Simple file dialog C++ library

v1.0.4

Table of contents

Overview

SimpleFileDialog C++ library provides simple dialog to chose file in Windows and Linux (tested for Ubuntu 22.04, 22.10 and Windows 11). The library used in projects when simple file chose dialog needed. To provide dialog in Linux the library calls "**zenity**" application. The library uses C++17 standard and doesn't have third-party dependencies. **SimpleFileDialog.h** file includes declaration of **SimpleFileDialog** class. **SimpleFileDialog** class includes only one simple static method **dialod()**. The library is licensed under the **Apache 2.0** license.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	20.07.2023	First version.
1.0.2	02.08.2023	- Fixed std::string compiling error for Linux.
1.0.3	20.03.2024	- Documentation updated.
1.0.4	17.05.2024	- Documentation updated.

Library files

The library is supplied only by source code. The user is given a set of files in the form of a CMake project (repository). The repository structure is shown below:

```
CMakeLists.txt -------------------- Main CMake file of the library.
src ------------------------------- Folder with library source code.
    SimpleFileDialog.cpp ---------- C++ implementation file.
    SimpleFileDialog.h ------------ Library main header file.
    SimpleFileDialogVersion.h ----- Header file with library version.
    SimpleFileDialogVersion.h.in -- Service CMake file to generate version file.
test ------------------------------ Folder of the test application.
    CMakeLists.txt ---------------- CMake file of the test application.
    main.cpp ---------------------- Source C++ file of the test application.
```

Class declaration

SimpleDileDialog class declared in **SimpleFileDialog.h** file. Class declaration:

```cpp
/// @brief File dialog class.
class SimpleFileDialog
{
public:
    /// @brief Dialog function.
    static std::string dialog();
};
```

SimpleFileDialog class includes only one static method **dialog()** which shows file chose dialog to user. The **dialog()** method returns string of file name. If file not chosen the method will return empty string **""**. Method used without **SimpleFileDialog** class instance. Example:

```cpp
#include <iostream>
#include "SimpleFileDialog.h"

int main(void)
{
    // Open file dialog.
    std::string file = cr::utils::SimpleFileDialog::dialog();
    std::cout << "File: " << file << std::endl;

    return -1;
}
```

Build and connect to your project

Typical commands to build **SimpleFileDialog** library:

```
git clone https://github.com/ConstantRobotics-Ltd/SimpleFileDialog.git
cd SimpleFileDialog
mkdir build
cd build
cmake ..
make
```

If you want connect **SimpleFileDialog** library to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **SimpleFileDialog** as submodule by commands:

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/SimpleFileDialog.git
3rdparty/SimpleFileDialog
```

In you repository folder will be created folder **3rdparty/SimpleFileDialog** which contains all library files. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    SimpleFileDialog
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

################################################################################
## 3RD-PARTY
## dependencies for the project
################################################################################
project(3rdparty LANGUAGES CXX)

################################################################################
## SETTINGS
## basic 3rd-party settings before use
```

```
################################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

################################################################################
## CONFIGURATION
## 3rd-party submodules configuration
################################################################################
SET(${PARENT}_SUBMODULE_SIMPLE_FILE_DIALOG              ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_SIMPLE_FILE_DIALOG)
    SET(${PARENT}_SIMPLE_FILE_DIALOG                    ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_SIMPLE_FILE_DIALOG_TEST               OFF CACHE BOOL "" FORCE)
endif()

################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_SIMPLE_FILE_DIALOG)
    add_subdirectory(SimpleFileDialog)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **SimpleFileDialog** to your project and excludes test application (SimpleFileDialog class test applications) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    SimpleFileDialog
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include SimpleFileDialog library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} SimpleFileDialog)
```

Done!