UMI RTX for Tic-Tac-Toe

Generated by Doxygen 1.9.1

Chapter 1

UMI-RTX Robotic Arm for Tic-Tac-Toe

1.1 Authors

- MEGUIN Nathan < nathan.meguin@etu.uca.fr> (ISIMA - Software Engineering)

1.2 Description

This repository is designed to set up a ROS2 interface to make the UMI-RTX Arm play a Tic-Tac-Toe game against a human.

1.3 Manual Installation

This project is built and tested with **Ubuntu 22.04** and **ROS2 Iron**.

1.3.1 ROS Installation

Ensure you have ROS2 Iron installed. If not, follow the instructions provided here.

Set up your environment by sourcing ROS2. If you're not using bash, replace ".bash" with your shell type:

```
source /opt/ros/iron/setup.bash
```

To avoid sourcing it every time, consider adding this line to the end of your ~/.bashrc file.

1.3.2 Install Dependencies

Update your package lists:

```
sudo apt update
```

Install necessary dependencies including Pinocchio for inverse kinematics:

```
sudo apt install ros-iron-pinocchio -y
sudo apt install ros-iron-xacro -y
```

1.3.3 Colcon Installation

Install Colcon, the build tool required for this project:

```
sudo apt update
sudo apt install python3-colcon-common-extensions
```

1.3.4 Autocompletion

Enable autocompletion for Colcon:

```
source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
```

To avoid sourcing it every time, add this line to the end of your ∼/.bashrc file.

1.3.5 Build the Package

Navigate to your ROS workspace directory:

```
cd ROS_ws
```

Build the package using Colcon:

```
colcon build
```

Once built, source it to set up your environment:

```
source install/setup.bash
```

1.3.6 Run the Package

To run the simulation, execute:

```
ros2 launch umi_rtx_controller simu.launch.py
```

To use the arm, execute:

```
./start_arm.sh
```

1.4 Docker

1.4.1 Docker Installation

Install Docker and the NVIDIA Docker toolkit:

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | sudo tee
       /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update && sudo apt-get install -y nvidia-docker2 nvidia-container-toolkit
sudo systemctl daemon-reload
sudo systemctl restart docker
```

1.4.2 Build the Docker Image

```
# Navigate to umi_rtx_demos
docker build -t "name" .
```

Replace "name" with your chosen name for the Docker image.

1.4.3 Run the Docker Container

```
# Grant permission to use the screen
xhost +
# Launch the container (replace "name" with the name you chose)
docker run --gpus all -it --privileged -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix --rm
       "name":latest
```

1.4.4 Run the Program

Start by running:

```
cd ROS_ws/
colcon build
source install/setup.bash
cd ..
```

Then, to launch the simulation:

```
ros2 launch umi_rtx_controller simu.launch.py
```

And to use the arm:

```
./start_arm.sh
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 rapidxml Namespace Reference

Classes

- class parse_error
- class xml_node
- class xml_attribute
- class xml_document
- class memory_pool
- class xml_base

Enumerations

- enum node_type {
  node_document , node_element , node_data , node_cdata ,
  node_comment , node_declaration , node_doctype , node_pi }

Variables

- const int parse_no_data_nodes = 0x1
- const int parse_no_element_values = 0x2
- const int parse_no_string_terminators = 0x4
- const int parse_no_entity_translation = 0x8
- const int parse_no_utf8 = 0x10
- const int parse_declaration_node = 0x20
- const int parse_comment_nodes = 0x40
- const int parse_doctype_node = 0x80
- const int parse_pi_nodes = 0x100
- const int parse_validate_closing_tags = 0x200
- const int parse_trim_whitespace = 0x400
- const int parse_normalize_whitespace = 0x800
- const int parse_default = 0
- const int parse_non_destructive = parse_no_string_terminators | parse_no_entity_translation
- const int parse_fastest = parse_non_destructive | parse_no_data_nodes
- const int parse_full = parse_declaration_node | parse_comment_nodes | parse_doctype_node | parse_pi_nodes | parse_validate_closing_tags

6.1.1 Enumeration Type Documentation

6.1.1.1 node_type

```
enum rapidxml::node_type
```

Enumeration listing all node types produced by the parser. Use xml_node::type() function to query node type.

Enumerator

node_document	A document node. Name and value are empty.
node_element	An element node. Name contains element name. Value contains text of first data node.
node_data	A data node. Name is empty. Value contains data text.
node_cdata	A CDATA node. Name is empty. Value contains data text.
node_comment	A comment node. Name is empty. Value contains comment text.
node_declaration	A declaration node. Name and value are empty. Declaration parameters (version, encoding and standalone) are in node attributes.
node_doctype	A DOCTYPE node. Name is empty. Value contains DOCTYPE text.
node_pi	A PI node. Name contains target. Value contains instructions.

6.1.2 Variable Documentation

6.1.2.1 parse_comment_nodes

```
const int rapidxml::parse_comment_nodes = 0x40
```

Parse flag instructing the parser to create comments nodes. By default, comment nodes are not created. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.2 parse_declaration_node

```
const int rapidxml::parse_declaration_node = 0x20
```

Parse flag instructing the parser to create XML declaration node. By default, declaration node is not created. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.3 parse_default

```
const int rapidxml::parse_default = 0
```

Parse flags which represent default behaviour of the parser. This is always equal to 0, so that all other flags can be simply ored together. Normally there is no need to inconveniently disable flags by anding with their negated ($\sim$) values. This also means that meaning of each flag is a *negation* of the default setting. For example, if flag name is rapidxml::parse_no_utf8, it means that utf-8 is *enabled* by default, and using the flag will disable it.

See xml_document::parse() function.

6.1.2.4 parse_doctype_node

```
const int rapidxml::parse_doctype_node = 0x80
```

Parse flag instructing the parser to create DOCTYPE node. By default, doctype node is not created. Although W3C specification allows at most one DOCTYPE node, RapidXml will silently accept documents with more than one. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.5 parse_fastest

```
const int rapidxml::parse_fastest = parse_non_destructive | parse_no_data_nodes
```

A combination of parse flags resulting in fastest possible parsing, without sacrificing important data.

See xml_document::parse() function.

6.1.2.6 parse_full

```
const int rapidxml::parse_full = parse_declaration_node | parse_comment_nodes | parse_doctype_node
| parse_pi_nodes | parse_validate_closing_tags
```

A combination of parse flags resulting in largest amount of data being extracted. This usually results in slowest parsing.

See xml_document::parse() function.

6.1.2.7 parse_no_data_nodes

```
const int rapidxml::parse_no_data_nodes = 0x1
```

Parse flag instructing the parser to not create data nodes. Text of first data node will still be placed in value of parent element, unless rapidxml::parse_no_element_values flag is also specified. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.8 parse_no_element_values

```
const int rapidxml::parse_no_element_values = 0x2
```

Parse flag instructing the parser to not use text of first data node as a value of parent element. Can be combined with other flags by use of | operator. Note that child data nodes of element node take precendence over its value when printing. That is, if element has one or more child data nodes *and* a value, the value will be ignored. Use rapidxml::parse_no_data_nodes flag to prevent creation of data nodes if you want to manipulate data using values of elements.

See xml_document::parse() function.

6.1.2.9 parse_no_entity_translation

```
const int rapidxml::parse_no_entity_translation = 0x8
```

Parse flag instructing the parser to not translate entities in the source text. By default entities are translated, modifying source text. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.10 parse_no_string_terminators

```
const int rapidxml::parse_no_string_terminators = 0x4
```

Parse flag instructing the parser to not place zero terminators after strings in the source text. By default zero terminators are placed, modifying source text. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.11 parse_no_utf8

```
const int rapidxml::parse_no_utf8 = 0x10
```

Parse flag instructing the parser to disable UTF-8 handling and assume plain 8 bit characters. By default, UTF-8 handling is enabled. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.12 parse_non_destructive

```
const int rapidxml::parse_non_destructive = parse_no_string_terminators | parse_no_entity_translation
```

A combination of parse flags that forbids any modifications of the source text. This also results in faster parsing. However, note that the following will occur:

- names and values of nodes will not be zero terminated, you have to use xml_base::name_size() and xml_base::value_size() functions to determine where name and value ends
- entities will not be translated
- whitespace will not be normalized

See xml_document::parse() function.

6.1.2.13 parse_normalize_whitespace

```
const int rapidxml::parse_normalize_whitespace = 0x800
```

Parse flag instructing the parser to condense all whitespace runs of data nodes to a single space character. Trimming of leading and trailing whitespace of data is controlled by rapidxml::parse_trim_whitespace flag. By default, whitespace is not normalized. If this flag is specified, source text will be modified. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.14 parse_pi_nodes

```
const int rapidxml::parse_pi_nodes = 0x100
```

Parse flag instructing the parser to create PI nodes. By default, PI nodes are not created. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.15 parse_trim_whitespace

```
const int rapidxml::parse_trim_whitespace = 0x400
```

Parse flag instructing the parser to trim all leading and trailing whitespace of data nodes. By default, whitespace is not trimmed. This flag does not cause the parser to modify source text. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.1.2.16 parse_validate_closing_tags

```
const int rapidxml::parse_validate_closing_tags = 0x200
```

Parse flag instructing the parser to validate closing tag names. If not set, name inside closing tag is irrelevant to the parser. By default, closing tags are not validated. Can be combined with other flags by use of | operator.

See xml_document::parse() function.

6.2 rviz_common Namespace Reference

Chapter 7

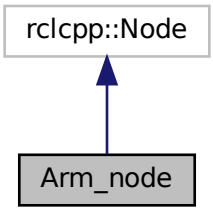
Class Documentation

7.1 Arm_node Class Reference

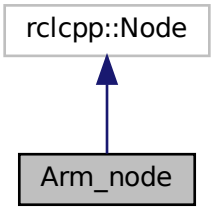
A ROS2 node that controls a robotic arm.

`#include <node_arm.hpp>`

Inheritance diagram for Arm_node:



Collaboration diagram for Arm_node:



Public Member Functions

- Arm_node ()

  Construct a new Arm_node object.

Private Member Functions

- void timer_callback ()

  Timer callback, actions that will be done at every iterations.
- void init_interfaces ()

  Initialize the timer, subscribers and publishers.
- void get_commands (const sensor_msgs::msg::JointState::SharedPtr msg)

  Get the commands that will be sent to the arm.
- void get_pose (const geometry_msgs::msg::Pose::SharedPtr msg)

  Get the targeted position.
- void get_grip (const std_msgs::msg::Float32::SharedPtr msg)

  Get the targeted grip.
- void set_motors ()

  Set the motors/joints to their required state to reach the desired position.
- void get_params ()

  Get the joints' parameters.
- string params2msg ()

  Converts motors_params into a string to publish more easily.

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- map< int, double > commands_motor
- map< int, map< int, int > > motors_params
- Arm full_arm
- double targ_x
- double targ_y
- double targ_z
- double x
- double y
- double z
- double target_yaw
- double target_pitch
- double target_roll
- double yaw
- double pitch
- double roll
- double target_grip
- double grip
- rclcpp::TimerBase::SharedPtr timer_
- rclcpp::Subscription< sensor_msgs::msg::JointState >::SharedPtr subscription_commands
- rclcpp::Subscription< geometry_msgs::msg::Pose >::SharedPtr pose_subscription
- rclcpp::Subscription< std_msgs::msg::Float32 >::SharedPtr grip_subscription
- rclcpp::Publisher< std_msgs::msg::String >::SharedPtr publisher_params

7.1.1 Detailed Description

A ROS2 node that controls a robotic arm.

This class initializes the node's interfaces, subscribes to topics for motor commands, target poses, and grip parameters, and manages the robotic arm's movements. It also publishes parameters and handles motor control logic.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Arm_node()

```
Arm_node::Arm_node ( )  [inline]
```

Construct a new Arm_node object.

7.1.3 Member Function Documentation

7.1.3.1 get_commands()

```
void Arm_node::get_commands (
            const sensor_msgs::msg::JointState::SharedPtr msg )  [private]
```

Get the commands that will be sent to the arm.

Parameters

msg	States of the joints required to reach the desired position

7.1.3.2 get_grip()

```
void Arm_node::get_grip (
            const std_msgs::msg::Float32::SharedPtr msg )  [private]
```

Get the targeted grip.

Parameters

msg	

7.1.3.3 get_params()

```
void Arm_node::get_params ( )  [private]
```

Get the joints' parameters.

7.1.3.4 get_pose()

```
void Arm_node::get_pose (
            const geometry_msgs::msg::Pose::SharedPtr msg )  [private]
```

Get the targeted position.

Parameters

msg	

7.1.3.5 init_interfaces()

```
void Arm_node::init_interfaces ( )  [private]
```

Initialize the timer, subscribers and publishers.

7.1.3.6 params2msg()

```
string Arm_node::params2msg ( )  [private]
```

Converts motors_params into a string to publish more easily.

Returns

string

7.1.3.7 set_motors()

```
void Arm_node::set_motors ( )  [private]
```

Set the motors/joints to their required state to reach the desired position.

7.1.3.8 timer_callback()

```
void Arm_node::timer_callback ( )  [private]
```

Timer callback, actions that will be done at every iterations.

7.1.4 Member Data Documentation

7.1.4.1 commands_motor

```
map<int,double> Arm_node::commands_motor  [private]
```

7.1.4.2 full_arm

```
Arm Arm_node::full_arm  [private]
```

7.1.4.3 grip

```
double Arm_node::grip  [private]
```

7.1.4.4 grip_subscription

```
rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr Arm_node::grip_subscription  [private]
```

7.1.4.5 loop_dt_

```
std::chrono::milliseconds Arm_node::loop_dt_ = 40ms  [private]
```

7.1.4.6 motors_params

```
map<int,map<int,int> > Arm_node::motors_params  [private]
```

7.1.4.7 pitch

```
double Arm_node::pitch  [private]
```

7.1.4.8 pose_subscription

```
rclcpp::Subscription<geometry_msgs::msg::Pose>::SharedPtr Arm_node::pose_subscription  [private]
```

7.1.4.9 publisher_params

```
rclcpp::Publisher<std_msgs::msg::String>::SharedPtr Arm_node::publisher_params  [private]
```

7.1.4.10 roll

```
double Arm_node::roll  [private]
```

7.1.4.11 subscription_commands

```
rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr Arm_node::subscription_commands
[private]
```

7.1.4.12 targ_x

```
double Arm_node::targ_x  [private]
```

7.1.4.13 targ_y

```
double Arm_node::targ_y  [private]
```

7.1.4.14 targ_z

```
double Arm_node::targ_z  [private]
```

7.1.4.15 target_grip

```
double Arm_node::target_grip  [private]
```

7.1.4.16 target_pitch

```
double Arm_node::target_pitch  [private]
```

7.1.4.17 target_roll

```
double Arm_node::target_roll  [private]
```

7.1.4.18 target_yaw

```
double Arm_node::target_yaw  [private]
```

7.1.4.19 timer_

```
rclcpp::TimerBase::SharedPtr Arm_node::timer_  [private]
```

7.1.4.20 x

```
double Arm_node::x  [private]
```

7.1.4.21 y

```
double Arm_node::y  [private]
```

7.1.4.22 yaw

```
double Arm_node::yaw  [private]
```

7.1.4.23 z

```
double Arm_node::z  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_arm.hpp
- ros2_ws/src/umi_rtx_controller/src/node_arm.cpp

7.2 rapidxml::xml_document< Ch >::attribute_name_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.2.1 Member Function Documentation

7.2.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::attribute_name_pred::test (
            Ch ch )  [inline], [static]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.3 rapidxml::xml_document< Ch >::attribute_value_pred< Quote > Struct Template Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.3.1 Member Function Documentation

7.3.1.1 test()

```
template<class Ch = char>
template<Ch Quote>
static unsigned char rapidxml::xml_document< Ch >::attribute_value_pred< Quote >::test (
            Ch ch )  [inline], [static]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.4 rapidxml::xml_document< Ch >::attribute_value_pure_pred< Quote > Struct Template Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.4.1 Member Function Documentation

7.4.1.1 test()

```
template<class Ch = char>
template<Ch Quote>
static unsigned char rapidxml::xml_document< Ch >::attribute_value_pure_pred< Quote >::test (
            Ch ch )  [inline], [static]
```

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

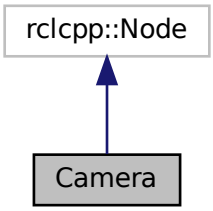
- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.5 Camera Class Reference

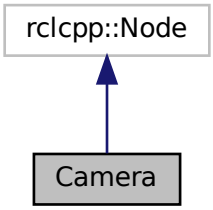
A ROS 2 node for managing and processing data from a RealSense camera.

`#include <node_camera.hpp>`

Inheritance diagram for Camera:



Collaboration diagram for Camera:



Public Member Functions

- Camera ()

  Construct a new Camera object.

Private Member Functions

- void timer_callback ()

  Callback function for processing frames and publishing data.
- void init_interfaces ()

  Initializes the ROS 2 interfaces for the Camera class.
- void init_camera ()

  Initializes the camera.

- void get_colored_objects (geometry_msgs::msg::Pose pose_msg, rs2::depth_frame depth)

  Detects colored objects in the frame.
- void get_angles (vector< cv::Point > &contours)

  Finds the fittest line with respect to the contour of the target and computes its orientation.
- void get_capture_step (const std_msgs::msg::Bool::SharedPtr msg)

  Callback function to capture a step.
- int find_box (double cx, double cy)

  Determines the box index based on the pawn coordinates.

