C++ based Automatic Mouse Clicker Documentation

Introduction

Welcome to Mini Mouse Manager (MMM), an automated mouse clicker software developed using Qt 6.6.1 and C++, built with CMake. MMM is a tool created as a learning project, aimed at streamlining click automation on the Windows platform.

About Mini Mouse Manager

Mini Mouse Manager (MMM) is a user-friendly automatic clicker designed to automate clicking actions precisely and flexibly. Its intuitive interface and customizable settings make automating clicks effortless for various tasks.

Key Features:

- **Control and Flexibility**: Users have precise control over clicking behavior through configurable settings. Utilizing the Windows.h library enhances control and direct interaction with Windows functionalities.
- **Intuitive Interface**: MMM boasts a user-friendly interface, simplifying configuration processes for a seamless user experience.
- **Platform Compatibility**: Tailored specifically for Windows, MMM utilizes the Windows.h library, ensuring consistent and reliable performance.
- **Enhanced Functionality**: MMM enables various clicking actions, including auto-clicking at the cursor's location, setting and clicking at specified coordinates, defining user-specific clicking areas, and configuring fixed or randomized time intervals between clicks.

Purpose of Documentation:

This documentation aims to comprehensively guide users in understanding and effectively utilizing Mini Mouse Manager (MMM). From installation instructions to detailed feature descriptions, this document intends to equip users with the necessary information to efficiently leverage MMM's functionalities on the Windows platform.

Note: Mini Mouse Manager (MMM) is developed under the LGPLv3 license, adhering to open-source principles, fostering a collaborative environment for further improvements and enhancements.

Getting Started with IDE

For the project to function properly, user must have either Qt Creator or Visual Studio IDE 2019 (or newer) installed on their system. These Integrated Development Environments (IDEs) provide the necessary tools and environment to compile, run, and work with the Qt_FileExplorer application seamlessly.

Prerequisites

QT

You will need Qt version 6.6.1 or newer, along with the Developer and Designer Tools. If you're unsure whether these tools are installed or if they are not installed, open the Qt Maintenance Tool by navigating to:

"Tools > Qt Maintenance Tool > Start Maintenance Tool" (select the necessary options in the 'Select Components' tab)

Visual Studio

If you prefer using the Visual Studio IDE, ensure that you have the necessary components installed. Access the Visual Studio Installer, modify your Visual Studio IDE, and search for the 'C++ CMake tools for Windows' in the 'Individual Components' tab. Install it if needed. Alternatively, you can install the C++ Windows package, ensuring to select CMake if you haven't installed C++ yet.

Getting project to run on Visual Studio

1. Navigate "File > Clone Repository..." and select folder for repository https://github.com/Magic146W/AutomaticClicker.git
2. Setting Up CMake for Visual Studio:
    a. Open Visual Studio and go to “File > Open > CMake…” and select CMakeLists.txt inside your working directory
    b. If you don’t find the CMake option, please go through prerequisites for VS

c. The first time you open this option, an overview of CMake in Visual Studio will appear. Once done, repeat the process of opening CMake from "File > Open" and select your working directory, then choose CMakeLists.txt.

d. Wait for the project to build.

3. To run the project, either click the green arrow with '*Qt Application*' written next to it or press F5.

Getting project to run on Qt Creator

1. Clone the repository to your local machine using cmd:

```
git clone https://github.com/Magic146W/AutomaticClicker.git
```

2. Open the project by clicking on *'CMakeLists.txt*' inside of the cloned directory.
3. When opening the project for the first time or if not previously configured, select the kit '*Desktop Qt 6.6.1 MSVC2019 64bit*' or a newer version and click on '*Configure Project*.'
4. Wait for the initial project setup and build to finish.
5. Run the project by clicking the green arrow or using the shortcut Ctrl+R.

Getting Started with executable file

Installation and Setup:

- Clone the repository, including the *'Executable'* directory, or download and extract the *'Executable'* directory from the repository.

Running the Application:

- Launch the *'Mini_Mouse_Manager'* executable file located within the *'Executable'* directory.
- Alternatively, run the application from your Integrated Development Environment (IDE) following the setup/installation instructions provided in this document's section **Getting Started with IDE**

Basic Functionalities:

Interface Overview:

- Upon application launch, a user-friendly interface featuring radio buttons and text fields will be displayed.

