artgslam_vsc

1.0

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

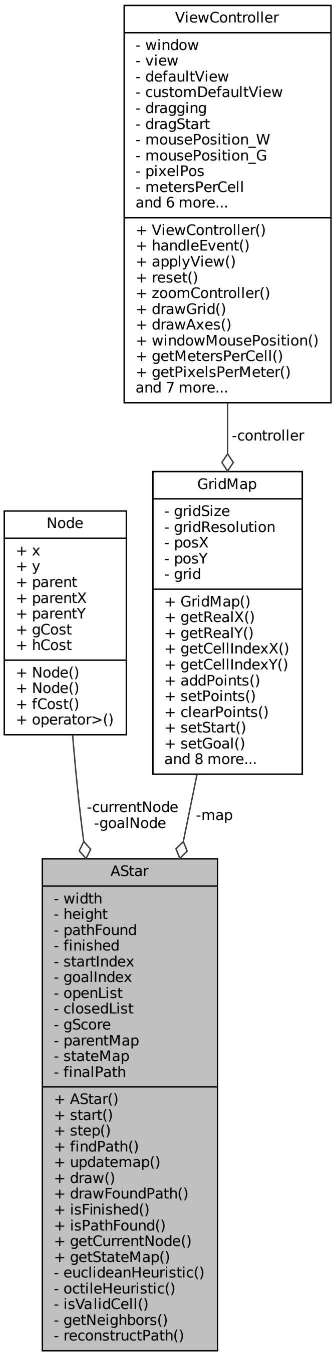
Class Documentation

3.1 AStar Class Reference

This class handles the A∗ algorithm for path planning. It receives start and goal coordinates from the GridMap, performs the algorithm step-by-step, and supports animation and path retrieval.

```
#include <AStar.hpp>
```

Collaboration diagram for AStar:



Public Types

- enum State {
  State::cStart, State::cEmpty, State::cObstacle, State::cClose,
  State::cPath, State::cGoal }

  Enum representing the visual state of each grid cell.

Public Member Functions

- AStar (GridMap &mapRef)

  Constructor with GridMap reference.
- void start ()

  Initializes the A$*$ algorithm.
- bool step ()

  Performs one step of the A$*$ algorithm.
- std::vector$<$ sf::Vector2i $>$ findPath ()

  Executes the full A$*$ algorithm from start to goal in one call.
- void updatemap ()

  Updates the internal state map based on the current GridMap.
- void draw (sf::RenderTarget &target, float pixelsPerMeter, float metersPerCell) const

  Draws the current algorithm state to the given SFML render target.
- void drawFoundPath (sf::RenderTarget &target, float pixelsPerMeter, float metersPerCell) const

  Draws the final path, if found.
- bool isFinished () const

  Returns whether the algorithm has finished executing.
- bool isPathFound () const

  Returns whether a path was found from start to goal.
- const Node & getCurrentNode () const

  Returns the current node being evaluated (useful for debugging or animation).
- const std::vector$<$ std::vector$<$ State $>$ $>$ & getStateMap () const

  Returns the current internal state map used for visualization.

Private Member Functions

- double euclideanHeuristic (int x1, int y1, int x2, int y2)

  Euclidean distance heuristic.
- double octileHeuristic (int x1, int y1, int x2, int y2)

  Octile distance heuristic (better suited for 8-connected grids).
- bool isValidCell (int x, int y) const

  Checks if the given cell coordinates are within the grid bounds.
- std::vector$<$ sf::Vector2i $>$ getNeighbors (const Node &node) const

  Returns the valid neighbor cells of a given node.
- std::vector$<$ sf::Vector2i $>$ reconstructPath (const Node &endNode) const

  Reconstructs the final path from the goal node by tracing the parent map.

Private Attributes

- GridMap & map

  Reference to the associated occupancy grid map.
- int width = 0

  Grid width.
- int height = 0

  Grid height.
- bool pathFound = false

  True if a valid path was found.
- bool finished = false

True if the algorithm has completed.

- sf::Vector2i startIndex = {-1, -1}

  Start position (grid indices)
- sf::Vector2i goalIndex = {-1, -1}

  Goal position (grid indices)
- Node currentNode

  Node currently being processed.
- Node goalNode

  Target goal node.
- std::priority_queue< Node, std::vector< Node >, std::greater< Node > > openList

  Priority queue for open nodes.
- std::vector< std::vector< bool > > closedList

  Tracks visited nodes.
- std::vector< std::vector< float > > gScore

  Cost from start to each cell.
- std::vector< std::vector< sf::Vector2i > > parentMap

  For path reconstruction.
- std::vector< std::vector< State > > stateMap

  State of each cell for visualization.
- std::vector< sf::Vector2i > finalPath

  Final path from start to goal.

3.1.1 Detailed Description

This class handles the A$*$ algorithm for path planning. It receives start and goal coordinates from the GridMap, performs the algorithm step-by-step, and supports animation and path retrieval.

Definition at line 13 of file AStar.hpp.

3.1.2 Member Enumeration Documentation

3.1.2.1 State

```
enum AStar::State  [strong]
```

Enum representing the visual state of each grid cell.

Enumerator

cStart	Start cell.
cEmpty	Unvisited cell.
cObstacle	Obstacle cell.
cClose	Closed/visited cell.
cPath	Final path cell.
cGoal	Goal cell.

Definition at line 17 of file AStar.hpp.

```
                {
        cStart,     ///< Start cell
        cEmpty,     ///< Unvisited cell
        cObstacle,  ///< Obstacle cell
        cClose,     ///< Closed/visited cell
        cPath,      ///< Final path cell
        cGoal       ///< Goal cell
    };
```

3.1.3 Constructor & Destructor Documentation

3.1.3.1 AStar()

```
AStar::AStar (
            GridMap & mapRef )
```

Constructor with GridMap reference.

Constructor initializes A∗ algorithm with a reference to the map.

Parameters

mapRef	Reference to the occupancy grid map

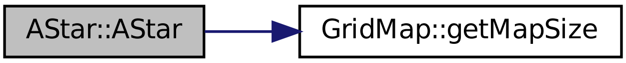
It prepares all necessary data structures to manage the pathfinding.

Definition at line 11 of file Astar.cpp.

```
    : map(mapRef) // Reference to the occupancy grid map
{
    width = map.getMapSize(); // Map width (assuming square map)
    height = width;

    // Initialize stateMap with all cells empty
    stateMap.resize(height, std::vector<State>(width, State::cEmpty));

    // Initialize closed list with false flags (unvisited)
    closedList.resize(height, std::vector<bool>(width, false));

    // Initialize parent map with invalid parent indices
    parentMap.resize(height, std::vector<sf::Vector2i>(width, {-1, -1}));

    // Initialize gScores to infinity (unexplored)
    gScore.resize(height, std::vector<float>(width, std::numeric_limits<float>::infinity()));

    // Initialize start and goal indices as invalid
    startIndex = {-1, -1};
    goalIndex = {-1, -1};

    pathFound = false; // No path found yet
    finished = false;  // Algorithm not finished yet
    finalPath.clear(); // Clear any previous path
}
```

References cEmpty, closedList, finalPath, finished, GridMap::getMapSize(), goalIndex, gScore, height, map, parentMap, pathFound, startIndex, stateMap, and width.

Here is the call graph for this function:



3.1.4 Member Function Documentation

3.1.4.1 draw()

```
void AStar::draw (
             sf::RenderTarget & target,
             float pixelsPerMeter,
             float metersPerCell ) const
```

Draws the current algorithm state to the given SFML render target.

Draws the current state of the map (open, closed, path, etc.) on the render target.

Can be used for animated A∗ visualizations.

Parameters

target	SFML render target to draw on.
pixelsPerMeter	Scaling factor for rendering.
metersPerCell	Size of each grid cell in meters.

Definition at line 247 of file Astar.cpp.

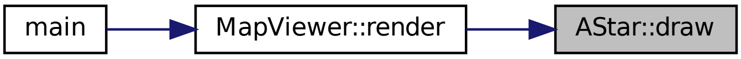
```
247                                                                                            {
248     float cellSize = metersPerCell * pixelsPerMeter;
249     float offsetX = -(width / 2.f) * cellSize;
250     float offsetY = -(height / 2.f) * cellSize;
251 
252     sf::RectangleShape cellShape(sf::Vector2f(cellSize, cellSize));
253     cellShape.setOutlineThickness(0);
254 
255     for (int y = 0; y < height; ++y) {
256         for (int x = 0; x < width; ++x) {
257             switch (stateMap[y][x]) {
258                 case State::cStart:
259                     cellShape.setFillColor(sf::Color::Red);
260                     break;
261                 case State::cEmpty:
262                     continue; // Skip empty cells
263                 case State::cObstacle:
264                     cellShape.setFillColor(sf::Color::Black);
265                     break;
266                 case State::cClose:
267                     cellShape.setFillColor(sf::Color(100, 149, 237, 180)); // Light blue,
       semi-transparent
268                     break;
269                 case State::cPath:
```

```
270                     cellShape.setFillColor(sf::Color::Yellow);
271                     break;
272                 case State::cGoal:
273                     cellShape.setFillColor(sf::Color::Green);
274                     break;
275                 default:
276                     continue;
277             }
278 
279             cellShape.setPosition(offsetX + x * cellSize, offsetY + y * cellSize);
280             target.draw(cellShape);
281         }
282     }
283 }
```

References cClose, cEmpty, cGoal, cObstacle, cPath, cStart, height, stateMap, and width.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.1.4.2 drawFoundPath()

```
void AStar::drawFoundPath (
            sf::RenderTarget & target,
            float pixelsPerMeter,
            float metersPerCell ) const
```

Draws the final path, if found.

Draws the final path found by the algorithm.

Parameters

target	SFML render target.
pixelsPerMeter	Pixels per meter scale.
metersPerCell	Size of grid cell in meters.

Definition at line 291 of file Astar.cpp.

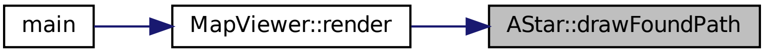
```
291                                                                                          {
292     if (finalPath.empty()) return; // No path to draw
293 
294     float cellSize = metersPerCell * pixelsPerMeter;
295     float offsetX = -(width / 2.f) * cellSize;
296     float offsetY = -(height / 2.f) * cellSize;
297 
298     sf::RectangleShape cellShape(sf::Vector2f(cellSize, cellSize));
299     cellShape.setFillColor(sf::Color::Yellow);
300 
```

```
    for (const auto& pos : finalPath) {
        if (pos == startIndex || pos == goalIndex) continue;

        float x = offsetX + pos.x * cellSize;
        float y = offsetY + pos.y * cellSize;
        cellShape.setPosition(x, y);

        target.draw(cellShape);
    }
}
```

References finalPath, goalIndex, height, startIndex, and width.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.1.4.3 euclideanHeuristic()

```
double AStar::euclideanHeuristic (
            int x1,
            int y1,
            int x2,
            int y2 )  [private]
```

Euclidean distance heuristic.

3.1.4.4 findPath()

```
std::vector< sf::Vector2i > AStar::findPath ( )
```

Executes the full A∗ algorithm from start to goal in one call.

Finds the complete path from start to goal.

Returns

Vector of grid coordinates representing the path.

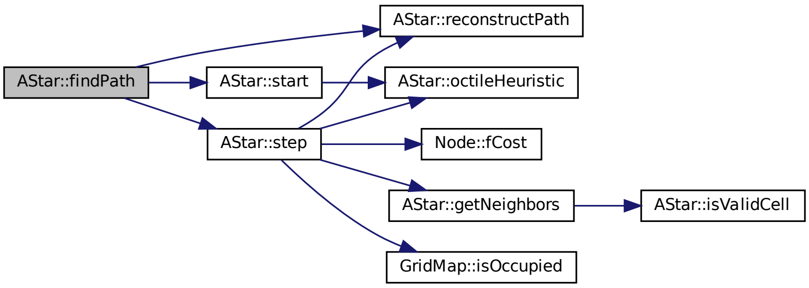
Vector of cell positions along the path.

Definition at line 193 of file Astar.cpp.

```
193                                          {
194     start();
195     while (!finished) {
196         if (!step()) break;
197     }
198     return pathFound ? reconstructPath(goalNode) : std::vector<sf::Vector2i>{};
199 }
```

References finished, goalNode, pathFound, reconstructPath(), start(), and step().

Here is the call graph for this function:



3.1.4.5 getCurrentNode()

```
const Node& AStar::getCurrentNode ( ) const  [inline]
```

Returns the current node being evaluated (useful for debugging or animation).

Definition at line 64 of file AStar.hpp.

```
64 { return currentNode; }
```

References currentNode.

3.1.4.6 getNeighbors()

```
std::vector< sf::Vector2i > AStar::getNeighbors (
                const Node & node ) const  [private]
```

Returns the valid neighbor cells of a given node.

Returns valid neighbors around a node in 8 directions.

Parameters

node	Current node.

Returns

Vector of neighbor cell coordinates.

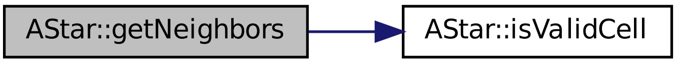
Definition at line 43 of file Astar.cpp.

```
43                                                                    {
44     // Directions: N, NE, E, SE, S, SW, W, NW
45     static const std::vector<sf::Vector2i> directions = {
46         { 0, -1}, { 1, -1}, { 1,  0}, { 1,  1},
47         { 0,  1}, {-1,  1}, {-1,  0}, {-1, -1}
48     };
49 
50     std::vector<sf::Vector2i> neighbors;
51 
52     for (const auto& dir : directions) {
53         int nx = node.x + dir.x;
54         int ny = node.y + dir.y;
55         if (isValidCell(nx, ny)) {
56             neighbors.emplace_back(nx, ny);
57         }
58     }
59 
60     return neighbors;
61 }
```

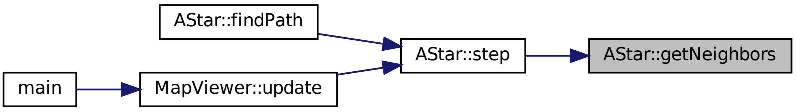
References isValidCell(), Node::x, and Node::y.

Referenced by step().

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.4.7 getStateMap()

```
const std::vector<std::vector<State> >& AStar::getStateMap ( ) const  [inline]
```

Returns the current internal state map used for visualization.

Definition at line 67 of file AStar.hpp.

```
67 { return stateMap; }
```

References stateMap.

3.1.4.8 isFinished()

```
bool AStar::isFinished ( ) const  [inline]
```

Returns whether the algorithm has finished executing.

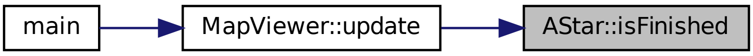
Definition at line 58 of file AStar.hpp.

```
58 { return finished; }
```

References finished.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.1.4.9 isPathFound()

```
bool AStar::isPathFound ( ) const  [inline]
```

Returns whether a path was found from start to goal.

Definition at line 61 of file AStar.hpp.

```
61 { return pathFound; }
```

References pathFound.

3.1.4.10 isValidCell()

```
bool AStar::isValidCell (
             int x,
             int y ) const  [inline], [private]
```

Checks if the given cell coordinates are within the grid bounds.

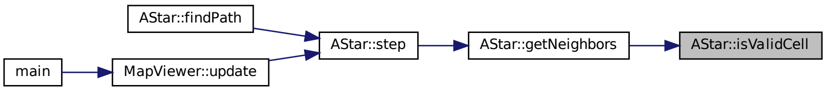
Definition at line 105 of file AStar.hpp.

```
105                                          {
106         return x >= 0 && x < width && y >= 0 && y < height;
107     }
```

References height.

Referenced by getNeighbors().

Here is the caller graph for this function:



3.1.4.11 octileHeuristic()

```
double AStar::octileHeuristic (
             int x1,
             int y1,
             int x2,
             int y2 )  [inline], [private]
```

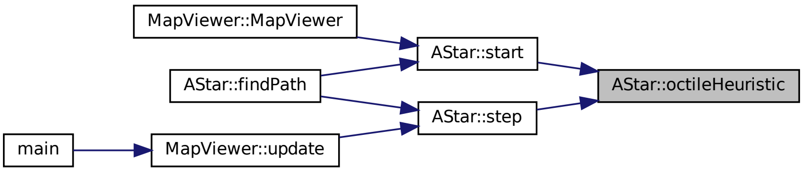
Octile distance heuristic (better suited for 8-connected grids).

Definition at line 98 of file AStar.hpp.

```
98                                                         {
99        int dx = std::abs(x2 - x1);
100        int dy = std::abs(y2 - y1);
101        return std::max(dx, dy) + (std::sqrt(2.0) - 1.0) * std::min(dx, dy);
102     }
```

Referenced by start(), and step().

Here is the caller graph for this function:



3.1.4.12 reconstructPath()

```
std::vector< sf::Vector2i > AStar::reconstructPath (
                const Node & endNode ) const  [private]
```

Reconstructs the final path from the goal node by tracing the parent map.

Reconstructs the path from the goal node back to the start node.

Parameters

endNode	The goal node from which to trace back.

Returns

Vector of cell positions representing the path.

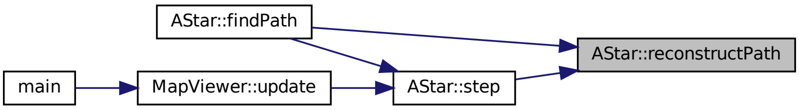
Definition at line 228 of file Astar.cpp.

```
                                                                          {
        std::vector<sf::Vector2i> path;
        sf::Vector2i current = {endNode.x, endNode.y};

        while (current != sf::Vector2i(-1, -1)) {
            path.push_back(current);
            current = parentMap[current.y][current.x];
        }

        std::reverse(path.begin(), path.end());
        return path;
}
```

References parentMap, Node::x, and Node::y.

Referenced by findPath(), and step().

Here is the caller graph for this function:



3.1.4.13 start()

```
void AStar::start ( )
```

Initializes the A∗ algorithm.

Initializes the pathfinding process.

Prepares internal structures and sets start/goal positions.

Resets all data structures and prepares the start node.

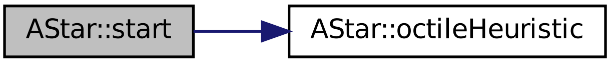
Definition at line 67 of file Astar.cpp.

```
67                   {
68     // Verify start and goal points are set
69     if (startIndex.x == -1 || goalIndex.x == -1) {
70         std::cerr « "Start or Goal not set properly.\n";
71         finished = true;
72         return;
73     }
74 
75     // Reset all pathfinding structures
76     stateMap.assign(height, std::vector<State>(width, State::cEmpty));
77     closedList.assign(height, std::vector<bool>(width, false));
78     parentMap.assign(height, std::vector<sf::Vector2i>(width, {-1, -1}));
79     gScore.assign(height, std::vector<float>(width, std::numeric_limits<float>::infinity()));
80     openList = {}; // Clear priority queue
81 
82     // Mark start and goal on stateMap
83     stateMap[startIndex.y][startIndex.x] = State::cStart;
84     stateMap[goalIndex.y][goalIndex.x] = State::cGoal;
85 
86     // Initialize the start node
87     Node startNode(startIndex.x, startIndex.y);
88     startNode.gCost = 0;
89     startNode.hCost = octileHeuristic(startIndex.x, startIndex.y, goalIndex.x, goalIndex.y);
90 
91     openList.push(startNode);
92     gScore[startIndex.y][startIndex.x] = 0.0f;
93 
94     pathFound = false;
95     finished = false;
96 }
```

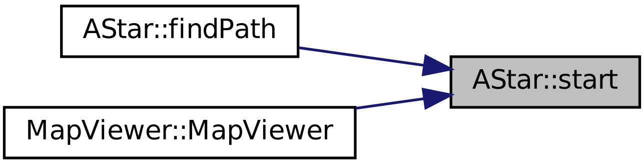
References cEmpty, cGoal, closedList, cStart, finished, Node::gCost, goalIndex, gScore, Node::hCost, height, octileHeuristic(), openList, parentMap, pathFound, startIndex, stateMap, and width.

Referenced by findPath(), and MapViewer::MapViewer().

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.4.14 step()

```
bool AStar::step ( )
```

Performs one step of the A∗ algorithm.

Performs one iteration (step) of the A∗ algorithm.

Returns

true if a step was taken, false if algorithm is finished.

true if more steps are needed; false if finished.

Definition at line 102 of file Astar.cpp.

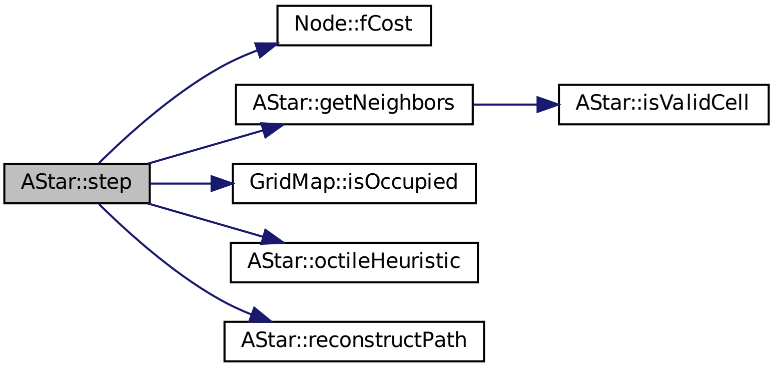
```
102                     {
103     if (finished) {
104         std::cout « "[DEBUG] finished == true, exiting step().\n";
105         return false;
106     }
107 
108     if (openList.empty()) {
109         // No nodes left to explore, path not found
110         finished = true;
111         pathFound = false;
112         std::cout « "[DEBUG] openList empty, no path found.\n";
113         return false;
114     }
115 
116     currentNode = openList.top();
117     openList.pop();
118 
119     std::cout « "[DEBUG] Processing node (" « currentNode.x « ", " « currentNode.y
120               « ") gCost: " « currentNode.gCost
121               « ", hCost: " « currentNode.hCost
122               « ", fCost: " « currentNode.fCost() « "\n";
123 
124     if (closedList[currentNode.y][currentNode.x]) {
125         std::cout « "[DEBUG] Node already closed, skipping.\n";
126         return true;
127     }
128 
129     closedList[currentNode.y][currentNode.x] = true;
130     stateMap[currentNode.y][currentNode.x] = State::cClose;
131 
132     if (currentNode.x == goalIndex.x && currentNode.y == goalIndex.y) {
133         finished = true;
134         pathFound = true;
135         goalNode = currentNode;
136 
137         std::cout « "[DEBUG] Goal node reached.\n";
138 
139         // Mark the path cells (except start/goal)
140         for (auto& pos : reconstructPath(goalNode)) {
141             if (pos != startIndex && pos != goalIndex) {
142                 stateMap[pos.y][pos.x] = State::cPath;
143                 std::cout « "[DEBUG] Marking path cell (" « pos.x « ", " « pos.y « ").\n";
144             }
145         }
146 
147         finalPath = reconstructPath(goalNode);
148         std::cout « "Path to goal reached\n";
149         return false;
150     }
151 
152     // Explore neighbors
153     for (const auto& neighborPos : getNeighbors(currentNode)) {
154         int nx = neighborPos.x;
155         int ny = neighborPos.y;
156 
157         int occupancy = map.isOccupied(nx, ny);
158 
159         std::cout « "[DEBUG] Neighbor (" « nx « ", " « ny « ") occupancy: " « occupancy
160                   « ", closed: " « closedList[ny][nx]
161                   « ", current gScore: " « gScore[ny][nx] « "\n";
162 
163         if (closedList[ny][nx] || occupancy == 1) {
164             std::cout « "[DEBUG] Neighbor discarded.\n";
165             continue;
```

```
        }

        float tentativeG = currentNode.gCost + octileHeuristic(currentNode.x, currentNode.y, nx, ny);

        std::cout « "[DEBUG] tentativeG for (" « nx « ", " « ny « "): " « tentativeG « "\n";

        if (tentativeG < gScore[ny][nx]) {
            std::cout « "[DEBUG] Updating gScore and parent for (" « nx « ", " « ny « ").\n";
            gScore[ny][nx] = tentativeG;
            parentMap[ny][nx] = {currentNode.x, currentNode.y};

            Node neighbor(nx, ny);
            neighbor.gCost = tentativeG;
            neighbor.hCost = octileHeuristic(nx, ny, goalIndex.x, goalIndex.y);

            openList.push(neighbor);
            std::cout « "[DEBUG] Neighbor (" « nx « ", " « ny « ") added to openList.\n";
        }
    }

    return true;
}
```

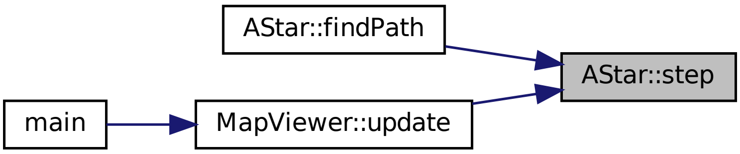
References cClose, closedList, cPath, currentNode, Node::fCost(), finalPath, finished, Node::gCost, getNeighbors(), goalIndex, goalNode, gScore, Node::hCost, GridMap::isOccupied(), map, octileHeuristic(), openList, parentMap, pathFound, reconstructPath(), startIndex, stateMap, Node::x, and Node::y.

Referenced by findPath(), and MapViewer::update().

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.4.15 updatemap()

```
void AStar::updatemap ( )
```

Updates the internal state map based on the current GridMap.

Updates the start and goal positions by scanning the map for 's' and 'g' markers.

Should be called if the GridMap is modified (e.g., new obstacles).

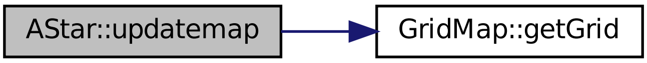
Definition at line 204 of file Astar.cpp.

```
204                        {
205     startIndex = {-1, -1};
206     goalIndex = {-1, -1};
207 
208     const auto& grid = map.getGrid();
209     for (int y = 0; y < height; ++y) {
210         for (int x = 0; x < width; ++x) {
211             int cell = grid[y][x];
212             if (cell == 's') {
213                 startIndex = {x, y};
214                 std::cout « "Updated startIndex: " « x « ", " « y « "\n";
215             } else if (cell == 'g') {
216                 goalIndex = {x, y};
217                 std::cout « "Updated goalIndex: " « x « ", " « y « "\n";
218             }
219         }
220     }
221 }
```

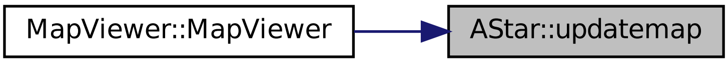
References GridMap::getGrid(), goalIndex, height, map, startIndex, and width.