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- rclcpp::TimerBase::SharedPtr timer_
- int board_state [9]
- int turn
- bool capture_step
- rs2::pipeline pipe
- rs2::config cfg
- rs2::align align_to_color
- rs2::colorizer color_map
- int available_device
- cv::Mat colorFrameCV
- cv::Mat depthFrameCV
- int m_frame_width
- int m_frame_height
- int m_depth_frame_width
- int m_depth_frame_height
- double m_cx
- double m_cy
- double m_cz
- double yaw
- double pitch
- double roll
- rclcpp::Publisher< sensor_msgs::msg::Image >::SharedPtr image_publisher
- rclcpp::Publisher< geometry_msgs::msg::Pose >::SharedPtr processed_pose_publisher
- rclcpp::Publisher< sensor_msgs::msg::Image >::SharedPtr depth_publisher
- rclcpp::Publisher< std_msgs::msg::Bool >::SharedPtr ready_to_play_publisher
- rclcpp::Publisher< umi_rtx_interfaces::msg::Board >::SharedPtr board_state_publisher
- rclcpp::Subscription< std_msgs::msg::Bool >::SharedPtr capture_step_subscriber

7.5.1 Detailed Description

A ROS 2 node for managing and processing data from a RealSense camera.

The `Camera` class is responsible for capturing images and depth data from a RealSense camera, processing this data to detect colored objects and their positions, and publishing relevant information to various ROS 2 topics. The class also handles initialization of ROS 2 interfaces and the RealSense camera.

The class includes methods to initialize ROS 2 publishers and subscribers, configure the RealSense camera, process image and depth data, and handle capture step commands. It also provides functionality to detect colored objects and determine box indices based on coordinates.

The `Camera` constructor sets up the node, initializes interfaces, and configures the camera.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Camera()

```
Camera::Camera ( )  [inline]
```

Construct a new Camera object.

The constructor also performs the following initializations:

- Calls `init_interfaces()` to set up publishers, subscribers, and timers for the camera node.
- Calls `init_camera()` to configure and start the RealSense camera, including setting up streams and configuring sensor options.

The `init_interfaces()` method creates and initializes ROS 2 publishers and subscribers required for image and pose data, board state, and capture step commands. The `init_camera()` method checks for available RealSense cameras, initializes the first available device, and configures its depth and color streams. It also sets sensor options for high accuracy.

Note

Ensure that the RealSense camera is properly connected and that the RealSense SDK is correctly installed before using this constructor.

7.5.3 Member Function Documentation

7.5.3.1 find_box()

```
int Camera::find_box (
            double cx,
            double cy )  [private]
```

Determines the box index based on the pawn coordinates.

This function calculates the box index based on the provided coordinates.

Parameters

cx	The x-coordinate.
cy	The y-coordinate.

Returns

The box index or -1 if the coordinates are out of bounds.

7.5.3.2 get_angles()

```
void Camera::get_angles (
            vector< cv::Point > & contours )  [private]
```

Finds the fittest line with respect to the contour of the target and computes its orientation.

Parameters

contours	The longest contours in the binarized image.

7.5.3.3 get_capture_step()

```
void Camera::get_capture_step (
            const std_msgs::msg::Bool::SharedPtr msg )  [private]
```

Callback function to capture a step.

This function is called when a message is received on the "capture_step" topic. It sets the capture_step flag based on the received message.

Parameters

msg	The received message.

7.5.3.4 get_colored_objects()

```
void Camera::get_colored_objects (
            geometry_msgs::msg::Pose pose_msg,
            rs2::depth_frame depth )  [private]
```

Detects colored objects in the frame.

This function processes the captured frames to detect colored objects. It identifies the position of the objects, determines their coordinates, and updates the board state accordingly.

Parameters

pose_msg	The pose message to be populated with object coordinates.
depth	The depth frame for distance measurement.

7.5.3.5 init_camera()

```
void Camera::init_camera ( )  [private]
```

Initializes the camera.

This function checks for available cameras, selects the first detected RealSense camera, and sets up the configuration for the depth and color streams.

7.5.3.6 init_interfaces()

```
void Camera::init_interfaces ( )  [private]
```

Initializes the ROS 2 interfaces for the Camera class.

This function initializes the various ROS 2 publishers, subscribers, and timers needed for the Camera class to function correctly.

7.5.3.7 timer_callback()

```
void Camera::timer_callback ( )  [private]
```

Callback function for processing frames and publishing data.

This function is called periodically by a timer and performs the following tasks:

- Waits for the next set of frames from the RealSense camera.
- Aligns the depth frame to the color frame for improved accuracy.
- Retrieves and processes the depth frame, colored depth frame, and color frame.
- Converts the frames to OpenCV matrices for further processing.
- If the `capture_step` flag is set, processes the frames to detect colored objects, updates the pose message with the detected object's position, and draws rectangles on the color frame to indicate regions of interest.
- Publishes the board state, processed pose message, and images (depth and color) to respective ROS 2 topics.
- Resets the `capture_step` flag after processing.

The function utilizes the RealSense `rs2::pipeline` to capture and process frames. The depth frame is aligned with the color frame to enhance accuracy. The `color_map` filter is applied to the depth frame to create a colored depth frame for visualization. Both the depth and color frames are converted to OpenCV matrices, which are then used to detect colored objects and draw rectangles on the color frame. The resulting images and pose data are published to the appropriate ROS 2 topics.

7.5.4 Member Data Documentation

7.5.4.1 align_to_color

```
rs2::align Camera::align_to_color  [private]
```

7.5.4.2 available_device

```
int Camera::available_device  [private]
```

7.5.4.3 board_state

```
int Camera::board_state[9]  [private]
```

7.5.4.4 board_state_publisher

```
rclcpp::Publisher<umi_rtx_interfaces::msg::Board>::SharedPtr Camera::board_state_publisher
[private]
```

7.5.4.5 capture_step

```
bool Camera::capture_step  [private]
```

7.5.4.6 capture_step_subscriber

```
rclcpp::Subscription<std_msgs::msg::Bool>::SharedPtr Camera::capture_step_subscriber  [private]
```

7.5.4.7 cfg

```
rs2::config Camera::cfg  [private]
```

7.5.4.8 color_map

```
rs2::colorizer Camera::color_map  [private]
```

7.5.4.9 colorFrameCV

```
cv::Mat Camera::colorFrameCV  [private]
```

7.5.4.10 depth_publisher

```
rclcpp::Publisher<sensor_msgs::msg::Image>::SharedPtr Camera::depth_publisher  [private]
```

7.5.4.11 depthFrameCV

```
cv::Mat Camera::depthFrameCV  [private]
```

7.5.4.12 image_publisher

```
rclcpp::Publisher<sensor_msgs::msg::Image>::SharedPtr Camera::image_publisher  [private]
```

7.5.4.13 loop_dt_

```
std::chrono::milliseconds Camera::loop_dt_ = 40ms  [private]
```

7.5.4.14 m_cx

```
double Camera::m_cx  [private]
```

7.5.4.15 m_cy

```
double Camera::m_cy  [private]
```

7.5.4.16 m_cz

```
double Camera::m_cz  [private]
```

7.5.4.17 m_depth_frame_height

```
int Camera::m_depth_frame_height  [private]
```

7.5.4.18 m_depth_frame_width

```
int Camera::m_depth_frame_width  [private]
```

7.5.4.19 m_frame_height

```
int Camera::m_frame_height  [private]
```

7.5.4.20 m_frame_width

```
int Camera::m_frame_width  [private]
```

7.5.4.21 pipe

```
rs2::pipeline Camera::pipe  [private]
```

7.5.4.22 pitch

```
double Camera::pitch  [private]
```

7.5.4.23 processed_pose_publisher

```
rclcpp::Publisher<geometry_msgs::msg::Pose>::SharedPtr Camera::processed_pose_publisher  [private]
```

7.5.4.24 ready_to_play_publisher

```
rclcpp::Publisher<std_msgs::msg::Bool>::SharedPtr Camera::ready_to_play_publisher  [private]
```

7.5.4.25 roll

```
double Camera::roll  [private]
```

7.5.4.26 timer_

```
rclcpp::TimerBase::SharedPtr Camera::timer_  [private]
```

7.5.4.27 turn

```
int Camera::turn  [private]
```

7.5.4.28 yaw

```
double Camera::yaw  [private]
```

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

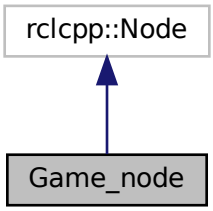
- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_camera.hpp
- ros2_ws/src/umi_rtx_controller/src/node_camera.cpp

7.6 Game_node Class Reference

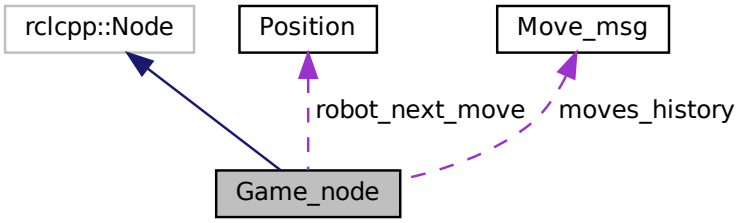
ROS 2 node for managing and controlling a game.

`#include <node_game.hpp>`

Inheritance diagram for Game_node:



Collaboration diagram for Game_node:



Public Member Functions

- Game_node ()

  Construct a new Game_node object.

Private Member Functions

- void init_interfaces ()

  Initialize the timer, subscribers, and publishers.
- void timer_callback ()

  Timer callback, actions that will be done at every iteration.
- void get_board (const umi_rtx_interfaces::msg::Board::SharedPtr msg)

  Callback to get the board state from the subscription.

- void get_ready_to_play (const std_msgs::msg::Bool::SharedPtr msg)

  Callback to get the has_played state from the subscription.
- Position getAIMove ()

  Get a random move for the AI.
- vector< Position > getAvailablePositions ()

  Get available positions on the board.
- bool hasWinner (int player)

  Check if a player has won.
- void update_turn ()

  Determine the current player's turn.
- bool isBoardFull ()

  Check if the board is full.
- void displayBoard ()

  Display the board on the console.
- Position findBestMove ()

  Find the best move for the robot using the minimax algorithm.
- int minimax (int depth, bool isMaximizer)

  Minimax algorithm to evaluate the board state.
- bool isGameOver ()

  Check if the game is over.
- int evaluate ()

  Evaluate the board state to determine the score.
- bool check_endgame ()

  Check for endgame conditions.
- bool check_changes ()

  Check for changes in the board and update move history accordingly.
- void add_move (int box, int player)

  Add a move to the history.
- void delete_move (int box)

  Delete a move from the history.

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- bool game_is_started
- bool is_initialized
- int board [3][3]
- int last_board [3][3]
- Position robot_next_move
- int turn
- int player_turn
- int starter
- Move_msg moves_history [9]
- std::string primary_msg
- std::string secondary_msg
- bool ready_to_play
- bool is_finished
- std_msgs::msg::String last_move_msg
- umi_rtx_interfaces::msg::GameData data_msg
- rclcpp::TimerBase::SharedPtr timer_
- rclcpp::Publisher< umi_rtx_interfaces::msg::GameData >::SharedPtr game_data_publisher
- rclcpp::Publisher< std_msgs::msg::Int32 >::SharedPtr robot_next_move_publisher
- rclcpp::Publisher< std_msgs::msg::String >::SharedPtr last_move_publisher
- rclcpp::Subscription< umi_rtx_interfaces::msg::Board >::SharedPtr board_state_subscription
- rclcpp::Subscription< std_msgs::msg::Bool >::SharedPtr ready_to_play_subscription

7.6.1 Detailed Description

ROS 2 node for managing and controlling a game.

This class handles the logic for a game of Tic-Tac-Toe, including managing game state, processing moves, and interacting with other components via ROS 2 messages and services.

The Game_node class inherits from `rclcpp::Node` and provides functionality to:

- Initialize game settings and ROS 2 interfaces.
- Handle game logic such as determining moves for AI player.
- Publish game state and move information.
- Subscribe to game state updates and readiness signals.

It includes mechanisms for:

- Randomly selecting the starting player.
- Performing game moves and evaluating game state.
- Using the minimax algorithm for AI decision-making.
- Publishing and subscribing to relevant ROS 2 topics.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Game_node()

```
Game_node::Game_node ( )  [inline]
```

Construct a new Game_node object.

7.6.3 Member Function Documentation

7.6.3.1 add_move()

```
void Game_node::add_move (
            int box,
            int player )  [private]
```

Add a move to the history.

Parameters

box	The position on the board (0-8) where the move occurred.
player	The player making the move (ROBOT or HUMAN).

7.6.3.2 check_changes()

```
bool Game_node::check_changes ( )  [private]
```

Check for changes in the board and update move history accordingly.

Returns

true if changes are detected, false otherwise.

7.6.3.3 check_endgame()

```
bool Game_node::check_endgame ( )  [private]
```

Check for endgame conditions.

Returns

true if the game has ended, false otherwise.

7.6.3.4 delete_move()

```
void Game_node::delete_move (
            int box )  [private]
```

Delete a move from the history.

Parameters

box	The position on the board (0-8) of the move to be deleted.

7.6.3.5 displayBoard()

```
void Game_node::displayBoard ( )  [private]
```

Display the board on the console.

7.6.3.6 evaluate()

```
int Game_node::evaluate ( )  [private]
```

Evaluate the board state to determine the score.

Returns

int The evaluation score.

7.6.3.7 findBestMove()

```
Position Game_node::findBestMove ( )  [private]
```

Find the best move for the robot using the minimax algorithm.

Returns

Position The best position for the robot's next move.

7.6.3.8 get_board()

```
void Game_node::get_board (
            const umi_rtx_interfaces::msg::Board::SharedPtr msg )  [private]
```

Callback to get the board state from the subscription.

Parameters

msg	Shared pointer to the received board state message.

7.6.3.9 get_ready_to_play()

```
void Game_node::get_ready_to_play (
            const std_msgs::msg::Bool::SharedPtr msg )  [private]
```

Callback to get the has_played state from the subscription.

Parameters

msg	Shared pointer to the received boolean message.

7.6.3.10 getAIMove()

```
Position Game_node::getAIMove ( )  [private]
```

Get a random move for the AI.

Returns

Position The chosen position.

7.6.3.11 getAvailablePositions()

```
vector< Position > Game_node::getAvailablePositions ( )  [private]
```

Get available positions on the board.

Returns

vector<Position> A vector of available positions.

7.6.3.12 hasWinner()

```
bool Game_node::hasWinner (
            int player )  [private]
```

Check if a player has won.

Parameters

player	The player to check for victory (ROBOT or HUMAN).

Returns

true If the player has won.

false If the player has not won.

7.6.3.13 init_interfaces()

```
void Game_node::init_interfaces ( )  [private]
```

Initialize the timer, subscribers, and publishers.

7.6.3.14 isBoardFull()

```
bool Game_node::isBoardFull ( )  [private]
```

Check if the board is full.

Returns

true If the board is full.

false If the board is not full.

7.6.3.15 isGameOver()

```
bool Game_node::isGameOver ( )  [private]
```

Check if the game is over.

Returns

true If the game is over.

false If the game is not over.

7.6.3.16 minimax()

```
int Game_node::minimax (
            int depth,
            bool isMaximizer )  [private]
```

Minimax algorithm to evaluate the board state.

Parameters

depth	Current depth in the minimax tree.
isMaximizer	Boolean flag to indicate if the current node is maximizing or minimizing.

Returns

int The evaluation score.

7.6.3.17 timer_callback()

```
void Game_node::timer_callback ( )  [private]
```

Timer callback, actions that will be done at every iteration.

7.6.3.18 update_turn()

```
void Game_node::update_turn ( )  [private]
```

Determine the current player's turn.

Returns

int The current player's turn (ROBOT or HUMAN).