Configuration:

- The basic configuration ensures user safety by permitting only numeric input in text fields. Specific range validators, like 0-59 for seconds or 0-999 for milliseconds, are implemented as needed. Modifying settings is straightforward; users can easily adjust settings by clicking on the circular field next to the desired setting.
- Additionally, users can save their layout and inputted settings by utilizing the 'Save' or 'Load' options. 'Load' will function only if the user's preferences were previously saved.

Starting the Program:

- Initiate the automated clicking process by clicking the "START(...)" button.
- Alternatively, use the specified hotkey indicated in the text on the "START(F6)" button. Users have the ability to change the hotkey as needed.

Customization:

- Select various clicking actions using the provided radio buttons.
- Specify fixed or random time intervals using the described text fields. Utilize special time input fields for random time intervals.
- Access the hotkey dialog using the “HOTKEY” button to modify the start/stop keyboard signal.
- Use the “SET LOCATION” button to set mouse cursor coordinates or input area size in pixels.

Initiating Actions:

- After configuring settings, initiate the automated clicking process by clicking "Start" or using the specified hotkey.

Hotkey Usage:

- Employ the specified hotkey for quick access and action initiation. This method is especially useful for swiftly halting the program, particularly when the mouse is clicking every 10 milliseconds.

Caution: The hotkey can be triggered even when the software is minimized and another window is selected due to the software's use of the hook system from the windows.h library.

Code Documentation

Class Hierarchy

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

QDialog

- ChangeHotkeyDialog

QMainWindow

- MainWindow

QObject

- GlobalMouseHook
- HookWorker
- InputManager
- MouseManager
- WindowsHookManager

MouseManager::ThreadData

Class List

MainWindow.cpp

Responsible for the user interface (UI).

Manages user interactions, GUI elements, and displays information to the user.

InputManager.cpp

Acts as the central control hub, managing connections and crucial data exchanges among MainWindow and the rest of the classes.

Orchestrates data flow between the MainWindow.cpp and other classes.

ChangeHotkeyDialog.cpp

Handles the secondary window where users can select a new hotkey.

Manages user input for modifying the hotkey.

WindowsHookManager.cpp

Coordinates and manages all hook-related classes, including keyboard and mouse hooks.

Serves as a centralized manager for hook-based functionalities.

GlobalMouseHook.cpp

Monitors the mouse hook,tracking mouse click actions.

HookWorker.cpp

Listens for keyboard hook events, intercepting user input at the Windows layer.

Responsible for managing keyboard-related interactions.

MouseManager.cpp

Executes mouse actions based on user instructions received from the InputManager.cpp.

Responsible for simulating mouse clicks and movements.

Interaction Flow

The primary interaction occurs between MainWindow.cpp and InputManager.cpp, with InputManager.cpp serving as a mediator between the main window and other classes. Each class has a specific role:

- Managing User Input: ChangeHotkeyDialog.cpp
- Overseeing Hook-Based Functionalities: WindowsHookManager.cpp
- Working with Mouse and Keyboard Hook Events: GlobalMouseHook.cpp, HookWorker.cpp
- Executing Mouse Actions: MouseManager.cpp

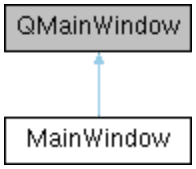
This architectural design provides a structured understanding of how the primary and secondary classes collaborate, defining their respective responsibilities within the software.

Class Documentation

*Made with doxygen

MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Slots

```
void mouseLocationUpdate (int x, int y)
```

Updates the displayed mouse cursor coordinates in the UI.

```
void updateButtonsText (const QString &newHotkey)
```

Updates the text of start and stop buttons with a new hotkey string.

```
void updateButtonsText (const QString &newHotkey)
```

Initiates the application by loading selected radio buttons, retrieving time and repetition settings, and emitting a signal to update the application run process accordingly.

```
void blockUIElements (bool isBlocked)
```

Enables or disables multiple UI elements based on the given boolean.

