWindowDevice::getPathFromWindow) (DeskUpWindowDevice *_this)`

A pointer to function that is used to get the path to the executable that created the window.

Parameters

_this	The very same instance

Returns

An `filesystem::path` representing the path of the window

Version

0.1.0

Date

2025

8.3.2.6 getWindowHeight

`DeskUp::Result< unsigned int >(* DeskUpWindowDevice::getWindowHeight) (DeskUpWindowDevice *_this)`

A pointer to function that is used to get the height of a window.

Parameters

_this	The very same instance

Returns

An `unsigned int` representing the height of the window

Version

0.1.0

Date

2025

8.3.2.7 getWindowWidth

`DeskUp::Result< unsigned int >(* DeskUpWindowDevice::getWindowWidth) (DeskUpWindowDevice *_this)`

A pointer to function that is used to get the width of a window.

Parameters

_this	The very same instance

Returns

An `unsigned int` representing the width of the window

Version

0.1.0

Date

2025

8.3.2.8 getWindowXPos

`DeskUp::Result< int >(* DeskUpWindowDevice::getWindowXPos) (DeskUpWindowDevice *_this)`
A pointer to function that is used to get the X position of a the top left corner of a window.

Parameters

_this	The very same instance

Returns

An `int` representing the X position of the top left corner of a window

Version

0.1.0

Date

2025

8.3.2.9 getWindowYPos

`DeskUp::Result< int >(* DeskUpWindowDevice::getWindowYPos) (DeskUpWindowDevice *_this)`
A pointer to function that is used to get the Y position of a the top left corner of a window.

Parameters

_this	The very same instance

Returns

An `int` representing the Y position of the top left corner of a window

Version

0.1.0

Date

2025

8.3.2.10 internalData

`void* DeskUpWindowDevice::internalData`
A pointer that points to the specific information needed by each backend.
each backend defines WindowData, which is the template to seek the values of this pointer. Whenever a device call wants to access backend-specific info, this pointer gets casted to `windowData*` by `std::reinterpret_cast`. Then the backend accesses whatever information it needs. Finally, the same call casts back this pointer to `void*` , leaving everything as it was. One can interpret this pointer as a "black box", or a "pouch", where the info gets thrown inside, and then reinterpreted back whenever it is needed

Version

0.1.0

Date

2025

8.3.2.11 loadWindowFromPath

`DeskUp::Status(* DeskUpWindowDevice::loadWindowFromPath) (DeskUpWindowDevice *_this, const fs::path &path)`

A pointer to function that is used to open a window from a given path.
If the path is empty,

Parameters

_this	The very same instance
path	a `const char*` to the executable

Returns

`void`

Version

0.2.0

Date

2025

8.3.2.12 recoverSavedWindow

`DeskUp::Result< windowDesc >(* DeskUpWindowDevice::recoverSavedWindow) (DeskUpWindowDevice *_this, const fs::path &filePath)`

A pointer to function that is used to recover a window from a deskUp file, which shall be located inside appData\DeskUp.

Parameters

_this	The very same instance
path	a `const char*` to the executable

Returns

A `windowDesc` representing the recovered window. If any of the recovery processes fails, the associated field will be set to the default value for it's type (int 0 and string "")

Version

0.2.0

Date

2025

8.3.2.13 resizeWindow

`DeskUp::Status(* DeskUpWindowDevice::resizeWindow) (DeskUpWindowDevice *_this, const windowDesc window)`

A pointer to function that is used to resize a given window.
Information about the window whose geometry is intended to modify must be specified inside the `_this->internalData` parameter

Parameters

_this	The very same instance
path	a `windowDesc` instance whose geometry wants to be used for the resizing

Returns

A `void`

Version

0.2.0

Date

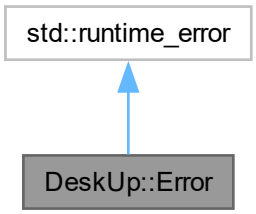
2025

The documentation for this struct was generated from the following file:

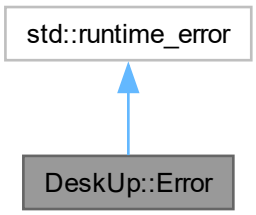
- desk_up_window_device.h

8.4 DeskUp::Error Class Reference

Centralized representation of a DeskUp runtime error.
`#include <desk_up_error.h>`
Inheritance diagram for DeskUp::Error:



Collaboration diagram for DeskUp::Error:



Public Member Functions

- Error ()

  Default constructor (represents a non-error).
- Error (Level l, ErrType err, unsigned int t, std::string msg)

  Constructs an error with full metadata.
- ErrType type () const noexcept

  Returns the error type.
- Level level () const noexcept

  Returns the error severity level.
- int attempts () const noexcept

  Returns how many times the operation was retried.
- windowDesc whichWindow () const noexcept

  Returns the affected window (if available).
- bool **isFatal** () const noexcept

  Whether the error is fatal.
- bool **isSkippable** () const noexcept

  Whether the error is skipable.
- bool **isWarning** () const noexcept

  Whether the error is a warning.
- bool **isError** () const noexcept

  Whether the error is a system error.
- bool **isRetryable** () const noexcept

  Whether the error can be retried.
- **operator bool** () const noexcept

  Converts to `true` if this instance represents an actual error.

Static Public Member Functions

- static Error fromSaveError (int e)

  Converts a `SaveErrorCode` (from `window_desc.cc`) into a structured error.

8.4.1 Detailed Description

Centralized representation of a DeskUp runtime error.
The `Error` class extends `std::runtime_error` to attach structured metadata to exceptions thrown across DeskUp subsystems. Each error carries:

- A **Level** (severity classification)
- An **ErrType** (category)
- The number of **retries** attempted (if applicable)
- An optional **windowDesc** structure indicating the affected window

Conversion utilities (`fromLastWinError()`, `fromSaveError()`) map Windows system errors and DeskUp-specific codes to structured `Error` instances.

See also

Level

ErrType

windowDesc

Version

0.2.1

Date

2025

8.4.2 Constructor & Destructor Documentation

8.4.2.1 Error() [1/2]

```
DeskUp::Error::Error () [inline]
```

Default constructor (represents a non-error).

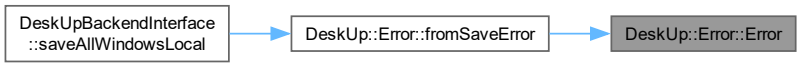
Version

0.2.1

Date

2025

Here is the caller graph for this function:



8.4.2.2 Error() [2/2]

```
DeskUp::Error::Error (
            Level l,
            ErrType err,
            unsigned int t,
            std::string msg) [inline]
```

Constructs an error with full metadata.

Parameters

l	Error level (severity).
err	Error type (category).
t	Number of retries attempted before failure.
msg	Descriptive error message.

Version

0.2.1

Date

2025

8.4.3 Member Function Documentation

8.4.3.1 attempts()

```
int DeskUp::Error::attempts () const  [inline], [noexcept]
```

Returns how many times the operation was retried.

Returns

Integer representing the retry count.

8.4.3.2 fromSaveError()

```
DeskUp::Error DeskUp::Error::fromSaveError (
            int e)  [static]
```

Converts a `SaveErrorCode` (from `window_desc.cc`) into a structured error.

Parameters

e	Integer error code returned by a save operation.

Returns

A structured Error describing the save failure.

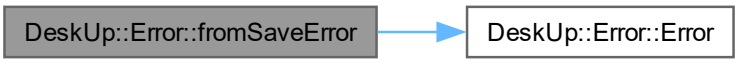
Version

0.2.1

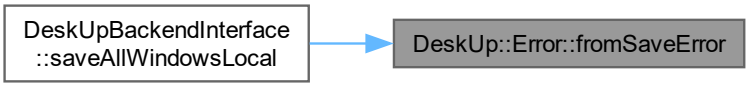
Date

2025

Here is the call graph for this function:



Here is the caller graph for this function:



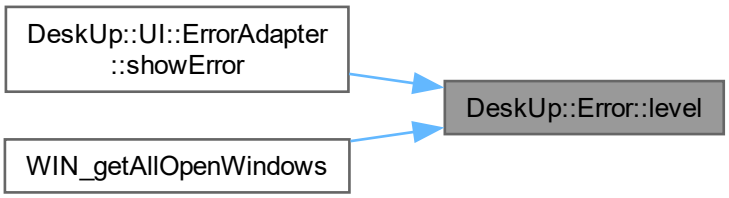
8.4.3.3 level()

`Level DeskUp::Error::level () const [inline], [noexcept]`
Returns the error severity level.

Returns

The associated Level.