7.6.4 Member Data Documentation

7.6.4.1 board

```
int Game_node::board[3][3]  [private]
```

7.6.4.2 board_state_subscription

```
rclcpp::Subscription<umi_rtx_interfaces::msg::Board>::SharedPtr Game_node::board_state_↩
subscription  [private]
```

7.6.4.3 data_msg

```
umi_rtx_interfaces::msg::GameData Game_node::data_msg  [private]
```

7.6.4.4 game_data_publisher

```
rclcpp::Publisher<umi_rtx_interfaces::msg::GameData>::SharedPtr Game_node::game_data_publisher
[private]
```

7.6.4.5 game_is_started

```
bool Game_node::game_is_started  [private]
```

7.6.4.6 is_finished

```
bool Game_node::is_finished  [private]
```

7.6.4.7 is_initialized

```
bool Game_node::is_initialized  [private]
```

7.6.4.8 last_board

```
int Game_node::last_board[3][3]  [private]
```

7.6.4.9 last_move_msg

```
std_msgs::msg::String Game_node::last_move_msg  [private]
```

7.6.4.10 last_move_publisher

```
rclcpp::Publisher<std_msgs::msg::String>::SharedPtr Game_node::last_move_publisher  [private]
```

7.6.4.11 loop_dt_

```
std::chrono::milliseconds Game_node::loop_dt_ = 40ms  [private]
```

7.6.4.12 moves_history

```
Move_msg Game_node::moves_history[9]  [private]
```

7.6.4.13 player_turn

```
int Game_node::player_turn  [private]
```

7.6.4.14 primary_msg

```
std::string Game_node::primary_msg  [private]
```

7.6.4.15 ready_to_play

```
bool Game_node::ready_to_play  [private]
```

7.6.4.16 ready_to_play_subscription

```
rclcpp::Subscription<std_msgs::msg::Bool>::SharedPtr Game_node::ready_to_play_subscription
[private]
```

7.6.4.17 robot_next_move

```
Position Game_node::robot_next_move  [private]
```

7.6.4.18 robot_next_move_publisher

```
rclcpp::Publisher<std_msgs::msg::Int32>::SharedPtr Game_node::robot_next_move_publisher  [private]
```

7.6.4.19 secondary_msg

```
std::string Game_node::secondary_msg  [private]
```

7.6.4.20 starter

```
int Game_node::starter  [private]
```

7.6.4.21 timer_

```
rclcpp::TimerBase::SharedPtr Game_node::timer_  [private]
```

7.6.4.22 turn

```
int Game_node::turn  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_game.hpp
- ros2_ws/src/umi_rtx_controller/src/node_game.cpp

7.7 GridSquare Struct Reference

```
#include <node_camera.hpp>
```

Public Attributes

- cv::Point2f center
- std::vector< cv::Point > contours
- int area

7.7.1 Member Data Documentation

7.7.1.1 area

```
int GridSquare::area
```

7.7.1.2 center

```
cv::Point2f GridSquare::center
```

7.7.1.3 contours

```
std::vector<cv::Point> GridSquare::contours
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_camera.hpp

7.8 rapidxml::memory_pool< Ch >::header Struct Reference

Public Attributes

- char * previous_begin

7.8.1 Member Data Documentation

7.8.1.1 previous_begin

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::header::previous_begin
```

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

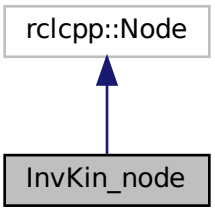
- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.9 InvKin_node Class Reference

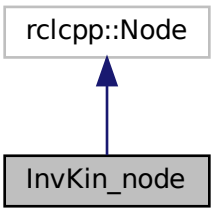
ROS 2 node for performing inverse kinematics.

`#include <node_invkin.hpp>`

Inheritance diagram for InvKin_node:



Collaboration diagram for InvKin_node:



Public Member Functions

- InvKin_node ()

  Construct a new InvKin_node object.

Private Member Functions

- void init_interfaces ()

  Initialize the timer, subscribers and publishers.
- void timer_callback ()

  Timer callback, actions that will be done at every iterations.
- void get_pose (const geometry_msgs::msg::Pose::SharedPtr msg)

  Get the targeted position.

- void get_grip (const std_msgs::msg::Float32::SharedPtr msg)

  Get the targeted grip.
- void get_state (double x, double y, double z)

  Processed the joints' states required to reach the desired pose, using an inverse kinematics algorithm.
- void correct_angle (Eigen::VectorXd &q)

  Put joints' angles in [-pi,pi].

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- map< int, double > state
- float L = 0.15
- double last_x
- double last_y
- double last_z
- int ROLL =6
- int PITCH =7
- double target_yaw
- double last_yaw
- double target_pitch
- double last_pitch
- double target_roll
- double last_roll
- double target_grip
- double lats_grip
- string urdf_file = ament_index_cpp::get_package_share_directory("umi_rtx_controller")+"/urdf/umi_rtx.urdf"
- pinocchio::Model model
- pinocchio::Data data
- const int JOINT_ID = 6
- const double eps = 1e-2
- const int IT_MAX = 1000
- const double DT = 1e-2
- const double damp = 1e-12
- pinocchio::Data::Matrix6x J
- Eigen::VectorXd q
- rclcpp::TimerBase::SharedPtr timer_
- rclcpp::Subscription< geometry_msgs::msg::Pose >::SharedPtr pose_subscription
- rclcpp::Subscription< std_msgs::msg::Float32 >::SharedPtr grip_subscription
- rclcpp::Publisher< sensor_msgs::msg::JointState >::SharedPtr angles_publisher

7.9.1 Detailed Description

ROS 2 node for performing inverse kinematics.

This class represents a ROS 2 node that performs inverse kinematics to compute joint states required for a robotic arm to reach a specified target pose and grip. It uses the Pinocchio library to handle kinematics calculations based on the URDF model of the robotic arm.

The `InvKin_node` class inherits from `rclcpp::Node` and provides functionality to:

- Initialize ROS 2 interfaces including publishers and subscribers.

- Read the URDF description of the robotic arm for kinematics calculations.
- Compute the required joint angles to achieve a given target pose and grip using inverse kinematics.
- Publish the computed joint states.

It includes methods for:

- Handling incoming target pose and grip commands.
- Calculating joint states through inverse kinematics.
- Correcting joint angles to be within a specified range.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 InvKin_node()

```
InvKin_node::InvKin_node ( )  [inline]
```

Construct a new InvKin_node object.

7.9.3 Member Function Documentation

7.9.3.1 correct_angle()

```
void InvKin_node::correct_angle (
            Eigen::VectorXd & q )  [private]
```

Put joints' angles in [-pi,pi].

Parameters

q	

7.9.3.2 get_grip()

```
void InvKin_node::get_grip (
            const std_msgs::msg::Float32::SharedPtr msg )  [private]
```

Get the targeted grip.

Parameters

msg	

7.9.3.3 get_pose()

```
void InvKin_node::get_pose (
            const geometry_msgs::msg::Pose::SharedPtr msg )  [private]
```

Get the targeted position.

Parameters

msg	

7.9.3.4 get_state()

```
void InvKin_node::get_state (
            double x,
            double y,
            double z )  [private]
```

Processed the joints' states required to reach the desired pose, using an inverse kinematics algorithm.

Parameters

x	Targeted x
y	Targeted y
z	Targeted z

7.9.3.5 init_interfaces()

```
void InvKin_node::init_interfaces ( )  [private]
```

Initialize the timer, subscribers and publishers.

7.9.3.6 timer_callback()

```
void InvKin_node::timer_callback ( )  [private]
```

Timer callback, actions that will be done at every iterations.

7.9.4 Member Data Documentation

7.9.4.1 angles_publisher

```
rclcpp::Publisher<sensor_msgs::msg::JointState>::SharedPtr InvKin_node::angles_publisher
[private]
```

7.9.4.2 damp

```
const double InvKin_node::damp = 1e-12  [private]
```

7.9.4.3 data

```
pinocchio::Data InvKin_node::data  [private]
```

7.9.4.4 DT

```
const double InvKin_node::DT = 1e-2  [private]
```

7.9.4.5 eps

```
const double InvKin_node::eps = 1e-2  [private]
```

7.9.4.6 grip_subscription

```
rclcpp::Subscription<std_msgs::msg::Float32>::SharedPtr InvKin_node::grip_subscription  [private]
```

7.9.4.7 IT_MAX

```
const int InvKin_node::IT_MAX = 1000  [private]
```

7.9.4.8 J

```
pinocchio::Data::Matrix6x InvKin_node::J  [private]
```

7.9.4.9 JOINT_ID

```
const int InvKin_node::JOINT_ID = 6  [private]
```

7.9.4.10 L

```
float InvKin_node::L = 0.15  [private]
```

7.9.4.11 last_pitch

```
double InvKin_node::last_pitch  [private]
```

7.9.4.12 last_roll

```
double InvKin_node::last_roll  [private]
```

7.9.4.13 last_x

```
double InvKin_node::last_x  [private]
```

7.9.4.14 last_y

```
double InvKin_node::last_y  [private]
```

7.9.4.15 last_yaw

```
double InvKin_node::last_yaw  [private]
```

7.9.4.16 last_z

```
double InvKin_node::last_z  [private]
```

7.9.4.17 lats_grip

```
double InvKin_node::lats_grip  [private]
```

7.9.4.18 loop_dt_

```
std::chrono::milliseconds InvKin_node::loop_dt_ = 40ms  [private]
```

7.9.4.19 model

```
pinocchio::Model InvKin_node::model  [private]
```

7.9.4.20 PITCH

```
int InvKin_node::PITCH =7  [private]
```

7.9.4.21 pose_subscription

```
rclcpp::Subscription<geometry_msgs::msg::Pose>::SharedPtr InvKin_node::pose_subscription
[private]
```

7.9.4.22 q

```
Eigen::VectorXd InvKin_node::q  [private]
```

7.9.4.23 ROLL

```
int InvKin_node::ROLL =6  [private]
```

7.9.4.24 state

```
map<int,double> InvKin_node::state  [private]
```

7.9.4.25 target_grip

```
double InvKin_node::target_grip  [private]
```

7.9.4.26 target_pitch

```
double InvKin_node::target_pitch  [private]
```

7.9.4.27 target_roll

```
double InvKin_node::target_roll  [private]
```

7.9.4.28 target_yaw

```
double InvKin_node::target_yaw  [private]
```

7.9.4.29 timer_

```
rclcpp::TimerBase::SharedPtr InvKin_node::timer_  [private]
```

7.9.4.30 urdf_file

```
string InvKin_node::urdf_file = ament_index_cpp::get_package_share_directory("umi_rtx_controller")+"/urdf/umi
_rtx.urdf"  [private]
```

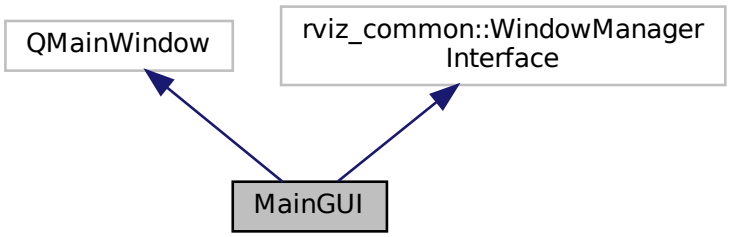
The documentation for this class was generated from the following files:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_invkin.hpp
- ros2_ws/src/umi_rtx_controller/src/node_invkin.cpp

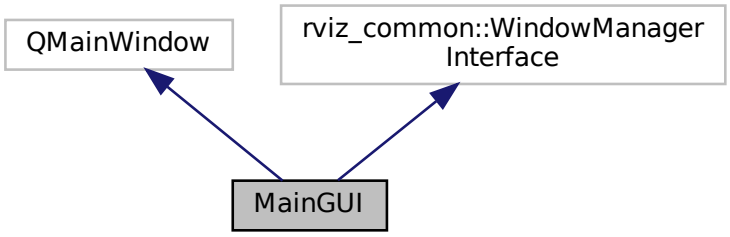
7.10 MainGUI Class Reference

```
#include <main_gui.hpp>
```

Inheritance diagram for MainGUI:



Collaboration diagram for MainGUI:



Public Member Functions

- MainGUI (QApplication *app, const std::shared_ptr< Objective_node > &ros2_node, rviz_common::ros_↩ integration::RosNodeAbstractionIface::WeakPtr rviz_ros_node, QWidget *parent=nullptr)

  Construct a new MainGUI object.
- ~MainGUI () override

  Destroy the Main GUI object.
- QWidget * getParentWindow () override

  Get the Parent Window object, override of the QMainWindow property, necessary for compilation but useless here.
- rviz_common::PanelDockWidget * addPane (const QString &name, QWidget *pane, Qt::DockWidgetArea area, bool floating) override

  Add a DockWidget to our window, override of the QMainWindow property, necessary for compilation but useless here.
- void setStatus (const QString &message) override

  Set the Status object, override of the QMainWindow property, necessary for compilation but useless here.

Public Attributes

- double x =0.
- double y =0.6
- double z =0.6
- double yaw =0.
- double pitch =0.
- double roll =0.
- double grip =0.02
- double raw_yaw =0.
- bool is_started_game = false
- bool manual_on = true
- int frame_stream = 0

Protected Member Functions

- void resizeEvent (QResizeEvent *event) override

  Handles the resizing of the widget and updates the displayed images.

Private Slots

- void closeEvent (QCloseEvent *event)

  Event necessary to compile, close the window and shutdown the ros node.
- void updateFrameAndInterface ()

  Update the frame and interface elements based on new camera data.
- void addSlider (QGridLayout *layout, const QString &label, QSlider *&slider, QDoubleSpinBox *&spinBox, int min, int max, int singleStep, int value)

  Add a slider with its associated label and spin box to the layout.
- void connectSlidersWithSpinBoxes ()

  Connect sliders with spin boxes for synchronized updates.

Private Member Functions

- void initializeRViz ()

  Initialise the RViz2 object, in order to integrate in our interface.

Private Attributes

- const shared_ptr< Objective_node > ros2_node
- QApplication * app_
- QWidget * main_widget
- QPushButton * switchButton
- QImage * image
- QLabel * Title
- QLabel * videoLabel
- QTimer * timer
- QPushButton * gameButton
- QDoubleSpinBox * spinBox_x
- QDoubleSpinBox * spinBox_y
- QDoubleSpinBox * spinBox_z
- QDoubleSpinBox * spinBox_yaw
- QDoubleSpinBox * spinBox_pitch
- QDoubleSpinBox * spinBox_roll
- QDoubleSpinBox * spinBox_grip
- QHBoxLayout * main_layout
- QVBoxLayout * game_layout
- QGridLayout * board_layout
- QVBoxLayout * info_layout
- QVBoxLayout * history_layout
- QVBoxLayout * umi_layout
- QLabel * turn_label
- QLabel * player_label
- QLabel * info_labels [9]
- QLabel * board_labels [3][3]
- int board [3][3]
- std::vector< std::string > msgs
- QPixmap case0
- QPixmap case1
- QPixmap case2
- rviz_common::RenderPanel * render_panel_
- rviz_common::Display * TF_
- rviz_common::Display * Model_
- rviz_common::VisualizationManager * manager_
- rviz_common::ros_integration::RosNodeAbstractionIface::WeakPtr rviz_ros_node_
- cv::Mat * frame

7.10.1 Constructor & Destructor Documentation

7.10.1.1 MainGUI()

```
MainGUI::MainGUI (
            QApplication * app,
            const std::shared_ptr< Objective_node > & ros2_node,
            rviz_common::ros_integration::RosNodeAbstractionIface::WeakPtr rviz_ros_node,
            QWidget * parent = nullptr )
```

Construct a new MainGUI object.

Initializes the main GUI components, including the widgets, layouts, sliders, buttons, and integrates RViz for 3D visualization.

Parameters

app	QApplication object that will be used for the GUI
ros2_node	The command node that works in pair with this interface
rviz_ros_node	The RViz ROS node that is necessary to run RViz2 in our interface
parent	

7.10.1.2 ∼MainGUI()

```
MainGUI::∼MainGUI ( )  [override]
```

Destroy the Main GUI object.

7.10.2 Member Function Documentation

7.10.2.1 addPane()

```
rviz_common::PanelDockWidget * MainGUI::addPane (
            const QString & name,
            QWidget * pane,
            Qt::DockWidgetArea area,
            bool floating )  [override]
```

Add a DockWidget to our window, override of the QMainWindow property, necessary for compilation but useless here.

Parameters

name	Name of the new DockWidget
pane	Type of the desired DOckWidget
area	Size of the Widget
floating	

Returns

rviz_common::PanelDockWidget∗

7.10.2.2 addSlider

```
void MainGUI::addSlider (
            QGridLayout * layout,
```

```
            const QString & label,
            QSlider *& slider,
            QDoubleSpinBox *& spinBox,
            int min,
            int max,
            int singleStep,
            int value )  [private], [slot]
```

Add a slider with its associated label and spin box to the layout.

Parameters

layout	Layout where the slider will be added.
labelText	Text for the slider label.
slider	Pointer to the QSlider instance.
spinBox	Pointer to the QSpinBox instance.
min	Minimum value for the slider.
max	Maximum value for the slider.
step	Step value for the slider.
value	Initial value for the slider.

7.10.2.3 closeEvent

```
void MainGUI::closeEvent (
            QCloseEvent * event )  [private], [slot]
```

Event necessary to compile, close the window and shutdown the ros node.

Parameters

event	

7.10.2.4 connectSlidersWithSpinBoxes

```
void MainGUI::connectSlidersWithSpinBoxes ( )  [private], [slot]
```

Connect sliders with spin boxes for synchronized updates.

7.10.2.5 getParentWindow()

```
QWidget * MainGUI::getParentWindow ( )  [override]
```

Get the Parent Window object, override of the QMainWindow property, necessary for compilation but useless here.

Returns

QWidget*

7.10.2.6 initializeRViz()

```
void MainGUI::initializeRViz ( )  [private]
```

Initialise the RViz2 object, in order to integrate in our interface.

7.10.2.7 resizeEvent()

```
void MainGUI::resizeEvent (
            QResizeEvent * event )  [inline], [override], [protected]
```

Handles the resizing of the widget and updates the displayed images.

This function is called automatically when the widget is resized. It performs the following actions:

- Loads images for different board states from the package directory.
- Scales these images to fit the size of the board labels while maintaining their aspect ratio.
- Updates the pixmap for each board label based on the current state of the board.
- Scales and updates the video feed image to fit the size of the video label while maintaining its aspect ratio.

Parameters

event	A pointer to the QResizeEvent object that contains information about the resize event.

Note

This function uses `ament_index_cpp::get_package_share_directory` to obtain the directory of the package and load images from it. It assumes that the images are located in the "images" subdirectory of the package's share directory.

7.10.2.8 setStatus()