Signals

```
void startCursorPositionGrab ()
```

Initiates the process to grab the cursor's position.

```
void updateApplicationRunProcess (int fixedTime, int randomTime, const int
&repeatTimes, const QPoint &location, const int &area)
```

Updates the application's running process with provided parameters.

```
void stopApplication (bool stop)
```

Stops the application based on the provided flag.

Public Member Functions

```
MainWindow (QWidget *parent=nullptr)
```

This class controls the main window's user interface elements and handles communication between user inputs and the program.

Private Slots

```
void on_PushButton_SetLocation_clicked ()
```

Triggers the capturing of the current cursor position.

```
void on_PushButton_SetLocation_clicked ()
```

*Opens the **ChangeHotkeyDialog** to allow users to modify the hotkey settings.*

```
void on_PushButton_Start_clicked ()
```

*Handles the click event of the 'Start' button by updating the process state via **InputManager**.*

```
void on_PushButton_Stop_clicked ()
```

Handles the click event of the 'Stop' button by emitting a signal to stop the application.

```
void on_PushButton_Save_clicked ()
```

Saves user settings and interface data.

```
void on_PushButton_Load_clicked ()
```

Loads previously saved settings and updates UI elements accordingly.

```
void updateInputManager (QAbstractButton *button)
```

*Updates the **InputManager** based on the state change of a provided button in various button groups.*

```
void validateTimeRange ()
```

Validates and adjusts the time range input in TimeEdit widgets.

Private Member Functions

```
bool eventFilter (QObject *watched, QEvent *event)
```

Filters and handles specific events for LineEdit and TimeEdit widgets.

```
void radioButtonGroupSetUp ()
```

Sets up radio button groups for specific functionalities within the main window.

```
template<typename... Args> void setUpButtonGroup (const QString
buttonGroupName, QButtonGroup *buttonGroup, Args... buttons)
```

Sets up a button group with specified buttons and connects signals for event handling.

```
void handleButtonUpdate (QAbstractButton *button, const QString &option,
    QButtonGroup *buttonGroup)
```

*Handles the update of a specific button's state within a button group and communicates the change to the **InputManager**.*

```
void connectValidatorForLineEdit ()
```

Connects validators to LineEdit widgets for input validation.

```
void loadSelectedRadioButtons ()
```

Loads the previously selected radio buttons' state and handles any unchecked buttons, making sure no null data is send further.

```
int getFixedTime ()
```

Retrieves the fixed time input from LineEdit widgets and returns the time in milliseconds.

Private Attributes

```
Ui::MainWindow * ui
QButtonGroup clickingIntervalData
QButtonGroup pressTypeData
QButtonGroup repeatClickData
QButtonGroup locationOptionsData
QList< QButtonGroup * > allButtonGroups
QVector< QLineEdit * > lineEditList
```

MainWindow Class Documentation

This class controls the main window's user interface elements and handles communication between user inputs and the program. As well as oversees the graphical user interface (GUI) elements within the main window and orchestrates the communication between user inputs, such as events or actions triggered by the user, and the underlying program logic.

The MainWindow class is responsible for:

Managing the layout and appearance of the main window's graphical user interface (GUI).

Handling user interactions and events within the main window.

Acting as a central hub for communication between different parts of the program.

Initiating and coordinating actions based on user inputs or system events.

Note

This class plays a crucial role in facilitating the program's UI and overall functionality.

MainWindow Member Function Documentation

```
bool MainWindow::eventFilter (QObject *watched, QEvent *   event)[private]
```

Filters and handles specific events for LineEdit and TimeEdit widgets.

Overrides the eventFilter method to filter events and handle focus-out events for LineEdit and TimeEdit widgets.

Parameters

watched	The watched object receiving the event.
event	The event being processed.

Returns

Returns true if the event was handled, otherwise returns false.

```
int MainWindow::getFixedTime ()[private]
```

Retrieves the fixed time input from LineEdit widgets and returns the time in milliseconds.

Retrieves 'Minutes', 'Seconds', and 'Milliseconds' inputs from LineEdit widgets and constructs a QTime object in 'mm:ss:zzz' format. If the resulting time is less than 10 milliseconds, adjusts 'Milliseconds' to 10.

