child process

ChildProcess C++ library

v1.0.0

Table of contents

Overview

ChildProcess C++ library provides running additional program under new process. The library runs child process and returns control from you. The library doesn't show child process output, just runs.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	16.08.2023	First version

ChildProcess class description

ChildProcess class declaration

ChildProcess class declared in **ChildProcess.h** file. Class declaration:

```
class ChildProcess
{
public:
    /**
     * @brief Get library version.
     * @return String of current library version: Major.Minor.Patch.
     */
    static std::string getVersion();
    /**
     * @brief Class constructor.
     */
    ChildProcess();
    /**
     * @brief Class destructor.
     */
    ~ChildProcess();
    /**
     * @brief Create a child process and run given command with provided arguments.
     * @param command Command to run.
     * @param params Arguments of command.
     * @return TRUE if command run or FALSE if not.
     */
    bool run(std::string command, std::string params = "");
    /**
     * @brief Close device.
     */
    void close();
};
```

getVersion method

getVersion() method returns string of current version of **ChildProcess** class. Method declaration:

```
static std::string getVersion();
```

Method can be used without **ChildProcess** class instance:

```
cout << "ChildProcess class version: " << ChildProcess::getVersion() << endl;
```

Console output:

```
ChildProcess class version: 1.0.0
```

run method

run() method runs given command with provided arguments under new process.

```
bool run(std::string command, std::string params = "");
```

Parameter	Value
command	Command/program to run (for example: mkdir or myProgram etc).
params	Optional arguments for command (for example: -l, newDirectory or myProgramInput.txt etc).

Returns: TRUE if a new process is started or FALSE if not.

close method

close() method kills created child process. Method declaration:

```
void close();
```

Build and connect to your project

Typical commands to build **ChildProcess** library:

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

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

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

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

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

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

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

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(${PARENT}_SUBMODULE_CHILD_PROCESS                   ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_CHILD_PROCESS)
    SET(${PARENT}_CHILD_PROCESS                         ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_CHILD_PROCESS_TEST                    OFF CACHE BOOL "" FORCE)
endif()

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

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

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

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 ChildProcess library in your **src/CMakeLists.txt** file:

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

Done!