Here is the caller graph for this function:



8.4.3.4 type()

`ErrType DeskUp::Error::type () const [inline], [noexcept]`
Returns the error type.

Returns

The associated ErrType.

8.4.3.5 whichWindow()

`windowDesc DeskUp::Error::whichWindow () const [inline], [noexcept]`
Returns the affected window (if available).

Returns

A windowDesc associated with this error.

The documentation for this class was generated from the following files:

- desk_up_error.h
- desk_up_error.cc

8.5 DeskUp::UI::ErrorAdapter Class Reference

Utility class to bridge DeskUp's error system with the Qt graphical interface.
`#include <desk_up_error_gui_converter.h>`

Static Public Member Functions

- static int showError (const DeskUp::Error &err)

  Displays a Qt dialog box for a given error.
- static QString **getUserMessage** (const DeskUp::Error &err)
- static std::pair< QString, QMessageBox::Icon > **mapLevel** (Level lvl)

8.5.1 Detailed Description

Utility class to bridge DeskUp's error system with the Qt graphical interface.
Converts `DeskUp::Error` instances into localized and user-friendly messages displayed via `QMessageBox`.

- Translates error **levels** into dialog icons and titles
- Translates error **types** into human-readable messages
- Displays the result via `QMessageBox`

See also

DeskUp::Error

Level

ErrType

Version

0.3.0

Date

2025

8.5.2 Member Function Documentation

8.5.2.1 showError()

```
int DeskUp::UI::ErrorAdapter::showError (
            const DeskUp::Error & err)  [static]
```

Displays a Qt dialog box for a given error.

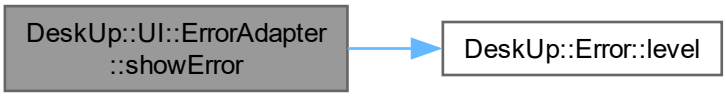
Parameters

err	A DeskUp::Error instance to display.

Returns

Integer result code from `QMessageBox::exec()`.

Here is the call graph for this function:

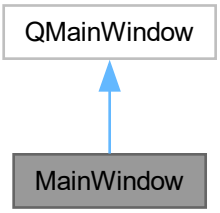


The documentation for this class was generated from the following files:

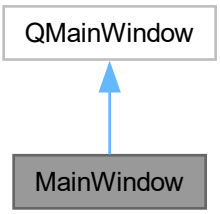
- desk_up_error_gui_converter.h
- desk_up_error_gui_converter.cc

8.6 MainWindow Class Reference

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

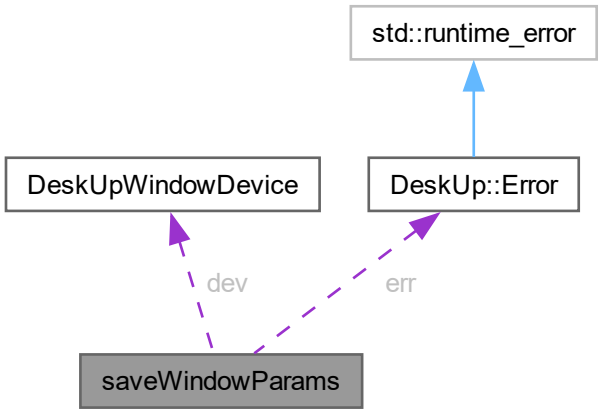
- **MainWindow** (QWidget ∗parent=nullptr)

The documentation for this class was generated from the following files:

- mainWindow.h
- mainWindow.cpp

8.7 saveWindowParams Struct Reference

Collaboration diagram for saveWindowParams:



Public Attributes

- DeskUpWindowDevice * **dev**
- std::vector< windowDesc > * **res**
- DeskUp::Error * **err**

The documentation for this struct was generated from the following file:

- desk_up_win.cc

8.8 windowData Struct Reference

Public Attributes

- HWND **hwnd**

The documentation for this struct was generated from the following file:

- desk_up_win.cc

8.9 windowDesc Struct Reference

Describes a single window instance in the DeskUp system.
`#include <window_desc.h>`

Public Member Functions

- **windowDesc** ()

  Constructs a window descriptor with default parameters (integers to 0 and strings empty).
- windowDesc (std::string n, int xPos, int yPos, int width, int height, std::string p)

  Constructs a window descriptor with explicit name, geometry, and executable path.
- int saveTo (fs::path path)

  Saves the current window description to a file.

- bool operator! () const

  Returns whether the window description is invalid or empty.

Public Attributes

- std::string name

  Constructs a window descriptor with explicit name, geometry, and executable path.
- int **x**

  The X coordinate (top-left corner) of the window.
- int **y**

  The Y coordinate (top-left corner) of the window.
- int **w**

  The window width in pixels.
- int **h**

  The window height in pixels.
- fs::path **pathToExec**

  The absolute path to the executable that owns this window.

8.9.1 Detailed Description

Describes a single window instance in the DeskUp system.
This structure holds all relevant information needed to represent a window in the abstract DeskUp format. It allows storing the executable path, name, and geometry (x, y, width, height).
Instances of this struct are typically generated by backend functions such as `WIN_getAllOpenWindows()`, and can later be serialized using `saveTo()`.

See also

windowDesc::saveTo()

WIN_getAllOpenWindows()

8.9.2 Constructor & Destructor Documentation

8.9.2.1 windowDesc()

```
windowDesc::windowDesc (
            std::string n,
            int xPos,
            int yPos,
            int width,
            int height,
            std::string p)  [inline]
```

Constructs a window descriptor with explicit name, geometry, and executable path.

Parameters

n	window name.
x	X coordinate (top-left) in pixels.
y	Y coordinate (top-left) in pixels.
w	Width in pixels.
h	Height in pixels.
p	Absolute path to the owning executable.

8.9.3 Member Function Documentation

8.9.3.1 operator"!()

```
bool windowDesc::operator!  () const  [inline]
```

Returns whether the window description is invalid or empty.
This operator checks if all window geometry attributes (x, y, w, h) are zero, which indicates an uninitialized or invalid state.

Returns

`true` if all coordinates and dimensions are zero, `false` otherwise.

8.9.3.2 saveTo()

```
int windowDesc::saveTo (
            fs::path path)
```

Saves the current window description to a file.
Writes the window's executable path and geometry (x, y, width, height) to the specified file, each on a new line.
This function returns a `SaveErrorCode` value to indicate whether the operation succeeded or failed, allowing fine-grained error handling. Positive values indicate success; negative values specify different failure modes.
Format written to file (one value per line):

1. Path to executable (`pathToExec`)
2. X coordinate
3. Y coordinate
4. Width
5. Height

Parameters

path	Absolute or relative path to the file where data will be stored.

Returns

One of the following `SaveErrorCode` values:

- `SAVE_SUCCESS` (1): File successfully written.
- `ERR_EMPTY_PATH` (-1): The path parameter is empty.
- `ERR_FILE_NOT_OPEN` (-2): File could not be opened.
- `ERR_NO_PERMISSION` (-3): Insufficient permissions to write the file.
- `ERR_FILE_NOT_FOUND` (-4): Parent directory missing or invalid path.
- `ERR_DISK_FULL` (-5): Disk out of space.
- `ERR_UNKNOWN` (-6): Unexpected write failure.

Note

The function performs basic validation and never throws exceptions. Callers are expected to check the return value and propagate or log the appropriate error code using `DeskUp::Error::fromSaveError()`.

See also

windowDesc

SaveErrorCode

Version

0.2.0

Date

2025

8.9.4 Member Data Documentation

8.9.4.1 name

`std::string windowDesc::name`

Constructs a window descriptor with explicit name, geometry, and executable path.

Parameters

n	window name.
x	X coordinate (top-left) in pixels.
y	Y coordinate (top-left) in pixels.
w	Width in pixels.
h	Height in pixels.
p	Absolute path to the owning executable.

The window name.
Usually derived from the executable name or window title.
The documentation for this struct was generated from the following files:

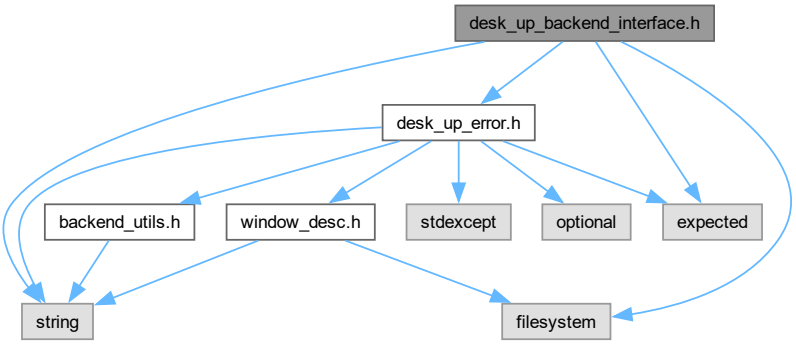
- window_desc.h
- window_desc.cc

Chapter 9

File Documentation

9.1 desk_up_backend_interface.h File Reference

High-level window operations for DeskUp (frontend utilities over the backend device).
Include dependency graph for desk_up_backend_interface.h:



Classes

- struct DeskUpBackendInterface

  Convenience façade for workspace-level window operations.