Returns

An integer representing the fixed time in milliseconds.

```
void MainWindow::handleButtonUpdate (QAbstractButton *button, const QString &option, QButtonGroup * buttonGroup)[private]
```

Handles the update of a specific button's state within a button group and communicates the change to the **InputManager**.

This method manages the state change of the provided 'button' within the given 'buttonGroup'. It updates the corresponding 'option' in the **InputManager** with the text of the provided 'button'.

Parameters

button	The QAbstractButton whose state is being updated.
option	The option identifier related to the button's state change in the **InputManager**.
buttonGroup	The QButtonGroup to which the button belongs.

```
void MainWindow::mouseLocationUpdate (int x, int y)[slot]
```

Updates the displayed mouse cursor coordinates in the UI.

Parameters

x	The x-coordinate of the mouse cursor.
y	The y-coordinate of the mouse cursor.

```
template<typename... Args> void MainWindow::setUpButtonGroup (const QString buttonGroupName, QButtonGroup *buttonGroup, Args...  buttons)[private]
```

Sets up a button group with specified buttons and connects signals for event handling.

Template Parameters

Args	Variadic template for QAbstractButton pointers representing buttons.

Parameters

buttonGroupName	The name assigned to the button group.
buttonGroup	Pointer to the QButtonGroup.

buttons...	Variadic arguments of QAbstractButton pointers representing buttons to be added.

```
void MainWindow::updateButtonsText (const QString &newHotkey)[slot]
```

Updates the text of start and stop buttons with a new hotkey string.

Parameters

newHotkey	The new hotkey string to be displayed in button texts.

```
void MainWindow::updateInputManager (QAbstractButton *button)[private], [slot]
```

Updates the **InputManager** based on the state change of a provided button in various button groups.

Parameters

button	The QAbstractButton whose state is being updated.

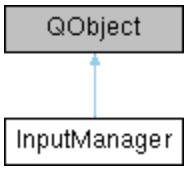
```
void MainWindow::validateTimeRange ()[private], [slot]
```

Validates and adjusts the time range input in TimeEdit widgets.

Adjusts 'tillTime' to ensure a minimum of 10 milliseconds if below the threshold. Maintains a valid time range by adjusting 'TimeEdit_Till' if 'tillTime' is smaller than 'fromTime'.

InputManager Class Reference

Inheritance diagram for InputManager:



Public Slots

```
void updateProcessState (bool stop)
```

Updates the state of the process based on the given boolean value.

```
void updateUserHotkey (const QString pressedKeys)
```

Updates and sets the user-defined hotkey, emitting a signal upon change.

```
void updateUserCursorLocationSet (int x, int y)
```

Emits a signal to return the updated cursor position.

```
void updateUserData (int fixedTime, int randomTime, const int &repeatTimes,
const QPoint &location, const int &area)
```

Updates user data based on the provided parameters, then triggers the application start accordingly.

Signals

```
void hotkeyChangePassed (QString newHotkey)
```

Signals that a new hotkey has been set.

```
void getCursorPosition ()
```

Signals the need to retrieve the cursor's position.

```
void returnCursorPosition (int x, int y)
```

Signals the cursor's position.

```
void startApplication (const int &clickTime, const int &timeBetweenClicks,
const QChar &type, const int &repetitions, const QChar &location, const
QPoint &xy, const int &area)
```

Signals the start of an application run based on provided parameters.

```
void stopApplication ()
```

Signals to stop the application process.

```
void initializeStartProcess ()
```

Signals the initialization of the application start process.

```
void isDialogOpen (bool isDialog)
```

Signals the state of the dialog window.

```
void blockUIElements (bool isBlock)
```

Signals the need to block or unblock UI elements.

Public Member Functions

```
void setMainWindowInstance (MainWindow *instance)
```

*Sets the **MainWindow** instance and establishes connections between **InputManager** signals and **MainWindow** slots.*

```
void changeButton (const QString &buttonType, const QString &value)
```

Updates the value of a specific button type from mainwindow class.

```
QString getUserHotkey ()
```

Retrieves the current user-defined hotkey.