Referenced by MapViewer::MapViewer().

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.5 Member Data Documentation

3.1.5.1 closedList

```
std::vector<std::vector<bool> > AStar::closedList  [private]
```

Tracks visited nodes.

Definition at line 86 of file AStar.hpp.

Referenced by AStar(), start(), and step().

3.1.5.2 currentNode

```
Node AStar::currentNode  [private]
```

Node currently being processed.

Definition at line 81 of file AStar.hpp.

Referenced by getCurrentNode(), and step().

3.1.5.3 finalPath

```
std::vector<sf::Vector2i> AStar::finalPath  [private]
```

Final path from start to goal.

Definition at line 90 of file AStar.hpp.

Referenced by AStar(), drawFoundPath(), and step().

3.1.5.4 finished

```
bool AStar::finished = false  [private]
```

True if the algorithm has completed.

Definition at line 77 of file AStar.hpp.

Referenced by AStar(), findPath(), isFinished(), start(), and step().

3.1.5.5 goalIndex

```
sf::Vector2i AStar::goalIndex = {-1, -1}  [private]
```

Goal position (grid indices)

Definition at line 80 of file AStar.hpp.

Referenced by AStar(), drawFoundPath(), start(), step(), and updatemap().

3.1.5.6 goalNode

```
Node AStar::goalNode  [private]
```

Target goal node.

Definition at line 82 of file AStar.hpp.

Referenced by findPath(), and step().

3.1.5.7 gScore

```
std::vector<std::vector<float> > AStar::gScore  [private]
```

Cost from start to each cell.

Definition at line 87 of file AStar.hpp.

Referenced by AStar(), start(), and step().

3.1.5.8 height

```
int AStar::height = 0  [private]
```

Grid height.

Definition at line 75 of file AStar.hpp.

Referenced by AStar(), draw(), drawFoundPath(), isValidCell(), start(), and updatemap().

3.1.5.9 map

```
GridMap& AStar::map [private]
```

Reference to the associated occupancy grid map.

Definition at line 70 of file AStar.hpp.

Referenced by AStar(), step(), and updatemap().

3.1.5.10 openList

```
std::priority_queue<Node, std::vector<Node>, std::greater<Node> > AStar::openList [private]
```

Priority queue for open nodes.

Definition at line 85 of file AStar.hpp.

Referenced by start(), and step().

3.1.5.11 parentMap

```
std::vector<std::vector<sf::Vector2i> > AStar::parentMap [private]
```

For path reconstruction.

Definition at line 88 of file AStar.hpp.

Referenced by AStar(), reconstructPath(), start(), and step().

3.1.5.12 pathFound

```
bool AStar::pathFound = false [private]
```

True if a valid path was found.

Definition at line 76 of file AStar.hpp.

Referenced by AStar(), findPath(), isPathFound(), start(), and step().

3.1.5.13 startIndex

```
sf::Vector2i AStar::startIndex = {-1, -1}  [private]
```

Start position (grid indices)

Definition at line 79 of file AStar.hpp.

Referenced by AStar(), drawFoundPath(), start(), step(), and updatemap().

3.1.5.14 stateMap

```
std::vector<std::vector<State> > AStar::stateMap  [private]
```

State of each cell for visualization.

Definition at line 89 of file AStar.hpp.

Referenced by AStar(), draw(), getStateMap(), start(), and step().

3.1.5.15 width

```
int AStar::width = 0  [private]
```

Grid width.

Definition at line 74 of file AStar.hpp.

Referenced by AStar(), draw(), drawFoundPath(), start(), and updatemap().

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

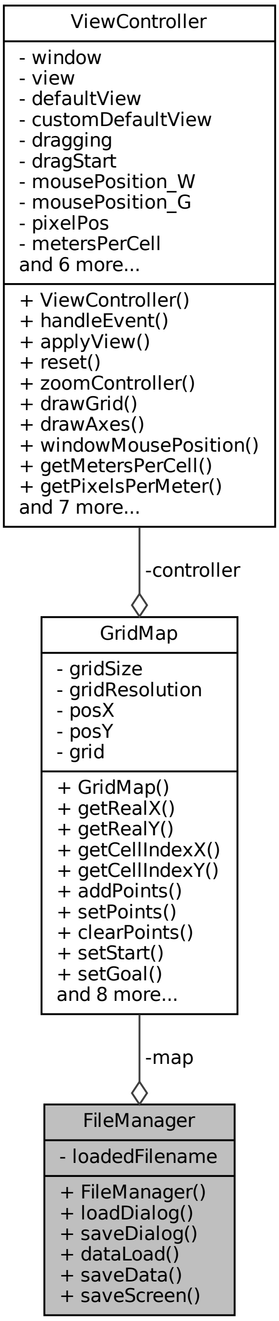
- include/artgslam_vsc/AStar.hpp
- src/Astar.cpp

3.2 FileManager Class Reference

Manages file input/output operations for grid map visualization.

```
#include <FileManager.hpp>
```

Collaboration diagram for FileManager:



Public Member Functions

- FileManager (GridMap &mapRef)

  Constructor.
- void loadDialog ()

  Opens a file dialog to select a file and loads its data into the map.
- void saveDialog ()

  Opens a file dialog to save the current data (coordinates).
- void dataLoad (const std::string &filename, std::vector< double > &x, std::vector< double > &y)

  Loads coordinate data from a file into x and y vectors.
- void saveData (const std::string &filename, std::vector< double > &x, std::vector< double > &y)

  Saves x and y coordinate data to a file.
- void saveScreen (const std::string &filename)

  Saves a screenshot of the current visualization to an image file.

Private Attributes

- GridMap & map

  Reference to the grid map object.
- std::string loadedFilename

  Path of the last loaded or saved file.

3.2.1 Detailed Description

Manages file input/output operations for grid map visualization.

This class handles loading data from files to populate the map, saving coordinate data, and exporting screenshots.

Definition at line 18 of file FileManager.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 FileManager()

```
FileManager::FileManager (
            GridMap & mapRef )
```

Constructor.

Constructor that stores a reference to the GridMap instance for interaction.

Parameters

mapRef	Reference to the GridMap to work with
mapRef	Reference to the GridMap.

Definition at line 7 of file FileManager.cpp.

```
    : map(mapRef)
{
}
```

3.2.3 Member Function Documentation

3.2.3.1 dataLoad()

```
void FileManager::dataLoad (
            const std::string & filename,
            std::vector< double > & x,
            std::vector< double > & y )
```

Loads coordinate data from a file into x and y vectors.

Loads (x,y) data points from a CSV file into the provided vectors.

Parameters

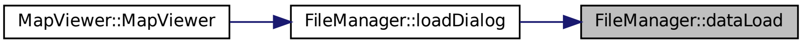
filename	Path of the file to read from
x	Output vector of x coordinates
y	Output vector of y coordinates
filename	Path to the CSV file.
x	Output vector for x coordinates.
y	Output vector for y coordinates.

Definition at line 78 of file FileManager.cpp.

```
{
    x.clear();
    y.clear();

    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr « "Error opening file: " « filename « std::endl;
        return;
    }

    std::string line;
    size_t count = 0;

    // Read file line by line
    while (std::getline(file, line)) {
        std::stringstream ss(line);
        std::string token;
        std::vector<std::string> tokens;

        // Tokenize by commas
        while (std::getline(ss, token, ',')) {
            tokens.push_back(token);
        }

        if (tokens.size() >= 2) {
            try {
                double xVal, yVal;

                // Support CSV files with two or more columns
                if (tokens.size() == 2) {
                    // Format: x,y
                    xVal = std::stod(tokens[0]);
                    yVal = std::stod(tokens[1]);
                } else {
                    // Format: skip first column, use second and third columns as x,y
                    xVal = std::stod(tokens[1]);
                    yVal = std::stod(tokens[2]);
                }

                x.push_back(xVal);
                y.push_back(yVal);
                ++count;
            } catch (const std::exception& e) {
                std::cerr « "Error parsing line: " « line « " (" « e.what() « ")\n";
            }
        } else {
            std::cerr « "Invalid line format (less than 2 columns): " « line « "\n";
        }
    }

    std::cout « "Loaded " « count « " points\n";
}
```

Referenced by loadDialog().

Here is the caller graph for this function:



3.2.3.2 loadDialog()

```
void FileManager::loadDialog ( )
```

Opens a file dialog to select a file and loads its data into the map.

Opens a file dialog to select a CSV dataset file, loads the data, and updates the map accordingly.

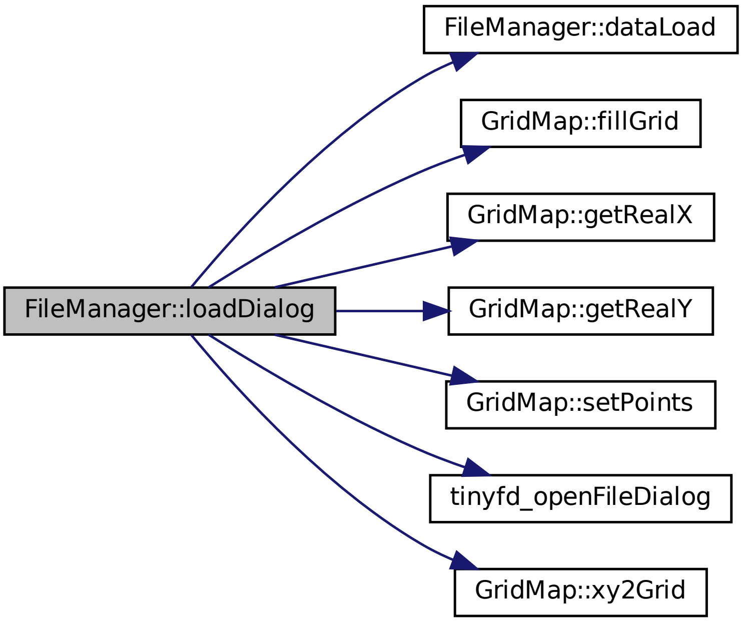
Definition at line 16 of file FileManager.cpp.

```
{
    const char* path = tinyfd_openFileDialog(
        "Open Dataset", // Dialog title
        "",             // Default path
        0,              // Number of filters (0 = none)
        nullptr,        // Filter patterns (ignored when count=0)
        nullptr,        // Filter description
        0               // Allow multiple selection? (0 = no)
    );

    if (path) {
        std::cout « "Selected file: " « path « std::endl;
        loadedFilename = path;

        // Load CSV data into vectors
        std::vector<double> xData, yData;
        dataLoad(loadedFilename, xData, yData);

        // Update map points with loaded real-world coordinates
        map.setPoints(xData, yData);

        // Convert real-world coordinates to grid indices and fill the occupancy grid
        std::vector<int> xGrid, yGrid;
        map.xy2Grid(map.getRealX(), map.getRealY(), xGrid, yGrid);
        map.fillGrid(xGrid, yGrid);
    } else {
        std::cout « "No file was selected.\n";
    }
}
```

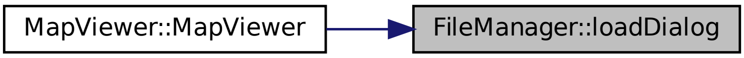
References dataLoad(), GridMap::fillGrid(), GridMap::getRealX(), GridMap::getRealY(), loadedFilename, map, GridMap::setPoints(), tinyfd_openFileDialog(), and GridMap::xy2Grid().

Referenced by MapViewer::MapViewer().

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3.3 saveData()

```
void FileManager::saveData (
            const std::string & filename,
            std::vector< double > & x,
            std::vector< double > & y )
```

Saves x and y coordinate data to a file.

Saves vectors of x and y coordinates to a CSV file.

Parameters

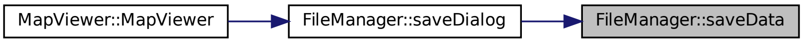
filename	Path of the file to write to
x	Vector of x coordinates
y	Vector of y coordinates
filename	Path to the output file.
x	Vector of x coordinates.
y	Vector of y coordinates.

Definition at line 138 of file FileManager.cpp.

```
{
    if (x.size() != y.size()) {
        std::cerr « "Error: x and y vector sizes do not match.\n";
        return;
    }

    std::ofstream outFile(filename);
    if (!outFile) {
        std::cerr « "Error: Cannot open file for writing.\n";
        return;
    }

    // Use high precision for floating point output
    outFile « std::fixed « std::setprecision(17);
    for (size_t i = 0; i < x.size(); ++i) {
        outFile « x[i] « "," « y[i] « "\n";
    }

    outFile.close();
    std::cout « "Text file saved successfully.\n";
}
```

Referenced by saveDialog().

Here is the caller graph for this function:



3.2.3.4 saveDialog()

```
void FileManager::saveDialog ( )
```

Opens a file dialog to save the current data (coordinates).

Opens a save file dialog and saves current real-world map data to a CSV file.

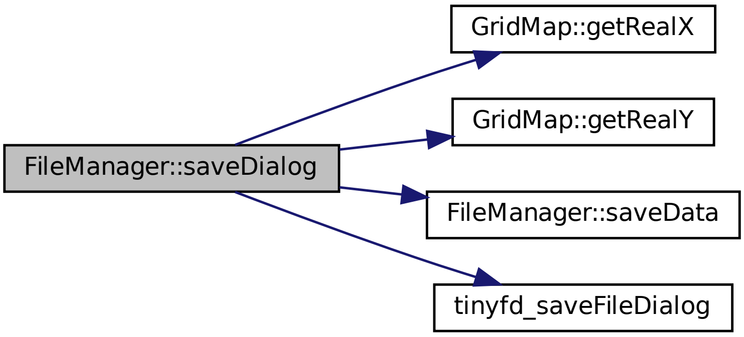
Definition at line 50 of file FileManager.cpp.

```
{
    const char* path = tinyfd_saveFileDialog(
        "Save File",       // Dialog title
        "output.txt",      // Default filename
        0,                 // Number of filters (0 = none)
        nullptr,           // Filter patterns
        nullptr            // Filter description
    );

    if (path) {
        std::cout « "Saving to: " « path « std::endl;

        // Save the real-world coordinates to the file
        saveData(path,
                const_cast<std::vector<double>&>(map.getRealX()),
                const_cast<std::vector<double>&>(map.getRealY()));
    } else {
        std::cout « "Save operation was canceled.\n";
    }
}
```

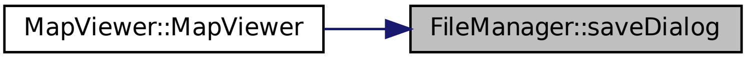
References GridMap::getRealX(), GridMap::getRealY(), map, saveData(), and tinyfd_saveFileDialog().

Referenced by MapViewer::MapViewer().

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3.5 saveScreen()

```
void FileManager::saveScreen (
            const std::string & filename )
```

Saves a screenshot of the current visualization to an image file.

Placeholder for saving a screenshot of the map window.

Parameters

filename	Path of the image file to create
filename	Suggested filename for saving.

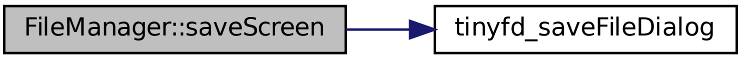
Note: Implementation to capture and save SFML window content is pending.

Definition at line 167 of file FileManager.cpp.

```
{
    const char* path = tinyfd_saveFileDialog(
        "Save Image",
        "Map.png",
        0,
        nullptr,
        nullptr
    );

    if (path) {
        // TODO: Implement screenshot capture and saving using SFML RenderWindow
        return;
    }
}
```

References tinyfd_saveFileDialog().

Here is the call graph for this function:



3.2.4 Member Data Documentation

3.2.4.1 loadedFilename

`std::string FileManager::loadedFilename [private]`

Path of the last loaded or saved file.

Definition at line 22 of file FileManager.hpp.

Referenced by loadDialog().

3.2.4.2 map

```
GridMap& FileManager::map [private]
```

Reference to the grid map object.

Definition at line 21 of file FileManager.hpp.

Referenced by loadDialog(), and saveDialog().

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

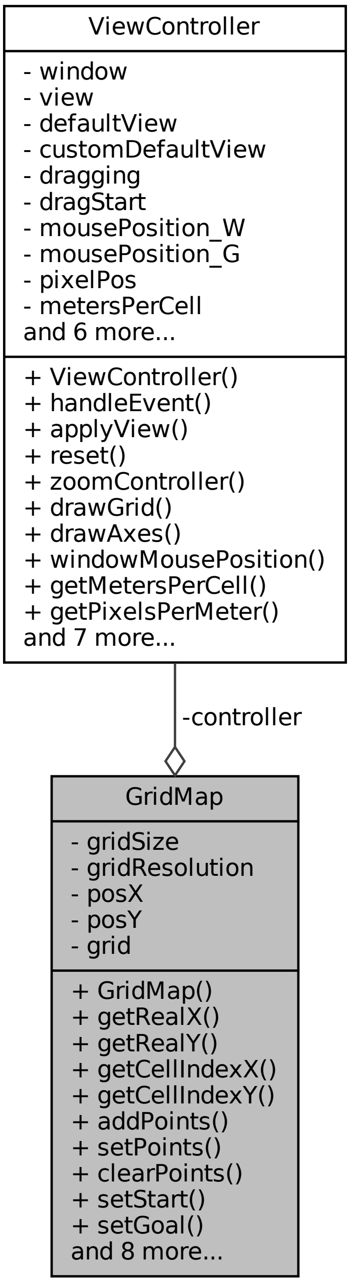
- include/artgslam_vsc/FileManager.hpp
- src/FileManager.cpp

3.3 GridMap Class Reference

Handles the creation and management of map data.

```
#include <GridMap.hpp>
```

Collaboration diagram for GridMap:



Public Member Functions

- GridMap (int size, double resolution, ViewController &controller)

  Constructor.
- const std::vector< double > & getRealX () const

  Getters for real-world x coordinates.
- const std::vector< double > & getRealY () const

Getters for real-world y coordinates.
- int getCellIndexX (double realX) const

  Convert a real-world x coordinate to grid index.
- int getCellIndexY (double realY) const

  Convert a real-world y coordinate to grid index.
- void addPoints (double x, double y)

  Add a point to the grid.
- void setPoints (const std::vector< double > &newX, const std::vector< double > &newY)

  Replace all current points with new ones.
- void clearPoints ()

  Clear all stored real-world points.
- void setStart (int i, int j)

  Set the start point in the grid (stored as ASCII 's').
- void setGoal (int i, int j)

  Set the goal point in the grid (stored as ASCII 'g').
- int isOccupied (int i, int j) const

  Query cell status.
- void xy2Grid (const std::vector< double > &x, const std::vector< double > &y, std::vector< int > &xGrid, std::vector< int > &yGrid)

  Convert real-world coordinates to grid indices.
- void fillGrid (const std::vector< int > &xGrid, const std::vector< int > &yGrid)

  Fill the occupancy grid using grid index vectors.
- const std::vector< std::vector< int > > & getGrid () const

  Get a const reference to the occupancy grid.
- void clearGridMap ()

  Clear the entire occupancy grid.
- void clearSetPoints (sf::Vector2i cellIndex)

  Clear start or goal markers from a cell.
- void draw (sf::RenderTarget &target, float pixelsPerMeter) const

  Draw the map using SFML.
- int getMapSize () const

  Get the number of cells per side.

Private Attributes

- int gridSize

  Number of cells in the grid (map size)
- double gridResolution

  Size of each cell in meters.
- std::vector< double > posX
- std::vector< double > posY

  Real-world coordinates from sonar data.
- std::vector< std::vector< int > > grid

  2D occupancy grid
- ViewController & controller

  Reference to view controller (for proper rendering)

3.3.1 Detailed Description

Handles the creation and management of map data.

It stores (x, y) coordinates from sonar readings, converts them into grid cell indices, and fills a 2D occupancy grid.

Definition at line 13 of file GridMap.hpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 GridMap()

```
GridMap::GridMap (
             int size,
             double resolution,
             ViewController & controller )
```

Constructor.

Constructs a GridMap with given size, resolution, and stores reference to the controller.

Parameters

size	Grid dimension (number of cells per side)
resolution	Size of each cell in meters
controller	Reference to the ViewController for rendering
size	Number of cells per grid side (grid is square).
resolution	Size of each cell in meters.
controller	Reference to the ViewController managing visualization.

Definition at line 11 of file GridMap.cpp.

```
    : gridSize(size), gridResolution(resolution), controller(controller)
{
    // Initialize the grid with all cells free (0)
    grid.assign(gridSize, std::vector<int>(gridSize, 0));
}
```

References grid, and gridSize.

3.3.3 Member Function Documentation

3.3.3.1 addPoints()

```
void GridMap::addPoints (
             double x,
             double y )
```

Add a point to the grid.

Adds a single real-world point to internal storage.

Parameters

x	Real-world x coordinate
y	Real-world y coordinate
x	X coordinate in meters.
y	Y coordinate in meters.

Definition at line 49 of file GridMap.cpp.

```
{
    posX.push_back(x);
    posY.push_back(y);
}
```

References posX, and posY.

3.3.3.2 clearGridMap()

```
void GridMap::clearGridMap ( )
```

Clear the entire occupancy grid.

Clears all cells in the grid, setting them to free (0).

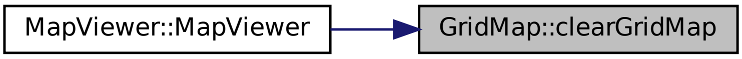
Definition at line 189 of file GridMap.cpp.

```
{
    grid.assign(gridSize, std::vector<int>(gridSize, 0));
}
```

References grid, and gridSize.

Referenced by MapViewer::MapViewer().

Here is the caller graph for this function:



3.3.3.3 clearPoints()

```
void GridMap::clearPoints ( )
```

Clear all stored real-world points.

Clears all stored real-world points.

Definition at line 69 of file GridMap.cpp.

```
{
    posX.clear();
    posY.clear();
}
```

References posX, and posY.

3.3.3.4 clearSetPoints()

```
void GridMap::clearSetPoints (
             sf::Vector2i cellIndex )
```

Clear start or goal markers from a cell.

Clears a specific grid cell by setting it free (0).

Parameters

cellIndex	Index of the cell to clear
cellIndex	Grid cell index to clear.

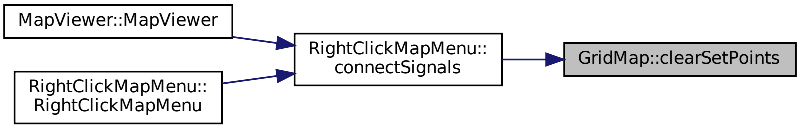
Definition at line 198 of file GridMap.cpp.

```
199 {
200     if (cellIndex.x >= 0 && cellIndex.y >= 0 &&
201         cellIndex.y < gridSize && cellIndex.x < gridSize) {
202         grid[cellIndex.y][cellIndex.x] = 0;
203     }
204 }
```

References grid, and gridSize.

Referenced by RightClickMapMenu::connectSignals().

Here is the caller graph for this function:



3.3.3.5 draw()

```
void GridMap::draw (
             sf::RenderTarget & target,
             float pixelsPerMeter ) const
```

Draw the map using SFML.

Draws the grid cells with obstacles, start and goal points on an SFML render target.

Only filled cells are rendered.

Parameters

target	Render target (usually the SFML window)
pixelsPerMeter	Scale factor for drawing
target	SFML RenderTarget to draw on.
pixelsPerMeter	Scaling factor for rendering.

Definition at line 211 of file GridMap.cpp.

```
{
    if (grid.empty() || grid[0].empty()) return;

    const int rows = static_cast<int>(grid.size());
    const int cols = static_cast<int>(grid[0].size());

    const float cellSize = gridResolution * pixelsPerMeter;

    // Center the grid rendering around origin (0,0)
    const float offsetX = -(cols / 2.f) * cellSize;
    const float offsetY = -(rows / 2.f) * cellSize;

    sf::RectangleShape cellShape({cellSize, cellSize});

    for (int row = 0; row < rows; ++row)
    {
        for (int col = 0; col < cols; ++col)
        {
            const int val = grid[row][col];

            // Assign colors based on cell value
            if (val == 1)
                cellShape.setFillColor(sf::Color::Yellow); // obstacle
            else if (val == 's')
                cellShape.setFillColor(sf::Color::Red);    // start
            else if (val == 'g')
                cellShape.setFillColor(sf::Color::Green);  // goal
            else
                continue; // skip free cells

            float x = offsetX + col * cellSize;
            float y = offsetY + row * cellSize;

            cellShape.setPosition(x, y);
            target.draw(cellShape);
        }
    }
}
```

References grid, and gridResolution.

Referenced by MapViewer::render().

Here is the caller graph for this function:

main
MapViewer::render
GridMap::draw

3.3.3.6 fillGrid()

```
void GridMap::fillGrid (
            const std::vector< int > & xGrid,
            const std::vector< int > & yGrid )
```

Fill the occupancy grid using grid index vectors.

Fills grid cells with obstacles based on given grid indices.

Parameters

xGrid	Vector of x indices
yGrid	Vector of y indices
xGrid	Vector of grid column indices.
yGrid	Vector of grid row indices.

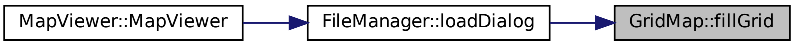
Definition at line 162 of file GridMap.cpp.

```
{
    // Reset grid to free cells
    grid.assign(gridSize, std::vector<int>(gridSize, 0));

    for (size_t i = 0; i < xGrid.size(); ++i) {
        int x = xGrid[i];
        int y = yGrid[i];

        if (x >= 0 && x < gridSize && y >= 0 && y < gridSize) {
            grid[y][x] = 1; // Mark obstacle
        }
    }
}
```

References grid, and gridSize.

Referenced by FileManager::loadDialog().

Here is the caller graph for this function:



3.3.3.7 getCellIndexX()

```
int GridMap::getCellIndexX (
              double realX ) const
```

Convert a real-world x coordinate to grid index.