9.1.1 Detailed Description

High-level window operations for DeskUp (frontend utilities over the backend device).
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

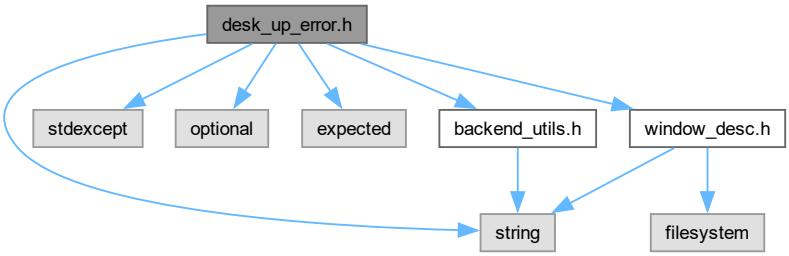
9.2 desk_up_backend_interface.h

Go to the documentation of this file.

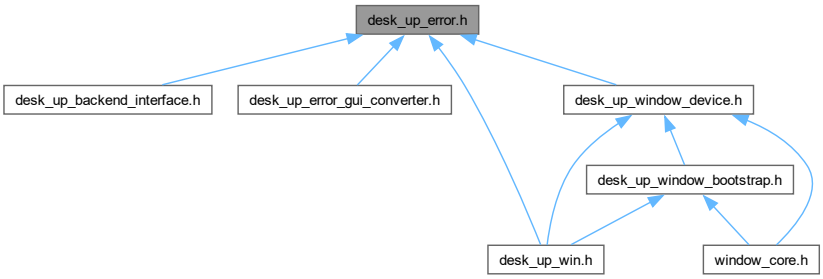
```
00001
00027
00028 #ifndef DESKUPBACKENDINTERFACE_H
00029 #define DESKUPBACKENDINTERFACE_H
00030
00031 #include <string>
00032 #include <expected>
00033 #include <filesystem>
00034
00035 #include "desk_up_error.h"
00036
00037 namespace fs = std::filesystem;
00038
00065 struct DeskUpBackendInterface{
00066
00096     static DeskUp::Status saveAllWindowsLocal(std::string workspaceName);
00097
00134     static DeskUp::Status restoreWindows(std::string workspaceName);
00135
00154     static bool isWorkspaceValid(const std::string& workspaceName);
00155
00173     static bool existsWorkspace(const std::string& workspaceName);
00174
00193     static int removeWorkspace(const std::string& workspaceName);
00194
00216     static bool existsFile(const fs::path& filePath);
00217
00218 };
00219
00220 #endif
```

9.3 desk_up_error.h File Reference

Error management system for DeskUp (centralized error representation and conversion utilities).
Include dependency graph for desk_up_error.h:



This graph shows which files directly or indirectly include this file:



Classes

- class DeskUp::Error

  Centralized representation of a DeskUp runtime error.

Typedefs

- template<typename T>
  using DeskUp::Result = std::expected<T, DeskUp::Error>

  Alias for an operation result that either holds a value or a DeskUp error.
- using DeskUp::Status = std::expected<void, DeskUp::Error>

  Alias for an operation that returns success or failure (void on success).

Enumerations

- enum class DeskUp::Level {
  Fatal , **Error** , **Warning** , **Retry** ,
  Info , **Debug** , **Default** , **Skip** ,
  None }

  Represents the severity of an error.
- enum class DeskUp::ErrType {
  InsufficientMemory , AccessDenied , SharingViolation , Io ,
  NotFound , DiskFull , DeviceNotFound , Timeout ,
  ResourceBusy , FileNotFound , InvalidFormat , InvalidInput ,
  CorruptedData , OutOfRange , NetworkError , ConnectionRefused ,
  HostUnreachable , ProtocolError , Unexpected , NotImplemented ,
  PolicyUpdated , FunctionFailed , Default , None }

  Represents the underlying type or origin of an error.

9.3.1 Detailed Description

Error management system for DeskUp (centralized error representation and conversion utilities).
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

Copyright (C) 2025 Nicolas Serrano Garcia

9.3.2 Typedef Documentation

9.3.2.1 Result

```
template<typename T>
using DeskUp::Result = std::expected<T, DeskUp::Error>
```

Alias for an operation result that either holds a value or a DeskUp error.

Template Parameters

T	The success type.

Version

0.2.1

Date

2025

9.3.2.2 Status

```
using DeskUp::Status = std::expected<void, DeskUp::Error>
```

Alias for an operation that returns success or failure (void on success).

Version

0.2.1

Date

2025

9.3.3 Enumeration Type Documentation

9.3.3.1 ErrType

```
enum class DeskUp::ErrType  [strong]
```

Represents the underlying type or origin of an error.
Each enumerator identifies a distinct class of errors so that DeskUp can map platform or library failures into portable internal types.

See also

DeskUp::Error

Version

0.2.1

Date

2025

Enumerator

InsufficientMemory	Memory allocation failed or system out of resources.
AccessDenied	Permission denied by the OS or file system.
SharingViolation	Another process holds an exclusive lock.
Io	Generic input/output error.
NotFound	Resource not found.
DiskFull	Storage volume full.
DeviceNotFound	Device unavailable or disconnected.
Timeout	Operation timed out.
ResourceBusy	Resource is in use.
FileNotFound	File could not be located.
InvalidFormat	Invalid file or data format.
InvalidInput	Invalid parameter passed by the caller.
CorruptedData	Data corruption detected.
OutOfRange	Index or parameter out of valid range.
NetworkError	Generic network failure.
ConnectionRefused	Connection attempt refused.
HostUnreachable	Target host cannot be reached.
ProtocolError	Violation of expected protocol behavior.
Unexpected	Unexpected runtime condition.
NotImplemented	Feature not yet implemented.
PolicyUpdated	External factors like dll dependencies have changed.
FunctionFailed	Generic function fail.
Default	Unspecified error type.
None	Represents no error.

9.3.3.2 Level

`enum class DeskUp::Level [strong]`

Represents the severity of an error.

This enumeration provides a consistent way to classify errors by their impact and intended handling behavior.

- **Fatal** → unrecoverable; execution must stop.
- **Error** → serious failure; operation aborted.
- **Warning** → non-critical issue; user notification recommended.
- **Retry** → recoverable error that may succeed upon retry.
- **Info** → informational message, not an error.

- **Debug** → debug-only diagnostic message.
- **Default** → unspecified severity.
- **None** → represents the absence of error.