Static Public Member Functions

```
static InputManager * getInstance (QObject *parent=nullptr)
```

Public Attributes

```
bool isProcessRunning
```

Private Member Functions

```
InputManager (QObject *parent=nullptr)
```

```
void updateProcessWithHook ()
```

Updates the current process state based on the hook's running status.

Private Attributes

```
WindowsHookManager * windowsHookManager
MouseManager * mouseManager
MainWindow * mainWindowInstance
QMap< QString, QString > radioButtonValues
QTimer debounceTimer
QString userHotkey
```

Static Private Attributes

```
static InputManager * instance = nullptr
```

InputManager Class Documentation

The InputManager class manages user inputs, interacts with the Windows hook system, and serves as a central control hub.

This class coordinates interactions between the user interface, various hook-related operations, handling key sequences, mouse movements, and application state changes. It connects user actions to Windows hook manager and mouse manager instances for processing.

InputManager Member Function Documentation

```
void InputManager::changeButton (const QString &buttonType, const QString &value)
```

Updates the value of a specific button type from mainwindow class.

Parameters

buttonType	The type of button to update.
value	The new value to set for the button.

```
void InputManager::setMainWindowInstance (MainWindow *instance)
```

Sets the **MainWindow** instance and establishes connections between **InputManager** signals and **MainWindow** slots.

Parameters

instance	A pointer to the **MainWindow** instance to be managed.

```
void InputManager::updateProcessState (bool stop)[slot]
```

Updates the state of the process based on the given boolean value.

Parameters

stop	A boolean indicating whether to stop (true) or start (false) the process.

```
void InputManager::updateUserCursorLocationSet (int x, int y)[slot]
```

Emits a signal to return the updated cursor position.

Parameters

x	The X-coordinate of the cursor.
y	The Y-coordinate of the cursor.

```
void InputManager::updateUserData (int fixedTime, int randomTime, const int
&  repeatTimes, const QPoint & xy, const int &area)[slot]
```

Updates user data based on the provided parameters, then triggers the application start accordingly.

Parameters

fixedTime	The fixed time interval for the process.
randomTime	The random time interval for the process.
repeatTimes	The number of repetitions for the process.
xy	The coordinates for the process location.
area	The area size for the process.

Modifies input parameters based on button values, then emits them further.

```
void InputManager::updateUserHotkey (const QString pressedKey)[slot]
```

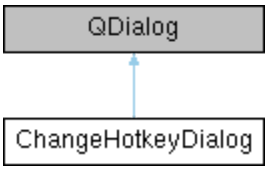
Updates and sets the user-defined hotkey, emitting a signal upon change.

Parameters

pressedKey	The newly pressed hotkey by the user.

ChangeHotkeyDialog Class Reference

Inheritance diagram for ChangeHotkeyDialog:



Signals

```
void updateHotkey (const QString &pressedKeys)
```

Updates the hotkey with the provided key sequence.

```
void isDialogOpen (bool isOpen)
```

Signals whether a dialog window is open or closed.

Public Member Functions

```
ChangeHotkeyDialog (QWidget *parent=nullptr)
```

*The **ChangeHotkeyDialog** class handles the secondary dialog responsible for changing hotkeys.*

Protected Member Functions

```
void keyPressEvent (QKeyEvent *event) override
```

Processes key events to capture the user input for setting a new hotkey.

Private Slots

```
void on_PushButton_Cancel_clicked ()
```

Closes the dialog window without changing hotkey.

```
void on_PushButton_Accept_clicked ()
```

Updates the hotkey and closes the dialog window.

```
void on_PushButton_Start_clicked ()
```

Initiates or resets the hotkey capturing process.

```
void on_pushButton_clicked ()
```

Displays a help message box providing guidance on changing hotkeys.

Private Attributes

```
Ui::ChangeHotkeyDialog * ui
QLineEdit * lineEdit
QString pressedKey
bool capturingKeys
```

ChangeHotkeyDialog Class Documentation

The ChangeHotkeyDialog class handles the secondary dialog responsible for changing hotkeys. This dialog captures keyboard input to set new hotkeys and displays a confirmation message to accept the newly set hotkey. The dialog comprises buttons to start capturing a new hotkey, cancel the operation, and accept the captured hotkey as the new input.