Converts real-world X coordinate to grid column index.

Parameters

realX	X coordinate in meters

Returns

Corresponding grid index

Parameters

realX	X coordinate in meters.

Returns

Grid column index corresponding to realX.

Definition at line 23 of file GridMap.cpp.

```
{
    double offset = (gridSize * gridResolution) / 2.0; // Centered origin offset
    int j = static_cast<int>(std::floor((realX + offset) / gridResolution));
    j = std::clamp(j, 0, gridSize - 1);
    return j;
}
```

References gridResolution, and gridSize.

3.3.3.8 getCellIndexY()

```
int GridMap::getCellIndexY (
              double realY ) const
```

Convert a real-world y coordinate to grid index.

Converts real-world Y coordinate to grid row index.

Parameters

realY	Y coordinate in meters

Returns

Corresponding grid index

Parameters

realY	Y coordinate in meters.

Returns

Grid row index corresponding to realY.

Definition at line 36 of file GridMap.cpp.

```
{
    double offset = (gridSize * gridResolution) / 2.0; // Centered origin offset
    int i = static_cast<int>(std::floor((realY + offset) / gridResolution));
    i = std::clamp(i, 0, gridSize - 1);
    return i;
}
```

References gridResolution, and gridSize.

3.3.3.9 getGrid()

```
const std::vector< std::vector< int > > & GridMap::getGrid ( ) const
```

Get a const reference to the occupancy grid.

Returns a const reference to the internal grid representation.

Returns

2D grid of occupancy values

Reference to grid.

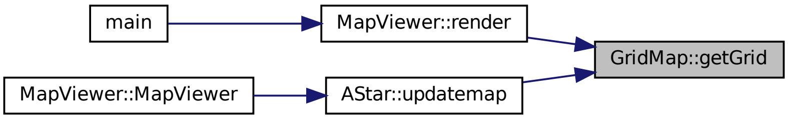
Definition at line 181 of file GridMap.cpp.

```
182 {
183     return grid;
184 }
```

References grid.

Referenced by MapViewer::render(), and AStar::updatemap().

Here is the caller graph for this function:



3.3.3.10 getMapSize()

```
int GridMap::getMapSize ( ) const  [inline]
```

Get the number of cells per side.

Returns

Grid size (width or height)

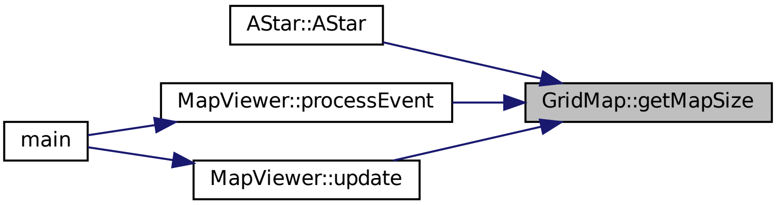
Definition at line 155 of file GridMap.hpp.

```
155 { return gridSize; }
```

References gridSize.

Referenced by AStar::AStar(), MapViewer::processEvent(), and MapViewer::update().

Here is the caller graph for this function:



3.3.3.11 getRealX()

```
const std::vector<double>& GridMap::getRealX ( ) const  [inline]
```

Getters for real-world x coordinates.

Returns

Vector of x coordinates

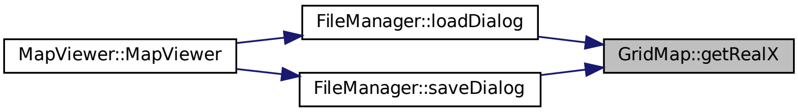
Definition at line 37 of file GridMap.hpp.

```
37 { return posX; }
```

References posX.

Referenced by FileManager::loadDialog(), and FileManager::saveDialog().

Here is the caller graph for this function:



3.3.3.12 getRealY()

```
const std::vector<double>& GridMap::getRealY ( ) const  [inline]
```

Getters for real-world y coordinates.

Returns

Vector of y coordinates

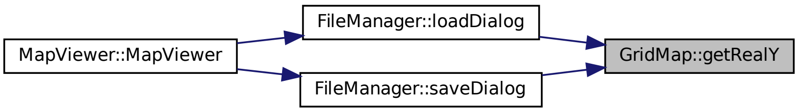
Definition at line 43 of file GridMap.hpp.

```
43 { return posY; }
```

References posY.

Referenced by FileManager::loadDialog(), and FileManager::saveDialog().

Here is the caller graph for this function:



3.3.3.13 isOccupied()

```
int GridMap::isOccupied (
             int i,
             int j ) const
```

Query cell status.

Checks if a grid cell is occupied or special.

Parameters

i	Row index
j	Column index

Returns

1 = occupied, 0 = free, ASCII 's' = start, ASCII 'g' = goal

Parameters

i	Column index in grid.
j	Row index in grid.

Returns

1 if obstacle, 0 if free, 's' for start, 'g' for goal, else 1 (occupied).

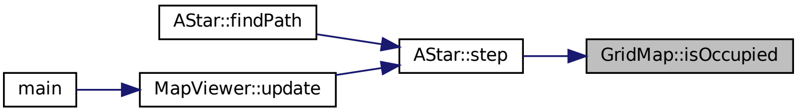
Definition at line 113 of file GridMap.cpp.

```
{
    if (grid[j][i] == 1) {
        return 1; // Obstacle
    }
    if (grid[j][i] == 0) {
        return 0; // Free cell
    }
    if (grid[j][i] == 's') {
        return 's'; // Start
    }
    if (grid[j][i] == 'g') {
        return 'g'; // Goal
    }
    // Default to occupied for unexpected values
    return 1;
}
```

References grid.

Referenced by AStar::step().

Here is the caller graph for this function:



3.3.3.14 setGoal()

```
void GridMap::setGoal (
              int col,
              int row )
```

Set the goal point in the grid (stored as ASCII 'g').

Sets the goal point on the grid if the cell is free.

Parameters

i	Row index
j	Column index
col	Column index in grid.
row	Row index in grid.

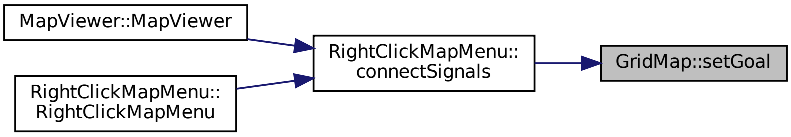
Definition at line 96 of file GridMap.cpp.

```
{
    std::cout « "[setGoal] col=" « col « " row=" « row « ’\n’;
    if (grid[row][col] == 0) {
        grid[row][col] = ’g’; // Mark cell as goal using ASCII code
        std::cout « "Goal set: " « grid[row][col] « std::endl;
     } else {
        std::cout « "[setGoal] Cell is occupied." « std::endl;
     }
}
```

References grid.

Referenced by RightClickMapMenu::connectSignals().

Here is the caller graph for this function:



3.3.3.15 setPoints()

```
void GridMap::setPoints (
            const std::vector< double > & newX,
            const std::vector< double > & newY )
```

Replace all current points with new ones.

Replaces stored points with new vectors of real-world coordinates.

Parameters

newX	New vector of x coordinates
newY	New vector of y coordinates
newX	Vector of X coordinates.
newY	Vector of Y coordinates.

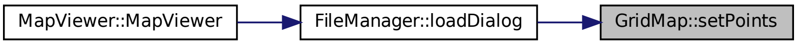
Definition at line 60 of file GridMap.cpp.

```
{
    posX = newX;
    posY = newY;
}
```

References posX, and posY.

Referenced by FileManager::loadDialog().

Here is the caller graph for this function:



3.3.3.16 setStart()

```
void GridMap::setStart (
              int col,
              int row )
```

Set the start point in the grid (stored as ASCII 's').

Sets the start point on the grid if the cell is free.

Parameters

i	Row index
j	Column index
col	Column index in grid.
row	Row index in grid.

Definition at line 80 of file GridMap.cpp.

```
{
    std::cout « "[setStart] col=" « col « " row=" « row « '\n';
    if (grid[row][col] == 0) {
        grid[row][col] = 's'; // Mark cell as start using ASCII code
        std::cout « "Start set: " « grid[row][col] « std::endl;
    } else {
        std::cout « "[setStart] Cell is occupied." « std::endl;
    }
}
```

References grid.

3.3.3.17 xy2Grid()

```
void GridMap::xy2Grid (
              const std::vector< double > & x,
              const std::vector< double > & y,
              std::vector< int > & xGrid,
              std::vector< int > & yGrid )
```

Convert real-world coordinates to grid indices.

Converts vectors of real-world coordinates to grid indices.

Parameters

x	Input vector of x coordinates
y	Input vector of y coordinates
xGrid	Output vector of x indices
yGrid	Output vector of y indices
x	Vector of X coordinates.
y	Vector of Y coordinates.
xGrid	Output vector of grid column indices.
yGrid	Output vector of grid row indices.

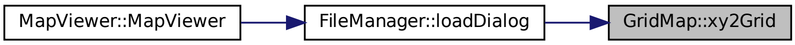
Definition at line 138 of file GridMap.cpp.

```
{
    const int halfGrid = gridSize / 2;
    size_t n = std::min(x.size(), y.size());

    xGrid.resize(n);
    yGrid.resize(n);

    for (size_t i = 0; i < n; ++i) {
        int gx = static_cast<int>(std::round(x[i] / gridResolution));
        int gy = static_cast<int>(std::round(y[i] / gridResolution));

        // Translate to centered origin indices
        xGrid[i] = gx + halfGrid;
        yGrid[i] = gy + halfGrid;
    }
}
```

References gridResolution, and gridSize.

Referenced by FileManager::loadDialog().

Here is the caller graph for this function:



3.3.4 Member Data Documentation

3.3.4.1 controller

```
ViewController& GridMap::controller  [private]
```

Reference to view controller (for proper rendering)

Definition at line 21 of file GridMap.hpp.

3.3.4.2 grid

```
std::vector<std::vector<int> > GridMap::grid  [private]
```

2D occupancy grid

Definition at line 19 of file GridMap.hpp.

Referenced by clearGridMap(), clearSetPoints(), draw(), fillGrid(), getGrid(), GridMap(), isOccupied(), setGoal(), and setStart().

3.3.4.3 gridResolution

```
double GridMap::gridResolution  [private]
```

Size of each cell in meters.

Definition at line 17 of file GridMap.hpp.

Referenced by draw(), getCellIndexX(), getCellIndexY(), and xy2Grid().

3.3.4.4 gridSize

```
int GridMap::gridSize  [private]
```

Number of cells in the grid (map size)

Definition at line 16 of file GridMap.hpp.

Referenced by clearGridMap(), clearSetPoints(), fillGrid(), getCellIndexX(), getCellIndexY(), getMapSize(), GridMap(), and xy2Grid().

3.3.4.5 posX

```
std::vector<double> GridMap::posX  [private]
```

Definition at line 18 of file GridMap.hpp.

Referenced by addPoints(), clearPoints(), getRealX(), and setPoints().

3.3.4.6 posY

```
std::vector<double> GridMap::posY  [private]
```

Real-world coordinates from sonar data.

Definition at line 18 of file GridMap.hpp.

Referenced by addPoints(), clearPoints(), getRealY(), and setPoints().

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

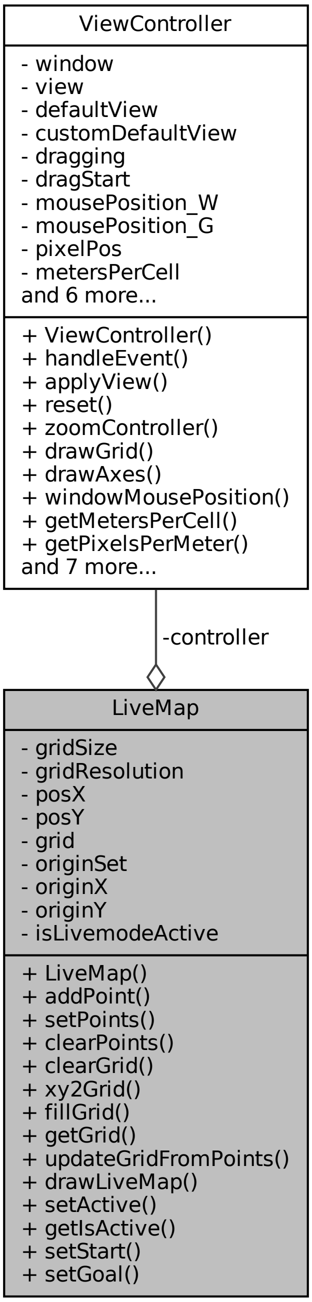
- include/artgslam_vsc/GridMap.hpp
- src/GridMap.cpp

3.4 LiveMap Class Reference

This class manages live map creation in "live mode", by dynamically updating the map from incoming ROS data (e.g., from RosHandler). It builds an occupancy grid in real time and draws it using a ViewController.

```
#include <LiveMap.hpp>
```

Collaboration diagram for LiveMap:



Public Member Functions

- LiveMap (int size, double resolution, ViewController &controller)

  Constructor: initializes map size, resolution and controller reference.
- void addPoint (double x, double y)

  Adds a single point (in real-world coordinates) to the internal point buffer.
- void setPoints (const std::vector< double > &newX, const std::vector< double > &newY)

Replaces current point buffer with a new set of coordinates.
- void clearPoints ()

  Clears all stored real-world points.
- void clearGrid ()

  Clears the occupancy grid (sets all cells to 0)
- void xy2Grid (const std::vector< double > &x, const std::vector< double > &y, std::vector< int > &xGrid, std::vector< int > &yGrid)

  Converts real-world coordinates to grid indices.
- void fillGrid (const std::vector< int > &xGrid, const std::vector< int > &yGrid)

  Fills the grid using the given grid indices, marking cells as occupied (1)
- const std::vector< std::vector< int > > & getGrid () const

  Returns a constant reference to the full occupancy grid.
- void updateGridFromPoints ()

  Updates the grid from the internal point buffer.
- void drawLiveMap (sf::RenderTarget &target) const

  Draws the live grid using the ViewController.
- void setActive (bool isActive)

  Activates or deactivates live mode (when true, dynamic updates occur)
- bool getIsActive () const

  Returns whether live mode is currently active.
- void setStart (int i, int j)

  Sets a special start cell in the grid (value = 's')
- void setGoal (int i, int j)

  Sets a special goal cell in the grid (value = 'g')

Private Attributes

- ViewController & controller

  Reference to the view controller for rendering.
- int gridSize

  Map size (number of cells per side)
- double gridResolution

  Size of each cell in real-world units.
- std::vector< double > posX
- std::vector< double > posY

  Buffer of real-world x/y points.
- std::vector< std::vector< int > > grid

  2D occupancy grid (0 = free, 1 = occupied, 's' = start, 'g' = goal)
- bool originSet = false

  Flag to know if the origin is initialized.
- double originX = 0.0
- double originY = 0.0

  Optional offset for positioning.
- bool isLivemodeActive = false

  Whether live updates are happening.

3.4.1 Detailed Description

This class manages live map creation in "live mode", by dynamically updating the map from incoming ROS data (e.g., from RosHandler). It builds an occupancy grid in real time and draws it using a ViewController.

Definition at line 13 of file LiveMap.hpp.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 LiveMap()

```
LiveMap::LiveMap (
            int size,
            double resolution,
            ViewController & controller )
```

Constructor: initializes map size, resolution and controller reference.

Constructs a LiveMap object with specified grid size and resolution.

Parameters

size	Number of grid cells per side
resolution	Size of each grid cell in meters
controller	Reference to the view controller

Initializes the occupancy grid and links to the ViewController.

Parameters

size	Number of grid cells per side (square grid).
resolution	Cell size in meters.
controller	Reference to ViewController for visualization parameters.

Definition at line 11 of file LiveMap.cpp.

```
    : gridSize(size), gridResolution(resolution), controller(controller), originSet(false)
{
    // Initialize grid to all free cells (0)
    grid.resize(gridSize, std::vector<int>(gridSize, 0));

    // Clear stored real-world points
    posX.clear();
    posY.clear();
}
```

References grid, gridSize, posX, and posY.

3.4.3 Member Function Documentation

3.4.3.1 addPoint()

```
void LiveMap::addPoint (
            double x,
            double y )
```

Adds a single point (in real-world coordinates) to the internal point buffer.

Adds a new real-world point to the list.

Parameters

x	Real-world X coordinate
y	Real-world Y coordinate

The first point added is used to set the origin for coordinate conversion.

Parameters

x	X coordinate in meters.
y	Y coordinate in meters.

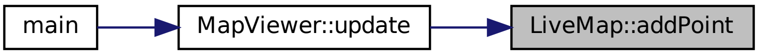
Definition at line 28 of file LiveMap.cpp.

```
{
    if (!originSet) {
        originX = x;
        originY = y;
        originSet = true;
    }
    posX.push_back(x);
    posY.push_back(y);
}
```

References originSet, originX, originY, posX, and posY.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.4.3.2 clearGrid()

```
void LiveMap::clearGrid ( )
```

Clears the occupancy grid (sets all cells to 0)

Resets the occupancy grid cells to free (0).

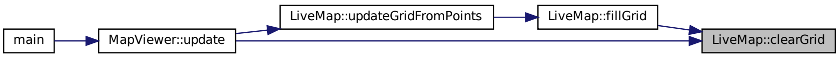
Definition at line 63 of file LiveMap.cpp.

```
{
    for (auto& row : grid) {
        std::fill(row.begin(), row.end(), 0);
    }
}
```

References grid.

Referenced by fillGrid(), and MapViewer::update().

Here is the caller graph for this function:



3.4.3.3 clearPoints()

```
void LiveMap::clearPoints ( )
```

Clears all stored real-world points.

Clears all stored real-world points and resets the origin flag.

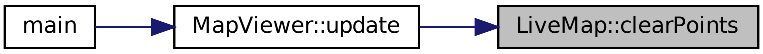
Definition at line 53 of file LiveMap.cpp.

```
54 {
55     posX.clear();
56     posY.clear();
57     originSet = false;
58 }
```

References originSet, posX, and posY.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.4.3.4 drawLiveMap()

```
void LiveMap::drawLiveMap (
             sf::RenderTarget & target ) const
```

Draws the live grid using the ViewController.

Draws occupied cells of the live map on the given SFML render target.

Parameters

target	SFML render target

The drawing is centered and scaled according to the controller parameters.

Parameters

target	SFML RenderTarget to draw on.

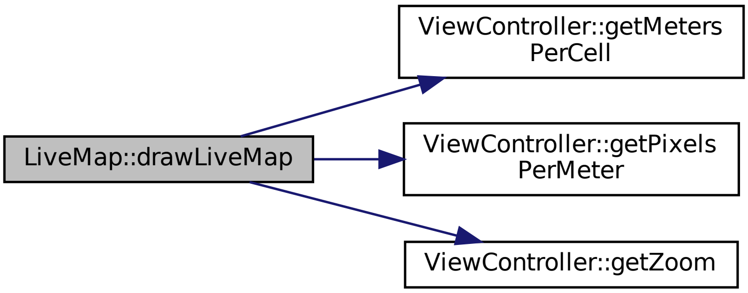
Definition at line 141 of file LiveMap.cpp.

```
{
    if (grid.empty()) return;

    float metersPerCell = controller.getMetersPerCell();   // e.g., 0.1
    float pixelsPerMeter = controller.getPixelsPerMeter(); // e.g., 50.0

    float cellSize = metersPerCell * pixelsPerMeter;

    int rows = static_cast<int>(grid.size());
    int cols = static_cast<int>(grid[0].size());

    int halfCols = cols / 2;
    int halfRows = rows / 2;

    float zoom = controller.getZoom();

    sf::RectangleShape cellShape;
    cellShape.setFillColor(sf::Color::Magenta);
    cellShape.setSize(sf::Vector2f(cellSize / zoom, cellSize / zoom));

    for (int row = 0; row < rows; ++row) {
        for (int col = 0; col < cols; ++col) {
            if (grid[row][col] == 1) {
                int cellX = col - halfCols;
                int cellY = row - halfRows;

                float x = static_cast<float>(cellX) * cellSize / zoom;
                float y = static_cast<float>(cellY) * cellSize / zoom;

                cellShape.setPosition(x, y);
                target.draw(cellShape);
            }
        }
    }
}
```

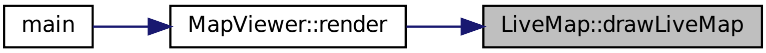
References controller, ViewController::getMetersPerCell(), ViewController::getPixelsPerMeter(), ViewController::getZoom(), and grid.

Referenced by MapViewer::render().

Here is the call graph for this function:


LiveMap::drawLiveMap
ViewController::getMeters PerCell
ViewController::getPixels PerMeter
ViewController::getZoom

Here is the caller graph for this function:



3.4.3.5 fillGrid()

```
void LiveMap::fillGrid (
            const std::vector< int > & xGrid,
            const std::vector< int > & yGrid )
```

Fills the grid using the given grid indices, marking cells as occupied (1)

Fills the grid with obstacles at specified grid indices.

Parameters

xGrid	Vector of X grid indices
yGrid	Vector of Y grid indices

Previous occupancy data is cleared.

Parameters

xGrid	Vector of X grid indices.
yGrid	Vector of Y grid indices.

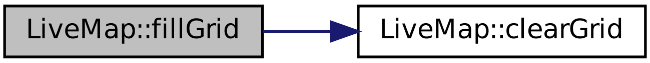
Definition at line 110 of file LiveMap.cpp.

```
{
    if (xGrid.size() != yGrid.size()) return;

    clearGrid();

    for (size_t i = 0; i < xGrid.size(); ++i) {
        int xIdx = xGrid[i];
        int yIdx = yGrid[i];

        grid[yIdx][xIdx] = 1; // Mark cell as occupied
    }
}
```

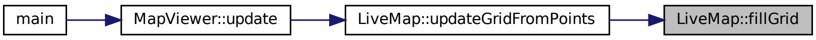
References clearGrid(), and grid.

Referenced by updateGridFromPoints().

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.6 getGrid()

```
const std::vector<std::vector<int> >& LiveMap::getGrid ( ) const  [inline]
```

Returns a constant reference to the full occupancy grid.

Returns

Constant reference to grid

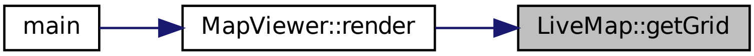
Definition at line 68 of file LiveMap.hpp.

```
68 { return grid; };
```

References grid.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.4.3.7 getIsActive()

```
bool LiveMap::getIsActive ( ) const  [inline]
```

Returns whether live mode is currently active.

Returns

True if live mode is active

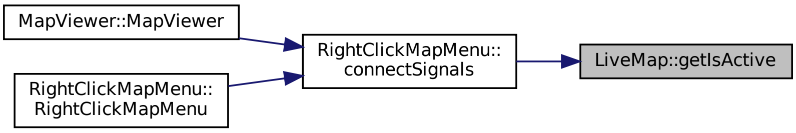
Definition at line 91 of file LiveMap.hpp.

```
91 { return isLivemodeActive; };
```

References isLivemodeActive.

Referenced by RightClickMapMenu::connectSignals().

Here is the caller graph for this function:


MapViewer::MapViewer
RightClickMapMenu::RightClickMapMenu
RightClickMapMenu::connectSignals
LiveMap::getIsActive

3.4.3.8 setActive()

```
void LiveMap::setActive (
             bool isActive )  [inline]
```

Activates or deactivates live mode (when true, dynamic updates occur)

Parameters

isActive	True to activate, false to deactivate

Definition at line 85 of file LiveMap.hpp.

```
85 { isLivemodeActive = true; };
```

References isLivemodeActive.

3.4.3.9 setGoal()

```
void LiveMap::setGoal (
             int i,
             int j )
```

Sets a special goal cell in the grid (value = 'g')

Marks a cell as the goal in the grid using ASCII code 'g' (103).

Parameters

i	Grid row index
j	Grid column index
i	Row index.
j	Column index.

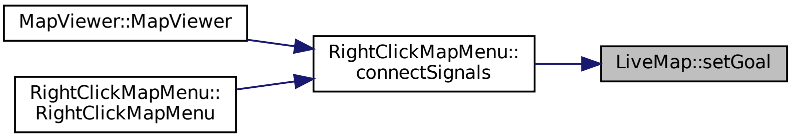
Definition at line 183 of file LiveMap.cpp.

```
184 {
185     grid[i][j] = 103; // ASCII 'g'
186 }
```

References grid.

Referenced by RightClickMapMenu::connectSignals().

Here is the caller graph for this function:



3.4.3.10 setPoints()

```
void LiveMap::setPoints (
            const std::vector< double > & newX,
            const std::vector< double > & newY )
```

Replaces current point buffer with a new set of coordinates.

Replaces stored points with a new set.

Parameters

newX	New set of X coordinates
newY	New set of Y coordinates
newX	Vector of X coordinates.
newY	Vector of Y coordinates.

Definition at line 44 of file LiveMap.cpp.

```
{
    posX = newX;
    posY = newY;
}
```

References posX, and posY.

3.4.3.11 setStart()

```
void LiveMap::setStart (
               int i,
               int j )
```

Sets a special start cell in the grid (value = 's')

Marks a cell as the start in the grid using ASCII code 's' (115).

Parameters

i	Grid row index
j	Grid column index
i	Row index.
j	Column index.

Definition at line 193 of file LiveMap.cpp.

```
{
    grid[i][j] = 115; // ASCII 's'
}
```

References grid.

3.4.3.12 updateGridFromPoints()

```
void LiveMap::updateGridFromPoints ( )
```

Updates the grid from the internal point buffer.

Updates the occupancy grid based on the stored real-world points.

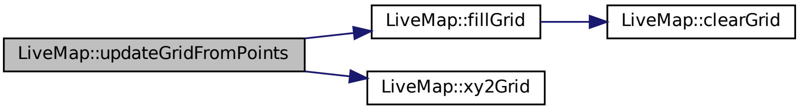
Definition at line 127 of file LiveMap.cpp.

```
{
    if (posX.empty() || posY.empty()) return;

    std::vector<int> xGrid, yGrid;
    xy2Grid(posX, posY, xGrid, yGrid);
    fillGrid(xGrid, yGrid);
}
```

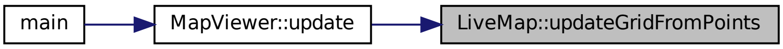
References fillGrid(), posX, posY, and xy2Grid().