See also

DeskUp::Error

Version

0.2.1

Date

2025

9.4 desk_up_error.h

Go to the documentation of this file.

```
00001
00027
00028 #ifndef DESKUPERROR_H
00029 #define DESKUPERROR_H
00030
00031 #include <string>
00032 #include <stdexcept>
00033 #include <optional>
00034 #include <expected>
00035 #ifdef _WIN32
00036     #include <Windows.h>
00037 #endif
00038
00039 #include "window_desc.h"
00040 #include "backend_utils.h"
00041
00042 namespace DeskUp {
00043
00065     enum class Level {
00066         Fatal,
00067         Error,
00068         Warning,
00069         Retry,
00070         Info,
00071         Debug,
00072         Default,
00073         Skip,
00074         None
00075     };
00076
00089     enum class ErrType {
00090         InsufficientMemory,
00091         AccessDenied,
00092         SharingViolation,
00093         Io,
00094         NotFound,
00095         DiskFull,
00096         DeviceNotFound,
00097         Timeout,
00098         ResourceBusy,
00099         FileNotFound,
00100         InvalidFormat,
00101         InvalidInput,
00102         CorruptedData,
00103         OutOfRange,
00104         NetworkError,
00105         ConnectionRefused,
00106         HostUnreachable,
00107         ProtocolError,
00108         Unexpected,
00109         NotImplemented,
00110         PolicyUpdated,
00111         FunctionFailed,
00112         Default,
00113         None
00114     };
00115
```

```
00139     class Error final : public std::runtime_error {
00140     public:
00147         Error() : std::runtime_error(""), lvl(Level::None), errType(ErrType::None), retries(0) {}
00148 
00159         Error(Level l, ErrType err, unsigned int t, std::string msg)
00160             : std::runtime_error(std::move(msg)), lvl(l), errType(err), retries(t) {}
00161 
00166         ErrType type() const noexcept { return errType; }
00167 
00172         Level level() const noexcept { return lvl; }
00173 
00178         int attempts() const noexcept { return retries; }
00179 
00184         windowDesc whichWindow() const noexcept { return affectedWindow; }
00185 
00187         bool isFatal() const noexcept { return lvl == Level::Fatal; }
00188 
00190         bool isSkippable() const noexcept { return lvl == Level::Skip; }
00191 
00193         bool isWarning() const noexcept { return lvl == Level::Warning; }
00194 
00196         bool isError() const noexcept { return lvl == Level::Error; }
00197 
00199         bool isRetryable() const noexcept { return lvl == Level::Retry; }
00200 
00202         explicit operator bool() const noexcept { return lvl != Level::None; }
00203 
00204         #ifdef _WIN32
00205 
00223         static Error fromLastWinError(DWORD error, std::string_view context = "",
      std::optional<unsigned int> tries = std::nullopt);
00224 
00234         static Error fromLastWinError(std::string_view context = "", std::optional<unsigned int> tries
     = std::nullopt);
00235 
00236         #endif
00237 
00246         static Error fromSaveError(int e);
00247 
00248     private:
00249         Level lvl;
00250         ErrType errType;
00251         unsigned int retries;
00252         windowDesc affectedWindow;
00253     };
00254 
00263     template <typename T>
00264     using Result = std::expected<T, DeskUp::Error>;
00265 
00273     using Status = std::expected<void, DeskUp::Error>;
00274 }
00275 
00276 #endif
```

9.5 desk_up_error_gui_converter.h

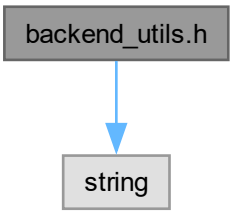
```
00001 
00026 
00027 #ifndef DESKUPERRORGUICONVERTER_H
00028 #define DESKUPERRORGUICONVERTER_H
00029 
00030 #include <QMessageBox>
00031 #include <QString>
00032 #include "desk_up_error.h"
00033 
00034 namespace DeskUp::UI {
00035 
00054     class ErrorAdapter {
00055     public:
00061         static int showError(const DeskUp::Error& err);
00062 
00063         static QString getUserMessage(const DeskUp::Error& err);
00064         static std::pair<QString, QMessageBox::Icon> mapLevel(Level lvl);
00065 
00066     };
00067 }
00068 
00069 #endif
```

9.6 mainWindow.h

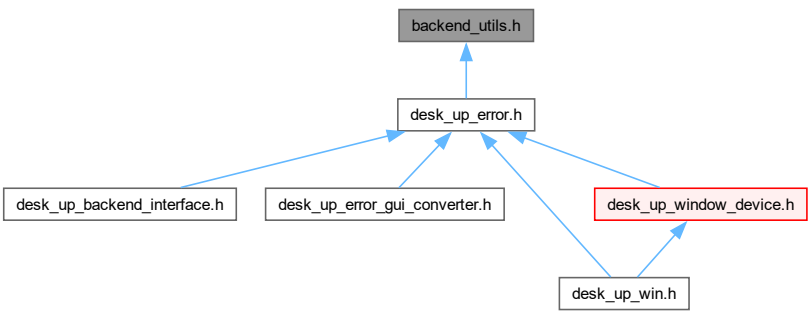
```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005
00006 class QAction;
00007
00008 class MainWindow : public QMainWindow
00009 {
00010     Q_OBJECT
00011
00012 public:
00013     explicit MainWindow(QWidget *parent = nullptr);
00014
00015 private slots:
00016     void onAddWorkspace();
00017     void onRestoreWorkspace();
00018     void onExit();
00019     void onAbout();
00020
00021 private:
00022     void setupMenus();
00023
00024     static void showSaveSuccessful();
00025     static void showRestoreSuccessful();
00026 };
00027
00028 #endif
```

9.7 backend_utils.h File Reference

Defines basic tools to easen the interaction with the Desk Up backend.
Include dependency graph for backend_utils.h:



This graph shows which files directly or indirectly include this file:



Functions

- std::string toLowerStr (const std::string &s)

  A function to convert a string to its lowercase version.
- std::string normalizePathLower (const std::string &p)

  A function to convert a generic path to a windows path using backslash path and later convert it to lowercase.
- std::wstring UTF8ToWide (const std::string &s)

  A function to convert a normal string into a wide one.

9.7.1 Detailed Description

Defines basic tools to easen the interaction with the Desk Up backend.
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

9.7.2 Function Documentation

9.7.2.1 normalizePathLower()

```
std::string normalizePathLower (
            const std::string & p)
```

A function to convert a generic path to a windows path using backslash path and later convert it to lowercase.
Note that it does not alter the original string.

Parameters

s	the string path to alter

Returns

`std::string` representing the altered path

Version

0.2.0

Date

2025

9.7.2.2 toLowerStr()

```
std::string toLowerStr (
            const std::string & s)
```

A function to convert a string to its lowercase version.
Note that it does not alter the original string

Parameters

s	the string to convert to lowercase

Returns

`std::string` representing the lowercase version

Version

0.2.0

Date

2025

9.7.2.3 UTF8ToWide()

```
std::wstring UTF8ToWide (
            const std::string & s)
```

A function to convert a normal string into a wide one.
Note that the original string doesn't get altered

Parameters

s	the string intended to convert

Returns

`std::wstring` representing the wide string

Version

0.2.0

Date

2025

9.8 backend_utils.h

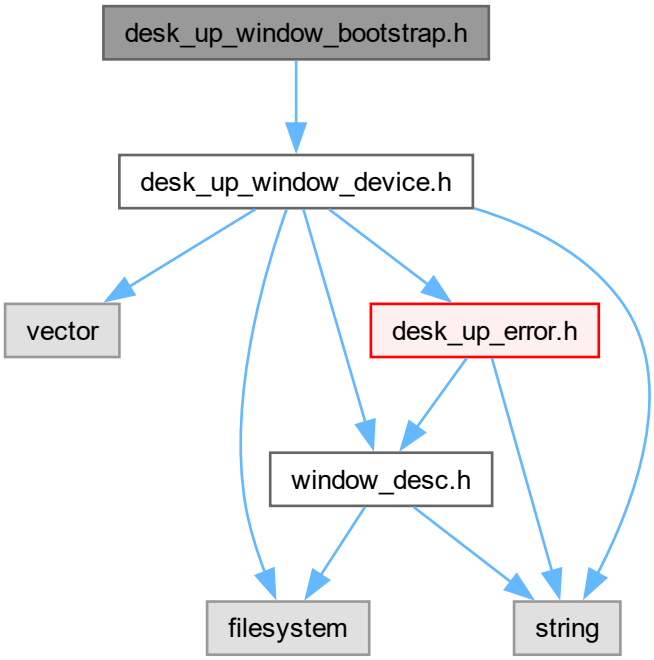
Go to the documentation of this file.

```
00001
00027
00028 #ifndef BACKENDUTILS_H
00029 #define BACKENDUTILS_H
00030
00031 #include <string>
00032
00033 #ifdef _WIN32
00034     #include <Windows.h>
00035
```

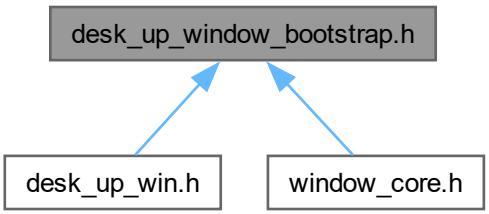
```
00044 std::string WideStringToUTF8(LPCWCH wideString);
00045
00055 std::string getSystemErrorMessageWindows(DWORD error, const std::string_view& contextMessage = "");
00056
00057 #endif
00058
00067 std::string toLowerStr(const std::string& s);
00068
00078 std::string normalizePathLower(const std::string& p);
00079
00088 std::wstring UTF8ToWide(const std::string& s);
00089
00090 #endif
```

9.9 desk_up_window_bootstrap.h File Reference

A struct wrapper used in the DeskUp backend.
Include dependency graph for desk_up_window_bootstrap.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct DeskUpWindowBootStrap

  This struct is a wrapper that holds a call to create a window device.