ChangeHotkeyDialog Member Function Documentation

```
void ChangeHotkeyDialog::keyPressEvent (QKeyEvent *event)[override], [protected]
```

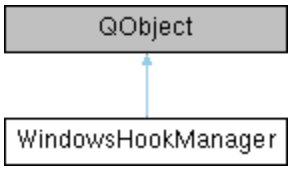
Processes key events to capture the user input for setting a new hotkey.

Parameters

event	The key event to be processed.

WindowsHookManager Class Reference

Inheritance diagram for WindowsHookManager:



Public Slots

```
void updateKeyboardVirtualKeys (const QString &keyCombination)
```

Updates the virtual key code corresponding to the provided key string.

```
void setDialogOpen (bool isOpen)
```

Controls the hook execution status based on the hotkey dialog state.

```
void prepareToGetCursorLocation ()
```

Prepares to retrieve the cursor's location on the screen.

Signals

```
void keyboardEventTriggered ()
```

Signals that a keyboard event has been triggered.

```
void passCursorLocation (int x, int y)
```

Emits the cursor's position on the screen.

Public Member Functions

```
WindowsHookManager (QObject *parent=nullptr)
```

Static Public Member Functions

```
static WindowsHookManager * getInstance (QObject *parent=nullptr)
```

Private Member Functions

```
void getCursorLocationOnScreen ()
```

Retrieves the cursor's location on the screen in response to a left button click.

Private Attributes

```
QMap< QString, int > keyMap
QThread * hookWorker
HookWorker * hookWorkerInstance
GlobalMouseHook * mouseHook
GlobalMouseHook * mouseHookInstance
bool shouldGrabMouse
```

Static Private Attributes

```
static WindowsHookManager * instance = nullptr
```

Coordinates and manages all hook-related classes, including keyboard and mouse hooks.

WindowsHookManager Class Documentation

Coordinates and manages all hook-related classes, including keyboard and mouse hooks.

Acts as a centralized manager for hook-based functionalities, overseeing keyboard and mouse hook operations. Initializes and manages instances of HookWorker and GlobalMouseHook, handles keyboard mapping, and grabs cursor locations.

WindowsHookManager Member Function Documentation

```
void WindowsHookManager::setDialogOpen (bool isOpen)[slot]
```

Controls the hook execution status based on the hotkey dialog state.

Parameters

isOpen	Indicates if a dialog window is open or closed.

```
void WindowsHookManager::updateKeyboardVirtualKeys (const QString &
key = nullptr)[slot]
```

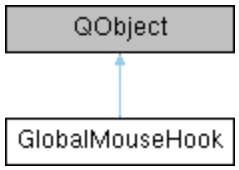
Updates the virtual key code corresponding to the provided key string.

Parameters

key	The string representation of the key to update the virtual key code.

GlobalMouseHook Class Reference

Inheritance diagram for GlobalMouseHook:



Signals

```
void mouseMoved (int x, int y)
```

Signals the movement of the mouse cursor.

```
void leftButtonClicked ()
```

Signals the left mouse button click event.

```
void rightButtonClicked ()
```

Signals the right mouse button click event.

```
void middleButtonClicked ()
```

Signals the middle mouse button click event.

Public Member Functions

```
GlobalMouseHook (QObject *parent=nullptr)
```

Static Public Member Functions

```
static GlobalMouseHook * getInstance ()
```

Static Private Member Functions

```
static LRESULT CALLBACK mouseProc (int nCode, WPARAM wParam, LPARAM
lParam)
```

Callback function for processing low-level mouse events.

Static Private Attributes

```
static HHOOK mouseHook = nullptr
```

*The **GlobalMouseHook** class manages mouse events globally through a Windows hook.*

```
static GlobalMouseHook * instance = nullptr
static GlobalMouseHook * mouseHookInstance
```

GlobalMouseHook Class Documentation

The GlobalMouseHook class manages mouse events globally through a Windows hook. This class sets up a low-level mouse hook (WH_MOUSE_LL) to capture and handle mouse events, including movements, left and right button clicks.
It utilizes SetWindowsHookEx to establish the hook and emits signals for specific mouse events, allowing external handling.