Referenced by MapViewer::update().

Here is the call graph for this function:



Here is the caller graph for this function:



3.4.3.13 xy2Grid()

```
void LiveMap::xy2Grid (
            const std::vector< double > & x,
            const std::vector< double > & y,
            std::vector< int > & xGrid,
            std::vector< int > & yGrid )
```

Converts real-world coordinates to grid indices.

Converts real-world coordinates to grid indices relative to the origin.

Parameters

x	Vector of real-world X coordinates
y	Vector of real-world Y coordinates
xGrid	Output vector for grid X indices
yGrid	Output vector for grid Y indices

Points outside the grid are ignored.

Parameters

x	Vector of X coordinates.
y	Vector of Y coordinates.
xGrid	Output vector of grid X indices.
yGrid	Output vector of grid Y indices.

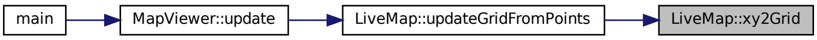
Definition at line 78 of file LiveMap.cpp.

```
{
    if (x.size() != y.size() || !originSet) return;

    xGrid.clear();
    yGrid.clear();

    int halfGrid = gridSize / 2;

    for (size_t i = 0; i < x.size(); ++i) {
        double shiftedX = x[i] - originX;
        double shiftedY = y[i] - originY;

        int xIdx = static_cast<int>(std::round(shiftedX / gridResolution)) + halfGrid;
        int yIdx = static_cast<int>(std::round(shiftedY / gridResolution)) + halfGrid;

        // Ignore points outside grid bounds
        if (xIdx < 0 || xIdx >= gridSize || yIdx < 0 || yIdx >= gridSize)
            continue;

        xGrid.push_back(xIdx);
        yGrid.push_back(yIdx);
    }
}
```

References gridResolution, gridSize, originSet, originX, and originY.

Referenced by updateGridFromPoints().

Here is the caller graph for this function:



3.4.4 Member Data Documentation

3.4.4.1 controller

`ViewController& LiveMap::controller [private]`

Reference to the view controller for rendering.

Definition at line 108 of file LiveMap.hpp.

Referenced by drawLiveMap().

3.4.4.2 grid

`std::vector<std::vector<int> > LiveMap::grid [private]`

2D occupancy grid (0 = free, 1 = occupied, 's' = start, 'g' = goal)

Definition at line 113 of file LiveMap.hpp.

Referenced by clearGrid(), drawLiveMap(), fillGrid(), getGrid(), LiveMap(), setGoal(), and setStart().

3.4.4.3 gridResolution

```
double LiveMap::gridResolution  [private]
```

Size of each cell in real-world units.

Definition at line 110 of file LiveMap.hpp.

Referenced by xy2Grid().

3.4.4.4 gridSize

```
int LiveMap::gridSize  [private]
```

Map size (number of cells per side)

Definition at line 109 of file LiveMap.hpp.

Referenced by LiveMap(), and xy2Grid().

3.4.4.5 isLivemodeActive

```
bool LiveMap::isLivemodeActive = false  [private]
```

Whether live updates are happening.

Definition at line 118 of file LiveMap.hpp.

Referenced by getIsActive(), and setActive().

3.4.4.6 originSet

```
bool LiveMap::originSet = false  [private]
```

Flag to know if the origin is initialized.

Definition at line 115 of file LiveMap.hpp.

Referenced by addPoint(), clearPoints(), and xy2Grid().

3.4.4.7 originX

```
double LiveMap::originX = 0.0  [private]
```

Definition at line 116 of file LiveMap.hpp.

Referenced by addPoint(), and xy2Grid().

3.4.4.8 originY

```
double LiveMap::originY = 0.0  [private]
```

Optional offset for positioning.

Definition at line 116 of file LiveMap.hpp.

Referenced by addPoint(), and xy2Grid().

3.4.4.9 posX

```
std::vector<double> LiveMap::posX  [private]
```

Definition at line 112 of file LiveMap.hpp.

Referenced by addPoint(), clearPoints(), LiveMap(), setPoints(), and updateGridFromPoints().

3.4.4.10 posY

```
std::vector<double> LiveMap::posY  [private]
```

Buffer of real-world x/y points.

Definition at line 112 of file LiveMap.hpp.

Referenced by addPoint(), clearPoints(), LiveMap(), setPoints(), and updateGridFromPoints().

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

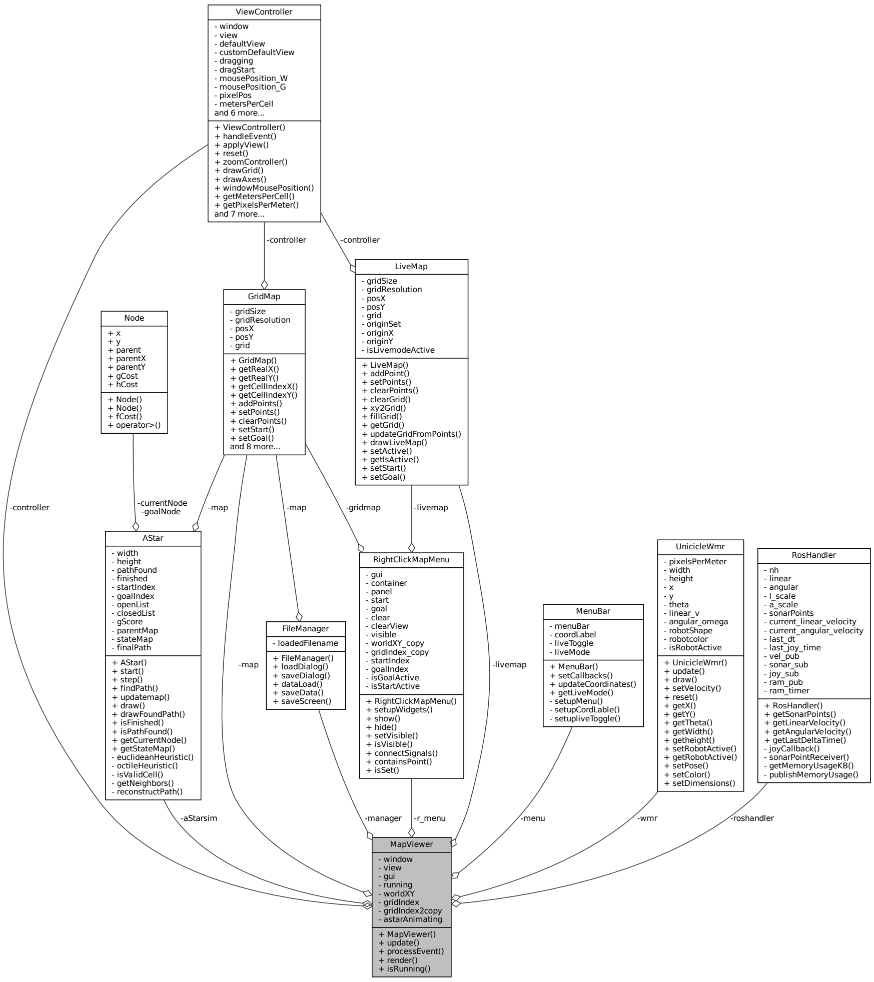
- include/artgslam_vsc/LiveMap.hpp
- src/LiveMap.cpp

3.5 MapViewer Class Reference

This is the main class. It manages mouse/keyboard events and GUI integration. It also coordinates the rendering and simulation components of the application.

```
#include <MapViewer.hpp>
```

Collaboration diagram for MapViewer:



Public Member Functions

- MapViewer (sf::RenderWindow &win)

  Constructor.
- void update ()

  Manages logic updates and user interaction.
- void processEvent ()

  Handles input events (mouse and keyboard)
- void render ()

  Renders all visual components to the screen.
- bool isRunning () const

  Returns whether the viewer should keep running.

Private Attributes

- sf::RenderWindow & window

  Reference to the SFML window for rendering.
- sf::View view

  SFML camera view.
- tgui::Gui gui

  GUI system for handling menus and widgets.
- MenuBar menu

  Instance of the custom menu bar.
- FileManager manager

  Manages file loading/saving.
- RosHandler roshandler

  Handles communication with ROS (publishing/subscribing)
- ViewController controller

  Manages zoom, panning, grid drawing, and coordinate conversions.
- GridMap map

  Represents and stores the occupancy grid.
- UnicicleWmr wmr

  Simulated unicycle WMR (Wheeled Mobile Robot)
- LiveMap livemap

  Handles live mapping mode based on ROS data.
- RightClickMapMenu r_menu

  Context menu for selecting start and goal positions.
- AStar aStarsim

  A∗ algorithm instance for path planning and animation.
- bool running = true

  Indicates whether the main loop is running.
- sf::Vector2f worldXY

  World coordinates (floating point)
- sf::Vector2i gridIndex

  Current hovered grid cell.
- sf::Vector2i gridIndex2copy

  Temporary copy of a selected grid index.
- bool astarAnimating = false

  Indicates whether the A∗ animation is currently running.

3.5.1 Detailed Description

This is the main class. It manages mouse/keyboard events and GUI integration. It also coordinates the rendering and simulation components of the application.

Definition at line 21 of file MapViewer.hpp.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 MapViewer()

```
MapViewer::MapViewer (
             sf::RenderWindow & win )
```

Constructor.

Constructs a MapViewer instance.

Parameters

win	Reference to the SFML render window

Initializes references to the SFML window, GUI, controller, map, menu, ROS handler, robot model, live map, right-click menu, and A∗ simulation. Sets up menu callbacks and connects right-click menu signals.

Parameters

win	Reference to the SFML RenderWindow where rendering occurs.

< Connect right-click menu event callbacks

< Load map file dialog

< Save map file dialog

< Exit application

< Reset camera view

< Clear the grid map

< Open robot creator tool

< Run A∗ pathfinding animation

< Update start and goal points

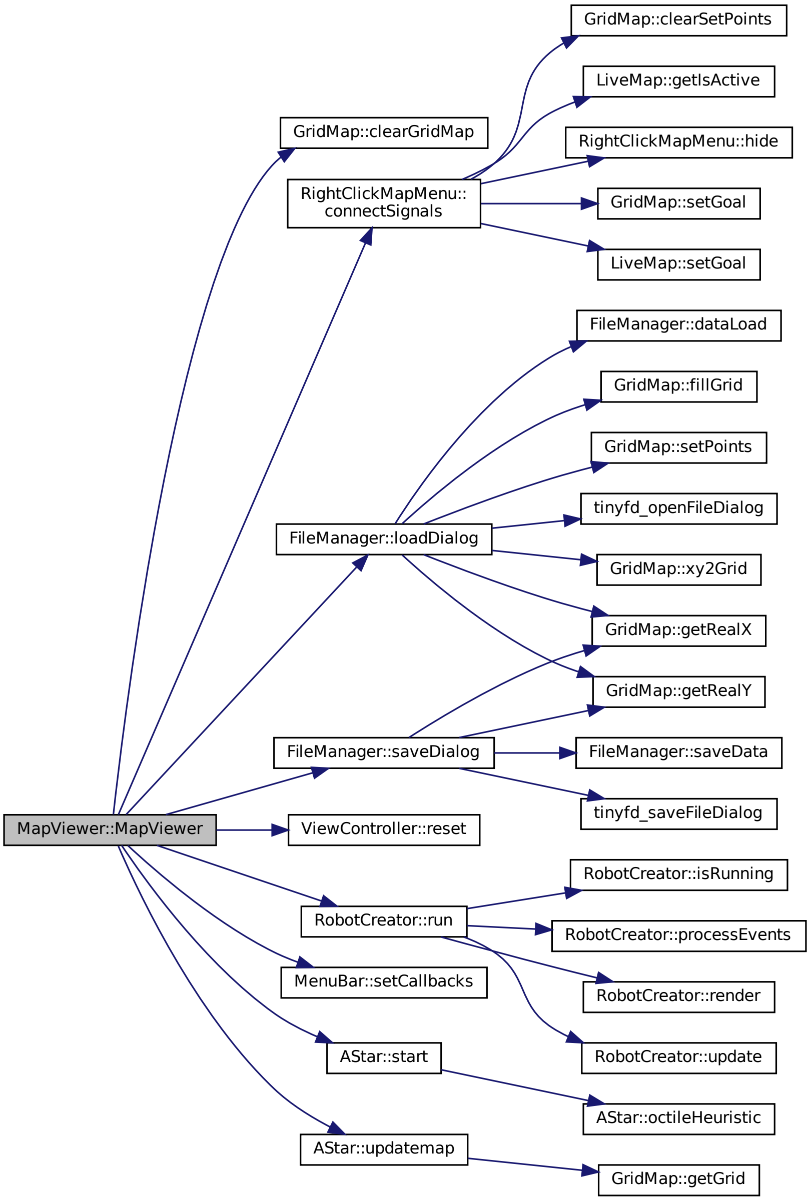
< Start A∗ animation

Definition at line 15 of file MapViewer.cpp.

```
    : window(win)                               // Store reference to SFML window
    , view(window.getDefaultView())            // Initialize default camera view
    , gui(win)                                // Initialize GUI with window reference
    , controller(win, 0.1f, 50.0f, view)      // Initialize ViewController with parameters
    , map(1000, 0.1, controller)               // Initialize GridMap with size, resolution, controller
    , manager(map)                             // FileManager with reference to GridMap
    , menu(gui)                               // MenuBar with GUI reference
    , roshandler()                           // ROS data handler for sensors and velocity
    , wmr()                                  // Wheeled Mobile Robot model
    , livemap(1000, 0.1, controller)          // LiveMap with same size and resolution
    , r_menu(gui, map, livemap)               // Right-click menu with GUI, map, and livemap refs
    , aStarsim(map)                           // A* pathfinding simulator with GridMap
{
    r_menu.connectSignals();  /**< Connect right-click menu event callbacks */

    // Set callback functions for menu bar actions
    menu.setCallbacks(
        [this]() { manager.loadDialog(); },              /**< Load map file dialog */
        [this]() { manager.saveDialog(); },              /**< Save map file dialog */
        [this]() { /* TODO: Implement image saving functionality */ },
        [this]() { running = false; window.close(); },   /**< Exit application */
        [this]() { controller.reset(); },                /**< Reset camera view */
        [this]() { map.clearGridMap(); },                /**< Clear the grid map */
        [this]() {                                        /**< Open robot creator tool */
            RobotCreator creator(wmr);
            creator.run();
        },
        [this]() {                                        /**< Run A* pathfinding animation */
            aStarsim.updatemap();  /**< Update start and goal points */
            aStarsim.start();      /**< Start A* animation */
            astarAnimating = true;
        }
    );
}
```

References astarAnimating, aStarsim, GridMap::clearGridMap(), RightClickMapMenu::connectSignals(), controller, FileManager::loadDialog(), manager, map, menu, r_menu, ViewController::reset(), RobotCreator::run(), running, FileManager::saveDialog(), MenuBar::setCallbacks(), AStar::start(), AStar::updatemap(), window, and wmr.

Here is the call graph for this function:



3.5.3 Member Function Documentation

3.5.3.1 isRunning()

```
bool MapViewer::isRunning ( ) const
```

Returns whether the viewer should keep running.

Checks if the MapViewer application is running.

Returns

True if running, false otherwise

true if the application is running and window is open, false otherwise.

Definition at line 218 of file MapViewer.cpp.

```
{
    return running && window.isOpen();
}
```

References running, and window.

Referenced by main().

Here is the caller graph for this function:

3.5.3.2 processEvent()

```
void MapViewer::processEvent ( )
```

Handles input events (mouse and keyboard)

Processes SFML window events.

Handles user inputs including window closing, mouse clicks, and GUI events. Shows or hides the right-click menu on mouse events. < Forward event to GUI system

< Forward event to view controller

< Close window and exit

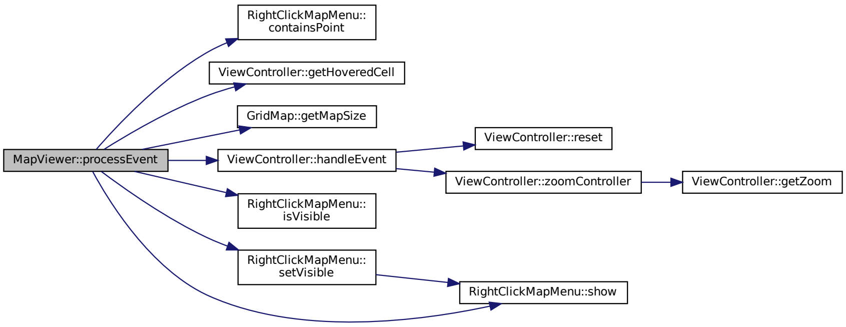
Definition at line 123 of file MapViewer.cpp.

```
{
    sf::Event event;
    while (window.pollEvent(event)) {
        gui.handleEvent(event);        /**< Forward event to GUI system */
        controller.handleEvent(event); /**< Forward event to view controller */

        if (event.type == sf::Event::Closed) {
            running = false;
            window.close();            /**< Close window and exit */
        }

        if (event.type == sf::Event::MouseButtonPressed) {
            const int gridSize = map.getMapSize();
            const sf::Vector2i cell = controller.getHoveredCell(gridSize);
            const sf::Vector2i pixelPos = sf::Mouse::getPosition(window);
            const sf::Vector2f pixelPosF(pixelPos);

            if (event.mouseButton.button == sf::Mouse::Right) {
                // Show context menu at mouse position for right-click
                r_menu.show(static_cast<float>(pixelPos.x),
                            static_cast<float>(pixelPos.y),
                            cell);
                r_menu.setVisible(true);
            } else if (event.mouseButton.button == sf::Mouse::Left) {
                // Hide menu if click is outside of the menu area
                if (r_menu.isVisible() && !r_menu.containsPoint(pixelPosF)) {
                    r_menu.setVisible(false);
                }
            }
        }
    }
}
```

References RightClickMapMenu::containsPoint(), controller, ViewController::getHoveredCell(), GridMap::getMapSize(), gui, ViewController::handleEvent(), RightClickMapMenu::isVisible(), map, r_menu, running, RightClickMapMenu::setVisible(), RightClickMapMenu::show(), and window.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:

main
MapViewer::processEvent

3.5.3.3 render()

```
void MapViewer::render ( )
```

Renders all visual components to the screen.

Renders the entire map viewer frame.

Clears the window, draws the grid and axes, live or stored maps, A∗ simulation, robot, GUI, and presents the final image. < Clear window with black background

< Apply current camera transform (zoom, pan)

< Draw grid lines

< Draw X and Y axes

< Draw live sensor data map

< Draw stored occupancy grid

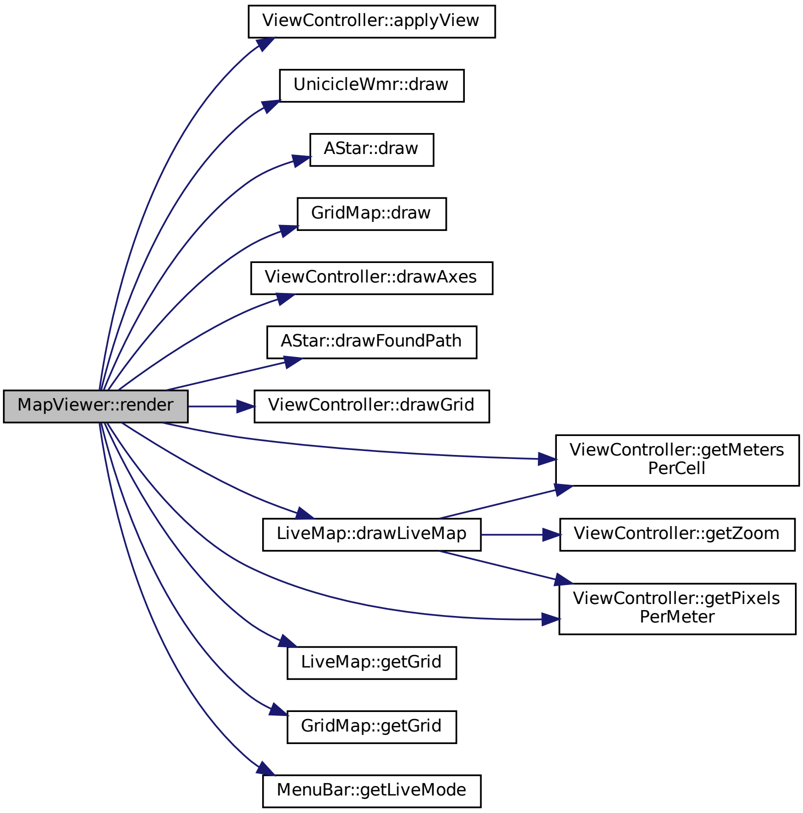
Definition at line 163 of file MapViewer.cpp.

```
{
    window.clear(sf::Color::Black);       /**< Clear window with black background */
    controller.applyView();                /**< Apply current camera transform (zoom, pan) */

    controller.drawGrid(window);           /**< Draw grid lines */
    controller.drawAxes(window);           /**< Draw X and Y axes */

    // Draw live map or stored grid map based on live mode status
    if (menu.getLiveMode()) {
        const auto& gridLive = livemap.getGrid();
        if (gridLive.empty() || gridLive[0].empty()) {
            std::cout « "Live grid is empty!" « std::endl;
            window.setView(window.getDefaultView());
            gui.draw();
            window.display();
            return;
        }
        livemap.drawLiveMap(window);       /**< Draw live sensor data map */
    } else {
        const auto& gridMap = map.getGrid();
        if (gridMap.empty() || gridMap[0].empty()) {
            std::cout « "Grid map is empty!" « std::endl;
            window.setView(window.getDefaultView());
            gui.draw();
            window.display();
            return;
        }
        map.draw(window, controller.getPixelsPerMeter()); /**< Draw stored occupancy grid */
    }

    // Draw A* algorithm visualization while animating
    if (astarAnimating) {
        aStarsim.draw(window, 50.0f, controller.getMetersPerCell());
    }

    // Draw the final path found by A* (if any)
    aStarsim.drawFoundPath(window, 50.0f, controller.getMetersPerCell());

    // Draw robot’s current pose and orientation
    wmr.draw(window);

    // Draw GUI elements on top (using default view)
    window.setView(window.getDefaultView());
    gui.draw();

    // Display everything on screen
    window.display();
}
```

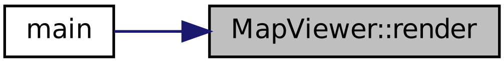
References ViewController::applyView(), astarAnimating, aStarsim, controller, UnicicleWmr::draw(), AStar::draw(), GridMap::draw(), ViewController::drawAxes(), AStar::drawFoundPath(), ViewController::drawGrid(), LiveMap::drawLiveMap(), LiveMap::getGrid(), GridMap::getGrid(), MenuBar::getLiveMode(), ViewController::getMetersPerCell(), ViewController::getPixelsPerMeter(), gui, livemap, map, menu, window, and wmr.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.3.4 update()

```
void MapViewer::update ( )
```

Manages logic updates and user interaction.

Updates application logic per frame.

Updates mouse position and grid cell info, live mode sensor data, A∗ animation, and robot model position. < Clear existing sonar points

< Clear live occupancy grid

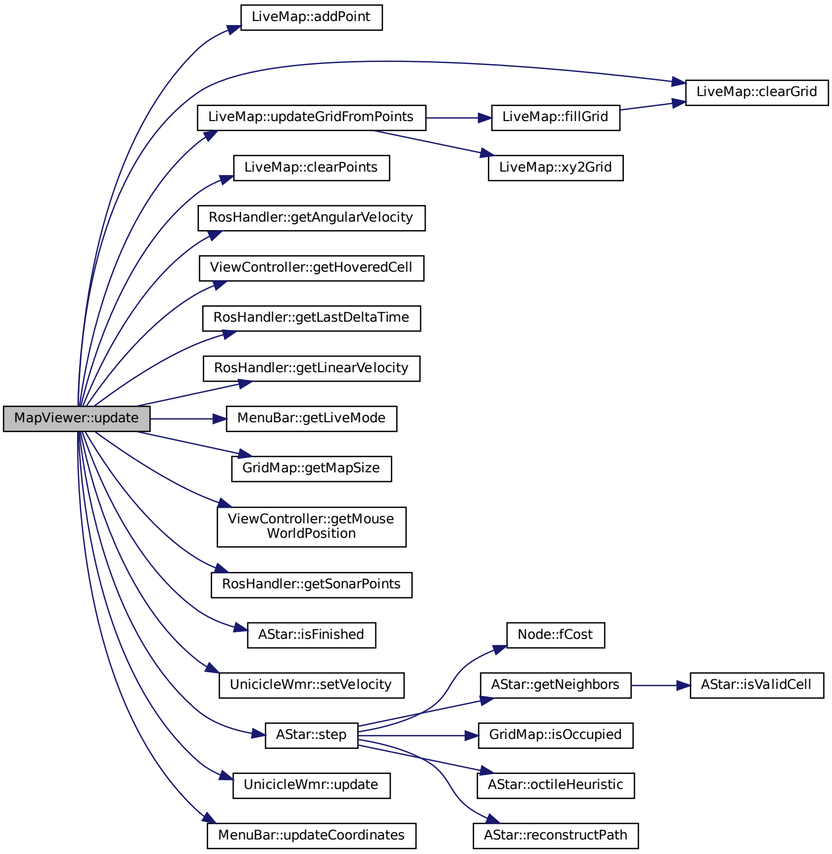
Definition at line 57 of file MapViewer.cpp.

```
{
    int gridSize = map.getMapSize();

    // Get the grid cell currently under the mouse cursor
    sf::Vector2i gridIndex = controller.getHoveredCell(gridSize);

    // Get mouse position in world coordinates (meters)
    sf::Vector2f worldPos = controller.getMouseWorldPosition();

    // Prepare coordinate status string with fixed precision
    std::ostringstream oss;
    oss « std::fixed « std::setprecision(2)
        « "Mouse: (" « worldPos.x « ", " « worldPos.y « ") m";

    if (gridIndex.x != -1 && gridIndex.y != -1) {
        oss « " | Grid: (" « gridIndex.x « ", " « gridIndex.y « ")";
    } else {
        oss « " | Out of bounds";
    }

    // Update coordinate display in status bar
    menu.updateCoordinates(oss.str());

    // -------- Live Mode: real-time sensor updates --------
    if (menu.getLiveMode()) {
        livemap.clearPoints();  /**< Clear existing sonar points */
        livemap.clearGrid();    /**< Clear live occupancy grid */

        // Update robot velocity from ROS data
        double v = roshandler.getLinearVelocity();
        double w = roshandler.getAngularVelocity();
        wmr.setVelocity(v, w);

        // Add sonar points to live map
        const auto& sonar = roshandler.getSonarPoints();
        for (const auto& p : sonar) {
            livemap.addPoint(p.x, p.y);
        }

        // Update live map grid with sonar points
        livemap.updateGridFromPoints();
    }

    // -------- A* Animation: step-by-step simulation --------
    if (astarAnimating) {
        if (!aStarsim.isFinished()) {
            bool continueAnim = aStarsim.step();
            if (!continueAnim) {
                astarAnimating = false;
            }
        } else {
            astarAnimating = false;
        }
    }

    // Update robot model position using ROS delta time
    wmr.update(roshandler.getLastDeltaTime());
}
```

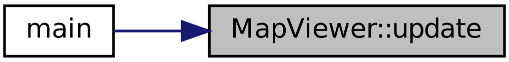
References LiveMap::addPoint(), astarAnimating, aStarsim, LiveMap::clearGrid(), LiveMap::clearPoints(), controller, RosHandler::getAngularVelocity(), ViewController::getHoveredCell(), RosHandler::getLastDeltaTime(), RosHandler::getLinearVelocity(), MenuBar::getLiveMode(), GridMap::getMapSize(), ViewController::getMouse←WorldPosition(), RosHandler::getSonarPoints(), gridIndex, AStar::isFinished(), livemap, map, menu, roshandler, UnicicleWmr::setVelocity(), AStar::step(), UnicicleWmr::update(), MenuBar::updateCoordinates(), LiveMap←::updateGridFromPoints(), and wmr.