```
void MainGUI::setStatus (
            const QString & message )  [override]
```

Set the Status object, override of the QMainWindow property, necessary for compilation but useless here.

Parameters

message	

7.10.2.9 updateFrameAndInterface

```
void MainGUI::updateFrameAndInterface ( )  [private], [slot]
```

Update the frame and interface elements based on new camera data.

7.10.3 Member Data Documentation

7.10.3.1 app_

```
QApplication* MainGUI::app_  [private]
```

7.10.3.2 board

```
int MainGUI::board[3][3]  [private]
```

7.10.3.3 board_labels

```
QLabel* MainGUI::board_labels[3][3]  [private]
```

7.10.3.4 board_layout

```
QGridLayout* MainGUI::board_layout  [private]
```

7.10.3.5 case0

```
QPixmap MainGUI::case0  [private]
```

7.10.3.6 case1

```
QPixmap MainGUI::case1  [private]
```

7.10.3.7 case2

```
QPixmap MainGUI::case2  [private]
```

7.10.3.8 frame

```
cv::Mat* MainGUI::frame  [private]
```

7.10.3.9 frame_stream

```
int MainGUI::frame_stream = 0
```

7.10.3.10 game_layout

```
QVBoxLayout* MainGUI::game_layout  [private]
```

7.10.3.11 gameButton

```
QPushButton* MainGUI::gameButton  [private]
```

7.10.3.12 grip

```
double MainGUI::grip =0.02
```

7.10.3.13 history_layout

```
QVBoxLayout* MainGUI::history_layout  [private]
```

7.10.3.14 image

```
QImage* MainGUI::image  [private]
```

7.10.3.15 info_labels

```
QLabel* MainGUI::info_labels[9]  [private]
```

7.10.3.16 info_layout

```
QVBoxLayout* MainGUI::info_layout  [private]
```

7.10.3.17 is_started_game

```
bool MainGUI::is_started_game = false
```

7.10.3.18 main_layout

```
QHBoxLayout* MainGUI::main_layout  [private]
```

7.10.3.19 main_widget

```
QWidget* MainGUI::main_widget  [private]
```

7.10.3.20 manager_

```
rviz_common::VisualizationManager* MainGUI::manager_  [private]
```

7.10.3.21 manual_on

```
bool MainGUI::manual_on = true
```

7.10.3.22 Model_

```
rviz_common::Display * MainGUI::Model_  [private]
```

7.10.3.23 msgs

```
std::vector<std::string> MainGUI::msgs  [private]
```

7.10.3.24 pitch

```
double MainGUI::pitch =0.
```

7.10.3.25 player_label

```
QLabel* MainGUI::player_label  [private]
```

7.10.3.26 raw_yaw

```
double MainGUI::raw_yaw =0.
```

7.10.3.27 render_panel_

```
rviz_common::RenderPanel* MainGUI::render_panel_  [private]
```

7.10.3.28 roll

```
double MainGUI::roll =0.
```

7.10.3.29 ros2_node

```
const shared_ptr<Objective_node> MainGUI::ros2_node  [private]
```

7.10.3.30 rviz_ros_node_

```
rviz_common::ros_integration::RosNodeAbstractionIface::WeakPtr MainGUI::rviz_ros_node_  [private]
```

7.10.3.31 spinBox_grip

```
QDoubleSpinBox * MainGUI::spinBox_grip  [private]
```

7.10.3.32 spinBox_pitch

```
QDoubleSpinBox * MainGUI::spinBox_pitch  [private]
```

7.10.3.33 spinBox_roll

```
QDoubleSpinBox * MainGUI::spinBox_roll  [private]
```

7.10.3.34 spinBox_x

```
QDoubleSpinBox* MainGUI::spinBox_x  [private]
```

7.10.3.35 spinBox_y

```
QDoubleSpinBox * MainGUI::spinBox_y  [private]
```

7.10.3.36 spinBox_yaw

```
QDoubleSpinBox * MainGUI::spinBox_yaw  [private]
```

7.10.3.37 spinBox_z

```
QDoubleSpinBox * MainGUI::spinBox_z  [private]
```

7.10.3.38 switchButton

```
QPushButton* MainGUI::switchButton  [private]
```

7.10.3.39 TF_

```
rviz_common::Display* MainGUI::TF_  [private]
```

7.10.3.40 timer

```
QTimer* MainGUI::timer  [private]
```

7.10.3.41 Title

```
QLabel* MainGUI::Title  [private]
```

7.10.3.42 turn_label

```
QLabel* MainGUI::turn_label  [private]
```

7.10.3.43 umi_layout

```
QVBoxLayout* MainGUI::umi_layout  [private]
```

7.10.3.44 videoLabel

```
QLabel* MainGUI::videoLabel  [private]
```

7.10.3.45 x

```
double MainGUI::x =0.
```

7.10.3.46 y

```
double MainGUI::y =0.6
```

7.10.3.47 yaw

```
double MainGUI::yaw =0.
```

7.10.3.48 z

```
double MainGUI::z =0.6
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/main_gui.hpp
- ros2_ws/src/umi_rtx_controller/src/main_gui.cpp

7.11 rapidxml::memory_pool< Ch > Class Template Reference

```
#include <rapidxml.hpp>
```

Classes

- struct header

Public Member Functions

- memory_pool ()

  Constructs empty pool with default allocator functions.
- ~memory_pool ()
- xml_node< Ch > * allocate_node (node_type type, const Ch *name=0, const Ch *value=0, std::size_t name_size=0, std::size_t value_size=0)
- xml_attribute< Ch > * allocate_attribute (const Ch *name=0, const Ch *value=0, std::size_t name_size=0, std::size_t value_size=0)
- Ch * allocate_string (const Ch *source=0, std::size_t size=0)
- xml_node< Ch > * clone_node (const xml_node< Ch > *source, xml_node< Ch > *result=0)
- void clear ()
- void set_allocator (alloc_func *af, free_func *ff)

Private Member Functions

- void init ()
- char * align (char *ptr)
- char * allocate_raw (std::size_t size)
- void * allocate_aligned (std::size_t size)

Private Attributes

- char * m_begin
- char * m_ptr
- char * m_end
- char m_static_memory [RAPIDXML_STATIC_POOL_SIZE]
- alloc_func * m_alloc_func
- free_func * m_free_func

7.11.1 Detailed Description

template<class Ch = char>
class rapidxml::memory_pool< Ch >

This class is used by the parser to create new nodes and attributes, without overheads of dynamic memory allocation. In most cases, you will not need to use this class directly. However, if you need to create nodes manually or modify names/values of nodes, you are encouraged to use memory_pool of relevant xml_document to allocate the memory. Not only is this faster than allocating them by using `new` operator, but also their lifetime will be tied to the lifetime of document, possibly simplyfing memory management.

Call allocate_node() or allocate_attribute() functions to obtain new nodes or attributes from the pool. You can also call allocate_string() function to allocate strings. Such strings can then be used as names or values of nodes without worrying about their lifetime. Note that there is no `free()` function – all allocations are freed at once when clear() function is called, or when the pool is destroyed.

It is also possible to create a standalone memory_pool, and use it to allocate nodes, whose lifetime will not be tied to any document.

Pool maintains `RAPIDXML_STATIC_POOL_SIZE` bytes of statically allocated memory. Until static memory is exhausted, no dynamic memory allocations are done. When static memory is exhausted, pool allocates additional blocks of memory of size `RAPIDXML_DYNAMIC_POOL_SIZE` each, by using global `new[]` and `delete[]` operators. This behaviour can be changed by setting custom allocation routines. Use set_allocator() function to set them.

Allocations for nodes, attributes and strings are aligned at `RAPIDXML_ALIGNMENT` bytes. This value defaults to the size of pointer on target architecture.

To obtain absolutely top performance from the parser, it is important that all nodes are allocated from a single, contiguous block of memory. Otherwise, cache misses when jumping between two (or more) disjoint blocks of memory can slow down parsing quite considerably. If required, you can tweak `RAPIDXML_STATIC_POOL_SIZE`, `RAPIDXML_DYNAMIC_POOL_SIZE` and `RAPIDXML_ALIGNMENT` to obtain best wasted memory to performance compromise. To do it, define their values before rapidxml.hpp file is included.

Parameters

Ch	Character type of created nodes.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 memory_pool()

```
template<class Ch = char>
rapidxml::memory_pool< Ch >::memory_pool ( )  [inline]
```

Constructs empty pool with default allocator functions.

7.11.2.2 ~memory_pool()

```
template<class Ch = char>
rapidxml::memory_pool< Ch >::~memory_pool ( )  [inline]
```

Destroys pool and frees all the memory. This causes memory occupied by nodes allocated by the pool to be freed. Nodes allocated from the pool are no longer valid.

7.11.3 Member Function Documentation

7.11.3.1 align()

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::align (
            char * ptr )  [inline], [private]
```

7.11.3.2 allocate_aligned()

```
template<class Ch = char>
void* rapidxml::memory_pool< Ch >::allocate_aligned (
            std::size_t size )  [inline], [private]
```

7.11.3.3 allocate_attribute()

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::memory_pool< Ch >::allocate_attribute (
            const Ch * name = 0,
            const Ch * value = 0,
            std::size_t name_size = 0,
            std::size_t value_size = 0 )  [inline]
```

Allocates a new attribute from the pool, and optionally assigns name and value to it. If the allocation request cannot be accomodated, this function will throw `std::bad_alloc`. If exceptions are disabled by defining RAPIDXML↩ _NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

name	Name to assign to the attribute, or 0 to assign no name.
value	Value to assign to the attribute, or 0 to assign no value.
name_size	Size of name to assign, or 0 to automatically calculate size from name string.
value_size	Size of value to assign, or 0 to automatically calculate size from value string.

Returns

Pointer to allocated attribute. This pointer will never be NULL.

7.11.3.4 allocate_node()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::memory_pool< Ch >::allocate_node (
            node_type type,
            const Ch * name = 0,
            const Ch * value = 0,
            std::size_t name_size = 0,
            std::size_t value_size = 0 )  [inline]
```

Allocates a new node from the pool, and optionally assigns name and value to it. If the allocation request cannot be accomodated, this function will throw `std::bad_alloc`. If exceptions are disabled by defining RAPIDXML↩ _NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

type	Type of node to create.
name	Name to assign to the node, or 0 to assign no name.
value	Value to assign to the node, or 0 to assign no value.
name_size	Size of name to assign, or 0 to automatically calculate size from name string.
value_size	Size of value to assign, or 0 to automatically calculate size from value string.

Returns

Pointer to allocated node. This pointer will never be NULL.

7.11.3.5 allocate_raw()

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::allocate_raw (
            std::size_t size )  [inline], [private]
```

7.11.3.6 allocate_string()

```
template<class Ch = char>
Ch* rapidxml::memory_pool< Ch >::allocate_string (
            const Ch * source = 0,
            std::size_t size = 0 )  [inline]
```

Allocates a char array of given size from the pool, and optionally copies a given string to it. If the allocation request cannot be accomodated, this function will throw `std::bad_alloc`. If exceptions are disabled by defining RAPIDXML_NO_EXCEPTIONS, this function will call rapidxml::parse_error_handler() function.

Parameters

source	String to initialize the allocated memory with, or 0 to not initialize it.
size	Number of characters to allocate, or zero to calculate it automatically from source string length; if size is 0, source string must be specified and null terminated.

Returns

Pointer to allocated char array. This pointer will never be NULL.

7.11.3.7 clear()

```
template<class Ch = char>
void rapidxml::memory_pool< Ch >::clear ( )  [inline]
```

Clears the pool. This causes memory occupied by nodes allocated by the pool to be freed. Any nodes or strings allocated from the pool will no longer be valid.

7.11.3.8 clone_node()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::memory_pool< Ch >::clone_node (
            const xml_node< Ch > * source,
            xml_node< Ch > * result = 0 )  [inline]
```

Clones an xml_node and its hierarchy of child nodes and attributes. Nodes and attributes are allocated from this memory pool. Names and values are not cloned, they are shared between the clone and the source. Result node can be optionally specified as a second parameter, in which case its contents will be replaced with cloned source node. This is useful when you want to clone entire document.

Parameters

source	Node to clone.
result	Node to put results in, or 0 to automatically allocate result node

Returns

Pointer to cloned node. This pointer will never be NULL.

7.11.3.9 init()

```
template<class Ch = char>
void rapidxml::memory_pool< Ch >::init ( )  [inline], [private]
```

7.11.3.10 set_allocator()

```
template<class Ch = char>
void rapidxml::memory_pool< Ch >::set_allocator (
            alloc_func * af,
            free_func * ff )  [inline]
```

Sets or resets the user-defined memory allocation functions for the pool. This can only be called when no memory is allocated from the pool yet, otherwise results are undefined. Allocation function must not return invalid pointer on failure. It should either throw, stop the program, or use `longjmp()` function to pass control to other place of program. If it returns invalid pointer, results are undefined.

User defined allocation functions must have the following forms:

```
void *allocate(std::size_t size);
void free(void *pointer);
```

Parameters

af	Allocation function, or 0 to restore default function
ff	Free function, or 0 to restore default function

7.11.4 Member Data Documentation

7.11.4.1 m_alloc_func

```
template<class Ch = char>
alloc_func* rapidxml::memory_pool< Ch >::m_alloc_func  [private]
```

7.11.4.2 m_begin

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::m_begin  [private]
```

7.11.4.3 m_end

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::m_end  [private]
```

7.11.4.4 m_free_func

```
template<class Ch = char>
free_func* rapidxml::memory_pool< Ch >::m_free_func  [private]
```

7.11.4.5 m_ptr

```
template<class Ch = char>
char* rapidxml::memory_pool< Ch >::m_ptr  [private]
```

7.11.4.6 m_static_memory

```
template<class Ch = char>
char rapidxml::memory_pool< Ch >::m_static_memory[RAPIDXML_STATIC_POOL_SIZE]  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.12 Move_msg Struct Reference

```
#include <node_game.hpp>
```

Public Attributes

- std::string msg
- int box

7.12.1 Member Data Documentation

7.12.1.1 box

```
int Move_msg::box
```

7.12.1.2 msg

```
std::string Move_msg::msg
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_game.hpp

7.13 rapidxml::xml_document< Ch >::node_name_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.13.1 Member Function Documentation

7.13.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::node_name_pred::test (
            Ch ch )  [inline], [static]
```

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

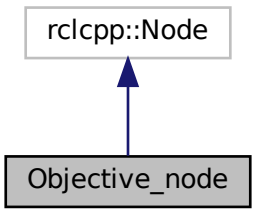
- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.14 Objective_node Class Reference

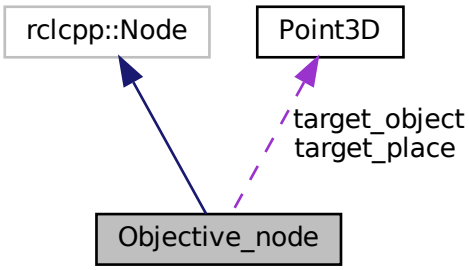
Manages and sends commands to a robotic arm based on image and game data.

`#include <node_commands.hpp>`

Inheritance diagram for Objective_node:



Collaboration diagram for Objective_node:



Public Member Functions

- Objective_node ()

  Construct a new Objective_node object.
- void update_state (double new_x, double new_y, double new_z, double new_yaw, double new_pitch, double new_roll, double new_grip)

  Function that will be used at each iteration in the GUI to update the target that will be communicated.

Public Attributes

- string mode ="manual"
- cv::Mat processed_frame
- cv::Mat depth_frame
- Point3D target_object
- Point3D target_place
- double x =0.
- double y =0.6
- double z =0.6
- double yaw =0.
- double pitch =0.
- double roll =0.
- double grip =0.8
- float t =0
- float dt =0.04
- bool is_robot_turn = false
- int board [3][3]
- std::string moves_history [9]
- std::string primary_msg = ""
- std::string secondary_msg = ""
- bool need_update

Private Member Functions

- void init_interfaces ()

  Initialize the timer, subscribers and publishers.
- void timer_callback ()

  Timer callback function for controlling the robotic arm.
- void get_processed_image (const sensor_msgs::msg::Image::SharedPtr msg)

  Get the processed images published.
- void get_depth_image (const sensor_msgs::msg::Image::SharedPtr msg)

  Get the depth images published.
- void get_robot_next_move (const std_msgs::msg::Int32::SharedPtr msg)

  Get the next move for the robot.
- void get_game_data (const umi_rtx_interfaces::msg::GameData::SharedPtr msg)

  Get the game data published.