9.9.1 Detailed Description

A struct wrapper used in the DeskUp backend.
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

9.10 desk_up_window_bootstrap.h

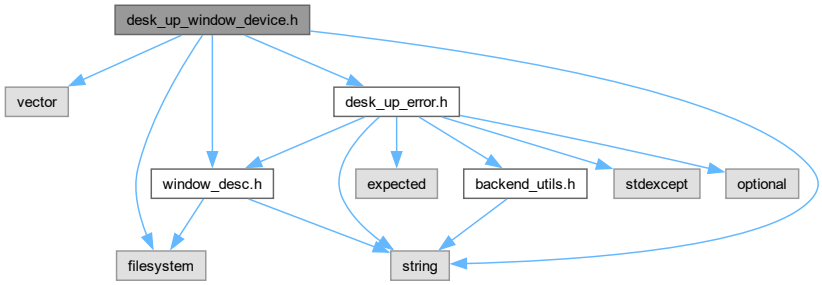
Go to the documentation of this file.

```
00001
00027
00028 #ifndef DESKUPWINDOWBOOSTRAP_H
00029 #define DESKUPWINDOWBOOSTRAP_H
00030
00031 #include "desk_up_window_device.h"
00032
00046 struct DeskUpWindowBootStrap{
00047
00057     const char * name;
00058
00067     DeskUpWindowDevice (*createDevice)();
```

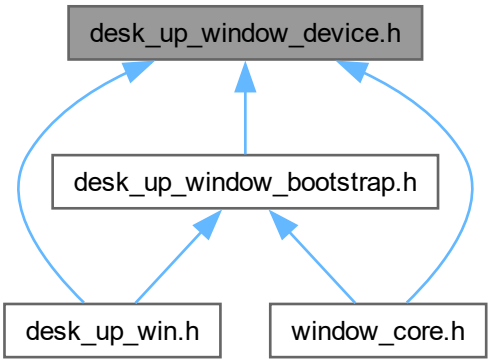
```
00068
00076     bool (*isAvailable)(void);
00077 };
00078
00079 #endif
```

9.11 desk_up_window_device.h File Reference

The struct descriptor of a window backend.
Include dependency graph for desk_up_window_device.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct DeskUpWindowDevice

  This abstract struct represents all the common calls that any backend must have.

9.11.1 Detailed Description

The struct descriptor of a window backend.
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

9.12 desk_up_window_device.h

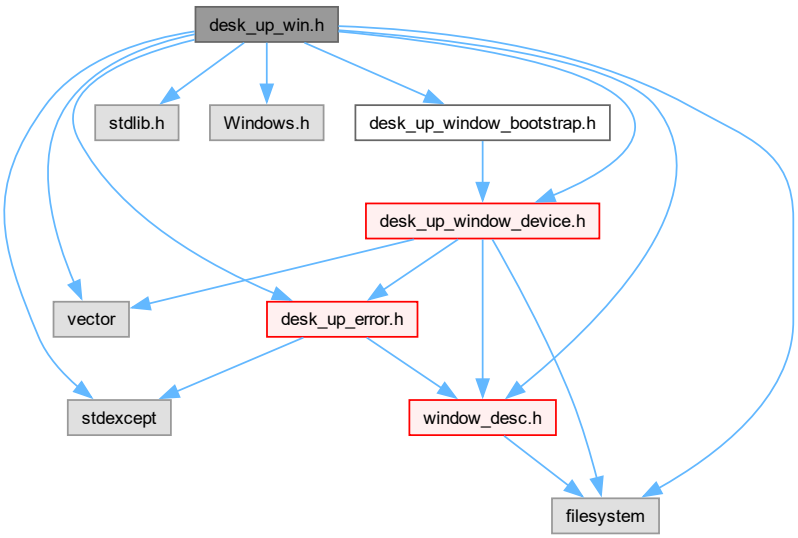
Go to the documentation of this file.

```
00001
00027
00028 #ifndef DESKUPWINDOWDEVICE_H
00029 #define DESKUPWINDOWDEVICE_H
00030
00031 #include <vector>
00032 #include <string>
00033 #include <filesystem>
00034
00035 #include "window_desc.h"
00036 #include "desk_up_error.h"
00037
00038 namespace fs = std::filesystem;
00039
00055 struct DeskUpWindowDevice{
00056
00065     DeskUp::Result<unsigned int> (*getWindowHeight)(DeskUpWindowDevice * _this);
00066
00075     DeskUp::Result<unsigned int> (*getWindowWidth)(DeskUpWindowDevice * _this);
00076
00085     DeskUp::Result<int> (*getWindowXPos)(DeskUpWindowDevice * _this);
00086
00095     DeskUp::Result<int> (*getWindowYPos)(DeskUpWindowDevice * _this);
00096
00105     DeskUp::Result<fs::path> (*getPathFromWindow)(DeskUpWindowDevice * _this);
00106
00117     DeskUp::Result<std::string> (*getDeskUpPath)(void);
00118
00128     DeskUp::Result<std::vector<windowDesc>> (*getAllOpenWindows)(DeskUpWindowDevice * _this);
00129
00139     DeskUp::Status (*loadWindowFromPath)(DeskUpWindowDevice * _this, const fs::path& path);
00140
00151     DeskUp::Result<windowDesc> (*recoverSavedWindow)(DeskUpWindowDevice * _this, const fs::path&
      filePath);
00152
00165     DeskUp::Status (*resizeWindow)(DeskUpWindowDevice * _this, const windowDesc window);
00166
00177     DeskUp::Result<unsigned int> (*closeProcessFromPath)(DeskUpWindowDevice * _this, const fs::path&
      path, bool allowForce);
00178
00191     void * internalData;
00192
00201     void (*DestroyDevice)(DeskUpWindowDevice * _this);
00202 };
00203
00204 #endif
```

9.13 desk_up_win.h File Reference

Bootstrap and functions for the Windows window backend (DeskUp).

Include dependency graph for desk_up_win.h:



Functions

- bool WIN_isAvailable () noexcept

  Returns whether the Windows backend is available.
- DeskUpWindowDevice WIN_CreateDevice () noexcept

  Creates a Windows `DeskUpWindowDevice`.
- void WIN_destroyDevice (DeskUpWindowDevice *_this) noexcept

  Deletes a Windows `DeskUpWindowDevice`.
- DeskUp::Result< std::string > WIN_getDeskUpPath () noexcept

  Returns the base DeskUp working path on the system.
- DeskUp::Result< int > WIN_getWindowXPos (DeskUpWindowDevice *_this) noexcept

  Gets the X position (top-left corner) of the active (client) window in the device.
- DeskUp::Result< int > WIN_getWindowYPos (DeskUpWindowDevice *_this) noexcept

  Gets the Y position (top-left corner) of the active (client) window in the device.
- DeskUp::Result< unsigned int > WIN_getWindowWidth (DeskUpWindowDevice *_this) noexcept

  Gets the width of the active (client) window in the device.
- DeskUp::Result< unsigned int > WIN_getWindowHeight (DeskUpWindowDevice *_this) noexcept

  Gets the height of the active (client) window in the device.
- DeskUp::Result< fs::path > WIN_getPathFromWindow (DeskUpWindowDevice *_this) noexcept

  Gets the absolute path of the executable that owns the active window.
- DeskUp::Result< std::vector< windowDesc > > WIN_getAllOpenWindows (DeskUpWindowDevice *_this) noexcept

  Enumerates all visible/non-minimized windows on the desktop.
- DeskUp::Result< windowDesc > WIN_recoverSavedWindow (DeskUpWindowDevice *_this, const fs::path &path) noexcept

  Loads a window description from a saved workspace file.
- DeskUp::Status WIN_loadProcessFromPath (DeskUpWindowDevice *_this, const fs::path &path) noexcept

  Creates a process from the specified path.
- DeskUp::Status WIN_resizeWindow (DeskUpWindowDevice *_this, const windowDesc window) noexcept

  Resizes a window according to the windowDesc parameter geometry.

- DeskUp::Result< unsigned int > WIN_closeProcessFromPath (DeskUpWindowDevice *, const fs::path &path, bool allowForce) noexcept

  This function closes all the instances associated with an executable, specified by the `path` parameter.
- void WIN_TEST_setHWND (DeskUpWindowDevice *_this, HWND hwnd)

  Test-only helper to set the internal HWND for the device.

Variables

- DeskUpWindowBootStrap winWindowDevice

  Windows backend bootstrap descriptor.

9.13.1 Detailed Description

Bootstrap and functions for the Windows window backend (DeskUp).
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

9.13.2 Function Documentation

9.13.2.1 WIN_closeProcessFromPath()

```
DeskUp::Result< unsigned int > WIN_closeProcessFromPath (
            DeskUpWindowDevice * ,
            const fs::path & path,
            bool allowForce)  [noexcept]
```

This function closes all the instances associated with an executable, specified by the `path` parameter.