Referenced by main().

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.4 Member Data Documentation

3.5.4.1 astarAnimating

```
bool MapViewer::astarAnimating = false  [private]
```

Indicates whether the A∗ animation is currently running.

Definition at line 43 of file MapViewer.hpp.

Referenced by MapViewer(), render(), and update().

3.5.4.2 aStarsim

```
AStar MapViewer::aStarsim  [private]
```

A∗ algorithm instance for path planning and animation.

Definition at line 36 of file MapViewer.hpp.

Referenced by MapViewer(), render(), and update().

3.5.4.3 controller

```
ViewController MapViewer::controller  [private]
```

Manages zoom, panning, grid drawing, and coordinate conversions.

Definition at line 31 of file MapViewer.hpp.

Referenced by MapViewer(), processEvent(), render(), and update().

3.5.4.4 gridIndex

```
sf::Vector2i MapViewer::gridIndex  [private]
```

Current hovered grid cell.

Definition at line 40 of file MapViewer.hpp.

Referenced by update().

3.5.4.5 gridIndex2copy

```
sf::Vector2i MapViewer::gridIndex2copy  [private]
```

Temporary copy of a selected grid index.

Definition at line 41 of file MapViewer.hpp.

3.5.4.6 gui

```
tgui::Gui MapViewer::gui  [private]
```

GUI system for handling menus and widgets.

Definition at line 26 of file MapViewer.hpp.

Referenced by processEvent(), and render().

3.5.4.7 livemap

```
LiveMap MapViewer::livemap  [private]
```

Handles live mapping mode based on ROS data.

Definition at line 34 of file MapViewer.hpp.

Referenced by render(), and update().

3.5.4.8 manager

```
FileManager MapViewer::manager  [private]
```

Manages file loading/saving.

Definition at line 29 of file MapViewer.hpp.

Referenced by MapViewer().

3.5.4.9 map

```
GridMap MapViewer::map  [private]
```

Represents and stores the occupancy grid.

Definition at line 32 of file MapViewer.hpp.

Referenced by MapViewer(), processEvent(), render(), and update().

3.5.4.10 menu

```
MenuBar MapViewer::menu  [private]
```

Instance of the custom menu bar.

Definition at line 28 of file MapViewer.hpp.

Referenced by MapViewer(), render(), and update().

3.5.4.11 r_menu

```
RightClickMapMenu MapViewer::r_menu  [private]
```

Context menu for selecting start and goal positions.

Definition at line 35 of file MapViewer.hpp.

Referenced by MapViewer(), and processEvent().

3.5.4.12 roshandler

```
RosHandler MapViewer::roshandler  [private]
```

Handles communication with ROS (publishing/subscribing)

Definition at line 30 of file MapViewer.hpp.

Referenced by update().

3.5.4.13 running

```
bool MapViewer::running = true  [private]
```

Indicates whether the main loop is running.

Definition at line 38 of file MapViewer.hpp.

Referenced by isRunning(), MapViewer(), and processEvent().

3.5.4.14 view

```
sf::View MapViewer::view  [private]
```

SFML camera view.

Definition at line 25 of file MapViewer.hpp.

3.5.4.15 window

```
sf::RenderWindow& MapViewer::window  [private]
```

Reference to the SFML window for rendering.

Definition at line 23 of file MapViewer.hpp.

Referenced by isRunning(), MapViewer(), processEvent(), and render().

3.5.4.16 wmr

```
UnicicleWmr MapViewer::wmr  [private]
```

Simulated unicycle WMR (Wheeled Mobile Robot)

Definition at line 33 of file MapViewer.hpp.

Referenced by MapViewer(), render(), and update().

3.5.4.17 worldXY

```
sf::Vector2f MapViewer::worldXY  [private]
```

World coordinates (floating point)

Definition at line 39 of file MapViewer.hpp.

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

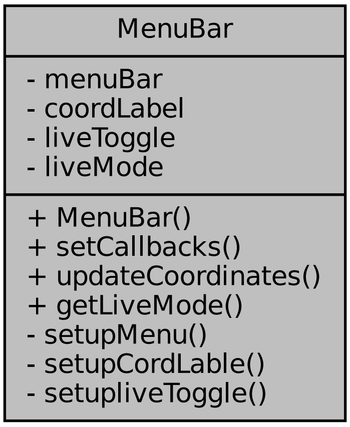
- include/artgslam_vsc/MapViewer.hpp
- src/MapViewer.cpp

3.6 MenuBar Class Reference

Manages the menu bar located at the bottom of the screen. Provides options for file handling, view controls, robot creation, and live mode toggle.

```
#include <MenuBar.hpp>
```

Collaboration diagram for MenuBar:



Public Member Functions

- MenuBar (tgui::Gui &gui)

  Constructor: initializes and places the menu elements within the given GUI.
- void setCallbacks (std::function< void()> onOpen, std::function< void()> onSave, std::function< void()> onSaveImage, std::function< void()> onClose, std::function< void()> onResetView, std::function< void()> onClearView, std::function< void()> onCreateRobot, std::function< void()> onSimulation)

  Sets the callback functions for each menu option:
- void updateCoordinates (const std::string &text)

  Updates the label showing the current grid coordinates of the mouse.
- bool getLiveMode () const

  Returns whether live mode is currently enabled.

Private Member Functions

- void setupMenu (tgui::Gui &gui)

  Helper to set up the main menu bar structure and entries.
- void setupCordLable (tgui::Gui &gui)

  Helper to set up the label that shows mouse grid coordinates.
- void setupliveToggle (tgui::Gui &gui)

  Helper to set up the toggle button for live mode (ROS streaming).

Private Attributes

- tgui::MenuBar::Ptr menuBar

  Pointer to the TGUI MenuBar widget.
- tgui::Label::Ptr coordLabel

  Label to show mouse position in grid coordinates.
- tgui::ToggleButton::Ptr liveToggle

  Toggle button to activate/deactivate live mode.
- bool liveMode = false

  Indicates whether live mode is active.

3.6.1 Detailed Description

Manages the menu bar located at the bottom of the screen. Provides options for file handling, view controls, robot creation, and live mode toggle.

Definition at line 14 of file MenuBar.hpp.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 MenuBar()

```
MenuBar::MenuBar (
            tgui::Gui & gui )
```

Constructor: initializes and places the menu elements within the given GUI.

Constructs the MenuBar and initializes all components.

Parameters

gui	Reference to the TGUI GUI object where menu elements will be placed.

Creates the main menu bar, the coordinate label, and the live mode toggle button.

Parameters

gui	Reference to the TGUI GUI instance where widgets will be added.

< Create the main menu bar

< Create coordinate display label

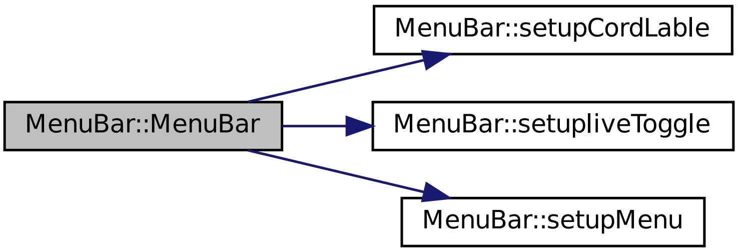
< Create toggle button for live mode

Definition at line 14 of file MenuBar.cpp.

```
                           {
    setupMenu(gui);        /**< Create the main menu bar */
    setupCordLable(gui);  /**< Create coordinate display label */
    setupliveToggle(gui);  /**< Create toggle button for live mode */
}
```

References setupCordLable(), setupliveToggle(), and setupMenu().

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 getLiveMode()

`bool MenuBar::getLiveMode ( ) const  [inline]`

Returns whether live mode is currently enabled.

Returns

true if live mode is active, false otherwise.

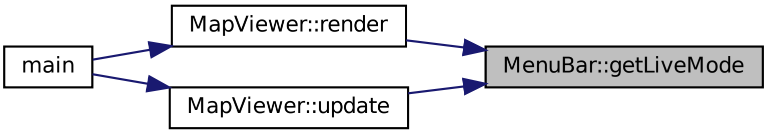
Definition at line 63 of file MenuBar.hpp.

```
63 { return liveMode; };
```

References liveMode.

Referenced by MapViewer::render(), and MapViewer::update().

Here is the caller graph for this function:



3.6.3.2 setCallbacks()

```
void MenuBar::setCallbacks (
            std::function< void()> onOpen,
            std::function< void()> onSave,
            std::function< void()> onSaveImage,
            std::function< void()> onClose,
            std::function< void()> onResetView,
            std::function< void()> onClearView,
            std::function< void()> onCreateRobot,
            std::function< void()> onSimulation )
```

Sets the callback functions for each menu option:

Connects external callback functions to the menu items.

- onOpen: Open a map file
- onSave: Save current map state
- onSaveImage: Export current map as an image
- onClose: Exit or close the application
- onResetView: Reset camera view
- onClearView: Clear the current grid/map
- onCreateRobot: Trigger robot creation interface
- onSimulation: Start or stop the simulation

Parameters

onOpen	Callback for "Open" menu option
onSave	Callback for "Save" menu option
onSaveImage	Callback for "Save Image" menu option
onClose	Callback for "Close" menu option
onResetView	Callback for "Reset View" menu option
onClearView	Callback for "Clear View" menu option
onCreateRobot	Callback for "Create Robot" menu option
onSimulation	Callback for "Simulation" menu option

Allows external code to respond to menu item selections.

Parameters

onOpen	Callback invoked when "Open" is selected.
onSave	Callback invoked when "Save" is selected.
onSaveImage	Callback invoked when "Save2Image" is selected.
onClose	Callback invoked when "Close" is selected.
onResetView	Callback invoked when "ResetView" is selected.
onClearView	Callback invoked when "ClearView" is selected.
onCreateRobot	Callback invoked when "WMR" is selected under "Create object".
onSimulation	Callback invoked when "A∗" is selected under "Simulation".

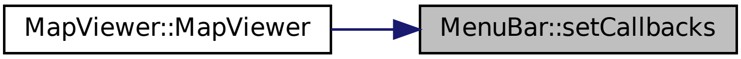
Definition at line 133 of file MenuBar.cpp.

```
{
    menuBar->connectMenuItem("File", "Open", [onOpen]() {
        if (onOpen) onOpen();
    });

    menuBar->connectMenuItem("File", "Save", [onSave]() {
        if (onSave) onSave();
    });

    menuBar->connectMenuItem("File", "Save2Image", [onSaveImage]() {
        if (onSaveImage) onSaveImage();
    });

    menuBar->connectMenuItem("File", "Close", [onClose]() {
        if (onClose) onClose();
    });

    menuBar->connectMenuItem("View", "ResetView", [onResetView]() {
        if (onResetView) onResetView();
    });

    menuBar->connectMenuItem("View", "ClearView", [onClearView]() {
        if (onClearView) onClearView();
    });

    menuBar->connectMenuItem("Create object", "WMR", [onCreateRobot]() {
        if (onCreateRobot) onCreateRobot();
    });

    menuBar->connectMenuItem("Simulation", "A*", [onSimulation]() {
        if (onSimulation) onSimulation();
    });
}
```

References menuBar.

Referenced by MapViewer::MapViewer().

Here is the caller graph for this function:



3.6.3.3 setupCordLable()

```
void MenuBar::setupCordLable (
             tgui::Gui & gui )  [private]
```

Helper to set up the label that shows mouse grid coordinates.

Sets up the coordinate label displayed at the bottom right of the window.

Parameters

gui	Reference to the TGUI GUI object.

The label initially shows placeholder text and has transparent background with black text.

Parameters

gui	Reference to the TGUI GUI instance.

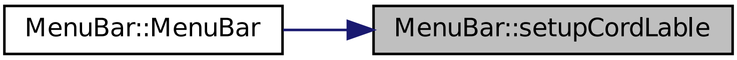
Definition at line 63 of file MenuBar.cpp.

```
                                      {
      coordLabel = tgui::Label::create("Grid: -, Real: -");
      coordLabel->setTextSize(14);
      coordLabel->setPosition("100% - 400", "100% - 20"); // Bottom right, offset leftwards
      coordLabel->getRenderer()->setBackgroundColor(tgui::Color::Transparent);
      coordLabel->getRenderer()->setTextColor(tgui::Color::Black);
      gui.add(coordLabel, "CoordLabel");
  }
```

References coordLabel.

Referenced by MenuBar().

Here is the caller graph for this function:



3.6.3.4 setupliveToggle()

```
void MenuBar::setupliveToggle (
            tgui::Gui & gui )  [private]
```

Helper to set up the toggle button for live mode (ROS streaming).

Creates and configures the live mode toggle button.

Parameters

gui	Reference to the TGUI GUI object.

The button switches between ON/OFF states with color feedback (green/red). Positioned near the coordinate label.

Parameters

gui	Reference to the TGUI GUI instance.

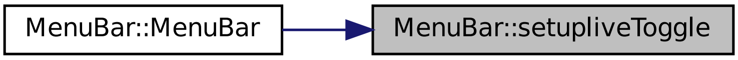
Definition at line 80 of file MenuBar.cpp.

```
                                          {
    liveToggle = tgui::ToggleButton::create();
    liveToggle->setSize(50, 20);
    liveToggle->setText("Live");
    liveToggle->setPosition("100% - 60", "100% - 20");

    auto renderer = liveToggle->getRenderer();
    renderer->setRoundedBorderRadius(10);
    renderer->setTextColor(tgui::Color::White);
    renderer->setBorderColor(tgui::Color::White);

    // Default OFF (red) colors
    renderer->setBackgroundColor(tgui::Color(120, 0, 0));
    renderer->setBackgroundColorHover(tgui::Color(180, 60, 60));
    renderer->setBackgroundColorDown(tgui::Color(255, 100, 100));

    gui.add(liveToggle);

    liveMode = false;

    // Change toggle button color based on state (green for ON, red for OFF)
    liveToggle->onToggle([this](bool state) {
        liveMode = state;
        auto renderer = liveToggle->getRenderer();

        if (state) {
            // ON state colors (green)
            renderer->setBackgroundColor(tgui::Color(0, 180, 0));
            renderer->setBackgroundColorHover(tgui::Color(80, 220, 80));
            renderer->setBackgroundColorDown(tgui::Color(100, 255, 100));
        } else {
            // OFF state colors (red)
            renderer->setBackgroundColor(tgui::Color(120, 0, 0));
            renderer->setBackgroundColorHover(tgui::Color(180, 60, 60));
            renderer->setBackgroundColorDown(tgui::Color(255, 100, 100));
        }
    });
}
```

References liveMode, and liveToggle.

Referenced by MenuBar().

Here is the caller graph for this function:



3.6.3.5 setupMenu()

```
void MenuBar::setupMenu (
            tgui::Gui & gui )  [private]
```

Helper to set up the main menu bar structure and entries.

Creates and configures the main menu bar with categorized menu items.

Parameters

gui	Reference to the TGUI GUI object.

The menu includes File, View, Create Object, and Simulation categories with their items. The menu bar is positioned at the bottom and menus open upward.

Parameters

gui	Reference to the TGUI GUI instance.

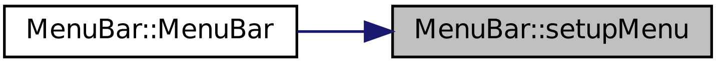
Definition at line 28 of file MenuBar.cpp.

```
                                      {
    menuBar = tgui::MenuBar::create();
    menuBar->setSize("100%", 20);
    menuBar->setPosition(0, "100% - 20");  // Bottom of the window
    menuBar->setInvertedMenuDirection(true); // Menus open upward
    gui.add(menuBar);

    // File menu and items
    menuBar->addMenu("File");
    menuBar->addMenuItem("File", "Open");
    menuBar->addMenuItem("File", "Save");
    menuBar->addMenuItem("File", "Save2Image");
    menuBar->addMenuItem("File", "Close");

    // View menu and items
    menuBar->addMenu("View");
    menuBar->addMenuItem("View", "ResetView");
    menuBar->addMenuItem("View", "ClearView");

    // Create object menu and items
    menuBar->addMenu("Create object");
    menuBar->addMenuItem("Create object", "WMR");

    // Simulation menu and items
    menuBar->addMenu("Simulation");
    menuBar->addMenuItem("Simulation", "A*");
}
```

References menuBar.

Referenced by MenuBar().

Here is the caller graph for this function:



3.6.3.6 updateCoordinates()

```
void MenuBar::updateCoordinates (
              const std::string & text )
```

Updates the label showing the current grid coordinates of the mouse.

Updates the coordinate label text.

Parameters

text	The text to display (usually formatted coordinates)
text	New coordinate string to display.

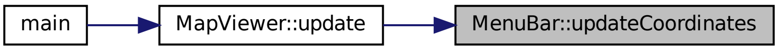
Definition at line 181 of file MenuBar.cpp.

```
181                                                             {
182     if (coordLabel)
183         coordLabel->setText(text);
184 }
```

References coordLabel.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.6.4 Member Data Documentation

3.6.4.1 coordLabel

```
tgui::Label::Ptr MenuBar::coordLabel  [private]
```

Label to show mouse position in grid coordinates.

Definition at line 85 of file MenuBar.hpp.

Referenced by setupCordLable(), and updateCoordinates().

3.6.4.2 liveMode

```
bool MenuBar::liveMode = false  [private]
```

Indicates whether live mode is active.

Definition at line 88 of file MenuBar.hpp.

Referenced by getLiveMode(), and setupliveToggle().

3.6.4.3 liveToggle

```
tgui::ToggleButton::Ptr MenuBar::liveToggle  [private]
```

Toggle button to activate/deactivate live mode.

Definition at line 86 of file MenuBar.hpp.

Referenced by setupliveToggle().

3.6.4.4 menuBar

```
tgui::MenuBar::Ptr MenuBar::menuBar  [private]
```

Pointer to the TGUI MenuBar widget.

Definition at line 84 of file MenuBar.hpp.

Referenced by setCallbacks(), and setupMenu().

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

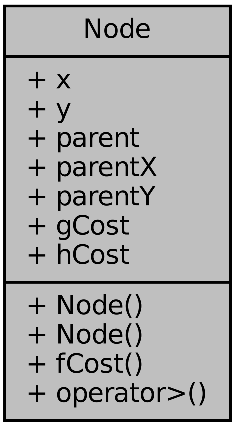
- include/artgslam_vsc/MenuBar.hpp
- src/MenuBar.cpp

3.7 Node Class Reference

Represents a node in a grid used for path planning algorithms (e.g., A∗). Each node corresponds to a cell and holds all relevant data for cost calculation and path reconstruction.

```
#include <Node.hpp>
```

Collaboration diagram for Node:



Public Member Functions

- Node ()

  Default constructor: initializes all coordinates and costs to invalid/default values.
- Node (int x, int y)

  Constructor with node position.
- float fCost () const

  Returns the total estimated cost (f-cost) of the node: f = g + h.
- bool operator> (const Node &other) const

  Comparison operator for use with priority queues (min-heap).

Public Attributes

- int x
- int y

  Current node (cell) position in grid coordinates.
- sf::Vector2i parent

  Parent node position (used for path reconstruction)
- int parentX
- int parentY

  Redundant storage of parent coordinates.
- float gCost

  Cost from the start node to this node.
- float hCost

  Heuristic cost estimate from this node to the goal.

3.7.1 Detailed Description

Represents a node in a grid used for path planning algorithms (e.g., A∗). Each node corresponds to a cell and holds all relevant data for cost calculation and path reconstruction.

Definition at line 9 of file Node.hpp.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Node() `[1/2]`

```
Node::Node ( )  [inline]
```

Default constructor: initializes all coordinates and costs to invalid/default values.

Definition at line 17 of file Node.hpp.

```
17 : x(-1), y(-1), parentX(-1), parentY(-1), gCost(0), hCost(0) {}
```

3.7.2.2 Node() `[2/2]`

```
Node::Node (
            int x,
            int y )  [inline]
```

Constructor with node position.

Parameters

x	Grid x-coordinate of the node.
y	Grid y-coordinate of the node.

Definition at line 24 of file Node.hpp.

```
        : x(x), y(y), parentX(-1), parentY(-1), gCost(0), hCost(0)
    {
        parent = sf::Vector2i(parentX, parentY);
    }
```

References parent, parentX, and parentY.

3.7.3 Member Function Documentation

3.7.3.1 fCost()

```
float Node::fCost ( ) const  [inline]
```

Returns the total estimated cost (f-cost) of the node: f = g + h.

This value is used in most path planning algorithms (e.g., A∗) to prioritize nodes.

Returns

Sum of gCost and hCost.

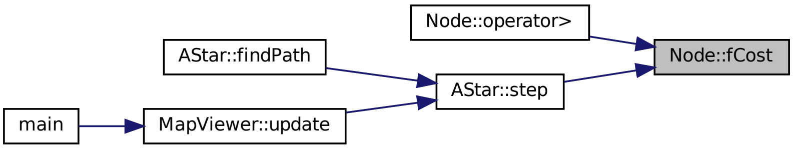
Definition at line 41 of file Node.hpp.

```
{ return gCost + hCost; }
```

References gCost, and hCost.

Referenced by operator>(), and AStar::step().

Here is the caller graph for this function:



3.7.3.2 operator>()

```
bool Node::operator> (
             const Node & other ) const  [inline]
```

Comparison operator for use with priority queues (min-heap).

Returns true if this node has a higher f-cost than another node. Nodes with lower f-costs are given higher priority.

Parameters

other	The other node to compare with.

Returns

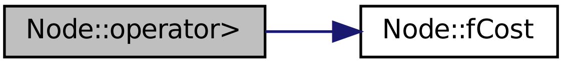
true if this node's fCost is greater than the other's.

Definition at line 50 of file Node.hpp.

```
50                                    {
51         return this->fCost() > other.fCost();
52     }
```

References fCost().

Here is the call graph for this function:



3.7.4 Member Data Documentation

3.7.4.1 gCost

```
float Node::gCost
```

Cost from the start node to this node.

Definition at line 33 of file Node.hpp.

Referenced by fCost(), AStar::start(), and AStar::step().

3.7.4.2 hCost

```
float Node::hCost
```

Heuristic cost estimate from this node to the goal.

Definition at line 34 of file Node.hpp.

Referenced by fCost(), AStar::start(), and AStar::step().

3.7.4.3 parent

```
sf::Vector2i Node::parent
```

Parent node position (used for path reconstruction)

Definition at line 31 of file Node.hpp.

Referenced by Node().

3.7.4.4 parentX

```
int Node::parentX
```

Definition at line 32 of file Node.hpp.

Referenced by Node().

3.7.4.5 parentY

```
int Node::parentY
```

Redundant storage of parent coordinates.

Definition at line 32 of file Node.hpp.

Referenced by Node().

3.7.4.6 x

```
int Node::x
```

Definition at line 30 of file Node.hpp.

Referenced by AStar::getNeighbors(), AStar::reconstructPath(), and AStar::step().

3.7.4.7 y

```
int Node::y
```

Current node (cell) position in grid coordinates.

Definition at line 30 of file Node.hpp.

Referenced by AStar::getNeighbors(), AStar::reconstructPath(), and AStar::step().