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- int robot_next_move
- bool is_game_started = false
- int count
- double x0
- double y0
- double z0
- double yaw0
- double pitch0
- double roll0
- double t0
- double processed_x

- double processed_y
- double processed_z
- double processed_yaw
- double processed_pitch
- double processed_roll
- double target_x
- double target_y
- double target_z
- double final_x
- double final_y
- double final_z
- bool is_initialized =false
- rclcpp::TimerBase::SharedPtr timer_
- rclcpp::Publisher< geometry_msgs::msg::Pose >::SharedPtr pose_publisher
- rclcpp::Publisher< std_msgs::msg::Float32 >::SharedPtr grip_publisher
- rclcpp::Publisher< std_msgs::msg::Bool >::SharedPtr capture_step_publisher
- rclcpp::Subscription< geometry_msgs::msg::Pose >::SharedPtr pose_subscriber
- rclcpp::Subscription< sensor_msgs::msg::Image >::SharedPtr processed_image_subscriber
- rclcpp::Subscription< sensor_msgs::msg::Image >::SharedPtr depth_image_subscriber
- rclcpp::Subscription< umi_rtx_interfaces::msg::GameData >::SharedPtr game_data_subscriber
- rclcpp::Subscription< std_msgs::msg::Int32 >::SharedPtr robot_next_move_subscriber

7.14.1 Detailed Description

Manages and sends commands to a robotic arm based on image and game data.

The Objective_node class handles the publication of target poses, grip parameters, and step capture signals. It subscribes to image data and game data, processes this information, and adjusts the arm's actions accordingly. The class is designed to break down the robot's movements into multiple steps to ensure precise and accurate control.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 Objective_node()

```
Objective_node::Objective_node ( )  [inline]
```

Construct a new Objective_node object.

Initializes the Objective_node with default values, sets up interfaces, and initializes the game board and moves history that will be used for the interface.

7.14.3 Member Function Documentation

7.14.3.1 get_depth_image()

```
void Objective_node::get_depth_image (
            const sensor_msgs::msg::Image::SharedPtr msg )  [private]
```

Get the depth images published.

This function is a callback that processes the depth image received from a ROS topic. It converts the ROS image message to an OpenCV image and stores it in the `depth_frame` member variable.

Parameters

msg	The ROS image message containing the depth image.

7.14.3.2 get_game_data()

```
void Objective_node::get_game_data (
            const umi_rtx_interfaces::msg::GameData::SharedPtr msg )  [private]
```

Get the game data published.

This function is a callback that processes the game data received from a ROS topic. It updates the internal state of the `Objective_node` based on the received game data.

Parameters

msg	The ROS message containing the game data.

7.14.3.3 get_processed_image()

```
void Objective_node::get_processed_image (
            const sensor_msgs::msg::Image::SharedPtr msg )  [private]
```

Get the processed images published.

This function is a callback that processes the processed image received from a ROS topic. It converts the ROS image message to an OpenCV image and stores it in the `processed_frame` member variable.

Parameters

msg	The ROS image message containing the processed image.

7.14.3.4 get_robot_next_move()

```
void Objective_node::get_robot_next_move (
            const std_msgs::msg::Int32::SharedPtr msg )  [private]
```

Get the next move for the robot.

This function is a callback that processes the next move data received from a ROS topic. It updates the `robot_next_move` member variable with the data from the received message.

Parameters

msg	The ROS message containing the box where the robot will play.

7.14.3.5 init_interfaces()

```
void Objective_node::init_interfaces ( )  [private]
```

Initialize the timer, subscribers and publishers.

7.14.3.6 timer_callback()

```
void Objective_node::timer_callback ( )  [private]
```

Timer callback function for controlling the robotic arm.

This function is executed at regular intervals and handles the state updates and movements of the robotic arm. It adjusts the arm's position, orientation, and grip based on the current mode and game state. The function also publishes updated pose, grip, and capture step messages.

The function performs the following steps:

- Moves the robotic arm to a capture position.
- Waits for a brief period.
- If it is the robot's turn and the game has started:
  - Moves the arm to the detected pawn position.
  - Closes the grip to grab the pawn.
  - Raises the arm.
  - Moves the arm to the target placement position.
  - Lowers the arm.
  - Opens the grip to place the pawn.
  - Returns the arm to the capture position.
- If the arm is in manual mode, it adapts the origin pose for automatic procedures.

Note

This function is part of the state machine controlling the robotic arm.

7.14.3.7 update_state()

```
void Objective_node::update_state (
            double new_x,
            double new_y,
            double new_z,
            double new_yaw,
            double new_pitch,
            double new_roll,
            double new_grip )
```

Function that will be used at each iteration in the GUI to update the target that will be communicated.

This function updates the target state parameters including position, orientation, and grip, which will be communicated to the robotic arm.

Parameters

new_x	New x-coordinate of the target position.
new_y	New y-coordinate of the target position.
new_z	New z-coordinate of the target position.
new_yaw	New yaw angle of the target orientation.
new_pitch	New pitch angle of the target orientation.
new_roll	New roll angle of the target orientation.
new_grip	New grip parameter for the robotic arm.

7.14.4 Member Data Documentation

7.14.4.1 board

```
int Objective_node::board[3][3]
```

7.14.4.2 capture_step_publisher

```
rclcpp::Publisher<std_msgs::msg::Bool>::SharedPtr Objective_node::capture_step_publisher
[private]
```

7.14.4.3 count

```
int Objective_node::count  [private]
```

7.14.4.4 depth_frame

```
cv::Mat Objective_node::depth_frame
```

7.14.4.5 depth_image_subscriber

```
rclcpp::Subscription<sensor_msgs::msg::Image>::SharedPtr Objective_node::depth_image_subscriber
[private]
```

7.14.4.6 dt

```
float Objective_node::dt =0.04
```

7.14.4.7 final_x

```
double Objective_node::final_x  [private]
```

7.14.4.8 final_y

```
double Objective_node::final_y  [private]
```

7.14.4.9 final_z

```
double Objective_node::final_z  [private]
```

7.14.4.10 game_data_subscriber

```
rclcpp::Subscription<umi_rtx_interfaces::msg::GameData>::SharedPtr Objective_node::game_data↩
_subscriber  [private]
```

7.14.4.11 grip

```
double Objective_node::grip =0.8
```

7.14.4.12 grip_publisher

```
rclcpp::Publisher<std_msgs::msg::Float32>::SharedPtr Objective_node::grip_publisher  [private]
```

7.14.4.13 is_game_started

```
bool Objective_node::is_game_started = false  [private]
```

7.14.4.14 is_initialized

```
bool Objective_node::is_initialized =false  [private]
```

7.14.4.15 is_robot_turn

```
bool Objective_node::is_robot_turn = false
```

7.14.4.16 loop_dt_

```
std::chrono::milliseconds Objective_node::loop_dt_ = 40ms  [private]
```

7.14.4.17 mode

```
string Objective_node::mode ="manual"
```

7.14.4.18 moves_history

```
std::string Objective_node::moves_history[9]
```

7.14.4.19 need_update

```
bool Objective_node::need_update
```

7.14.4.20 pitch

```
double Objective_node::pitch =0.
```

7.14.4.21 pitch0

```
double Objective_node::pitch0  [private]
```

7.14.4.22 pose_publisher

```
rclcpp::Publisher<geometry_msgs::msg::Pose>::SharedPtr Objective_node::pose_publisher  [private]
```

7.14.4.23 pose_subscriber

```
rclcpp::Subscription<geometry_msgs::msg::Pose>::SharedPtr Objective_node::pose_subscriber
[private]
```

7.14.4.24 primary_msg

```
std::string Objective_node::primary_msg = ""
```

7.14.4.25 processed_frame

```
cv::Mat Objective_node::processed_frame
```

7.14.4.26 processed_image_subscriber

```
rclcpp::Subscription<sensor_msgs::msg::Image>::SharedPtr Objective_node::processed_image_↩
subscriber  [private]
```

7.14.4.27 processed_pitch

```
double Objective_node::processed_pitch  [private]
```

7.14.4.28 processed_roll

```
double Objective_node::processed_roll  [private]
```

7.14.4.29 processed_x

```
double Objective_node::processed_x  [private]
```

7.14.4.30 processed_y

```
double Objective_node::processed_y  [private]
```

7.14.4.31 processed_yaw

```
double Objective_node::processed_yaw  [private]
```

7.14.4.32 processed_z

```
double Objective_node::processed_z  [private]
```

7.14.4.33 robot_next_move

```
int Objective_node::robot_next_move  [private]
```

7.14.4.34 robot_next_move_subscriber

```
rclcpp::Subscription<std_msgs::msg::Int32>::SharedPtr Objective_node::robot_next_move_subscriber
[private]
```

7.14.4.35 roll

```
double Objective_node::roll =0.
```

7.14.4.36 roll0

```
double Objective_node::roll0  [private]
```

7.14.4.37 secondary_msg

```
std::string Objective_node::secondary_msg = ""
```

7.14.4.38 t

```
float Objective_node::t =0
```

7.14.4.39 t0

```
double Objective_node::t0  [private]
```

7.14.4.40 target_object

```
Point3D Objective_node::target_object
```

7.14.4.41 target_place

```
Point3D Objective_node::target_place
```

7.14.4.42 target_x

```
double Objective_node::target_x  [private]
```

7.14.4.43 target_y

```
double Objective_node::target_y  [private]
```

7.14.4.44 target_z

```
double Objective_node::target_z  [private]
```

7.14.4.45 timer_

```
rclcpp::TimerBase::SharedPtr Objective_node::timer_  [private]
```

7.14.4.46 x

```
double Objective_node::x =0.
```

7.14.4.47 x0

```
double Objective_node::x0  [private]
```

7.14.4.48 y

```
double Objective_node::y =0.6
```

7.14.4.49 y0

```
double Objective_node::y0  [private]
```

7.14.4.50 yaw

```
double Objective_node::yaw =0.
```

7.14.4.51 yaw0

```
double Objective_node::yaw0  [private]
```

7.14.4.52 z

```
double Objective_node::z =0.6
```

7.14.4.53 z0

```
double Objective_node::z0  [private]
```

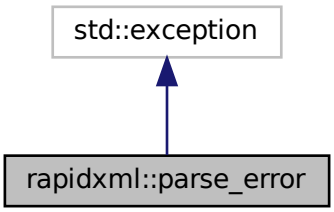
The documentation for this class was generated from the following files:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_commands.hpp
- ros2_ws/src/umi_rtx_controller/src/node_commands.cpp

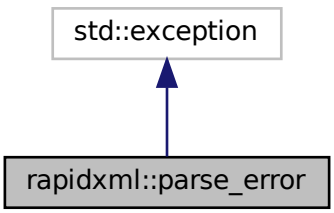
7.15 rapidxml::parse_error Class Reference

```
#include <rapidxml.hpp>
```

Inheritance diagram for rapidxml::parse_error:



Collaboration diagram for rapidxml::parse_error:



Public Member Functions

- parse_error (const char *what, void *where)
  Constructs parse error.
- virtual const char * what () const throw ()
- template<class Ch >
  Ch * where () const

Private Attributes

- const char * m_what
- void * m_where

7.15.1 Detailed Description

Parse error exception. This exception is thrown by the parser when an error occurs. Use what() function to get human-readable error message. Use where() function to get a pointer to position within source text where error was detected.

If throwing exceptions by the parser is undesirable, it can be disabled by defining RAPIDXML_NO_EXCEPTIONS macro before rapidxml.hpp is included. This will cause the parser to call rapidxml::parse_error_handler() function instead of throwing an exception. This function must be defined by the user.

This class derives from `std::exception` class.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 parse_error()

```
rapidxml::parse_error::parse_error (
            const char * what,
            void * where )  [inline]
```

Constructs parse error.

7.15.3 Member Function Documentation

7.15.3.1 what()

```
virtual const char* rapidxml::parse_error::what ( ) const throw ( )   [inline], [virtual]
```

Gets human readable description of error.

Returns

Pointer to null terminated description of the error.

7.15.3.2 where()

```
template<class Ch >
Ch* rapidxml::parse_error::where ( ) const  [inline]
```

Gets pointer to character data where error happened. Ch should be the same as char type of xml_document that produced the error.

Returns

Pointer to location within the parsed string where error occured.

7.15.4 Member Data Documentation

7.15.4.1 m_what

```
const char* rapidxml::parse_error::m_what  [private]
```

7.15.4.2 m_where

```
void* rapidxml::parse_error::m_where  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.16 Point3D Struct Reference

```
#include <node_commands.hpp>
```

Public Attributes

- double x
- double y
- double z

7.16.1 Member Data Documentation

7.16.1.1 x

```
double Point3D::x
```

7.16.1.2 y

```
double Point3D::y
```

7.16.1.3 z

```
double Point3D::z
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_commands.hpp

7.17 Position Struct Reference

```
#include <node_game.hpp>
```

Public Attributes

- int row
- int col

7.17.1 Member Data Documentation

7.17.1.1 col

```
int Position::col
```

7.17.1.2 row

```
int Position::row
```

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

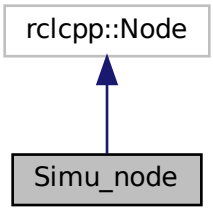
- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_game.hpp

7.18 Simu_node Class Reference

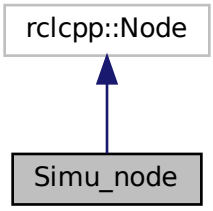
ROS 2 node for simulating a robotic arm.

`#include <node_simu.hpp>`

Inheritance diagram for Simu_node:



Collaboration diagram for Simu_node:



Public Member Functions

- Simu_node ()

  Construct a new Simu_node object.

Private Member Functions

- void init_interfaces ()

  Initialize the timer, subscribers and publishers.
- void timer_callback ()

  Timer callback, actions that will be done at every iterations.
- void init_urdf ()

  Read the URDF description of the arm, to get the joints informations.
- void get_commands (const sensor_msgs::msg::JointState::SharedPtr msg)

  Get the commands that will be sent to the arm.

Private Attributes

- std::chrono::milliseconds loop_dt_ = 40ms
- map< string, map< string, double > > dependent_joints
- map< string, map< string, double > > free_joints
- map< string, double > zeros
- vector< string > joint_list
- vector< string > names
- string urdf_file = ament_index_cpp::get_package_share_directory("umi_rtx_controller")+"/urdf/umi_rtx.urdf"
- rclcpp::TimerBase::SharedPtr timer_
- rclcpp::Subscription< sensor_msgs::msg::JointState >::SharedPtr invkin_subscriber
- rclcpp::Publisher< sensor_msgs::msg::JointState >::SharedPtr simu_publisher

7.18.1 Detailed Description

ROS 2 node for simulating a robotic arm.

This class represents a ROS 2 node designed for simulating the operation of a robotic arm. It handles the initialization of the node's interfaces, processes commands for joint movements, and manages the simulation of the robotic arm based on the URDF description.

The `Simu_node` class inherits from `rclcpp::Node` and provides functionality to:

- Initialize ROS 2 publishers and subscribers.
- Read the URDF description to extract joint information.
- Process commands for joint states and simulate their effects.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 Simu_node()

```
Simu_node::Simu_node ( )  [inline]
```

Construct a new Simu_node object.

7.18.3 Member Function Documentation

7.18.3.1 get_commands()

```
void Simu_node::get_commands (
            const sensor_msgs::msg::JointState::SharedPtr msg )  [private]
```

Get the commands that will be sent to the arm.

Parameters

msg	States of the joints required to reach the desired position sent through

7.18.3.2 init_interfaces()

```
void Simu_node::init_interfaces ( )  [private]
```

Initialize the timer, subscribers and publishers.

7.18.3.3 init_urdf()

```
void Simu_node::init_urdf ( )  [private]
```

Read the URDF description of the arm, to get the joints informations.

7.18.3.4 timer_callback()

```
void Simu_node::timer_callback ( )  [private]
```

Timer callback, actions that will be done at every iterations.