GlobalMouseHook Member Function Documentation

```
LRESULT CALLBACK GlobalMouseHook::mouseProc (int nCode, WPARAM
wParam, LPARAM lParam)[static], [private]
```

Callback function for processing low-level mouse events.

Parameters

nCode	A hook code. If less than zero, the function must return the result of calling the next hook procedure in the hook chain.
wParam	The identifier of the mouse message.
lParam	A pointer to an MSLLHOOKSTRUCT structure that contains details of the event.

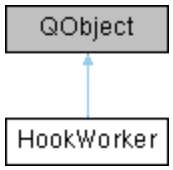
This function is called when a low-level mouse event occurs. It interprets and handles different mouse events such as mouse movements, left button clicks, and right button clicks. Upon receiving a particular mouse event, it emits corresponding signals to notify the connected slots.

Returns

The result of calling the next hook procedure in the hook chain.

HookWorker Class Reference

Inheritance diagram for HookWorker:



Signals

```
void keyboardEventTriggered ()
```

Signals the occurrence of a keyboard event.

Public Member Functions

```
HookWorker (QObject *parent=nullptr)
```

```
void processHooks ()
```

Triggers the keyboard event signal to indicate the detection of a specific key sequence.

```
void blockHook (bool block)
```

Blocks or unblocks the keyboard hook based on the provided boolean parameter.

```
int getCurrentVKCode ()
```

Retrieves the currently set Virtual Key (VK) code being monitored.

```
void setVkCode (int newVKCode)
```

Sets a new Virtual Key (VK) code to monitor a specific key.

```
void stopHook (bool run)
```

Controls the status of the keyboard hook, either stopping or resuming it.

Static Public Member Functions

```
static HookWorker * getInstance ()
```

Static Private Member Functions

```
static LRESULT CALLBACK KeyboardProc (int nCode, WPARAM wParam,
LPARAM lParam)
```

Handles low-level keyboard events through a Windows hook, monitoring specific key sequences and controlling hook blocking until the required key sequence is released.

Private Attributes

```
HHOOK globalKeyboardHook
bool isRunning
int vkCode = 117
bool isHookBlocked
```

Static Private Attributes

```
static HookWorker * instance = nullptr
```

*The **HookWorker** class manages a Windows low-level keyboard hook to monitor and respond to keyboard events globally.*

HookWorker Class Documentation

The HookWorker class manages a Windows low-level keyboard hook to monitor and respond to keyboard events globally.

This class sets up and controls a low-level keyboard hook (WH_KEYBOARD_LL) for capturing keyboard events like key presses and releases. It enables the detection and handling of specific key sequences in the system.

The HookWorker class initializes and manages a single instance of a global keyboard hook. It listens to keyboard events (key down and key up) and triggers specific actions based on a defined virtual key code (VKCode). The class also provides functionalities to stop and resume the keyboard hook.

HookWorker Member Function Documentation

```
void HookWorker::blockHook (bool block)
```

Blocks or unblocks the keyboard hook based on the provided boolean parameter.

Parameters

block	A boolean flag to block or unblock the keyboard hook.

```
int HookWorker::getCurrentVKCode ()
```

Retrieves the currently set Virtual Key (VK) code being monitored.

Returns

An integer representing the currently monitored Virtual Key (VK) code.

```
LRESULT CALLBACK HookWorker::KeyboardProc (int nCode, WPARAM  wParam,
LPARAM lParam)[static], [private]
```

Handles low-level keyboard events through a Windows hook, monitoring specific key sequences and controlling hook blocking until the required key sequence is released.

This function captures keyboard events, allowing identification of when a user presses down and releases a specific key. It blocks the hook until the required key sequence is released during WM_KEYUP, ensuring accurate detection of key presses and releases.

Parameters

nCode	The hook code indicating the action that should be taken.
wParam	The type of keyboard message (e.g., WM_KEYDOWN, WM_KEYUP).
lParam	A pointer to a KBDLLHOOKSTRUCT structure containing details about the keyboard event.

Returns

Returns the result of the next hook in the chain.

```
void HookWorker::setVkCode (int newVKCode)
```

Sets a new Virtual Key (VK) code to monitor a specific key.