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

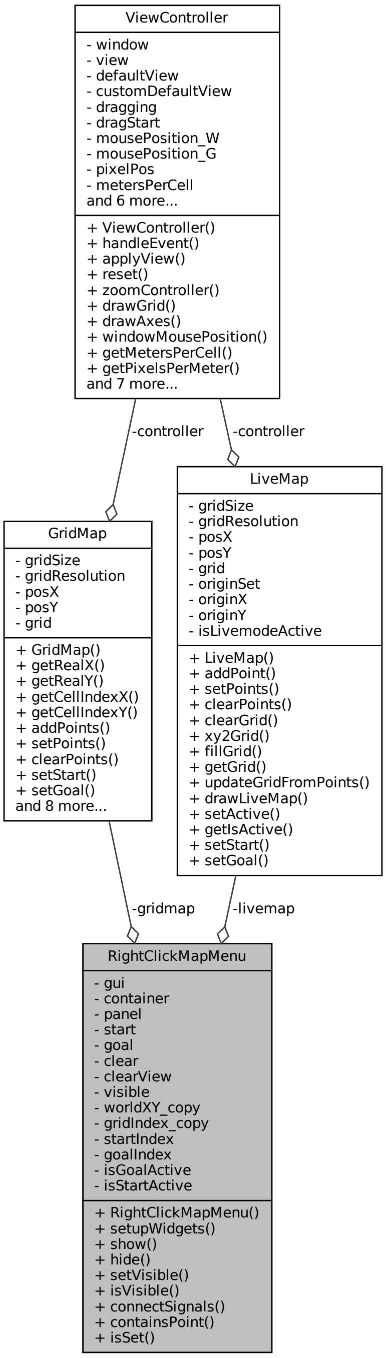
- include/artgslam_vsc/Node.hpp

3.8 RightClickMapMenu Class Reference

Manages a contextual right-click menu displayed on top of a map. Allows users to set the start and goal positions or clear the grid through TGUI buttons.

```
#include <RightClickMapMenu.hpp>
```

Collaboration diagram for RightClickMapMenu:



Public Member Functions

- RightClickMapMenu (tgui::Gui &guiRef, GridMap &mapRef, LiveMap &livemapRef)

  Constructor: initializes the menu with references to the TGUI GUI system and map instances.
- void setupWidgets ()

  Sets up the menu layout and widgets (start, goal, clear, etc.).
- void show (float x, float y, const sf::Vector2i &gridIndex)

  Displays the context menu at the given screen coordinates.
- void hide ()

  Hides the context menu.
- void setVisible (bool show)

  Sets the visibility state of the menu.
- bool isVisible () const

  Returns whether the menu is currently visible.
- void connectSignals ()

  Connects internal widget signals to their callbacks (e.g., button clicks).
- bool containsPoint (const sf::Vector2f &point) const

  Checks if a world-space point is inside the menu bounds.
- bool isSet () const

  Returns true if both the start and goal have been set by the user.

Private Attributes

- tgui::Gui & gui

  GUI reference (owned externally)
- GridMap & gridmap

  Reference to the static grid map.
- LiveMap & livemap

  Reference to the live map.
- tgui::Group::Ptr container

  Main container for the popup menu.
- tgui::ChildWindow::Ptr panel

  Panel that acts as the context menu window.
- tgui::Button::Ptr start

  Sets the start position.
- tgui::Button::Ptr goal

  Sets the goal position.
- tgui::Button::Ptr clear

  Clears the map.
- tgui::Button::Ptr clearView

  Clears overlays (visited cells, path, etc.)
- bool visible = false

  Visibility flag.
- sf::Vector2f worldXY_copy

  Click position in world coordinates.
- sf::Vector2i gridIndex_copy

  Corresponding cell index.
- sf::Vector2i startIndex

  Current start cell index.
- sf::Vector2i goalIndex

Current goal cell index.

- bool isGoalActive = false

  Flag: goal has been set.

- bool isStartActive = false

  Flag: start has been set.

3.8.1 Detailed Description

Manages a contextual right-click menu displayed on top of a map. Allows users to set the start and goal positions or clear the grid through TGUI buttons.

Definition at line 17 of file RightClickMapMenu.hpp.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 RightClickMapMenu()

```
RightClickMapMenu::RightClickMapMenu (
            tgui::Gui & guiRef,
            GridMap & mapRef,
            LiveMap & livemapRef )
```

Constructor: initializes the menu with references to the TGUI GUI system and map instances.

Constructs the RightClickMapMenu.

Parameters

guiRef	TGUI GUI object (must persist).
mapRef	Reference to the static grid map.
livemapRef	Reference to the live map (dynamic overlays, e.g., visited paths).

Initializes references to GUI, GridMap, and LiveMap, then sets up the menu widgets and connects button signals.

Parameters

guiRef	Reference to the TGUI GUI instance.
mapRef	Reference to the GridMap instance.
livemapRef	Reference to the LiveMap instance.

< Load and configure GUI widgets

< Connect button event handlers

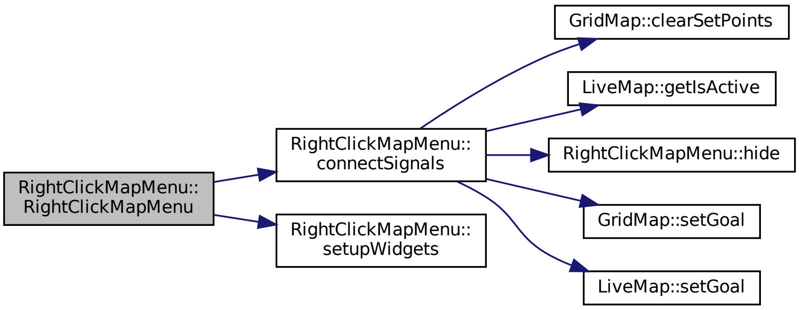
Definition at line 15 of file RightClickMapMenu.cpp.

```
    : gui(guiRef), gridmap(mapRef), livemap(livemapRef)
```

```
{
    std::cout « " RightClickMapMenu: loading..." « std::endl;
    setupWidgets();    /**< Load and configure GUI widgets */
    connectSignals();  /**< Connect button event handlers */
}
```

References connectSignals(), and setupWidgets().

Here is the call graph for this function:



3.8.3 Member Function Documentation

3.8.3.1 connectSignals()

```
void RightClickMapMenu::connectSignals ( )
```

Connects internal widget signals to their callbacks (e.g., button clicks).

Connects button press signals to their respective logic functions.

Handles setting start/goal points, clearing points, and other button actions.

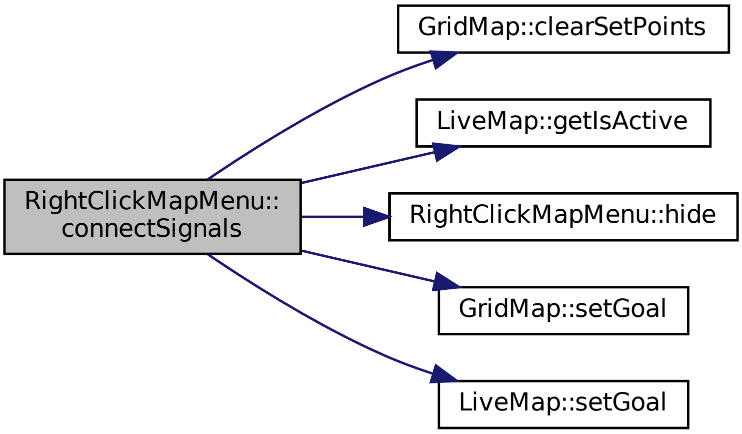
Definition at line 159 of file RightClickMapMenu.cpp.

```
{
    if (start) {
        start->onPress([this]() {
            if (!isStartActive) {
                if (livemap.getIsActive()) {
                    livemap.setStart(gridIndex_copy.x, gridIndex_copy.y);
                    std::cout « " Start set on LiveMap." « std::endl;
                } else {
                    gridmap.setStart(gridIndex_copy.x, gridIndex_copy.y);
                    std::cout « " Start set on GridMap." « std::endl;
                }
                startIndex = gridIndex_copy;
                isStartActive = true;
            } else {
                std::cout « " Start is already active. Clear it first." « std::endl;
            }
            hide();
        });
    }

    if (goal) {
        goal->onPress([this]() {
            if (!isGoalActive) {
                if (livemap.getIsActive()) {
                    livemap.setGoal(gridIndex_copy.x, gridIndex_copy.y);
                    std::cout « " Goal set on LiveMap." « std::endl;
                } else {
                    gridmap.setGoal(gridIndex_copy.x, gridIndex_copy.y);
                    std::cout « " Goal set on GridMap." « std::endl;
                }
                goalIndex = gridIndex_copy;
                isGoalActive = true;
            } else {
                std::cout « " Goal is already active. Clear it first." « std::endl;
            }
            hide();
        });
    }

    if (clear) {
        clear->onPress([this]() {
            if (isStartActive) {
                gridmap.clearSetPoints(startIndex);
                isStartActive = false;
            }
            if (isGoalActive) {
                gridmap.clearSetPoints(goalIndex);
                isGoalActive = false;
            }
            std::cout « " Start and Goal cleared." « std::endl;
            hide();
        });
    }

    if (clearView) {
        clearView->onPress([this]() {
            std::cout « " Clear View button pressed (no action assigned)." « std::endl;
            hide();
        });
    }
}
```

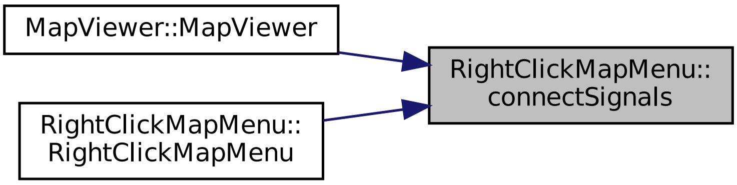
References clear, GridMap::clearSetPoints(), clearView, LiveMap::getIsActive(), goal, goalIndex, gridIndex_copy, gridmap, hide(), isGoalActive, isStartActive, livemap, GridMap::setGoal(), LiveMap::setGoal(), start, and startIndex.

Referenced by MapViewer::MapViewer(), and RightClickMapMenu().

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.3.2 containsPoint()

```
bool RightClickMapMenu::containsPoint (
             const sf::Vector2f & point ) const
```

Checks if a world-space point is inside the menu bounds.

Determines if a given point lies inside the menu boundaries.

Useful to avoid menu clicks being interpreted as map clicks.

Parameters

point	World coordinate point.

Returns

True if point is inside the menu, false otherwise.

Parameters

point	Point in screen coordinates to test.

Returns

true if point is inside menu bounds, false otherwise.

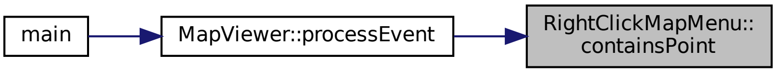
Definition at line 143 of file RightClickMapMenu.cpp.

```
{
    if (!panel) return false;

    sf::Vector2f pos = panel->getAbsolutePosition();
    sf::Vector2f size = panel->getSize();
    sf::FloatRect bounds(pos.x, pos.y, size.x, size.y);

    return bounds.contains(point);
152 }
```

References panel.

Referenced by MapViewer::processEvent().

Here is the caller graph for this function:



3.8.3.3 hide()

```
void RightClickMapMenu::hide ( )
```

Hides the context menu.

Hides the right-click menu.

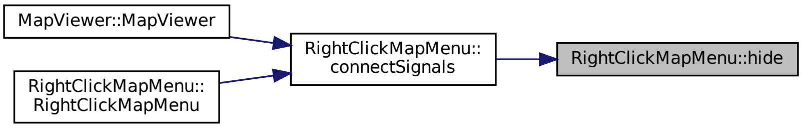
Definition at line 106 of file RightClickMapMenu.cpp.

```
107 {
108     if (panel) {
109         panel->setVisible(false);
110         visible = false;
111     }
112 }
```

References panel, and visible.

Referenced by connectSignals().

Here is the caller graph for this function:



3.8.3.4 isSet()

```
bool RightClickMapMenu::isSet ( ) const  [inline]
```

Returns true if both the start and goal have been set by the user.

Used to determine when path planning can begin.

Returns

True if start and goal are active.

Definition at line 75 of file RightClickMapMenu.hpp.

```
75 { return isGoalActive && isStartActive; }
```

References isGoalActive, and isStartActive.

3.8.3.5 isVisible()

```
bool RightClickMapMenu::isVisible ( ) const
```

Returns whether the menu is currently visible.

Returns

True if visible, false otherwise.

true if visible, false otherwise.

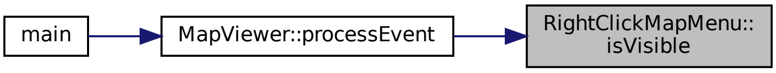
Definition at line 132 of file RightClickMapMenu.cpp.

```
133 {
134     return visible;
135 }
```

References visible.

Referenced by MapViewer::processEvent().

Here is the caller graph for this function:



3.8.3.6 setupWidgets()

```
void RightClickMapMenu::setupWidgets ( )
```

Sets up the menu layout and widgets (start, goal, clear, etc.).

Loads GUI widgets from external TGUI form and applies styles.

Retrieves the ROS package path to locate the GUI form file, loads widgets into a container, then retrieves and styles the individual buttons. Hides the panel by default. < Hide panel initially

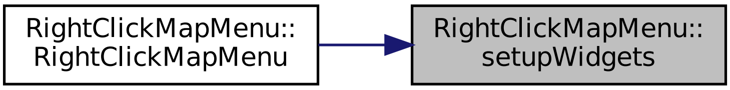
Definition at line 29 of file RightClickMapMenu.cpp.

```
{
    std::string package_path = ros::package::getPath("artgslam_vsc");
    std::string formPath = package_path + "/assets/forms/Right_Click_Menu.txt";

    try {
        container = tgui::Group::create();
        container->loadWidgetsFromFile(formPath);
        gui.add(container);

        panel = container->get<tgui::ChildWindow>("ChildWindow1");
        if (!panel) {
            std::cerr « " ’ChildWindow1’ widget not found in form." « std::endl;
            return;
        }

        auto layout = panel->get<tgui::VerticalLayout>("VerticalLayout1");
        if (!layout) {
            std::cerr « " ’VerticalLayout1’ widget not found in panel." « std::endl;
            return;
        }

        start     = layout->get<tgui::Button>("start");
        goal      = layout->get<tgui::Button>("goal");
        clear     = layout->get<tgui::Button>("Clear");
        clearView = layout->get<tgui::Button>("ClearView");

        if (!start || !goal || !clear || !clearView) {
            std::cerr « " One or more buttons not found in layout." « std::endl;
        } else {
            auto applyHoverStyle = [](tgui::Button::Ptr btn) {
                auto renderer = btn->getRenderer();
                renderer->setBackgroundColor(sf::Color::Transparent);
                renderer->setBackgroundColorHover(sf::Color(0, 120, 215)); // Windows 10 blue
                renderer->setTextColor(sf::Color::Black);
                renderer->setTextColorHover(sf::Color::White);
                renderer->setBorderColor(sf::Color::Transparent);
                renderer->setBorderColorHover(sf::Color(0, 120, 215));
                renderer->setBorders({1, 1, 1, 1}); // 1-pixel border
            };

            applyHoverStyle(start);
            applyHoverStyle(goal);
            applyHoverStyle(clear);
            applyHoverStyle(clearView);
        }

        panel->setVisible(false); /**< Hide panel initially */

    } catch (const tgui::Exception& e) {
        std::cerr « " Failed to load TGUI form: " « e.what() « std::endl;
    }
}
```

References clear, clearView, container, goal, gui, panel, and start.

Referenced by RightClickMapMenu().

Here is the caller graph for this function:



3.8.3.7 setVisible()

```
void RightClickMapMenu::setVisible (
             bool show )
```

Sets the visibility state of the menu.

Explicitly sets the visibility of the menu.

Parameters

show	True to show the menu, false to hide.

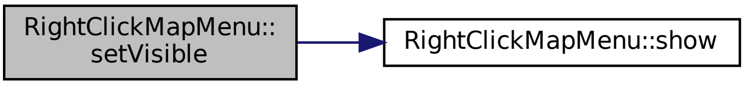
Definition at line 119 of file RightClickMapMenu.cpp.

```
{
    if (panel) {
        panel->setVisible(show);
        visible = show;
    }
}
```

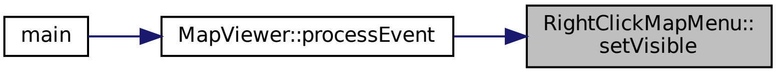
References panel, show(), and visible.

Referenced by MapViewer::processEvent().

Here is the call graph for this function:



Here is the caller graph for this function:



3.8.3.8 show()

```
void RightClickMapMenu::show (
             float x,
             float y,
             const sf::Vector2i & gridIndex )
```

Displays the context menu at the given screen coordinates.

Shows the right-click menu at specified screen coordinates with a selected grid index.

Parameters

x	X coordinate in world space.
y	Y coordinate in world space.
gridIndex	Cell index corresponding to the click.
x	Screen x position.
y	Screen y position.
gridIndex	Grid cell index selected.

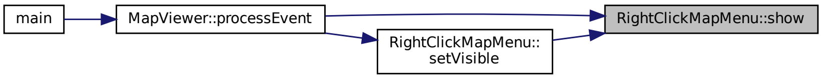
Definition at line 90 of file RightClickMapMenu.cpp.

```
{
    if (!panel) return;

    gridIndex_copy = gridIndex;

    std::cout « " Selected index: (" « gridIndex.x « ", " « gridIndex.y « ")" « std::endl;

    panel->setPosition(x, y);
    panel->setVisible(true);
     visible = true;
}
```

References gridIndex_copy, panel, and visible.

Referenced by MapViewer::processEvent(), and setVisible().

Here is the caller graph for this function:



3.8.4 Member Data Documentation

3.8.4.1 clear

```
tgui::Button::Ptr RightClickMapMenu::clear  [private]
```

Clears the map.

Definition at line 87 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and setupWidgets().

3.8.4.2 clearView

```
tgui::Button::Ptr RightClickMapMenu::clearView  [private]
```

Clears overlays (visited cells, path, etc.)

Definition at line 88 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and setupWidgets().

3.8.4.3 container

```
tgui::Group::Ptr RightClickMapMenu::container  [private]
```

Main container for the popup menu.

Definition at line 82 of file RightClickMapMenu.hpp.

Referenced by setupWidgets().

3.8.4.4 goal

```
tgui::Button::Ptr RightClickMapMenu::goal  [private]
```

Sets the goal position.

Definition at line 86 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and setupWidgets().

3.8.4.5 goalIndex

```
sf::Vector2i RightClickMapMenu::goalIndex  [private]
```

Current goal cell index.

Definition at line 96 of file RightClickMapMenu.hpp.

Referenced by connectSignals().

3.8.4.6 gridIndex_copy

```
sf::Vector2i RightClickMapMenu::gridIndex_copy  [private]
```

Corresponding cell index.

Definition at line 93 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and show().

3.8.4.7 gridmap

```
GridMap& RightClickMapMenu::gridmap  [private]
```

Reference to the static grid map.

Definition at line 79 of file RightClickMapMenu.hpp.

Referenced by connectSignals().

3.8.4.8 gui

```
tgui::Gui& RightClickMapMenu::gui  [private]
```

GUI reference (owned externally)

Definition at line 78 of file RightClickMapMenu.hpp.

Referenced by setupWidgets().

3.8.4.9 isGoalActive

```
bool RightClickMapMenu::isGoalActive = false  [private]
```

Flag: goal has been set.

Definition at line 98 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and isSet().

3.8.4.10 isStartActive

```
bool RightClickMapMenu::isStartActive = false  [private]
```

Flag: start has been set.

Definition at line 99 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and isSet().

3.8.4.11 livemap

```
LiveMap& RightClickMapMenu::livemap  [private]
```

Reference to the live map.

Definition at line 80 of file RightClickMapMenu.hpp.

Referenced by connectSignals().

3.8.4.12 panel

```
tgui::ChildWindow::Ptr RightClickMapMenu::panel  [private]
```

Panel that acts as the context menu window.

Definition at line 83 of file RightClickMapMenu.hpp.

Referenced by containsPoint(), hide(), setupWidgets(), setVisible(), and show().

3.8.4.13 start

```
tgui::Button::Ptr RightClickMapMenu::start  [private]
```

Sets the start position.

Definition at line 85 of file RightClickMapMenu.hpp.

Referenced by connectSignals(), and setupWidgets().

3.8.4.14 startIndex

```
sf::Vector2i RightClickMapMenu::startIndex  [private]
```

Current start cell index.

Definition at line 95 of file RightClickMapMenu.hpp.

Referenced by connectSignals().

3.8.4.15 visible

```
bool RightClickMapMenu::visible = false  [private]
```

Visibility flag.

Definition at line 90 of file RightClickMapMenu.hpp.

Referenced by hide(), isVisible(), setVisible(), and show().

3.8.4.16 worldXY_copy

```
sf::Vector2f RightClickMapMenu::worldXY_copy  [private]
```

Click position in world coordinates.

Definition at line 92 of file RightClickMapMenu.hpp.

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

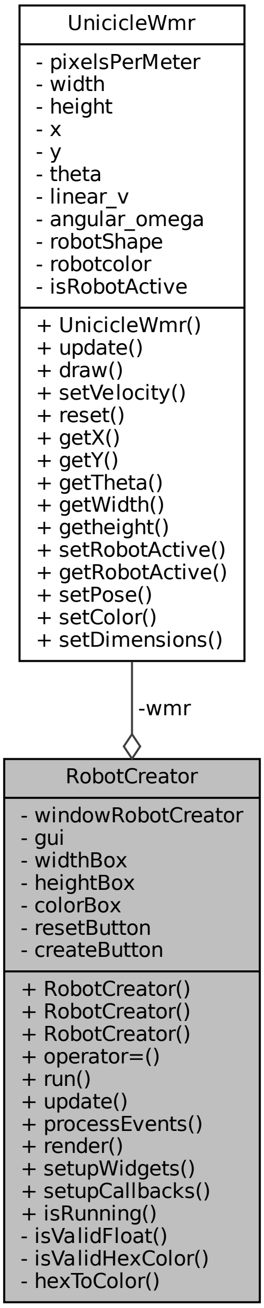
- include/artgslam_vsc/RightClickMapMenu.hpp
- src/RightClickMapMenu.cpp

3.9 RobotCreator Class Reference

Manages a GUI window for creating and configuring a Unicycle Wheeled Mobile Robot (WMR). Allows user input for robot parameters like width, height, and color, then applies these to the robot model.

```
#include <RobotCreator.hpp>
```

Collaboration diagram for RobotCreator:



Public Member Functions

- RobotCreator (UnicicleWmr &wmrRef)

  Explicit constructor that requires a reference to the UnicicleWmr instance to configure.
- RobotCreator ()=delete

  Deleted default constructor to avoid creating an instance without a robot reference.
- RobotCreator (const RobotCreator &)=delete

  Deleted copy constructor to prevent copying.
- RobotCreator & operator= (const RobotCreator &)=delete

  Deleted copy assignment operator to prevent copying.
- void run ()

  Main loop entry point to run the creation window.
- void update ()

  Updates internal state and GUI elements each frame.
- void processEvents ()

  Handles user input events like keyboard and mouse.
- void render ()

  Renders the GUI window and widgets.
- void setupWidgets ()

  Initializes and sets up all GUI widgets (EditBoxes, Buttons)
- void setupCallbacks ()

  Connects callbacks for button presses and other GUI events.
- bool isRunning () const

  Returns true if the creation window is currently open and running.

Private Member Functions

- bool isValidFloat (const std::string &str)

  Helper function to check if a string represents a valid floating-point number.
- bool isValidHexColor (const std::string &str)

  Helper function to validate if a string is a valid hexadecimal color code.
- sf::Color hexToColor (const std::string &hex)

  Converts a hexadecimal color string to an SFML Color object.

Private Attributes

- sf::RenderWindow windowRobotCreator

  SFML window for robot creation GUI.
- tgui::Gui gui

  TGUI GUI manager attached to the window.
- UnicicleWmr & wmr

  Reference to the robot being configured.
- tgui::EditBox::Ptr widthBox

  GUI widgets for robot parameters input.
- tgui::EditBox::Ptr heightBox

  Input for robot height.
- tgui::EditBox::Ptr colorBox

  Input for robot color (hexadecimal string)
- tgui::Button::Ptr resetButton

  Buttons for user actions.
- tgui::Button::Ptr createButton

  Applies parameters and creates/configures the robot.

3.9.1 Detailed Description

Manages a GUI window for creating and configuring a Unicycle Wheeled Mobile Robot (WMR). Allows user input for robot parameters like width, height, and color, then applies these to the robot model.

Definition at line 19 of file RobotCreator.hpp.

3.9.2 Constructor & Destructor Documentation

3.9.2.1 RobotCreator() [1/3]

```
RobotCreator::RobotCreator (
              UnicicleWmr & wmrRef )  [explicit]
```

Explicit constructor that requires a reference to the UnicicleWmr instance to configure.

Constructor for RobotCreator.

The explicit keyword prevents unintended implicit conversions.

Parameters

wmrRef	Reference to the wheeled mobile robot instance

Loads the GUI form, initializes the window and widgets, and sets up callbacks.

Parameters

wmrRef	Reference to the UnicicleWmr robot model to configure.

< Link widget variables to GUI elements

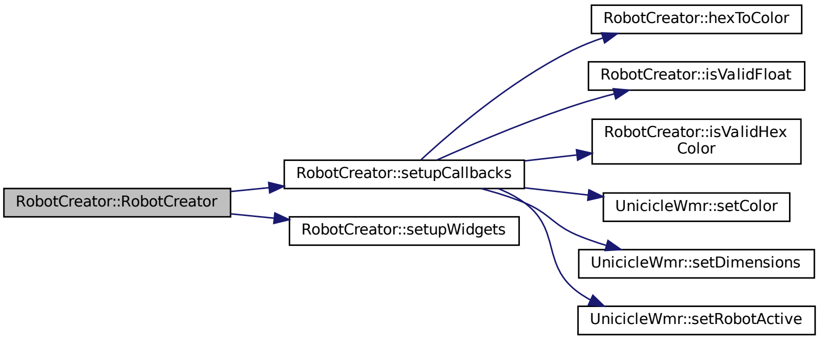
< Setup button callbacks

Definition at line 13 of file RobotCreator.cpp.

```
    : windowRobotCreator(sf::VideoMode(400, 250), "Robot Creator"),
      gui(windowRobotCreator), wmr(wmrRef)
{
    // Load form file path from ROS package
    std::string package_path = ros::package::getPath("artgslam_vsc");
    std::string formPath = package_path + "/assets/forms/createRobot.txt";

    try {
        gui.loadWidgetsFromFile(formPath);
    } catch (const tgui::Exception& e) {
        std::cerr « "Error loading GUI form: " « e.what() « std::endl;
    }

    setupWidgets();   /**< Link widget variables to GUI elements */
    setupCallbacks(); /**< Setup button callbacks */
}
```

References gui, setupCallbacks(), and setupWidgets().

Here is the call graph for this function:


RobotCreator::RobotCreator
RobotCreator::setupCallbacks
RobotCreator::setupWidgets
RobotCreator::hexToColor
RobotCreator::isValidFloat
RobotCreator::isValidHex Color
UnicicleWmr::setColor
UnicicleWmr::setDimensions
UnicicleWmr::setRobotActive

3.9.2.2 RobotCreator() [2/3]

```
RobotCreator::RobotCreator ( )  [delete]
```

Deleted default constructor to avoid creating an instance without a robot reference.