7.18.4 Member Data Documentation

7.18.4.1 dependent_joints

```
map<string,map<string, double> > Simu_node::dependent_joints  [private]
```

7.18.4.2 free_joints

```
map<string,map<string, double> > Simu_node::free_joints  [private]
```

7.18.4.3 invkin_subscriber

```
rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr Simu_node::invkin_subscriber
[private]
```

7.18.4.4 joint_list

```
vector<string> Simu_node::joint_list  [private]
```

7.18.4.5 loop_dt_

```
std::chrono::milliseconds Simu_node::loop_dt_ = 40ms  [private]
```

7.18.4.6 names

```
vector<string> Simu_node::names  [private]
```

7.18.4.7 simu_publisher

```
rclcpp::Publisher<sensor_msgs::msg::JointState>::SharedPtr Simu_node::simu_publisher  [private]
```

7.18.4.8 timer_

```
rclcpp::TimerBase::SharedPtr Simu_node::timer_  [private]
```

7.18.4.9 urdf_file

```
string Simu_node::urdf_file = ament_index_cpp::get_package_share_directory("umi_rtx_controller")+"/urdf/umi↩
_rtx.urdf"  [private]
```

7.18.4.10 zeros

```
map<string,double> Simu_node::zeros  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_simu.hpp
- ros2_ws/src/umi_rtx_controller/src/node_simu.cpp

7.19 rapidxml::xml_document< Ch >::text_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.19.1 Member Function Documentation

7.19.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::text_pred::test (
            Ch ch )  [inline], [static]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.20 rapidxml::xml_document< Ch >::text_pure_no_ws_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.20.1 Member Function Documentation

7.20.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::text_pure_no_ws_pred::test (
            Ch ch )  [inline], [static]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.21 rapidxml::xml_document< Ch >::text_pure_with_ws_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.21.1 Member Function Documentation

7.21.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::text_pure_with_ws_pred::test (
            Ch ch )  [inline], [static]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.22 rapidxml::xml_document< Ch >::whitespace_pred Struct Reference

Static Public Member Functions

- static unsigned char test (Ch ch)

7.22.1 Member Function Documentation

7.22.1.1 test()

```
template<class Ch = char>
static unsigned char rapidxml::xml_document< Ch >::whitespace_pred::test (
             Ch ch )  [inline], [static]
```

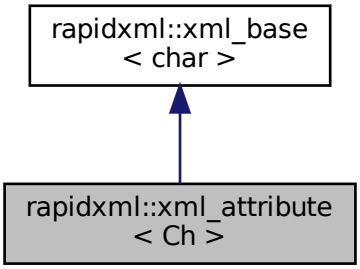
The documentation for this struct was generated from the following file:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

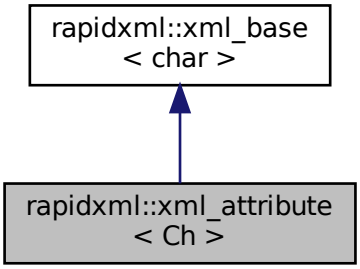
7.23 rapidxml::xml_attribute< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inheritance diagram for rapidxml::xml_attribute< Ch >:



Collaboration diagram for rapidxml::xml_attribute< Ch >:



Public Member Functions

- xml_attribute ()
- xml_document< Ch > * document () const
- xml_attribute< Ch > * previous_attribute (const Ch *name=0, std::size_t name_size=0, bool case_↩ sensitive=true) const
- xml_attribute< Ch > * next_attribute (const Ch *name=0, std::size_t name_size=0, bool case_↩ sensitive=true) const

Private Attributes

- xml_attribute< Ch > * m_prev_attribute
- xml_attribute< Ch > * m_next_attribute

Friends

- class xml_node< Ch >

Additional Inherited Members

7.23.1 Detailed Description

template<class Ch = char>
class rapidxml::xml_attribute< Ch >

Class representing attribute node of XML document. Each attribute has name and value strings, which are available through name() and value() functions (inherited from xml_base). Note that after parse, both name and value of attribute will point to interior of source text used for parsing. Thus, this text must persist in memory for the lifetime of attribute.

Parameters

Ch	Character type to use.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 xml_attribute()

```
template<class Ch = char>
rapidxml::xml_attribute< Ch >::xml_attribute ( )  [inline]
```

Constructs an empty attribute with the specified type. Consider using memory_pool of appropriate xml_document if allocating attributes manually.

7.23.3 Member Function Documentation

7.23.3.1 document()

```
template<class Ch = char>
xml_document<Ch>* rapidxml::xml_attribute< Ch >::document ( ) const  [inline]
```

Gets document of which attribute is a child.

Returns

Pointer to document that contains this attribute, or 0 if there is no parent document.

7.23.3.2 next_attribute()

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::next_attribute (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets next attribute, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return next attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

7.23.3.3 previous_attribute()

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::previous_attribute (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets previous attribute, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return previous attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

7.23.4 Friends And Related Function Documentation

7.23.4.1 xml_node< Ch >

```
template<class Ch = char>
friend class xml_node< Ch >  [friend]
```

7.23.5 Member Data Documentation

7.23.5.1 m_next_attribute

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::m_next_attribute  [private]
```

7.23.5.2 m_prev_attribute

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_attribute< Ch >::m_prev_attribute  [private]
```

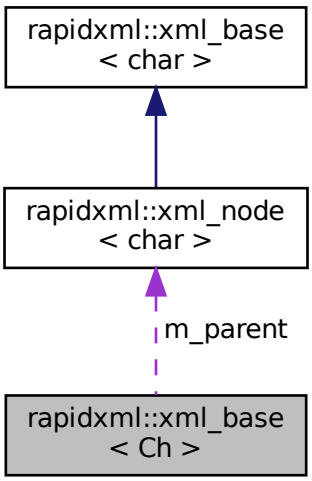
The documentation for this class was generated from the following file:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

7.24 rapidxml::xml_base< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Collaboration diagram for rapidxml::xml_base< Ch >:



Public Member Functions

- xml_base ()
- Ch * name () const
- std::size_t name_size () const
- Ch * value () const
- std::size_t value_size () const
- void name (const Ch *name, std::size_t size)
- void name (const Ch *name)
- void value (const Ch *value, std::size_t size)
- void value (const Ch *value)
- xml_node< Ch > * parent () const

Static Protected Member Functions

- static Ch * nullstr ()

Protected Attributes

- Ch * m_name
- Ch * m_value
- std::size_t m_name_size
- std::size_t m_value_size
- xml_node< Ch > * m_parent

7.24.1 Detailed Description

template<class Ch = char>
class rapidxml::xml_base< Ch >

Base class for xml_node and xml_attribute implementing common functions: name(), name_size(), value(), value_size() and parent().

Parameters

Ch	Character type to use

7.24.2 Constructor & Destructor Documentation

7.24.2.1 xml_base()

```
template<class Ch = char>
rapidxml::xml_base< Ch >::xml_base ( )  [inline]
```

7.24.3 Member Function Documentation

7.24.3.1 name() [1/3]

```
template<class Ch = char>
Ch* rapidxml::xml_base< Ch >::name ( ) const  [inline]
```

Gets name of the node. Interpretation of name depends on type of node. Note that name will not be zero-terminated if rapidxml::parse_no_string_terminators option was selected during parse.

Use name_size() function to determine length of the name.

Returns

Name of node, or empty string if node has no name.

7.24.3.2 name() [2/3]

```
template<class Ch = char>
void rapidxml::xml_base< Ch >::name (
            const Ch * name )  [inline]
```

Sets name of node to a zero-terminated string. See also ownership_of_strings and xml_node::name(const Ch *, std::size_t).

Parameters

name	Name of node to set. Must be zero terminated.

7.24.3.3 name() [3/3]

```
template<class Ch = char>
void rapidxml::xml_base< Ch >::name (
            const Ch * name,
            std::size_t size )  [inline]
```

Sets name of node to a non zero-terminated string. See ownership_of_strings.

Note that node does not own its name or value, it only stores a pointer to it. It will not delete or otherwise free the pointer on destruction. It is reponsibility of the user to properly manage lifetime of the string. The easiest way to achieve it is to use memory_pool of the document to allocate the string - on destruction of the document the string will be automatically freed.

Size of name must be specified separately, because name does not have to be zero terminated. Use name(const Ch *) function to have the length automatically calculated (string must be zero terminated).

Parameters

name	Name of node to set. Does not have to be zero terminated.
size	Size of name, in characters. This does not include zero terminator, if one is present.

7.24.3.4 name_size()

```
template<class Ch = char>
std::size_t rapidxml::xml_base< Ch >::name_size ( ) const  [inline]
```

Gets size of node name, not including terminator character. This function works correctly irrespective of whether name is or is not zero terminated.

Returns

Size of node name, in characters.

7.24.3.5 nullstr()

```
template<class Ch = char>
static Ch* rapidxml::xml_base< Ch >::nullstr ( )  [inline], [static], [protected]
```

7.24.3.6 parent()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_base< Ch >::parent ( ) const  [inline]
```

Gets node parent.

Returns

Pointer to parent node, or 0 if there is no parent.

7.24.3.7 value() [1/3]

```
template<class Ch = char>
Ch* rapidxml::xml_base< Ch >::value ( ) const  [inline]
```

Gets value of node. Interpretation of value depends on type of node. Note that value will not be zero-terminated if rapidxml::parse_no_string_terminators option was selected during parse.

Use value_size() function to determine length of the value.

Returns

Value of node, or empty string if node has no value.

7.24.3.8 value() [2/3]

```
template<class Ch = char>
void rapidxml::xml_base< Ch >::value (
            const Ch * value )  [inline]
```

Sets value of node to a zero-terminated string. See also ownership_of_strings and xml_node::value(const Ch *, std::size_t).

Parameters

value	Vame of node to set. Must be zero terminated.

7.24.3.9 value() [3/3]

```
template<class Ch = char>
void rapidxml::xml_base< Ch >::value (
```

```
const Ch * value,
std::size_t size )  [inline]
```

Sets value of node to a non zero-terminated string. See ownership_of_strings.

Note that node does not own its name or value, it only stores a pointer to it. It will not delete or otherwise free the pointer on destruction. It is reponsibility of the user to properly manage lifetime of the string. The easiest way to achieve it is to use memory_pool of the document to allocate the string - on destruction of the document the string will be automatically freed.

Size of value must be specified separately, because it does not have to be zero terminated. Use value(const Ch *) function to have the length automatically calculated (string must be zero terminated).

If an element has a child node of type node_data, it will take precedence over element value when printing. If you want to manipulate data of elements using values, use parser flag rapidxml::parse_no_data_nodes to prevent creation of data nodes by the parser.

Parameters

value	value of node to set. Does not have to be zero terminated.
size	Size of value, in characters. This does not include zero terminator, if one is present.

7.24.3.10 value_size()

```
template<class Ch = char>
std::size_t rapidxml::xml_base< Ch >::value_size ( ) const  [inline]
```

Gets size of node value, not including terminator character. This function works correctly irrespective of whether value is or is not zero terminated.

Returns

Size of node value, in characters.

7.24.4 Member Data Documentation

7.24.4.1 m_name

```
template<class Ch = char>
Ch* rapidxml::xml_base< Ch >::m_name  [protected]
```

7.24.4.2 m_name_size

```
template<class Ch = char>
std::size_t rapidxml::xml_base< Ch >::m_name_size  [protected]
```

7.24.4.3 m_parent

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_base< Ch >::m_parent  [protected]
```

7.24.4.4 m_value

```
template<class Ch = char>
Ch* rapidxml::xml_base< Ch >::m_value  [protected]
```

7.24.4.5 m_value_size

```
template<class Ch = char>
std::size_t rapidxml::xml_base< Ch >::m_value_size  [protected]
```

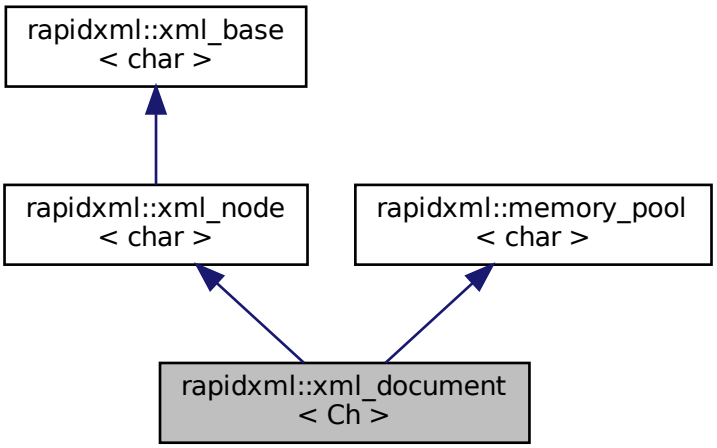
The documentation for this class was generated from the following file:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

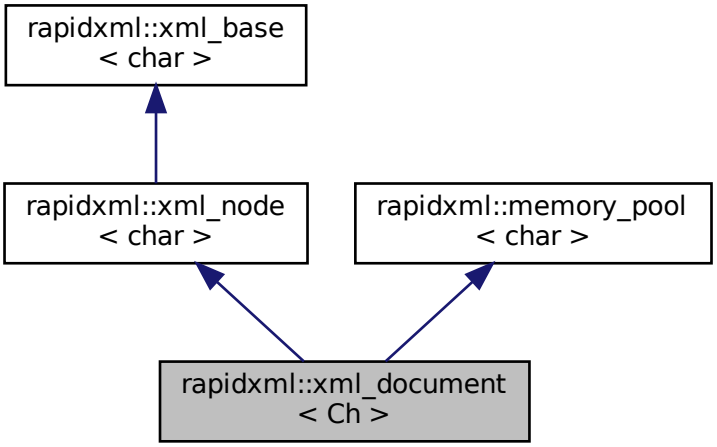
7.25 rapidxml::xml_document< Ch > Class Template Reference

```
#include <rapidxml.hpp>
```

Inheritance diagram for rapidxml::xml_document< Ch >:



Collaboration diagram for rapidxml::xml_document< Ch >:



Classes

- struct attribute_name_pred
- struct attribute_value_pred
- struct attribute_value_pure_pred
- struct node_name_pred
- struct text_pred
- struct text_pure_no_ws_pred
- struct text_pure_with_ws_pred
- struct whitespace_pred

Public Member Functions

- xml_document ()

  Constructs empty XML document.
- template<int Flags>
  void parse (Ch *text)
- void clear ()

Private Member Functions

- template<int Flags>
  void parse_bom (Ch *&text)
- template<int Flags>
  xml_node< Ch > * parse_xml_declaration (Ch *&text)
- template<int Flags>
  xml_node< Ch > * parse_comment (Ch *&text)
- template<int Flags>
  xml_node< Ch > * parse_doctype (Ch *&text)

- template<int Flags>
  xml_node< Ch > * parse_pi (Ch *&text)
- template<int Flags>
  Ch parse_and_append_data (xml_node< Ch > *node, Ch *&text, Ch *contents_start)
- template<int Flags>
  xml_node< Ch > * parse_cdata (Ch *&text)
- template<int Flags>
  xml_node< Ch > * parse_element (Ch *&text)
- template<int Flags>
  xml_node< Ch > * parse_node (Ch *&text)
- template<int Flags>
  void parse_node_contents (Ch *&text, xml_node< Ch > *node)
- template<int Flags>
  void parse_node_attributes (Ch *&text, xml_node< Ch > *node)

Static Private Member Functions

- template<int Flags>
  static void insert_coded_character (Ch *&text, unsigned long code)
- template<class StopPred , int Flags>
  static void skip (Ch *&text)
- template<class StopPred , class StopPredPure , int Flags>
  static Ch * skip_and_expand_character_refs (Ch *&text)

Additional Inherited Members

7.25.1 Detailed Description

template<class Ch = char>
class rapidxml::xml_document< Ch >

This class represents root of the DOM hierarchy. It is also an xml_node and a memory_pool through public inheritance. Use parse() function to build a DOM tree from a zero-terminated XML text string. parse() function allocates memory for nodes and attributes by using functions of xml_document, which are inherited from memory_pool. To access root node of the document, use the document itself, as if it was an xml_node.

Parameters

Ch	Character type to use.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 xml_document()

```
template<class Ch = char>
rapidxml::xml_document< Ch >::xml_document ( )  [inline]
```

Constructs empty XML document.

7.25.3 Member Function Documentation

7.25.3.1 clear()

```
template<class Ch = char>
void rapidxml::xml_document< Ch >::clear ( )  [inline]
```

Clears the document by deleting all nodes and clearing the memory pool. All nodes owned by document pool are destroyed.

7.25.3.2 insert_coded_character()

```
template<class Ch = char>
template<int Flags>
static void rapidxml::xml_document< Ch >::insert_coded_character (
            Ch *& text,
            unsigned long code )  [inline], [static], [private]
```

7.25.3.3 parse()

```
template<class Ch = char>
template<int Flags>
void rapidxml::xml_document< Ch >::parse (
            Ch * text )  [inline]
```

Parses zero-terminated XML string according to given flags. Passed string will be modified by the parser, unless rapidxml::parse_non_destructive flag is used. The string must persist for the lifetime of the document. In case of error, rapidxml::parse_error exception will be thrown.

If you want to parse contents of a file, you must first load the file into the memory, and pass pointer to its beginning. Make sure that data is zero-terminated.

Document can be parsed into multiple times. Each new call to parse removes previous nodes and attributes (if any), but does not clear memory pool.

Parameters

text	XML data to parse; pointer is non-const to denote fact that this data may be modified by the parser.

7.25.3.4 parse_and_append_data()

```
template<class Ch = char>
template<int Flags>
```

```
Ch rapidxml::xml_document< Ch >::parse_and_append_data (
            xml_node< Ch > * node,
            Ch *& text,
            Ch * contents_start )  [inline], [private]
```

7.25.3.5 parse_bom()

```
template<class Ch = char>
template<int Flags>
void rapidxml::xml_document< Ch >::parse_bom (
            Ch *& text )  [inline], [private]
```

7.25.3.6 parse_cdata()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_cdata (
            Ch *& text )  [inline], [private]
```

7.25.3.7 parse_comment()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_comment (
            Ch *& text )  [inline], [private]
```

7.25.3.8 parse_doctype()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_doctype (
            Ch *& text )  [inline], [private]
```

7.25.3.9 parse_element()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_element (
            Ch *& text )  [inline], [private]
```

7.25.3.10 parse_node()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_node (
            Ch *& text )  [inline], [private]
```

7.25.3.11 parse_node_attributes()

```
template<class Ch = char>
template<int Flags>
void rapidxml::xml_document< Ch >::parse_node_attributes (
            Ch *& text,
            xml_node< Ch > * node )  [inline], [private]
```

7.25.3.12 parse_node_contents()

```
template<class Ch = char>
template<int Flags>
void rapidxml::xml_document< Ch >::parse_node_contents (
            Ch *& text,
            xml_node< Ch > * node )  [inline], [private]
```

7.25.3.13 parse_pi()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_pi (
            Ch *& text )  [inline], [private]
```

7.25.3.14 parse_xml_declaration()

```
template<class Ch = char>
template<int Flags>
xml_node<Ch>* rapidxml::xml_document< Ch >::parse_xml_declaration (
            Ch *& text )  [inline], [private]
```

7.25.3.15 skip()