Parameters

newVKCode	The new Virtual Key (VK) code to be set for monitoring.

```
void HookWorker::stopHook (bool running)
```

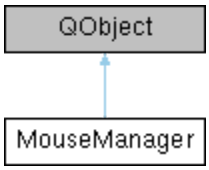
Controls the status of the keyboard hook, either stopping or resuming it.

Parameters

running	A boolean flag indicating whether the hook should be active or inactive.

MouseManager Class Reference

Inheritance diagram for MouseManager:



Classes

struct ThreadDataPublic Slots

```
void runClickingApplication (const int &clickTime, const int
&timeBetweenClicks, const QChar &type, const int &repetitions, const
QChar &location, const QPoint &xy, const int &area)
```

Initiates the mouse clicking application based on the provided parameters.

```
void stopClickingApplication ()
```

Stops the mouse clicking application if it's currently active.

Signals

```
void leftMouseClick ()
```

Signals a left mouse click event.

```
void mouseMoved (int x, int y)
```

Signals mouse movement to the specified coordinates.

```
void finished ()
```

Signals the completion of a process or operation.

Public Member Functions

```
MouseManager (QObject *parent=nullptr)
```

*Executes mouse actions based on user instructions received from the **InputManager**. Responsible for simulating mouse clicks and movements.*

```
void startMouseHook ()
```

```
void stopMouseHook ()
```

Public Attributes

```
bool isRunning
```

Private Slots

```
void runApplication ()
```

Executes the mouse clicking application based on defined parameters.

Private Member Functions

```
QPoint getRandomPointWithinCircle (const QPoint &centerOfCircle, int
circleRadius)
```

Generates a random point within a circle given its center and radius.

```
void leftClick ()
```

Simulates a left mouse button click.

```
bool repetitionsCompleted () const
```

Checks if the number of repetitions for the mouse clicking process is completed.

Private Attributes

```
QTimer * mouseTimer
bool isLocation
bool isArea
bool shouldStop = false
int repetitionCount = 0
ThreadData threadData
```

MouseManagerClass Documentation

Executes mouse actions based on user instructions received from the InputManager. Responsible for simulating mouse clicks and movements. It utilizes QTimer for controlling click intervals and facilitates left-click simulations and cursor movement based on the provided parameters like click time, time between clicks, click type, repetitions, and location. Also, it includes methods to start, stop, and check repetitions in the mouse click simulations.

Note

Uses QTimer for controlling click intervals and manages mouse event simulation based on given parameters.

MouseManagerClass Member Function Documentation

```
QPoint MouseManager::getRandomPointWithinCircle (const QPoint & center, int radius)[private]
```

Generates a random point within a circle given its center and radius.

Parameters

center	The center point of the circle.
radius	The radius of the circle.

Returns

Returns a random point within the specified circle.

This function calculates a random point within a circle using polar coordinates. It generates a random angle and distance within the range of the circle's radius and then converts these values to Cartesian coordinates (x, y) relative to the center.

```
void MouseManager::leftClick ()[private]
```

Simulates a left mouse button click.

This function generates a left mouse button click event by sending the corresponding input flags (down and up) using the Windows API. It emulates a left mouse button press and release action.

```
bool MouseManager::repetitionsCompleted () const[private]
```

Checks if the number of repetitions for the mouse clicking process is completed.

Returns

True if the desired number of repetitions is reached; otherwise, false.

```
void MouseManager::runApplication ()[private], [slot]
```

Executes the mouse clicking application based on defined parameters.

This function manages the execution of the mouse clicking application. It simulates mouse clicks and movements according to the provided settings until the specified number of repetitions is achieved or the process is stopped.

```
void MouseManager::runClickingApplication (const int &clickTime,
const int & timeBetweenClicks, const QChar &  type, const int &
repetitions, const QChar &location, const QPoint &  xy, const int &
area)[slot]
```

Initiates the mouse clicking application based on the provided parameters.

Parameters

clickTime	The time interval for mouse clicks.
timeBetweenClicks	The extra time interval between consecutive clicks.
type	The type of mouse click.
repetitions	The number of times to repeat the click action.
location	The location type for mouse action.
xy	The coordinates of the mouse action.
area	The area size for mouse action.

Configures the mouse clicking application based on provided parameters.