3.9.2.3 RobotCreator() [3/3]

```
RobotCreator::RobotCreator (
             const RobotCreator &  )  [delete]
```

Deleted copy constructor to prevent copying.

3.9.3 Member Function Documentation

3.9.3.1 hexToColor()

```
sf::Color RobotCreator::hexToColor (
             const std::string & hex )  [private]
```

Converts a hexadecimal color string to an SFML Color object.

Converts a hex color string to an SFML color object.

Assumes valid input format.

Parameters

hex	Hexadecimal color string

Returns

Corresponding sf::Color

Supports both 3-digit (#abc) and 6-digit (#aabbcc) hex formats.

Parameters

hex	The hex color string.

Returns

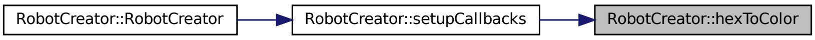
Corresponding sf::Color object.

Definition at line 209 of file RobotCreator.cpp.

```
{
    std::string h = hex;
    if (h.empty()) return sf::Color::White;
    if (h[0] == '#') h = h.substr(1);

    if (h.length() == 3)
    {
        h = {h[0], h[0], h[1], h[1], h[2], h[2]};
    }

    unsigned int r = std::stoul(h.substr(0, 2), nullptr, 16);
    unsigned int g = std::stoul(h.substr(2, 2), nullptr, 16);
    unsigned int b = std::stoul(h.substr(4, 2), nullptr, 16);

    return sf::Color(r, g, b);
}
```

Referenced by setupCallbacks().

Here is the caller graph for this function:



3.9.3.2 isRunning()

```
bool RobotCreator::isRunning ( ) const
```

Returns true if the creation window is currently open and running.

Checks if the window is still open.

Returns

true if the window is open, false otherwise.

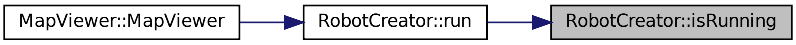
Definition at line 170 of file RobotCreator.cpp.

```
{
    return windowRobotCreator.isOpen();
}
```

References windowRobotCreator.

Referenced by run().

Here is the caller graph for this function:



3.9.3.3 isValidFloat()

```
bool RobotCreator::isValidFloat (
             const std::string & str )  [private]
```

Helper function to check if a string represents a valid floating-point number.

Validates if a string represents a valid floating point number.

Parameters

str	Input string

Returns

true if valid float, false otherwise

Parameters

str	The string to validate.

Returns

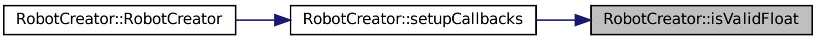
true if valid float format, false otherwise.

Definition at line 181 of file RobotCreator.cpp.

```
{
    if (str.empty()) return false;
    static const std::regex floatRegex(R"(^-?\d+(\.\d+)?$)");
    return std::regex_match(str, floatRegex);
}
```

Referenced by setupCallbacks().

Here is the caller graph for this function:


RobotCreator::RobotCreator
RobotCreator::setupCallbacks
RobotCreator::isValidFloat

3.9.3.4 isValidHexColor()

```
bool RobotCreator::isValidHexColor (
             const std::string & str )  [private]
```

Helper function to validate if a string is a valid hexadecimal color code.

Validates if a string is a valid hex color (#RGB or #RRGGBB).

Supports formats like "#RRGGBB" or "RRGGBB".

Parameters

str	Input string

Returns

true if valid hex color, false otherwise

Parameters

str	The hex color string.

Returns

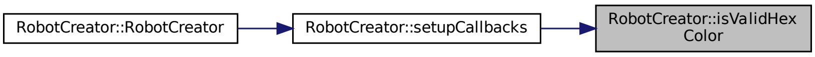
true if valid hex color format, false otherwise.

Definition at line 194 of file RobotCreator.cpp.

```
{
    static const std::regex hexShort(R"(#([A-Fa-f0-9]{3}))");
    static const std::regex hexLong(R"(#([A-Fa-f0-9]{6}))");
    return std::regex_match(str, hexShort) || std::regex_match(str, hexLong);
}
```

Referenced by setupCallbacks().

Here is the caller graph for this function:


RobotCreator::RobotCreator
RobotCreator::setupCallbacks
RobotCreator::isValidHex Color

3.9.3.5 operator=()

```
RobotCreator& RobotCreator::operator= (
              const RobotCreator & )  [delete]
```

Deleted copy assignment operator to prevent copying.

3.9.3.6 processEvents()

```
void RobotCreator::processEvents ( )
```

Handles user input events like keyboard and mouse.

Processes all window events.

Polls and handles SFML window events and delegates event handling to TGUI.

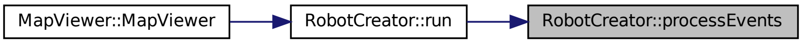
Definition at line 51 of file RobotCreator.cpp.

```
{
    sf::Event event;
    while (windowRobotCreator.pollEvent(event))
    {
        gui.handleEvent(event);

        if (event.type == sf::Event::Closed)
            windowRobotCreator.close();
    }
}
```

References gui, and windowRobotCreator.

Referenced by run().

Here is the caller graph for this function:



3.9.3.7 render()

```
void RobotCreator::render ( )
```

Renders the GUI window and widgets.

Clears the window and draws the GUI.

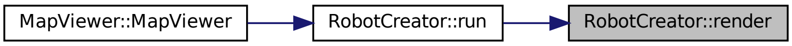
Definition at line 74 of file RobotCreator.cpp.

```
{
    windowRobotCreator.clear(sf::Color::White);
    gui.draw();
    windowRobotCreator.display();
}
```

References gui, and windowRobotCreator.

Referenced by run().

Here is the caller graph for this function:



3.9.3.8 run()

```
void RobotCreator::run ( )
```

Main loop entry point to run the creation window.

Runs the main loop of the Robot Creator window.

Processes events, updates state, and renders GUI until the window is closed. < Handle input and window events

< Update logic (currently unused)

< Render the GUI

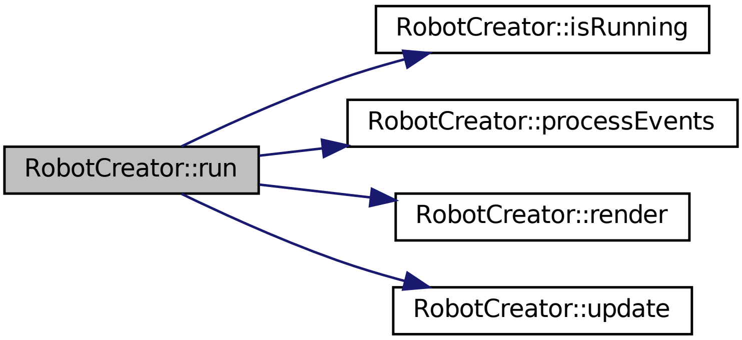
Definition at line 36 of file RobotCreator.cpp.

```
{
    while (isRunning())
    {
        processEvents(); /**< Handle input and window events */
        update();        /**< Update logic (currently unused) */
        render();        /**< Render the GUI */
    }
}
```

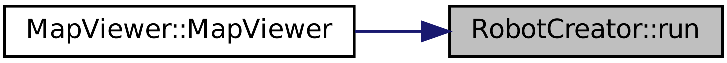
References isRunning(), processEvents(), render(), and update().

Referenced by MapViewer::MapViewer().

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.3.9 setupCallbacks()

```
void RobotCreator::setupCallbacks ( )
```

Connects callbacks for button presses and other GUI events.

Sets up callbacks for reset and create buttons.

- Resets all input fields on reset button press.
- Validates input and creates robot on create button press.

Definition at line 110 of file RobotCreator.cpp.

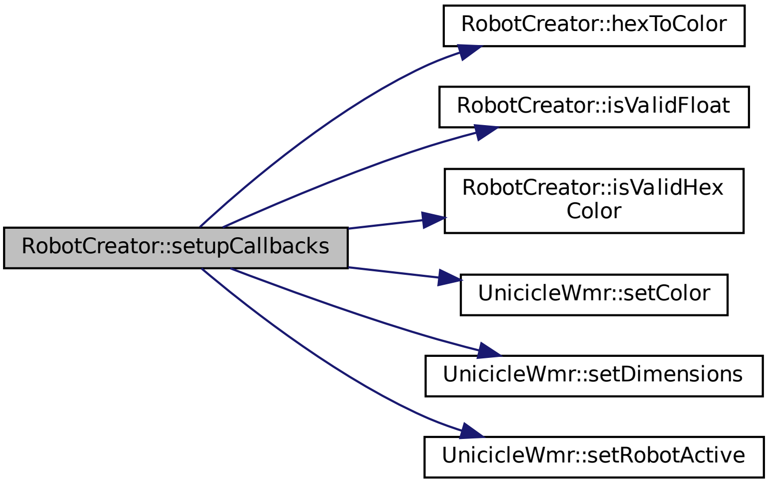
```
111 {
112     if (!resetButton || !createButton) return;
113 
114     resetButton->onPress([this]() {
115         widthBox->setText("");
116         heightBox->setText("");
117         colorBox->setText("");
118     });
119 
```

```
    createButton->onPress([this]() {
        std::string widthStr  = widthBox->getText().toStdString();
        std::string heightStr = heightBox->getText().toStdString();
        std::string colorStr  = colorBox->getText().toStdString();

        bool valid = true;

        if (!isValidFloat(widthStr)) {
            widthBox->setText("");
            valid = false;
        }

        if (!isValidFloat(heightStr)) {
            heightBox->setText("");
            valid = false;
        }

        if (!isValidHexColor(colorStr)) {
            colorBox->setText("");
            valid = false;
        }

        if (!valid) {
            std::cout « "Error: Invalid input. Please check the fields.\n";
            return;
        }

        float width  = std::stof(widthStr);
        float height = std::stof(heightStr);
        sf::Color color = hexToColor(colorStr);

        wmr.setDimensions(width, height);
        wmr.setColor(color);
        wmr.setRobotActive(true);

        std::cout « "Robot created:\n";
        std::cout « "Width: " « width « ", Height: " « height « "\n";
        std::cout « "Color RGB: ("
                  « static_cast<int>(color.r) « ", "
                  « static_cast<int>(color.g) « ", "
                  « static_cast<int>(color.b) « ")\n";

        windowRobotCreator.close();
    });
}
```

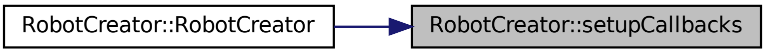
References colorBox, createButton, heightBox, hexToColor(), isValidFloat(), isValidHexColor(), resetButton, UnicicleWmr::setColor(), UnicicleWmr::setDimensions(), UnicicleWmr::setRobotActive(), widthBox, windowRobotCreator, and wmr.

Referenced by RobotCreator().

Here is the call graph for this function:


RobotCreator::setupCallbacks
RobotCreator::hexToColor
RobotCreator::isValidFloat
RobotCreator::isValidHex Color
UnicicleWmr::setColor
UnicicleWmr::setDimensions
UnicicleWmr::setRobotActive

Here is the caller graph for this function:



3.9.3.10 setupWidgets()

```
void RobotCreator::setupWidgets ( )
```

Initializes and sets up all GUI widgets (EditBoxes, Buttons)

Retrieves and validates GUI widgets from the loaded form.

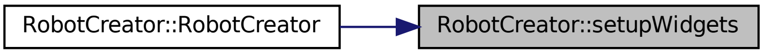
Definition at line 84 of file RobotCreator.cpp.

```
{
    widthBox  = gui.get<tgui::EditBox>("widthBox");
    heightBox = gui.get<tgui::EditBox>("heightBox");
    colorBox  = gui.get<tgui::EditBox>("colorBox");

    resetButton  = gui.get<tgui::Button>("Button1");
    createButton = gui.get<tgui::Button>("Button2");

    if (!widthBox || !heightBox || !colorBox || !resetButton || !createButton)
    {
        std::cerr « "[ERROR] One or more widgets could not be found.\n";
        return;
    }

    // Optional: set input validators (commented out)
    // widthBox->setInputValidator(R"(^-?\d*\.?\d+$)");
    // heightBox->setInputValidator(R"(^-?\d*\.?\d+$)");
}
```

References colorBox, createButton, gui, heightBox, resetButton, and widthBox.

Referenced by RobotCreator().

Here is the caller graph for this function:



3.9.3.11 update()

```
void RobotCreator::update ( )
```

Updates internal state and GUI elements each frame.

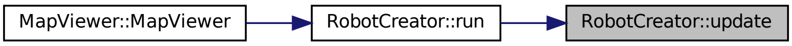
Placeholder for future update logic.

Definition at line 66 of file RobotCreator.cpp.

```
{
    // Currently no update logic
}
```

Referenced by run().

Here is the caller graph for this function:



3.9.4 Member Data Documentation

3.9.4.1 colorBox

```
tgui::EditBox::Ptr RobotCreator::colorBox  [private]
```

Input for robot color (hexadecimal string)

Definition at line 67 of file RobotCreator.hpp.

Referenced by setupCallbacks(), and setupWidgets().

3.9.4.2 createButton

```
tgui::Button::Ptr RobotCreator::createButton  [private]
```

Applies parameters and creates/configures the robot.

Definition at line 71 of file RobotCreator.hpp.

Referenced by setupCallbacks(), and setupWidgets().

3.9.4.3 gui

```
tgui::Gui RobotCreator::gui  [private]
```

TGUI GUI manager attached to the window.

Definition at line 61 of file RobotCreator.hpp.

Referenced by processEvents(), render(), RobotCreator(), and setupWidgets().

3.9.4.4 heightBox

```
tgui::EditBox::Ptr RobotCreator::heightBox  [private]
```

Input for robot height.

Definition at line 66 of file RobotCreator.hpp.

Referenced by setupCallbacks(), and setupWidgets().

3.9.4.5 resetButton

```
tgui::Button::Ptr RobotCreator::resetButton  [private]
```

Buttons for user actions.

Resets all input fields to default

Definition at line 70 of file RobotCreator.hpp.

Referenced by setupCallbacks(), and setupWidgets().

3.9.4.6 widthBox

```
tgui::EditBox::Ptr RobotCreator::widthBox  [private]
```

GUI widgets for robot parameters input.

Input for robot width

Definition at line 65 of file RobotCreator.hpp.

Referenced by setupCallbacks(), and setupWidgets().

3.9.4.7 windowRobotCreator

```
sf::RenderWindow RobotCreator::windowRobotCreator  [private]
```

SFML window for robot creation GUI.

Definition at line 60 of file RobotCreator.hpp.

Referenced by isRunning(), processEvents(), render(), and setupCallbacks().

3.9.4.8 wmr

```
UnicicleWmr& RobotCreator::wmr  [private]
```

Reference to the robot being configured.

Definition at line 62 of file RobotCreator.hpp.

Referenced by setupCallbacks().

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

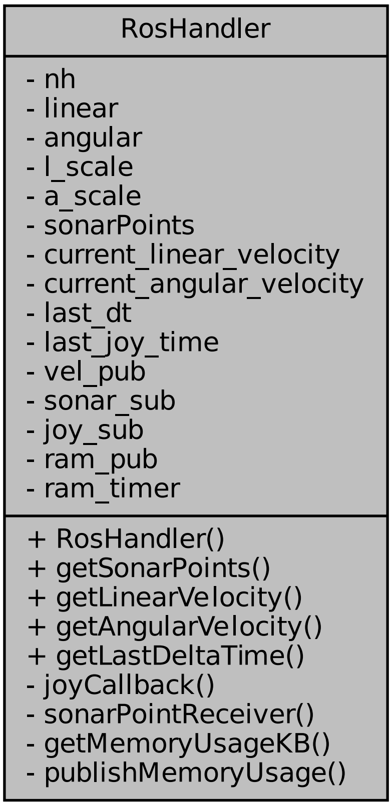
- include/artgslam_vsc/RobotCreator.hpp
- src/RobotCreator.cpp

3.10 RosHandler Class Reference

Handles ROS communication:

```
#include <RosHandler.hpp>
```

Collaboration diagram for RosHandler:



Public Member Functions

- RosHandler ()

  Constructor.
- const std::vector< geometry_msgs::Point32 > & getSonarPoints () const

  Gets the vector of sonar points received from the sensor.
- double getLinearVelocity () const
- double getAngularVelocity () const
- float getLastDeltaTime () const

Private Member Functions

- void joyCallback (const sensor_msgs::Joy::ConstPtr &joy)

  Callback for joystick messages.
- void sonarPointReceiver (const geometry_msgs::Point32::ConstPtr &sonar)

  Callback for sonar point messages.
- long getMemoryUsageKB ()

  Reads current RAM usage of the process in KB.
- void publishMemoryUsage (const ros::TimerEvent &)

  Periodically publishes the RAM usage to a ROS topic.

Private Attributes

- ros::NodeHandle nh

  ROS node handle.
- int linear
- int angular

  Raw joystick axis indices.
- double l_scale
- double a_scale

  Linear and angular velocity scaling factors.
- std::vector< geometry_msgs::Point32 > sonarPoints

  Points from sonar sensor.
- double current_linear_velocity = 0.0

  Current linear velocity.
- double current_angular_velocity = 0.0

  Current angular velocity.
- float last_dt = 0.0

  Time since last joystick message.
- ros::Time last_joy_time

  Timestamp of last joystick message received.
- ros::Publisher vel_pub

  Publisher for velocity commands.
- ros::Subscriber sonar_sub

  Subscriber for sonar point messages.
- ros::Subscriber joy_sub

  Subscriber for joystick messages.
- ros::Publisher ram_pub

  Publisher for RAM usage info.
- ros::Timer ram_timer

  Timer to trigger RAM publishing.

3.10.1 Detailed Description

Handles ROS communication:

- Publishes velocity commands
- Subscribes to joystick and sonar data
- Provides access to velocities and sonar points

Definition at line 26 of file RosHandler.hpp.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 RosHandler()

RosHandler::RosHandler ( )

Constructor.

Constructs a RosHandler object.

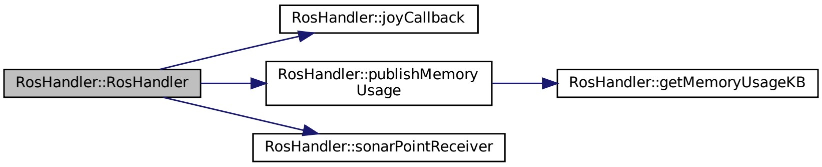
Initializes ROS parameters for joystick axes and scales, sets up publishers and subscribers for velocity commands, joystick inputs, sonar data, and RAM usage monitoring. Also initializes timers for periodic memory usage publishing.

Definition at line 11 of file RosHandler.cpp.

```
: linear(1), angular(0), l_scale(0.5), a_scale(0.5)
{
    // Read joystick axis parameters from ROS parameter server or use defaults
    nh.param("axis_linear", linear, linear);
    nh.param("axis_angular", angular, angular);
    nh.param("scale_angular", a_scale, a_scale);
    nh.param("scale_linear", l_scale, l_scale);

    // Publisher for velocity commands to the wheeled mobile robot using ROSARIA
    // Tested with Adept Mobile Robots Amigobot
    vel_pub = nh.advertise<geometry_msgs::Twist>("RosAria/cmd_vel", 1);

    // Subscribe to joystick inputs
    joy_sub = nh.subscribe<sensor_msgs::Joy>("joy", 10, &RosHandler::joyCallback, this);

    // Subscribe to sonar data published on the ROS topic "sonarFilterdata_bag"
    sonar_sub = nh.subscribe("sonarFilterdata_bag", 1000, &RosHandler::sonarPointReceiver, this);

    // Initialize last joystick input time for synchronizing robot movement (testing pending)
    last_joy_time = ros::Time::now();

    // Publisher for RAM usage data (in kilobytes) for monitoring memory usage
    ram_pub = nh.advertise<std_msgs::Int32>("ram_usage_kb", 10);

    // Timer to periodically publish RAM usage every 1 second
    ram_timer = nh.createTimer(ros::Duration(1.0), &RosHandler::publishMemoryUsage, this);
}
```

References a_scale, angular, joy_sub, joyCallback(), l_scale, last_joy_time, linear, nh, publishMemoryUsage(), ram_pub, ram_timer, sonar_sub, sonarPointReceiver(), and vel_pub.

Here is the call graph for this function:


RosHandler::joyCallback
RosHandler::RosHandler
RosHandler::publishMemory Usage
RosHandler::getMemoryUsageKB
RosHandler::sonarPointReceiver

3.10.3 Member Function Documentation

3.10.3.1 getAngularVelocity()

```
double RosHandler::getAngularVelocity ( ) const  [inline]
```

Returns

Current robot angular velocity for animation purposes

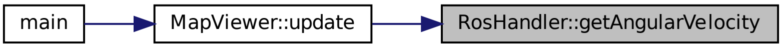
Definition at line 38 of file RosHandler.hpp.

```
38 { return current_angular_velocity; }
```

References current_angular_velocity.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.10.3.2 getLastDeltaTime()

```
float RosHandler::getLastDeltaTime ( ) const  [inline]
```

Returns

Time since last joystick update

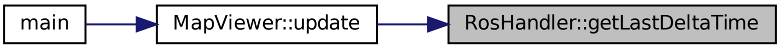
Definition at line 41 of file RosHandler.hpp.

```
41 { return last_dt; }
```

References last_dt.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.10.3.3 getLinearVelocity()

```
double RosHandler::getLinearVelocity ( ) const  [inline]
```

Returns

Current robot linear velocity for animation purposes

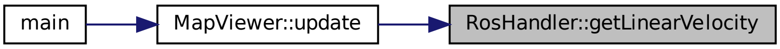
Definition at line 35 of file RosHandler.hpp.

```
35 { return current_linear_velocity; }
```

References current_linear_velocity.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.10.3.4 getMemoryUsageKB()

```
long RosHandler::getMemoryUsageKB ( )  [inline], [private]
```

Reads current RAM usage of the process in KB.

Reads the current process memory usage (Resident Set Size) in kilobytes.

Returns

RAM usage in kilobytes

Parses the "/proc/self/status" file to extract the VmRSS value.

Returns

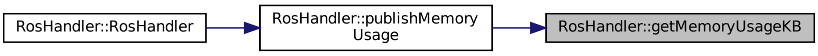
> Memory usage in kilobytes, or -1 if reading failed.

Definition at line 109 of file RosHandler.cpp.

```
109                                          {
110     std::ifstream status("/proc/self/status");
111     std::string line;
112     while (std::getline(status, line)) {
113         if (line.rfind("VmRSS:", 0) == 0) { // Line starts with "VmRSS:"
114             long kb;
115             sscanf(line.c_str(), "VmRSS: %ld kB", &kb);
116             return kb;
117         }
118     }
119     return -1; // Return -1 if memory info could not be read
120 }
```

Referenced by publishMemoryUsage().

Here is the caller graph for this function:



3.10.3.5 getSonarPoints()

```
const std::vector< geometry_msgs::Point32 > & RosHandler::getSonarPoints ( ) const
```

Gets the vector of sonar points received from the sensor.

Returns

> Sonar points received from ROS used to build the map
>
> const reference to a vector containing sonar points.

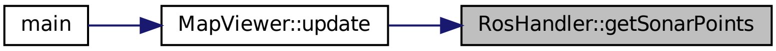
Definition at line 45 of file RosHandler.cpp.

```
46 {
47     return sonarPoints;
48 }
```

References sonarPoints.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.10.3.6 joyCallback()

```
void RosHandler::joyCallback (
             const sensor_msgs::Joy::ConstPtr & joy )  [private]
```

Callback for joystick messages.

Callback function for joystick input messages.

Reads joystick values and publishes velocity commands.

Parameters

joy	Incoming joystick message

Processes joystick inputs to compute linear and angular velocities, publishes velocity commands to control the robot, and logs the current velocity values.

Parameters

joy	Const pointer to the joystick message received.

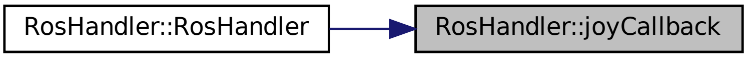
Definition at line 59 of file RosHandler.cpp.

```
{
    // Calculate time elapsed since last joystick message for potential synchronization
    ros::Time now = ros::Time::now();
    ros::Duration delta = now - last_joy_time;
    last_joy_time = now;
    last_dt = delta.toSec();

    // Create velocity command based on joystick input and scale parameters
    geometry_msgs::Twist twist;
    twist.angular.z = a_scale * joy->axes[angular];
    twist.linear.x = l_scale * joy->axes[linear];

    // Store current velocities for external access
    current_linear_velocity = twist.linear.x;
    current_angular_velocity = twist.angular.z;

    // Publish velocity command to control the robot
    vel_pub.publish(twist);

    // Log current velocities for debugging
    ROS_INFO_STREAM("[JOY] Linear Velocity: " « twist.linear.x
                    « " | Angular Velocity: " « twist.angular.z);
}
```

References a_scale, angular, current_angular_velocity, current_linear_velocity, l_scale, last_dt, last_joy_time, linear, and vel_pub.

Referenced by RosHandler().

Here is the caller graph for this function:



3.10.3.7 publishMemoryUsage()

```
void RosHandler::publishMemoryUsage (
             const ros::TimerEvent &  )  [inline], [private]
```

Periodically publishes the RAM usage to a ROS topic.

Timer callback function to publish the current memory usage.

Parameters

event	Timer event info

Publishes the memory usage (in KB) to a ROS topic for monitoring.

Parameters

event	ROS timer event information (unused).

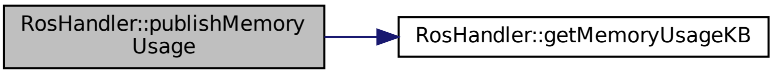
Definition at line 129 of file RosHandler.cpp.

```
129                                                        {
130     std_msgs::Int32 msg;
131     msg.data = static_cast<int>(getMemoryUsageKB());
132     ram_pub.publish(msg);
133 }
```

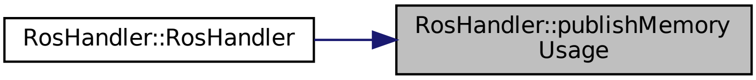
References getMemoryUsageKB(), and ram_pub.