```
template<class Ch = char>
template<class StopPred , int Flags>
static void rapidxml::xml_document< Ch >::skip (
            Ch *& text )  [inline], [static], [private]
```

7.25.3.16 skip_and_expand_character_refs()

```
template<class Ch = char>
template<class StopPred , class StopPredPure , int Flags>
static Ch* rapidxml::xml_document< Ch >::skip_and_expand_character_refs (
            Ch *& text )  [inline], [static], [private]
```

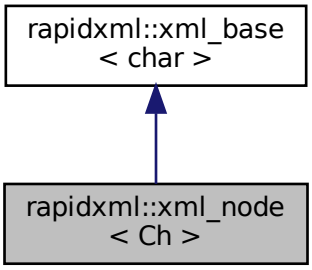
The documentation for this class was generated from the following file:

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

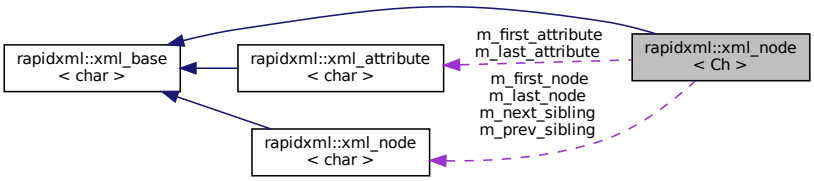
7.26 rapidxml::xml_node< Ch > Class Template Reference

`#include <rapidxml.hpp>`

Inheritance diagram for rapidxml::xml_node< Ch >:



Collaboration diagram for rapidxml::xml_node< Ch >:



Public Member Functions

- xml_node (node_type type)
- node_type type () const
- xml_document< Ch > * document () const
- xml_node< Ch > * first_node (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_node< Ch > * last_node (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_node< Ch > * previous_sibling (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_node< Ch > * next_sibling (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_attribute< Ch > * first_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- xml_attribute< Ch > * last_attribute (const Ch *name=0, std::size_t name_size=0, bool case_sensitive=true) const
- void type (node_type type)
- void prepend_node (xml_node< Ch > *child)
- void append_node (xml_node< Ch > *child)
- void insert_node (xml_node< Ch > *where, xml_node< Ch > *child)
- void remove_first_node ()
- void remove_last_node ()
- void remove_node (xml_node< Ch > *where)

  Removes specified child from the node.
- void remove_all_nodes ()

  Removes all child nodes (but not attributes).
- void prepend_attribute (xml_attribute< Ch > *attribute)
- void append_attribute (xml_attribute< Ch > *attribute)
- void insert_attribute (xml_attribute< Ch > *where, xml_attribute< Ch > *attribute)
- void remove_first_attribute ()
- void remove_last_attribute ()
- void remove_attribute (xml_attribute< Ch > *where)
- void remove_all_attributes ()

  Removes all attributes of node.

Private Member Functions

- xml_node (const xml_node &)
- void operator= (const xml_node &)

Private Attributes

- node_type m_type
- xml_node< Ch > * m_first_node
- xml_node< Ch > * m_last_node
- xml_attribute< Ch > * m_first_attribute
- xml_attribute< Ch > * m_last_attribute
- xml_node< Ch > * m_prev_sibling
- xml_node< Ch > * m_next_sibling

Additional Inherited Members

7.26.1 Detailed Description

template<class Ch = char>
class rapidxml::xml_node< Ch >

Class representing a node of XML document. Each node may have associated name and value strings, which are available through name() and value() functions. Interpretation of name and value depends on type of the node. Type of node can be determined by using type() function.

Note that after parse, both name and value of node, if any, will point interior of source text used for parsing. Thus, this text must persist in the memory for the lifetime of node.

Parameters

Ch	Character type to use.

7.26.2 Constructor & Destructor Documentation

7.26.2.1 xml_node() [1/2]

```
template<class Ch = char>
rapidxml::xml_node< Ch >::xml_node (
            node_type type )  [inline]
```

Constructs an empty node with the specified type. Consider using memory_pool of appropriate document to allocate nodes manually.

Parameters

type	Type of node to construct.

7.26.2.2 xml_node() [2/2]

```
template<class Ch = char>
rapidxml::xml_node< Ch >::xml_node (
            const xml_node< Ch > &  )  [private]
```

7.26.3 Member Function Documentation

7.26.3.1 append_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::append_attribute (
             xml_attribute< Ch > * attribute )  [inline]
```

Appends a new attribute to the node.

Parameters

attribute	Attribute to append.

7.26.3.2 append_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::append_node (
             xml_node< Ch > * child )  [inline]
```

Appends a new child node. The appended child becomes the last child.

Parameters

child	Node to append.

7.26.3.3 document()

```
template<class Ch = char>
xml_document<Ch>* rapidxml::xml_node< Ch >::document ( ) const  [inline]
```

Gets document of which node is a child.

Returns

Pointer to document that contains this node, or 0 if there is no parent document.

7.26.3.4 first_attribute()

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_node< Ch >::first_attribute (
             const Ch * name = 0,
             std::size_t name_size = 0,
             bool case_sensitive = true ) const  [inline]
```

Gets first attribute of node, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return first attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

7.26.3.5 first_node()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::first_node (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets first child node, optionally matching node name.

Parameters

name	Name of child to find, or 0 to return first child regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found child, or 0 if not found.

7.26.3.6 insert_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::insert_attribute (
            xml_attribute< Ch > * where,
            xml_attribute< Ch > * attribute )  [inline]
```

Inserts a new attribute at specified place inside the node. All attributes after and including the specified attribute are moved one position back.

Parameters

where	Place where to insert the attribute, or 0 to insert at the back.
attribute	Attribute to insert.

7.26.3.7 insert_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::insert_node (
             xml_node< Ch > * where,
             xml_node< Ch > * child )  [inline]
```

Inserts a new child node at specified place inside the node. All children after and including the specified node are moved one position back.

Parameters

where	Place where to insert the child, or 0 to insert at the back.
child	Node to insert.

7.26.3.8 last_attribute()

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_node< Ch >::last_attribute (
             const Ch * name = 0,
             std::size_t name_size = 0,
             bool case_sensitive = true ) const  [inline]
```

Gets last attribute of node, optionally matching attribute name.

Parameters

name	Name of attribute to find, or 0 to return last attribute regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found attribute, or 0 if not found.

7.26.3.9 last_node()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::last_node (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets last child node, optionally matching node name. Behaviour is undefined if node has no children. Use first_node() to test if node has children.

Parameters

name	Name of child to find, or 0 to return last child regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found child, or 0 if not found.

7.26.3.10 next_sibling()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::next_sibling (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets next sibling node, optionally matching node name. Behaviour is undefined if node has no parent. Use parent() to test if node has a parent.

Parameters

name	Name of sibling to find, or 0 to return next sibling regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found sibling, or 0 if not found.

7.26.3.11 operator=()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::operator= (
            const xml_node< Ch > &  )  [private]
```

7.26.3.12 prepend_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::prepend_attribute (
            xml_attribute< Ch > * attribute )  [inline]
```

Prepends a new attribute to the node.

Parameters

attribute	Attribute to prepend.

7.26.3.13 prepend_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::prepend_node (
            xml_node< Ch > * child )  [inline]
```

Prepends a new child node. The prepended child becomes the first child, and all existing children are moved one position back.

Parameters

child	Node to prepend.

7.26.3.14 previous_sibling()

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::previous_sibling (
            const Ch * name = 0,
            std::size_t name_size = 0,
            bool case_sensitive = true ) const  [inline]
```

Gets previous sibling node, optionally matching node name. Behaviour is undefined if node has no parent. Use parent() to test if node has a parent.

Parameters

name	Name of sibling to find, or 0 to return previous sibling regardless of its name; this string doesn't have to be zero-terminated if name_size is non-zero
name_size	Size of name, in characters, or 0 to have size calculated automatically from string
case_sensitive	Should name comparison be case-sensitive; non case-sensitive comparison works properly only for ASCII characters

Returns

Pointer to found sibling, or 0 if not found.

7.26.3.15 remove_all_attributes()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_all_attributes ( )  [inline]
```

Removes all attributes of node.

7.26.3.16 remove_all_nodes()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_all_nodes ( )  [inline]
```

Removes all child nodes (but not attributes).

7.26.3.17 remove_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_attribute (
            xml_attribute< Ch > * where )  [inline]
```

Removes specified attribute from node.

Parameters

where	Pointer to attribute to be removed.

7.26.3.18 remove_first_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_first_attribute ( )  [inline]
```

Removes first attribute of the node. If node has no attributes, behaviour is undefined. Use first_attribute() to test if node has attributes.

7.26.3.19 remove_first_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_first_node ( )  [inline]
```

Removes first child node. If node has no children, behaviour is undefined. Use first_node() to test if node has children.

7.26.3.20 remove_last_attribute()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_last_attribute ( )  [inline]
```

Removes last attribute of the node. If node has no attributes, behaviour is undefined. Use first_attribute() to test if node has attributes.

7.26.3.21 remove_last_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_last_node ( )  [inline]
```

Removes last child of the node. If node has no children, behaviour is undefined. Use first_node() to test if node has children.

7.26.3.22 remove_node()

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::remove_node (
            xml_node< Ch > * where )  [inline]
```

Removes specified child from the node.

7.26.3.23 type() [1/2]

```
template<class Ch = char>
node_type rapidxml::xml_node< Ch >::type ( ) const  [inline]
```

Gets type of node.

Returns

Type of node.

7.26.3.24 type() [2/2]

```
template<class Ch = char>
void rapidxml::xml_node< Ch >::type (
            node_type type )  [inline]
```

Sets type of node.

Parameters

type	Type of node to set.

7.26.4 Member Data Documentation

7.26.4.1 m_first_attribute

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_node< Ch >::m_first_attribute  [private]
```

7.26.4.2 m_first_node

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::m_first_node  [private]
```

7.26.4.3 m_last_attribute

```
template<class Ch = char>
xml_attribute<Ch>* rapidxml::xml_node< Ch >::m_last_attribute  [private]
```

7.26.4.4 m_last_node

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::m_last_node  [private]
```

7.26.4.5 m_next_sibling

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::m_next_sibling  [private]
```

7.26.4.6 m_prev_sibling

```
template<class Ch = char>
xml_node<Ch>* rapidxml::xml_node< Ch >::m_prev_sibling  [private]
```

7.26.4.7 m_type

```
template<class Ch = char>
node_type rapidxml::xml_node< Ch >::m_type  [private]
```

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

- ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/rapidxml.hpp

Chapter 8

File Documentation

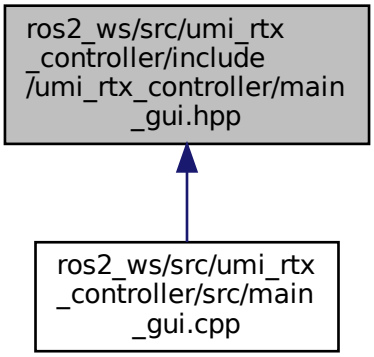
8.1 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/main_gui.hpp File Reference

```
#include <QApplication>
#include <QSlider>
#include <QLabel>
#include <QVBoxLayout>
#include <QDoubleSpinBox>
#include <QHBoxLayout>
#include <QObject>
#include <QMainWindow>
#include <QWidget>
#include <QPushButton>
#include <QPalette>
#include <QDockWidget>
#include <QProcess>
#include <QCheckBox>
#include <QResizeEvent>
#include <QTimer>
#include <QImage>
#include "umi_rtx_controller/node_commands.hpp"
#include "rclcpp/rclcpp.hpp"
#include "rclcpp/clock.hpp"
#include "rviz_common/display.hpp"
#include "rviz_common/window_manager_interface.hpp"
#include "rviz_common/ros_integration/ros_node_abstraction.hpp"
#include "rviz_common/render_panel.hpp"
#include "rviz_common/visualization_manager.hpp"
#include <rviz_common/config.hpp>
#include <rviz_common/yaml_config_reader.hpp>
#include <rviz_common/tool.hpp>
#include <rviz_common/tool_manager.hpp>
#include <rviz_common/view_controller.hpp>
#include "rviz_rendering/render_window.hpp"
#include "rviz_default_plugins/visibility_control.hpp"
#include <ament_index_cpp/get_package_share_directory.hpp>
#include <iostream>
```

```
#include <opencv2/opencv.hpp>
```

Include dependency graph for main_gui.hpp:

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



Classes

- class MainGUI

Namespaces

- rviz_common

8.2 ros2_ws/src/umi_rtx_controller/include/umi_rtx_↩ controller/mainpage.h File Reference

8.3 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ arm.hpp File Reference

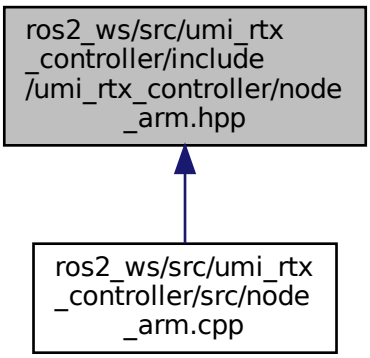
Node for controlling an arm in a robotics system.

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
#include "std_msgs/msg/float32.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "geometry_msgs/msg/pose.hpp"
```

```
#include "umi_rtx_controller/umi-drivers/armlib.h"
#include "umi_rtx_controller/umi-drivers/rtx.h"
#include "umi_rtx_controller/umi-drivers/armraw.h"
#include "umi_rtx_controller/umi-drivers/comm.h"
#include "umi_rtx_controller/umi-drivers/rtxd.h"
#include "umi_rtx_controller/arm_parts/arm.h"
#include "umi_rtx_controller/robotics/umi.h"
#include <sys/types.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <sys/fcntl.h>
#include <sys/un.h>
#include <sys/uio.h>
#include <sys/file.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <fcntl.h>
#include <signal.h>
#include <chrono>
#include <map>
#include <iostream>
#include <sstream>
```

Include dependency graph for node_arm.hpp:

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



Classes

- class Arm_node

  A ROS2 node that controls a robotic arm.

8.3.1 Detailed Description

Node for controlling an arm in a robotics system.

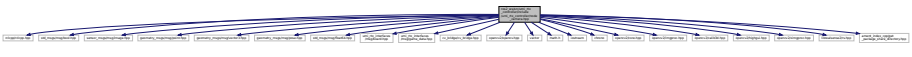
This file defines the Arm_node class which handles communication and control for a robotic arm. It includes functionality for receiving motor commands, target poses, and grip parameters, as well as for controlling the motors and publishing parameters.

8.4 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ camera.hpp File Reference

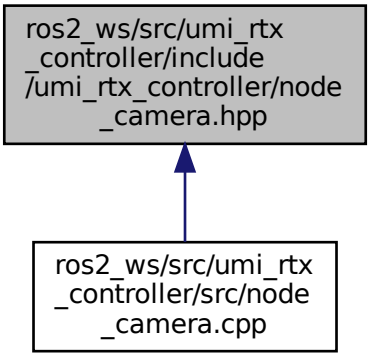
Implementation of the Camera class for handling RealSense camera data and ROS communication.