Parameters

_this	The same device instance.
path	A `std::string` instance representing the path to check.
allowForce	Whether if the call should force the windows it finds to close.

Returns

`The` number of associated windows closed in the process. @errors

- Level::Fatal, ErrType::InvalidInput → Empty path.
- Level::Retry, ErrType::Os → Process enumeration or termination failure.

Version

0.2.0

Date

2025

9.13.2.2 WIN_CreateDevice()

```
DeskUpWindowDevice WIN_CreateDevice ()  [noexcept]
```

Creates a Windows DeskUpWindowDevice.
Wires the device function pointers to the Windows backend implementations and allocates the internal data required.

Returns

An initialized DeskUpWindowDevice.

Version

0.1.0

Date

2025

9.13.2.3 WIN_destroyDevice()

```
void WIN_destroyDevice (
            DeskUpWindowDevice * _this)  [noexcept]
```

Deletes a Windows DeskUpWindowDevice.
Deletion is done as a wrapper function in window_core.cc, done by DU_destroy via the pointer function

Version

0.3.2

Date

2025

9.13.2.4 WIN_getAllOpenWindows()

```
DeskUp::Result< std::vector< windowDesc > > WIN_getAllOpenWindows (
            DeskUpWindowDevice * _this)  [noexcept]
```

Enumerates all visible/non-minimized windows on the desktop.

Parameters

_this	The same device instance.

Returns

`std::vector<windowDesc>` with the abstract description of each window. @errors

- Level::Fatal, ErrType::Unexpected → Device or windowData became corrupt during enumeration (explicit check after EnumDesktopWindows fails and error.level() == Error).
- Level::Warning, ErrType::Unexpected → EnumDesktopWindows returned false for an unexpected reason not classified as fatal/error (catch-all for unanticipated callback failures).

Note

Indirect errors (via `WIN_CreateAndSaveWindowProc` callback): During enumeration, each window is processed via a callback that invokes `WIN_getPathFromWindow`, `WIN_getWindowXPos`, `WIN_getWindowYPos`, `WIN_getWindowWidth`, and `WIN_getWindowHeight`. The callback may generate:

- Level::Error, ErrType::InvalidInput → Callback parameters missing (improbable).
- Level::Error, ErrType::DeviceNotFound → Missing device/internal data while processing a window.
- Fatal errors from any geometry/path call → Bubble up immediately, aborting enumeration.
- Skip errors (e.g., invalid HWND mid-enumeration) → Individual window skipped, enumeration continues.
- Other Error-level issues → Tolerated once (static flag `levelErrorHappened`), fatal on second consecutive occurrence. Since the geometry and path functions each call `retryOp`, refer to their individual documentation (`WIN_getWindowXPos`, `WIN_getWindowYPos`, `WIN_getWindowWidth`, `WIN_getWindowHeight`, `WIN_getPathFromWindow`) for complete Windows error code mappings that may propagate through the callback.

Version

0.1.0

Date

2025

Here is the call graph for this function:

9.13.2.5 WIN_getDeskUpPath()

```
DeskUp::Result< std::string > WIN_getDeskUpPath ()  [noexcept]
```

Returns the base DeskUp working path on the system.
Points to the top-level directory where user workspaces are stored. The directory is ensured to exist (created if necessary).

Returns

`std::string` with the absolute path to the DeskUp top-level folder. @errors

- Level::Retry, ErrType::Io → Filesystem creation failed (non-fatal)

Version

0.1.0

Date

2025

9.13.2.6 WIN_getPathFromWindow()


```
DeskUp::Result< fs::path > WIN_getPathFromWindow (
            DeskUpWindowDevice * _this)  [noexcept]
```

Gets the absolute path of the executable that owns the active window.

Parameters

_this	The same device instance.

Returns

`fs::path` with the process image path on success. @errors

- Level::Error, ErrType::DeviceNotFound → Missing or invalid device/internal data (explicit check).
- Level::Skip, ErrType::InvalidInput → No valid HWND bound (explicit check).

Note

Indirect errors (via `retryOp`): This function internally retries three Windows API sequences: `GetWindowThreadProcessId`, `OpenProcess` + `GetExitCodeProcess`, and path resolution (e.g. `QueryFullProcessImageName`). Failures inside those lambdas are converted by `Error::fromLastWinError`. The most relevant Windows codes and their mapped DeskUp classifications for this routine are:

- `ERROR_INVALID_WINDOW_HANDLE` → Level::Skip, ErrType::ConnectionRefused (invalid/closed HWND when obtaining PID).
- `ERROR_ACCESS_DENIED` (transformed to `ERROR_ACCESS_DISABLED_BY_POLICY` before returning) → Level::Skip, ErrType::Default (policy or permission restriction when opening the process).
- `ERROR_NOT_ENOUGH_MEMORY` / `ERROR_OUTOFMEMORY` / `ERROR_TOO_MANY_OPEN_FILES` → Level::Fatal, ErrType::InsufficientMemory (system resource exhaustion during any queried step).
- `ERROR_SHARING_VIOLATION`, `ERROR_INVALID_PARAMETER`, `ERROR_INVALID_NAME`, `ERROR_FILENAME_EXCED_RANGE`, `ERROR_BAD_FORMAT` → Level::Skip, ErrType::InvalidInput (unexpected invalid handle/arguments surfaced by underlying APIs).
- `ERROR_FILE_NOT_FOUND`, `ERROR_PATH_NOT_FOUND` → Level::Skip, ErrType::InvalidInput (executable image could not be resolved).
- `ERROR_DISK_FULL` → Level::Fatal, ErrType::Io (unlikely here, but propagated if encountered reading image name).
- `ERROR_WRITE_FAULT` / `ERROR_READ_FAULT` / `ERROR_CRC` / `ERROR_IO_DEVICE` / `ERROR_FUNCTION_FAILED` → Level::Retry, ErrType::Io or ErrType::Unexpected (transient or unexpected low-level I/O issues; retries attempted before surfacing).
- Other unmapped codes → Level::Default, ErrType::Default. These are not emitted directly by `WIN_getPathFromWindow`; they are rethrown from `retryOp` after classification. Only the explicit pre-check errors are listed in the main @errors section above.

Version

0.1.0

Date

2025

9.13.2.7 WIN_getWindowHeight()

```
DeskUp::Result< unsigned int > WIN_getWindowHeight (
            DeskUpWindowDevice * _this)  [noexcept]
```

Gets the height of the active (client) window in the device.

Parameters

_this	The same device instance.

Returns

`unsigned int` with the window height. @errors

- Level::Error, ErrType::DeviceNotFound → Missing or invalid device/internal data (explicit check).
- Level::Skip, ErrType::InvalidInput → No valid HWND bound (explicit check).

Note

Indirect errors: Relies on `retryOp` for `GetWindowInfo`; propagated errors are produced by `retryOp` and not enumerated here.

Version

0.1.0

Date

2025

9.13.2.8 WIN_getWindowWidth()

```
DeskUp::Result< unsigned int > WIN_getWindowWidth (
            DeskUpWindowDevice * _this)  [noexcept]
```

Gets the width of the active (client) window in the device.

Parameters

_this	The same device instance.

Returns

`unsigned int` with the window width. @errors

- Level::Error, ErrType::DeviceNotFound → Missing or invalid device/internal data (explicit check).
- Level::Skip, ErrType::InvalidInput → No valid HWND bound (explicit check).

Note

Indirect errors: Additional system errors may be returned via `retryOp` wrapping `GetWindowInfo`; see its implementation and `Error::fromLastWinError` for classification.

Version

0.1.0

Date

2025

9.13.2.9 WIN_getWindowXPos()

```
DeskUp::Result< int > WIN_getWindowXPos (
            DeskUpWindowDevice * _this)  [noexcept]
```

Gets the X position (top-left corner) of the active (client) window in the device.

Parameters

_this	The same device instance.

Returns

`int` with the X coordinate of the window. @errors

- Level::Error, ErrType::DeviceNotFound → Missing or invalid device/internal data (explicit check).
- Level::Skip, ErrType::InvalidInput → No valid HWND bound (explicit check).

Note

Indirect errors: This function wraps `GetWindowInfo` with `retryOp`, which may propagate additional system-derived errors (fatal/skip/warning/retry) classified via `Error::fromLastWinError`. Those are not enumerated here because they are not produced directly by this function but rethrown from `retryOp`.

Version

0.1.0

Date

2025

9.13.2.10 WIN_getWindowYPos()

```
DeskUp::Result< int > WIN_getWindowYPos (
            DeskUpWindowDevice * _this)  [noexcept]
```

Gets the Y position (top-left corner) of the active (client) window in the device.

Parameters

_this	The same device instance.

Returns

`int` with the Y coordinate of the window. @errors

- Level::Error, ErrType::DeviceNotFound → Missing or invalid device/internal data (explicit check).
- Level::Skip, ErrType::InvalidInput → No valid HWND bound (explicit check).

Note

Indirect errors: Uses `retryOp` around `GetWindowInfo`; any Windows-origin errors are classified and rethrown by `retryOp` (see `Error::fromLastWinError`). They are not listed here.

Version

0.1.0

Date

2025

9.13.2.11 WIN_isAvailable()

```
bool WIN_isAvailable () [noexcept]
```

Returns whether the Windows backend is available.

Returns

`true` when compiled/executed on Windows, `false` otherwise.

Version

0.1.0

Date

2025

9.13.2.12 WIN_loadProcessFromPath()

```
DeskUp::Status WIN_loadProcessFromPath (
            DeskUpWindowDevice * _this,
            const fs::path & path) [noexcept]
```

Creates a process from the specified path.

Parameters

_this	The same device instance.
path	a literal representing the path to the executable linked with the program.

Returns

`DeskUp::Status` indicating success or failure. @errors

- Level::Fatal, ErrType::InvalidInput → Empty or invalid path/device.
- Level::Retry, ErrType::NotFound → Process started but main HWND not found.
- Level::Retry, ErrType::Os → ShellExecuteEx failed.

Version

0.2.0

Date

2025

9.13.2.13 WIN_recoverSavedWindow()

```
DeskUp::Result< windowDesc > WIN_recoverSavedWindow (
            DeskUpWindowDevice * _this,
            const fs::path & path) [noexcept]
```

Loads a window description from a saved workspace file.

Parameters

_this	The same device instance.
path	Path to the saved window description file.

Returns

`windowDesc` with geometry and executable path. @errors

- Level::Fatal, ErrType::InvalidInput → File missing or incomplete.
- Level::Retry, ErrType::Io → Filesystem or parse failure.

Version

0.2.0

Date

2025

9.13.2.14 WIN_resizeWindow()

```
DeskUp::Status WIN_resizeWindow (
            DeskUpWindowDevice * _this,
            const windowDesc window)  [noexcept]
```

Resizes a window according to the windowDesc parameter geometry.
Information about the window whose geometry is intended to modify must be specified inside the _↩ `this->internalData` parameter.

Parameters

_this	The same device instance.
window	a windowDesc instance whose geometry will be applied to resize the window.

Returns

`DeskUp::Status` indicating success or failure. @errors

- Level::Fatal, ErrType::InvalidInput → Invalid window device.
- Level::Retry, ErrType::NotFound → HWND not found.
- Level::Warning, ErrType::InvalidInput → Zero or negative width/height.
- Level::Retry, ErrType::Os → SetWindowPos failed.

Version

0.2.0

Date

2025

9.13.2.15 WIN_TEST_setHWND()

```
void WIN_TEST_setHWND (
            DeskUpWindowDevice * _this,
            HWND hwnd)
```

Test-only helper to set the internal HWND for the device.

Parameters

_this	The device instance.
hwnd	The HWND to assign.

Note

Only available when DESKUP_ENABLE_WIN32_TEST_HOOKS is defined.

9.13.3 Variable Documentation

9.13.3.1 winWindowDevice

`DeskUpWindowBootStrap winWindowDevice [extern]`

Windows backend bootstrap descriptor.

This global is used by `DU_Init` to determine whether Windows is available on the device and to create the corresponding `DeskUpWindowDevice`. After creation, all calls should go through the device instance (e.g., via `current_window_device` in `window_core.h` or equivalent).

See also

DeskUpWindowBootStrap

DeskUpWindowDevice

Version

0.1.0

Date

2025

9.14 desk_up_win.h

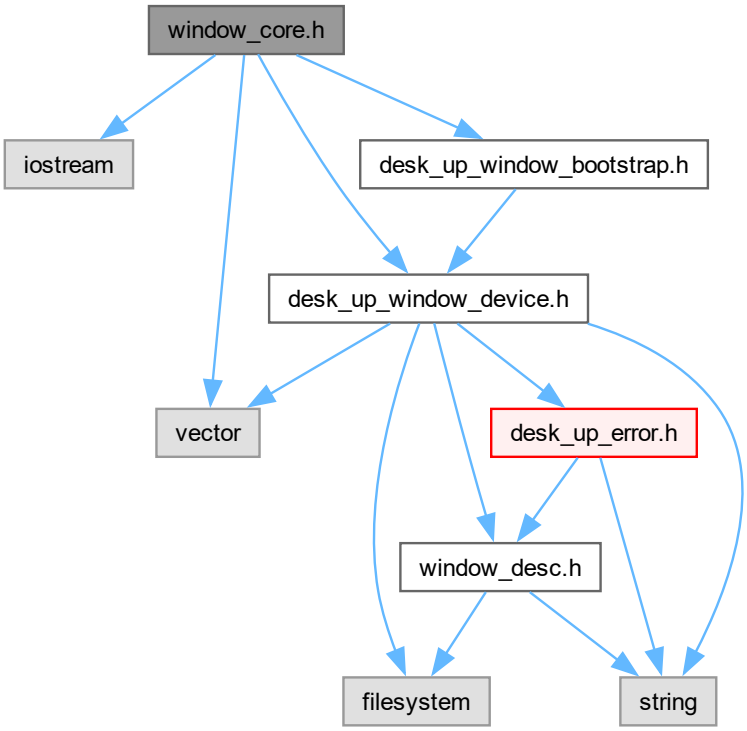
Go to the documentation of this file.

```
00001
00027
00028 #ifndef DESKUPWIN_H
00029 #define DESKUPWIN_H
00030
00031 #include <stdexcept>
00032 #include <vector>
00033 #include <filesystem>
00034
00035 #include <stdlib.h>
00036 #include <Windows.h>
00037
00038 #include "desk_up_window_bootstrap.h"
00039 #include "desk_up_window_device.h"
00040 #include "window_desc.h"
00041 #include "desk_up_error.h"
00042
00043 namespace fs = std::filesystem;
00044
00059 extern DeskUpWindowBootStrap winWindowDevice;
00060
00067 bool WIN_isAvailable() noexcept;
00068
00079 DeskUpWindowDevice WIN_CreateDevice() noexcept;
00080
00088 void WIN_destroyDevice(DeskUpWindowDevice* _this) noexcept;
00089
00102 DeskUp::Result<std::string> WIN_getDeskUpPath() noexcept;
00103
00116 DeskUp::Result<int> WIN_getWindowXPos(DeskUpWindowDevice * _this) noexcept;
00117
00130 DeskUp::Result<int> WIN_getWindowYPos(DeskUpWindowDevice * _this) noexcept;
00131
00144 DeskUp::Result<unsigned int> WIN_getWindowWidth(DeskUpWindowDevice * _this) noexcept;
00145
00158 DeskUp::Result<unsigned int> WIN_getWindowHeight(DeskUpWindowDevice * _this) noexcept;
```

```
00159
00181 DeskUp::Result<fs::path> WIN_getPathFromWindow(DeskUpWindowDevice * _this) noexcept;
00182
00201 DeskUp::Result<std::vector<windowDesc» WIN_getAllOpenWindows(DeskUpWindowDevice * _this) noexcept;
00202
00215 DeskUp::Result<windowDesc> WIN_recoverSavedWindow(DeskUpWindowDevice * _this, const fs::path& path)
      noexcept;
00216
00230 DeskUp::Status WIN_loadProcessFromPath(DeskUpWindowDevice * _this, const fs::path& path) noexcept;
00231
00248 DeskUp::Status WIN_resizeWindow(DeskUpWindowDevice * _this, const windowDesc window) noexcept;
00249
00263 DeskUp::Result<unsigned int> WIN_closeProcessFromPath(DeskUpWindowDevice*, const fs::path& path, bool
      allowForce) noexcept;
00264
00271 void WIN_TEST_setHWND(DeskUpWindowDevice* _this, HWND hwnd);
00272
00273 #endif
```

9.15 window_core.h File Reference

Declares basic code to interact with Desk Up.
Include dependency graph for window_core.h:



Functions

- int DU_Init ()

  Initializes the DeskUp backend system.

- void DU_Destroy ()

  Destroys and cleans up the DeskUp backend resources.