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/bool.hpp"
#include "sensor_msgs/msg/image.hpp"
#include "geometry_msgs/msg/point.hpp"
#include "geometry_msgs/msg/vector3.hpp"
#include "geometry_msgs/msg/pose.hpp"
#include "std_msgs/msg/float64.hpp"
#include "umi_rtx_interfaces/msg/board.hpp"
#include "umi_rtx_interfaces/msg/game_data.hpp"
#include <cv_bridge/cv_bridge.hpp>
#include <opencv2/opencv.hpp>
#include <vector>
#include <math.h>
#include <iostream>
#include <chrono>
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/ximgproc.hpp>
#include <librealsense2/rs.hpp>
#include <ament_index_cpp/get_package_share_directory.hpp>
```

Include dependency graph for node_camera.hpp:



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



Classes

- struct GridSquare
- class Camera

  A ROS 2 node for managing and processing data from a RealSense camera.

8.4.1 Detailed Description

Implementation of the Camera class for handling RealSense camera data and ROS communication.

This file contains the implementation of the Camera class, which initializes the camera, handles image processing, and communicates with ROS topics.

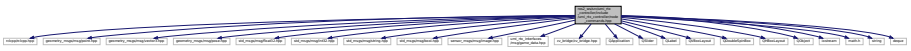
8.5 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ commands.hpp File Reference

Node for managing and sending commands to a robotic arm based on image and game data.

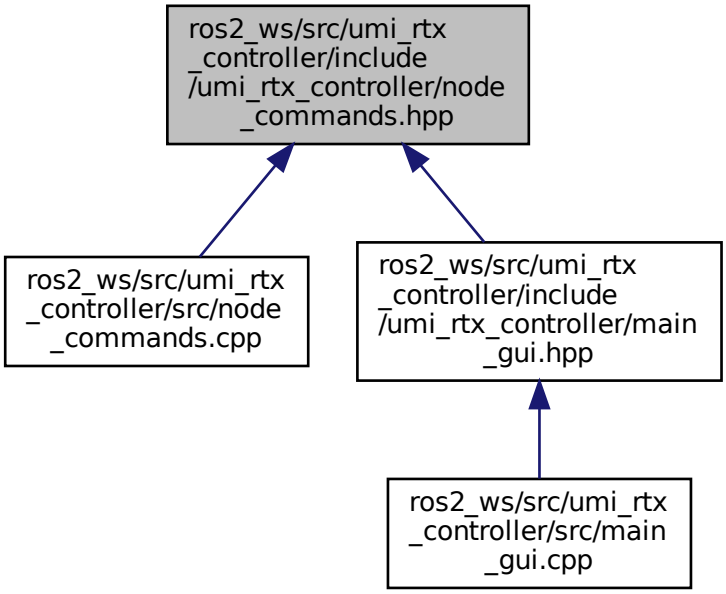
```
#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/point.hpp"
#include "geometry_msgs/msg/vector3.hpp"
#include "geometry_msgs/msg/pose.hpp"
#include "std_msgs/msg/float32.hpp"
#include "std_msgs/msg/int32.hpp"
#include "std_msgs/msg/string.hpp"
#include "std_msgs/msg/bool.hpp"
#include "sensor_msgs/msg/image.hpp"
#include "umi_rtx_interfaces/msg/game_data.hpp"
#include <cv_bridge/cv_bridge.hpp>
#include <QApplication>
```

```
#include <QSlider>
#include <QLabel>
#include <QVBoxLayout>
#include <QDoubleSpinBox>
#include <QHBoxLayout>
#include <QObject>
#include <iostream>
#include <math.h>
#include <string>
#include <deque>
```

Include dependency graph for node_commands.hpp:



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



Classes

- struct Point3D
- class Objective_node

  Manages and sends commands to a robotic arm based on image and game data.

8.5.1 Detailed Description

Node for managing and sending commands to a robotic arm based on image and game data.

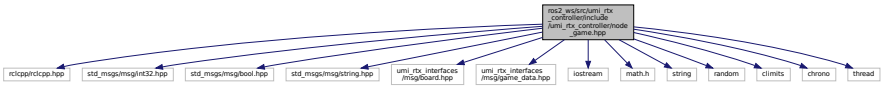
This file defines the `Objective_node` class which handles the publication of target poses, grip parameters, and step capture signals. It also subscribes to image data and game data, processes this information, and adjusts the arm's actions accordingly. The class is designed to break down the robot's movements into multiple steps to ensure precise and accurate control.

8.6 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ game.hpp File Reference

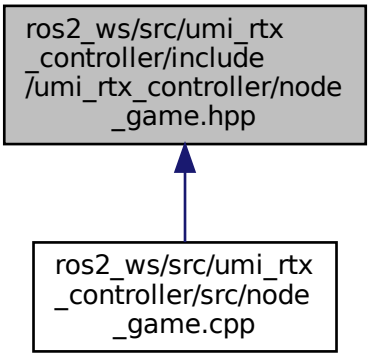
Node for managing the game logic and interactions for a tic-tac-toe game.

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/int32.hpp"
#include "std_msgs/msg/bool.hpp"
#include "std_msgs/msg/string.hpp"
#include "umi_rtx_interfaces/msg/board.hpp"
#include "umi_rtx_interfaces/msg/game_data.hpp"
#include <iostream>
#include <math.h>
#include <string>
#include <random>
#include <climits>
#include <chrono>
#include <thread>
```

Include dependency graph for node_game.hpp:



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



Classes

- struct Position
- struct Move_msg
- class Game_node

  ROS 2 node for managing and controlling a game.

Variables

- const int BOARD_SIZE = 3
- const int ROBOT = 1
- const int HUMAN = 2

8.6.1 Detailed Description

Node for managing the game logic and interactions for a tic-tac-toe game.

This file defines the `Game_node` class which handles game initialization, game state updates, move management, and communication with other ROS nodes.

The node subscribes to board state updates, and publishes game data and the robot's next move.

8.6.2 Variable Documentation

8.6.2.1 BOARD_SIZE

```
const int BOARD_SIZE = 3
```

8.6.2.2 HUMAN

```
const int HUMAN = 2
```

8.6.2.3 ROBOT

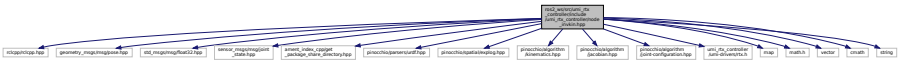
```
const int ROBOT = 1
```

8.7 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ invkin.hpp File Reference

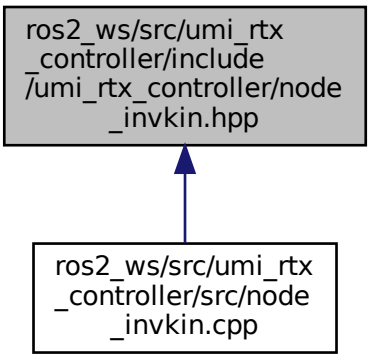
Déclaration des fonctions et méthodes pour le nœud d'inverse kinematics (IK).

```
#include "rclcpp/rclcpp.hpp"
#include "geometry_msgs/msg/pose.hpp"
#include "std_msgs/msg/float32.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include <ament_index_cpp/get_package_share_directory.hpp>
#include "pinocchio/parsers/urdf.hpp"
#include "pinocchio/spatial/explog.hpp"
#include "pinocchio/algorithm/kinematics.hpp"
#include "pinocchio/algorithm/jacobian.hpp"
#include "pinocchio/algorithm/joint-configuration.hpp"
#include "umi_rtx_controller/umi-drivers/rtx.h"
#include <map>
#include <math.h>
#include <vector>
#include <cmath>
#include <string>
```

Include dependency graph for node_invkin.hpp:



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



Classes

- class InvKin_node

  ROS 2 node for performing inverse kinematics.

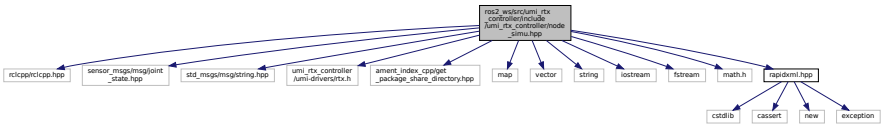
8.7.1 Detailed Description

Déclaration des fonctions et méthodes pour le nœud d'inverse kinematics (IK).

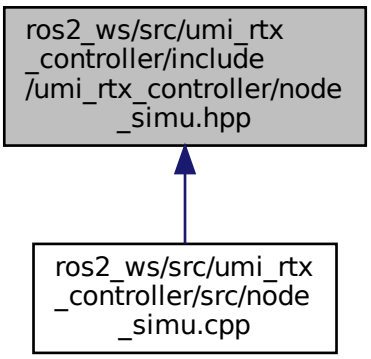
8.8 ros2_ws/src/umi_rtx_controller/include/umi_rtx_controller/node_↩ simu.hpp File Reference

```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"
#include "std_msgs/msg/string.hpp"
#include "umi_rtx_controller/umi-drivers/rtx.h"
#include <ament_index_cpp/get_package_share_directory.hpp>
#include <map>
#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <math.h>
#include "rapidxml.hpp"
```

Include dependency graph for node_simu.hpp:



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



Classes

- class Simu_node

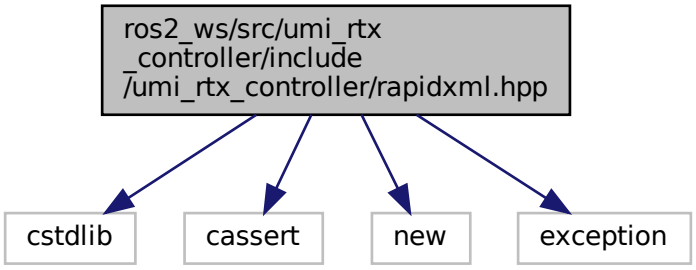
  ROS 2 node for simulating a robotic arm.

8.9 ros2_ws/src/umi_rtx_controller/include/umi_rtx_↩ controller/rapidxml.hpp File Reference

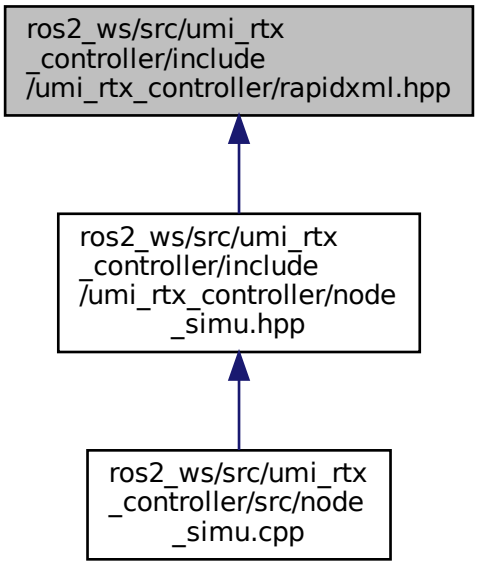
This file contains rapidxml parser and DOM implementation.

```
#include <cstdlib>
#include <cassert>
#include <new>
#include <exception>
```

Include dependency graph for rapidxml.hpp:



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



Classes

- class rapidxml::parse_error
- class rapidxml::memory_pool< Ch >
- struct rapidxml::memory_pool< Ch >::header
- class rapidxml::xml_base< Ch >
- class rapidxml::xml_attribute< Ch >
- class rapidxml::xml_node< Ch >
- class rapidxml::xml_document< Ch >
- struct rapidxml::xml_document< Ch >::whitespace_pred
- struct rapidxml::xml_document< Ch >::node_name_pred
- struct rapidxml::xml_document< Ch >::attribute_name_pred
- struct rapidxml::xml_document< Ch >::text_pred
- struct rapidxml::xml_document< Ch >::text_pure_no_ws_pred
- struct rapidxml::xml_document< Ch >::text_pure_with_ws_pred
- struct rapidxml::xml_document< Ch >::attribute_value_pred< Quote >
- struct rapidxml::xml_document< Ch >::attribute_value_pure_pred< Quote >

Namespaces

- rapidxml

Macros

- #define RAPIDXML_PARSE_ERROR(what, where) throw parse_error(what, where)
- #define RAPIDXML_STATIC_POOL_SIZE (64 ∗ 1024)
- #define RAPIDXML_DYNAMIC_POOL_SIZE (64 ∗ 1024)
- #define RAPIDXML_ALIGNMENT sizeof(void ∗)

Enumerations

- enum rapidxml::node_type {
  rapidxml::node_document , rapidxml::node_element , rapidxml::node_data , rapidxml::node_cdata ,
  rapidxml::node_comment , rapidxml::node_declaration , rapidxml::node_doctype , rapidxml::node_pi }

Variables

- const int rapidxml::parse_no_data_nodes = 0x1
- const int rapidxml::parse_no_element_values = 0x2
- const int rapidxml::parse_no_string_terminators = 0x4
- const int rapidxml::parse_no_entity_translation = 0x8
- const int rapidxml::parse_no_utf8 = 0x10
- const int rapidxml::parse_declaration_node = 0x20
- const int rapidxml::parse_comment_nodes = 0x40
- const int rapidxml::parse_doctype_node = 0x80
- const int rapidxml::parse_pi_nodes = 0x100
- const int rapidxml::parse_validate_closing_tags = 0x200
- const int rapidxml::parse_trim_whitespace = 0x400
- const int rapidxml::parse_normalize_whitespace = 0x800
- const int rapidxml::parse_default = 0
- const int rapidxml::parse_non_destructive = parse_no_string_terminators | parse_no_entity_translation
- const int rapidxml::parse_fastest = parse_non_destructive | parse_no_data_nodes
- const int rapidxml::parse_full = parse_declaration_node | parse_comment_nodes | parse_doctype_node | parse_pi_nodes | parse_validate_closing_tags

8.9.1 Detailed Description

This file contains rapidxml parser and DOM implementation.

8.9.2 Macro Definition Documentation

8.9.2.1 RAPIDXML_ALIGNMENT

```
#define RAPIDXML_ALIGNMENT sizeof(void *)
```

8.9.2.2 RAPIDXML_DYNAMIC_POOL_SIZE

```
#define RAPIDXML_DYNAMIC_POOL_SIZE (64 * 1024)
```

8.9.2.3 RAPIDXML_PARSE_ERROR

```
#define RAPIDXML_PARSE_ERROR(
              what,
              where ) throw parse_error(what, where)
```

8.9.2.4 RAPIDXML_STATIC_POOL_SIZE

```
#define RAPIDXML_STATIC_POOL_SIZE (64 * 1024)
```

8.10 ros2_ws/src/umi_rtx_controller/src/main_gui.cpp File Reference

Implementation of the MainGUI class and the main function for the UMI-RTX Interface.

```
#include "umi_rtx_controller/main_gui.hpp"
```

Include dependency graph for main_gui.cpp:

Functions

- int main (int argc, char ∗argv[ ])

8.10.1 Detailed Description

Implementation of the MainGUI class and the main function for the UMI-RTX Interface.

This file contains the implementation of the MainGUI class, which represents the main graphical user interface for the UMI-RTX controller. It includes widget initialization, layout management, slider and button functionalities, and integration with RViz for visualization. The file also contains the main function which initializes and runs the application.

Note

This file requires Qt and RViz libraries and assumes ROS2 integration.

8.10.2 Function Documentation

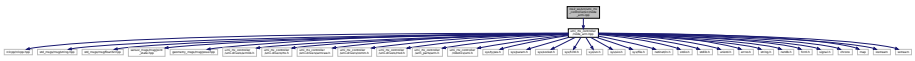
8.10.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

8.11 ros2_ws/src/umi_rtx_controller/src/node_arm.cpp File Reference

```
#include "umi_rtx_controller/node_arm.hpp"
```

Include dependency graph for node_arm.cpp:



Functions

- int main (int argc, char ∗argv[ ])

8.11.1 Function Documentation

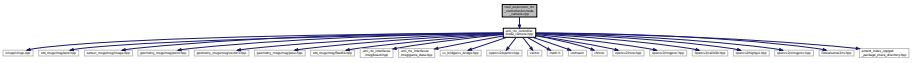
8.11.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

8.12 ros2_ws/src/umi_rtx_controller/src/node_camera.cpp File Reference

Implementation of the Camera class for controlling the UMI-RTX camera system.

`#include "umi_rtx_controller/node_camera.hpp"`
Include dependency graph for node_camera.cpp:



Functions

- int main (int argc, char ∗argv[ ])

8.12.1 Detailed Description

Implementation of the Camera class for controlling the UMI-RTX camera system.

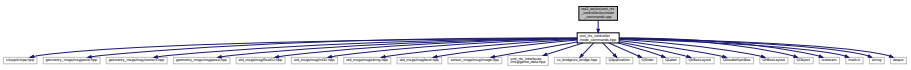
8.12.2 Function Documentation

8.12.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

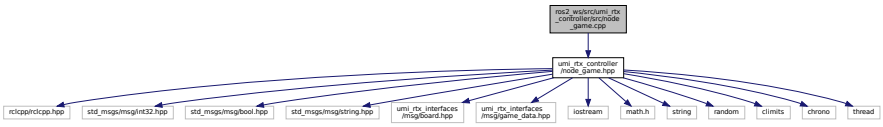
8.13 ros2_ws/src/umi_rtx_controller/src/node_commands.cpp File Reference

`#include "umi_rtx_controller/node_commands.hpp"`
Include dependency graph for node_commands.cpp:



8.14 ros2_ws/src/umi_rtx_controller/src/node_game.cpp File Reference

```
#include "umi_rtx_controller/node_game.hpp"
```
Include dependency graph for node_game.cpp:



Functions

- int main (int argc, char ∗argv[ ])

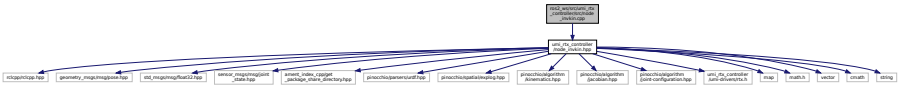
8.14.1 Function Documentation

8.14.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

8.15 ros2_ws/src/umi_rtx_controller/src/node_invkin.cpp File Reference

```
#include "umi_rtx_controller/node_invkin.hpp"
```
Include dependency graph for node_invkin.cpp:



Functions

- int main (int argc, char ∗argv[ ])

8.15.1 Function Documentation

8.15.1.1 main()

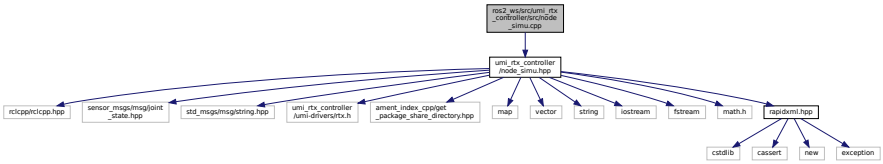
```
int main (
            int argc,
            char * argv[] )
```

8.16 ros2_ws/src/umi_rtx_controller/src/node_simu.cpp File Reference

Implementation of the Simu_node class for simulating joint states in ROS.

```
#include "umi_rtx_controller/node_simu.hpp"
```

Include dependency graph for node_simu.cpp:



Functions

- int main (int argc, char ∗argv[ ])

8.16.1 Detailed Description

Implementation of the Simu_node class for simulating joint states in ROS.

This file contains the implementation of the Simu_node class, which is responsible for:

- Initializing ROS interfaces (subscriptions and publications).
- Handling timer callbacks to publish joint states.
- Parsing a URDF file to initialize joint properties.
- Processing incoming joint commands and updating the joint states accordingly.

The main function initializes ROS, creates an instance of Simu_node, and starts spinning to process incoming messages.

8.16.2 Function Documentation

8.16.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Index