Variables

- std::string DESKUPDIR
- std::unique_ptr< DeskUpWindowDevice > current_window_backend

9.15.1 Detailed Description

Declares basic code to interact with Desk Up.
This file is part of DeskUp

Author

Nicolas Serrano Garcia serranogarcianicolas@gmail.com

Date

2025

Copyright

9.15.2 Function Documentation

9.15.2.1 DU_Destroy()

```
void DU_Destroy ()
```

Destroys and cleans up the DeskUp backend resources.
Releases the backend device and clears global state. Should be called when DeskUp is no longer needed to free resources. After calling this function, DU_Init() must be called again before using any DeskUp functionality.

Note

Safe to call even if DU_Init() was not called or failed.

See also

DU_Init()

Version

0.1.1

Date

2025

9.15.2.2 DU_Init()

```
int DU_Init ()
```

Initializes the DeskUp backend system.
This function must be called once before using any DeskUp backend feature. It iterates through the available backends (currently only the Windows backend) and:

- Calls the backend bootstrap function `isAvailable()` to check if it can be used.
- If available, calls `createDevice()` to create and configure the backend device.
- Calls `getDeskUpPath()` through the device to determine the workspace base directory.

Once initialization completes successfully:

- The global variable DESKUPDIR_anchor contains the DeskUp workspace path.
- The global pointer current_window_backend_anchor references the active backend device.

Note

The function currently supports only the Windows backend, which internally maps to:

- WIN_isAvailable()
- WIN_CreateDevice()
- WIN_getDeskUpPath()

Returns

1 if initialization succeeds, 0 otherwise.

See also

DeskUpWindowDevice
DeskUpWindowBootStrap
WIN_isAvailable()
WIN_CreateDevice()
WIN_getDeskUpPath()

Version

0.1.1

Date

2025

9.15.3 Variable Documentation

9.15.3.1 current_window_backend

```
std::unique_ptr<DeskUpWindowDevice> current_window_backend  [extern]
```

A unique pointer to the selected backend device for DeskUp.
This global pointer gets assigned when calling DU_Init(), so using it without initializing DeskUp will cause undefined behaviour. It provides access to backend-specific functions such as window enumeration, size, and position.

See also

DU_Init()
DeskUpWindowBootStrap

Version

0.1.0

Date

2025

9.15.3.2 DESKUPDIR

```
std::string DESKUPDIR  [extern]
```

The path to the DeskUp saved workspace.
This global variable gets assigned when calling DU_Init(), so using it without initializing DeskUp will cause undefined behaviour. In the process, it calls the device's getDeskUpPath() function, which is resolved to the specific backend implementation (e.g. WIN_getDeskUpPath on Windows).

See also

DU_Init()

DeskUpWindowDevice::getDeskUpPath

Version

0.1.0

Date

2025

9.16 window_core.h

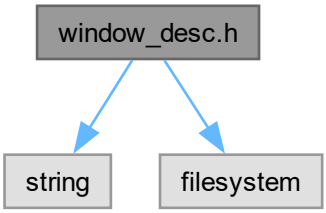
Go to the documentation of this file.

```
00001
00027
00028 #ifndef WINDOWGLOBAL_H
00029 #define WINDOWGLOBAL_H
00030
00031 #include <iostream>
00032 #include <vector>
00033 #include "desk_up_window_device.h"
00034 #include "desk_up_window_bootstrap.h"
00035
00050 extern std::string DESKUPDIR;
00051
00065 extern std::unique_ptr<DeskUpWindowDevice> current_window_backend;
00066
00095 int DU_Init();
00096
00112 void DU_Destroy();
00113
00114 #endif
```

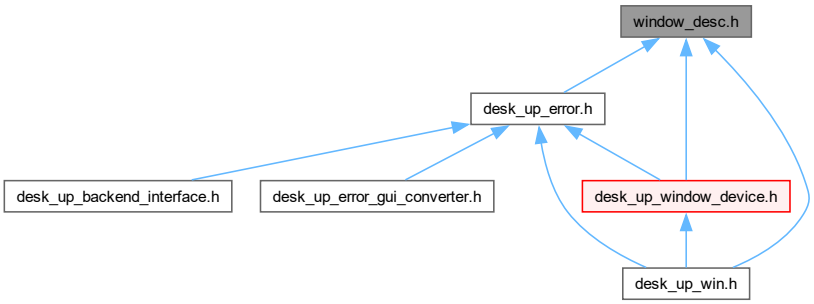
9.17 window_desc.h File Reference

Defines the abstract data structure used to represent a window in DeskUp.

Include dependency graph for window_desc.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct windowDesc

  Describes a single window instance in the DeskUp system.

Enumerations

- enum SaveErrorCode {
  SAVE_SUCCESS = 1 , ERR_EMPTY_PATH = -1 , ERR_FILE_NOT_OPEN = -2 , ERR_NO_PERMISSION = -3 ,
  ERR_FILE_NOT_FOUND = -4 , ERR_DISK_FULL = -5 , ERR_UNKNOWN = -6 }

  Enumerates possible results and error codes for `windowDesc::saveTo()`.

9.17.1 Detailed Description

Defines the abstract data structure used to represent a window in DeskUp.
This file is part of DeskUp
The `windowDesc` structure stores all the necessary information about a window so that DeskUp can identify and restore its original state (position, size, and associated executable). It acts as a generic abstraction that can be populated by any backend (e.g., Windows, Linux, macOS).
Each backend fills in the fields using its native APIs (for example, `WIN_getAllOpenWindows()` in the Windows implementation).

See also

windowDesc::saveTo()

WIN_getAllOpenWindows()

Version

0.1.0

Date

2025

Copyright

9.17.2 Enumeration Type Documentation

9.17.2.1 SaveErrorCode

`enum SaveErrorCode`

Enumerates possible results and error codes for `windowDesc::saveTo()`.
This enumeration defines all return codes used by `windowDesc::saveTo()` to indicate whether the save operation succeeded or failed, and if failed, what type of error occurred.
Positive values indicate success; negative values indicate different failure modes, allowing callers (and the backend) to interpret the cause precisely and propagate appropriate `DeskUp::Error` levels.

See also

windowDesc::saveTo()

Version

0.2.1

Date

2025

Enumerator

SAVE_SUCCESS	File successfully written. Indicates that the window description was saved without issues.
ERR_EMPTY_PATH	The provided file path is empty. Returned when `saveTo()` receives an empty path parameter.
ERR_FILE_NOT_OPEN	The target file could not be opened. The file may not exist or may be locked by another process.

ERR_NO_PERMISSION	Insufficient permissions to write the file. The current process lacks write access to the target directory.
ERR_FILE_NOT_FOUND	File not found or directory path invalid. Returned when the parent path of the target file does not exist.
ERR_DISK_FULL	Disk or storage device is full. The save operation could not complete because the disk ran out of space.
ERR_UNKNOWN	Unknown or unexpected error during save. A generic failure when no specific condition above matches.

9.18 window_desc.h

Go to the documentation of this file.

```
00001
00036
00037 #ifndef WINDOWDESC_H
00038 #define WINDOWDESC_H
00039
00040 #include <string>
00041 #include <filesystem>
00042
00043 namespace fs = std::filesystem;
00044
00060 struct windowDesc {
00061
00065     windowDesc();
00066
00076     windowDesc(std::string n, int xPos, int yPos, int width, int height, std::string p) : name(n),
      x(xPos), y(yPos), w(width), h(height), pathToExec(fs::path(p)) {}
00077
00078     // /**
00079     //  * @brief Constructs a window descriptor with explicit name, geometry, and executable path.
00080     //  * @param n window name.
00081     //  * @param x X coordinate (top-left) in pixels.
00082     //  * @param y Y coordinate (top-left) in pixels.
00083     //  * @param w Width in pixels.
00084     //  * @param h Height in pixels.
00085     //  * @param p Absolute path to the owning executable.
00086     //  */
00087     // windowDesc(std::string n, int xPos, int yPos, int width, int height, fs::path p) : name(n),
      x(xPos), y(yPos), w(width), h(height), pathToExec(p) {}
00088
00093     std::string name;
00094
00098     int x;
00099
00103     int y;
00104
00108     int w;
00109
00113     int h;
00114
00118     fs::path pathToExec;
00119
00157     int saveTo(fs::path path);
00158
00168     bool operator!() const {
00169         return !x && !y && !w && !h;
00170     }
00171
00172 };
00173
00191 enum SaveErrorCode {
00196     SAVE_SUCCESS = 1,
00197
00202     ERR_EMPTY_PATH = -1,
00203
00208     ERR_FILE_NOT_OPEN = -2,
00209
00214     ERR_NO_PERMISSION = -3,
00215
00220     ERR_FILE_NOT_FOUND = -4,
00221
00226     ERR_DISK_FULL = -5,
00227
00232     ERR_UNKNOWN = -6
00233 };
00234
00235 #endif
```