Referenced by RosHandler().

Here is the call graph for this function:



Here is the caller graph for this function:



3.10.3.8 sonarPointReceiver()

```
void RosHandler::sonarPointReceiver (
            const geometry_msgs::Point32::ConstPtr & msg )  [private]
```

Callback for sonar point messages.

Callback function to receive sonar points from ROS topic.

Receives sonar point data to be used in mapping.

Parameters

sonar	Incoming sonar point message

Adds received sonar point to internal buffer and logs the coordinates.

Parameters

msg	Const pointer to the sonar point message received.

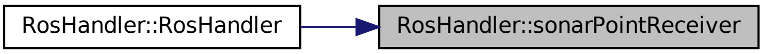
Definition at line 91 of file RosHandler.cpp.

```
{
    // Append new sonar point to vector
    sonarPoints.push_back(*msg);

    // Log received sonar point coordinates
    ROS_INFO_STREAM("[SONAR] Point received -> X: " « msg->x
                     « " | Y: " « msg->y
                     « " | Z: " « msg->z);
}
```

References sonarPoints.

Referenced by RosHandler().

Here is the caller graph for this function:



3.10.4 Member Data Documentation

3.10.4.1 a_scale

```
double RosHandler::a_scale  [private]
```

Linear and angular velocity scaling factors.

Definition at line 48 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

3.10.4.2 angular

```
int RosHandler::angular  [private]
```

Raw joystick axis indices.

Definition at line 47 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

3.10.4.3 current_angular_velocity

```
double RosHandler::current_angular_velocity = 0.0  [private]
```

Current angular velocity.

Definition at line 53 of file RosHandler.hpp.

Referenced by getAngularVelocity(), and joyCallback().

3.10.4.4 current_linear_velocity

```
double RosHandler::current_linear_velocity = 0.0  [private]
```

Current linear velocity.

Definition at line 52 of file RosHandler.hpp.

Referenced by getLinearVelocity(), and joyCallback().

3.10.4.5 joy_sub

```
ros::Subscriber RosHandler::joy_sub  [private]
```

Subscriber for joystick messages.

Definition at line 60 of file RosHandler.hpp.

Referenced by RosHandler().

3.10.4.6 l_scale

```
double RosHandler::l_scale  [private]
```

Definition at line 48 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

3.10.4.7 last_dt

```
float RosHandler::last_dt = 0.0  [private]
```

Time since last joystick message.

Definition at line 54 of file RosHandler.hpp.

Referenced by getLastDeltaTime(), and joyCallback().

3.10.4.8 last_joy_time

```
ros::Time RosHandler::last_joy_time  [private]
```

Timestamp of last joystick message received.

Definition at line 55 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

3.10.4.9 linear

```
int RosHandler::linear  [private]
```

Definition at line 47 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

3.10.4.10 nh

```
ros::NodeHandle RosHandler::nh  [private]
```

ROS node handle.

Definition at line 44 of file RosHandler.hpp.

Referenced by RosHandler().

3.10.4.11 ram_pub

```
ros::Publisher RosHandler::ram_pub  [private]
```

Publisher for RAM usage info.

Definition at line 63 of file RosHandler.hpp.

Referenced by publishMemoryUsage(), and RosHandler().

3.10.4.12 ram_timer

```
ros::Timer RosHandler::ram_timer  [private]
```

Timer to trigger RAM publishing.

Definition at line 64 of file RosHandler.hpp.

Referenced by RosHandler().

3.10.4.13 sonar_sub

```
ros::Subscriber RosHandler::sonar_sub  [private]
```

Subscriber for sonar point messages.

Definition at line 59 of file RosHandler.hpp.

Referenced by RosHandler().

3.10.4.14 sonarPoints

```
std::vector<geometry_msgs::Point32> RosHandler::sonarPoints  [private]
```

Points from sonar sensor.

Definition at line 51 of file RosHandler.hpp.

Referenced by getSonarPoints(), and sonarPointReceiver().

3.10.4.15 vel_pub

```
ros::Publisher RosHandler::vel_pub  [private]
```

Publisher for velocity commands.

Definition at line 58 of file RosHandler.hpp.

Referenced by joyCallback(), and RosHandler().

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

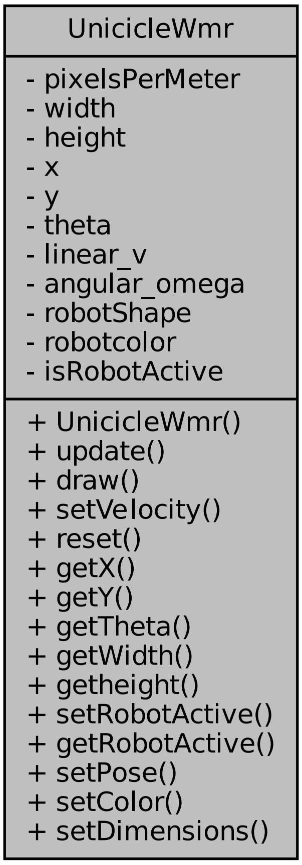
- include/artgslam_vsc/RosHandler.hpp
- src/RosHandler.cpp

3.11 UnicicleWmr Class Reference

Simulates a unicycle model robot with position, velocity, and rendering support.

```
#include <UnicicleWmr.hpp>
```

Collaboration diagram for UnicicleWmr:



Public Member Functions

- UnicicleWmr (float width_m=0.28f, float height_m=0.33f, float pixelsPerMeter=50.0f)

    Constructor.
- void update (float dt)

    Updates the robot's position and orientation using its velocity.
- void draw (sf::RenderWindow &window)

    Draws the robot on the screen.
- void setVelocity (float line_v, float angular_omega)

    Sets the linear and angular velocity of the robot.
- void reset (float x=0.0f, float y=0.0f, float theta=0.0f)

    Resets the robot pose and velocities.
- float getX () const
- float getY () const
- float getTheta () const
- float getWidth () const
- float getheight () const
- void setRobotActive (bool active)
- bool getRobotActive () const
- void setPose (float x, float y, float theta)

    Sets the robot's pose.
- void setColor (sf::Color color)

    Sets the color used for drawing the robot.
- void setDimensions (float width, float height)

    Updates the robot's dimensions and resizes the SFML shape accordingly.

Private Attributes

- float pixelsPerMeter

    Conversion factor from meters to pixels.
- float width

    Robot width in meters.
- float height

    Robot height in meters.
- float x

    X position in meters.
- float y

    Y position in meters.
- float theta

    Orientation in radians.
- float linear_v

    Linear velocity (m/s)
- float angular_omega

    Angular velocity (rad/s)
- sf::RectangleShape robotShape
- sf::Color robotcolor

    Color of the robot shape.
- bool isRobotActive = false

3.11.1 Detailed Description

Simulates a unicycle model robot with position, velocity, and rendering support.

Definition at line 10 of file UnicicleWmr.hpp.

3.11.2 Constructor & Destructor Documentation

3.11.2.1 UnicicleWmr()

```
UnicicleWmr::UnicicleWmr (
            float width_m = 0.28f,
            float height_m = 0.33f,
            float pixelsPerMeter = 50.0f )
```

Constructor.

Parameters

width_m	Width of the robot in meters.
height_m	Height of the robot in meters.
pixelsPerMeter	Conversion factor to render robot dimensions in pixels.

Initializes the wheeled mobile robot with dimensions in meters and pixels-per-meter scale. Sets initial position (0,0), orientation 0 radians, zero velocities, and default blue color.

Parameters

width_m	Width of the robot in meters.
height_m	Height of the robot in meters.
pixelsPerMeter	Scale factor converting meters to pixels.

Setup the SFML rectangle shape representing the robot with correct size and origin at center

Definition at line 13 of file UnicicleWmr.cpp.

```
    : width(width_m), height(height_m), pixelsPerMeter(pixelsPerMeter),
      x(0.f), y(0.f), theta(0.f),
      linear_v(0.f), angular_omega(0.f),
      robotcolor(sf::Color::Blue)
{
    /** Setup the SFML rectangle shape representing the robot with correct size and origin at center */
    robotShape.setSize(sf::Vector2f(width * pixelsPerMeter, height * pixelsPerMeter));
    robotShape.setOrigin(robotShape.getSize() / 2.f);
    robotShape.setFillColor(robotcolor);
}
```

References height, pixelsPerMeter, robotcolor, robotShape, and width.

3.11.3 Member Function Documentation

3.11.3.1 draw()

```
void UnicicleWmr::draw (
             sf::RenderWindow & window )
```

Draws the robot on the screen.

Draws the robot on the given SFML render window if active.

Parameters

window	SFML render window.

Positions and rotates the shape based on current pose.

Parameters

window	Reference to the SFML render window.

Convert radians to degrees

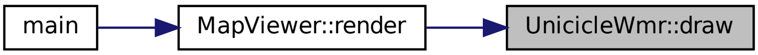
Definition at line 46 of file UnicicleWmr.cpp.

```
{
    if (isRobotActive) {
        robotShape.setPosition(x * pixelsPerMeter, y * pixelsPerMeter);
        robotShape.setRotation(theta * 180.f / 3.14159265f);  /** Convert radians to degrees */
        robotShape.setFillColor(robotcolor);
        window.draw(robotShape);
    }
}
```

References isRobotActive, pixelsPerMeter, robotcolor, robotShape, theta, x, and y.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.11.3.2 getheight()

```
float UnicicleWmr::getheight ( ) const  [inline]
```

Definition at line 54 of file UnicicleWmr.hpp.

```
{ return height; }
```

References height.

3.11.3.3 getRobotActive()

```
bool UnicicleWmr::getRobotActive ( ) const  [inline]
```

Definition at line 58 of file UnicicleWmr.hpp.

```
58 { return isRobotActive; }
```

References isRobotActive.

3.11.3.4 getTheta()

```
float UnicicleWmr::getTheta ( ) const  [inline]
```

Definition at line 50 of file UnicicleWmr.hpp.

```
50 { return theta; }
```

References theta.

3.11.3.5 getWidth()

```
float UnicicleWmr::getWidth ( ) const  [inline]
```

Definition at line 53 of file UnicicleWmr.hpp.

```
53 { return width; }
```

References width.

3.11.3.6 getX()

```
float UnicicleWmr::getX ( ) const  [inline]
```

Definition at line 48 of file UnicicleWmr.hpp.

```
48 { return x; }
```

References x.

3.11.3.7 getY()

```
float UnicicleWmr::getY ( ) const  [inline]
```

Definition at line 49 of file UnicicleWmr.hpp.

```
49 { return y; }
```

References y.

3.11.3.8 reset()

```
void UnicicleWmr::reset (
             float x = 0.0f,
             float y = 0.0f,
             float theta = 0.0f )
```

Resets the robot pose and velocities.

Resets the robot's pose to the specified position and orientation and stops movement.

Parameters

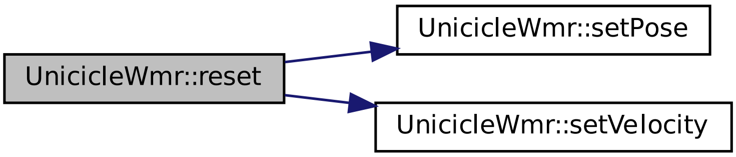
x	X position in meters.
y	Y position in meters.
theta	Orientation in radians.
x	New x-position in meters.
y	New y-position in meters.
theta	New orientation in radians.

Definition at line 75 of file UnicicleWmr.cpp.

```
{
    setPose(x, y, theta);
    setVelocity(0.f, 0.f);
}
```

References setPose(), setVelocity(), theta, x, and y.

Here is the call graph for this function:



3.11.3.9 setColor()

```
void UnicicleWmr::setColor (
            sf::Color color )
```

Sets the color used for drawing the robot.

Parameters

color	SFML color to use.

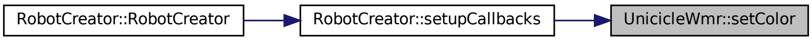
Definition at line 102 of file UnicicleWmr.cpp.

```
{
    robotcolor = color;
}
```

References robotcolor.

Referenced by RobotCreator::setupCallbacks().

Here is the caller graph for this function:



3.11.3.10 setDimensions()

```
void UnicicleWmr::setDimensions (
             float width,
             float height )
```

Updates the robot's dimensions and resizes the SFML shape accordingly.

Also resets the origin to the center for correct rotation and positioning.

Parameters

width	New width in meters.
height	New height in meters.

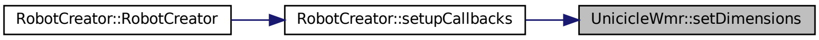
Definition at line 115 of file UnicicleWmr.cpp.

```
{
    this->width = width;
    this->height = height;
    robotShape.setSize(sf::Vector2f(width * pixelsPerMeter, height * pixelsPerMeter));
    robotShape.setOrigin(robotShape.getSize() / 2.f);
}
```

References height, pixelsPerMeter, robotShape, and width.

Referenced by RobotCreator::setupCallbacks().

Here is the caller graph for this function:


RobotCreator::RobotCreator
RobotCreator::setupCallbacks
UnicicleWmr::setDimensions

3.11.3.11 setPose()

```
void UnicicleWmr::setPose (
             float x,
             float y,
             float theta )
```

Sets the robot's pose.

Updates the position (x, y) and orientation (theta).

Parameters

x	X-position in meters.
y	Y-position in meters.
theta	Orientation in radians.

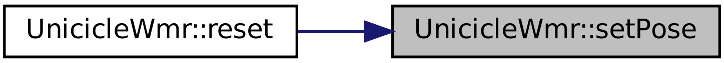
Definition at line 90 of file UnicicleWmr.cpp.

```
91 {
92     this->x = x;
93     this->y = y;
94     this->theta = theta;
95 }
```

References theta, x, and y.

Referenced by reset().

Here is the caller graph for this function:



3.11.3.12 setRobotActive()

```
void UnicicleWmr::setRobotActive (
             bool active )  [inline]
```

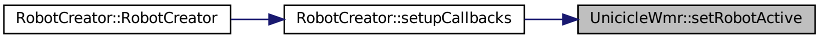
Definition at line 57 of file UnicicleWmr.hpp.

```
57 { isRobotActive = active; }
```

References isRobotActive.

Referenced by RobotCreator::setupCallbacks().

Here is the caller graph for this function:


RobotCreator::RobotCreator
RobotCreator::setupCallbacks
UnicicleWmr::setRobotActive

3.11.3.13 setVelocity()

```
void UnicicleWmr::setVelocity (
             float linear_v,
             float angular_omega )
```

Sets the linear and angular velocity of the robot.

Sets the linear and angular velocities of the robot.

Parameters

line_v	Linear velocity in m/s.
angular_omega	Angular velocity in rad/s.
linear_v	Linear velocity (forward) in meters per second.
angular_omega	Angular velocity (rotational) in radians per second.

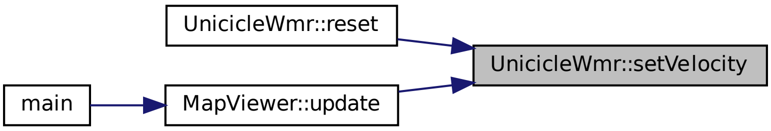
Definition at line 62 of file UnicicleWmr.cpp.

```
{
    this->linear_v = linear_v;
    this->angular_omega = angular_omega;
}
```

References angular_omega, and linear_v.

Referenced by reset(), and MapViewer::update().

Here is the caller graph for this function:



3.11.3.14 update()

```
void UnicicleWmr::update (
            float dt )
```

Updates the robot's position and orientation using its velocity.

Updates the robot's pose based on current velocities and elapsed time.

Parameters

dt	Time step in seconds.

Simple unicycle kinematics: update x, y, and theta accordingly.

Parameters

dt	Time interval in seconds since last update.

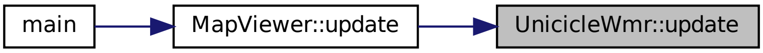
Definition at line 32 of file UnicicleWmr.cpp.

```
{
    x += linear_v * cos(theta) * dt;
    y += linear_v * sin(theta) * dt;
    theta += angular_omega * dt;
}
```

References angular_omega, linear_v, theta, x, and y.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.11.4 Member Data Documentation

3.11.4.1 angular_omega

```
float UnicicleWmr::angular_omega  [private]
```

Angular velocity (rad/s)

Definition at line 79 of file UnicicleWmr.hpp.

Referenced by setVelocity(), and update().

3.11.4.2 height

```
float UnicicleWmr::height  [private]
```

Robot height in meters.

Definition at line 70 of file UnicicleWmr.hpp.

Referenced by getheight(), setDimensions(), and UnicicleWmr().

3.11.4.3 isRobotActive

```
bool UnicicleWmr::isRobotActive = false  [private]
```

Definition at line 86 of file UnicicleWmr.hpp.

Referenced by draw(), getRobotActive(), and setRobotActive().

3.11.4.4 linear_v

```
float UnicicleWmr::linear_v  [private]
```

Linear velocity (m/s)

Definition at line 78 of file UnicicleWmr.hpp.

Referenced by setVelocity(), and update().

3.11.4.5 pixelsPerMeter

```
float UnicicleWmr::pixelsPerMeter  [private]
```

Conversion factor from meters to pixels.

Definition at line 66 of file UnicicleWmr.hpp.

Referenced by draw(), setDimensions(), and UnicicleWmr().

3.11.4.6 robotcolor

```
sf::Color UnicicleWmr::robotcolor  [private]
```

Color of the robot shape.

Definition at line 83 of file UnicicleWmr.hpp.

Referenced by draw(), setColor(), and UnicicleWmr().

3.11.4.7 robotShape

```
sf::RectangleShape UnicicleWmr::robotShape  [private]
```

Definition at line 82 of file UnicicleWmr.hpp.

Referenced by draw(), setDimensions(), and UnicicleWmr().

3.11.4.8 theta

```
float UnicicleWmr::theta  [private]
```

Orientation in radians.

Definition at line 75 of file UnicicleWmr.hpp.

Referenced by draw(), getTheta(), reset(), setPose(), and update().

3.11.4.9 width

```
float UnicicleWmr::width  [private]
```

Robot width in meters.

Definition at line 69 of file UnicicleWmr.hpp.

Referenced by getWidth(), setDimensions(), and UnicicleWmr().

3.11.4.10 x

```
float UnicicleWmr::x  [private]
```

X position in meters.

Definition at line 73 of file UnicicleWmr.hpp.

Referenced by draw(), getX(), reset(), setPose(), and update().

3.11.4.11 y

```
float UnicicleWmr::y  [private]
```

Y position in meters.

Definition at line 74 of file UnicicleWmr.hpp.

Referenced by draw(), getY(), reset(), setPose(), and update().

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

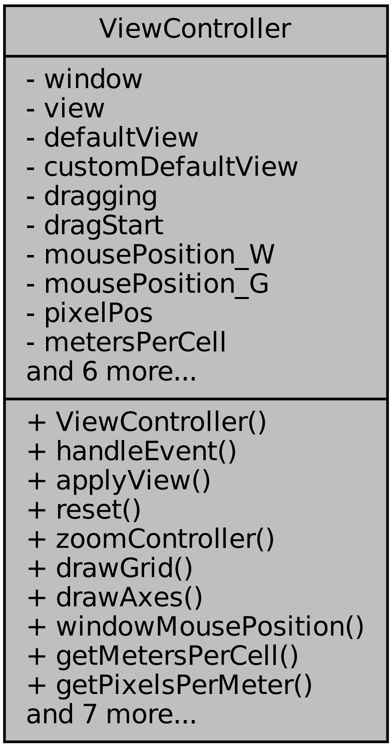
- include/artgslam_vsc/UnicicleWmr.hpp
- src/UnicicleWmr.cpp

3.12 ViewController Class Reference

Controls view, zoom, and panning.

```
#include <ViewController.hpp>
```

Collaboration diagram for ViewController:



Public Member Functions

- ViewController (sf::RenderWindow &win, float metersPerCell, float pixelsPerMeter, sf::View &view)

  Constructor.
- void handleEvent (const sf::Event &event)

  Handles mouse movement events for panning and zooming.
- void applyView ()

  Applies the current view settings to the window.
- void reset ()

  Resets the view to its default state.
- void zoomController (const sf::Event &event)

  Handles zoom in/out based on user input events.
- void drawGrid (sf::RenderTarget &target)

  Draws the grid lines on the provided render target.

- void drawAxes (sf::RenderTarget &target)

  Draws the axes lines and numeration on the render target.
- sf::Vector2i windowMousePosition () const

  Returns the mouse position in window pixel coordinates.
- float getMetersPerCell () const

  Returns the size of each grid cell in meters.
- float getPixelsPerMeter () const

  Returns the pixels per meter scale factor.
- sf::Vector2i getMousePixelPosition () const

  Returns mouse position in grid coordinates.
- sf::Vector2f getMouseWorldPosition () const

  Returns mouse position in world coordinates.
- sf::View getDefaultView () const

  Returns the default view of the window.
- int getMapSizeCells () const

  Returns the total number of grid cells per side.
- sf::View getView () const

  Returns the current view.
- float getZoom () const

  Returns the current zoom level.
- sf::Vector2i getHoveredCell (int gridSize) const

  Returns the grid cell index hovered by the mouse.

Private Attributes

- sf::RenderWindow & window

  Reference to the SFML window.
- sf::View & view

  Reference to the controlled view.
- sf::View defaultView

  Default view settings.
- sf::View customDefaultView

  Custom default view if needed.
- bool dragging = false

  True if currently dragging/panning.
- sf::Vector2i dragStart

  Mouse position where dragging started.
- sf::Vector2f mousePosition_W

  Mouse position in world coordinates.
- sf::Vector2i mousePosition_G

  Mouse position in grid coordinates.
- sf::Vector2i pixelPos

  Mouse position in window pixels.
- float metersPerCell

  Size of each grid cell in meters.
- float pixelsPerMeter

  Conversion factor from meters to pixels.
- const int mapSizeCells = 1000

  Number of cells per side in the map.
- sf::Font font

Font used for axis numbering.

- bool fontLoaded = false

  Flag indicating font load success.

- std::vector< std::vector< sf::Vector2f > > cellTopLeft_

  Cached coordinates of top-left corners of cells.

- bool cellCoordsValid_ = false

  Indicates if cached cell coordinates are valid.

3.12.1 Detailed Description

Controls view, zoom, and panning.

Also draws grid and axis numeration.

Definition at line 7 of file ViewController.hpp.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 ViewController()

```
ViewController::ViewController (
            sf::RenderWindow & win,
            float metersPerCell_,
            float pixelsPerMeter_,
            sf::View & view )
```

Constructor.

Parameters

win	Reference to the SFML render window.
metersPerCell	Size of each grid cell in meters.
pixelsPerMeter	Scale factor to convert meters to pixels.
view	Reference to the SFML view object to control.

Initializes the view controller with references to the SFML window and view, as well as parameters defining grid cell size and pixel scale. Loads a custom font for rendering axis labels.

Parameters

win	Reference to the SFML render window.
*metersPer↩ Cell_*	Size of each grid cell in meters.
*pixelsPer↩ Meter_*	Scale factor from meters to pixels.
view	Reference to the SFML view to control.

Initialize the view with a zoom level of 3 (closer zoom)

Load font from ROS package assets for axis labels

Definition at line 19 of file ViewController.cpp.

```
20     : window(win), metersPerCell(metersPerCell_), pixelsPerMeter(pixelsPerMeter_), view(view)
21 {
22     /** Initialize the view with a zoom level of 3 (closer zoom) */
23     defaultView = window.getDefaultView();
24 
25     view = defaultView;
26     view.setCenter(0.f, 0.f);
27     view.setSize(defaultView.getSize() / 3.f);
28     customDefaultView = view;
29 
30     /** Load font from ROS package assets for axis labels */
31     std::string package_path = ros::package::getPath("artgslam_vsc");
32     std::string fontPath = package_path + "/assets/fonts/NotoSansMath-Regular.ttf";
33 
34     fontLoaded = font.loadFromFile(fontPath);
35     if (!fontLoaded) {
36         std::cerr « " Error: Could not load font in ViewController from: " « fontPath « std::endl;
37     } else {
38         std::cout « " Font successfully loaded from: " « fontPath « std::endl;
39     }
40 }
```

References customDefaultView, defaultView, font, fontLoaded, view, and window.

3.12.3 Member Function Documentation

3.12.3.1 applyView()

`void ViewController::applyView ( )`

Applies the current view settings to the window.

Applies the current view to the SFML window.

Should be called before rendering to set the view properly.

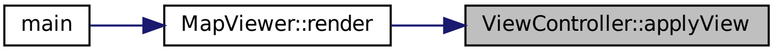
Definition at line 84 of file ViewController.cpp.

```
84                               {
85     window.setView(view);
86 }
```

References view, and window.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.12.3.2 drawAxes()

```
void ViewController::drawAxes (
            sf::RenderTarget & target )
```

Draws the axes lines and numeration on the render target.

Draws the X and Y axes centered at (0,0) with labels.

Parameters

target	SFML render target.

X axis is red, Y axis is blue. Labels are drawn every 5 cells along each axis, scaled according to current zoom level for readability.

Parameters

target	The render target on which to draw the axes.

Save current view

Use current zoomed/panned view

Calculate visible boundaries

Draw X axis (red) and Y axis (blue)

Draw axis labels on the HUD (default view)

Cell size in pixels

Clamp text scale between 0.5 and 1.5 for readability

Calculate visible cells in current view

Draw labels on X axis every 5 cells

Draw labels on Y axis every 5 cells

Restore original view

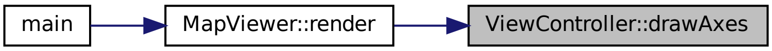
Definition at line 184 of file ViewController.cpp.

```
{
    if (!fontLoaded) return;

    /** Save current view */
    sf::View originalView = target.getView();
    target.setView(view);  /** Use current zoomed/panned view */

    sf::VertexArray axes(sf::Lines, 4);
    sf::Vector2f center = view.getCenter();
    sf::Vector2f size = view.getSize();

    /** Calculate visible boundaries */
    float left = center.x - size.x * 0.5f;
    float right = center.x + size.x * 0.5f;
    float top = center.y - size.y * 0.5f;
    float bottom = center.y + size.y * 0.5f;

    /** Draw X axis (red) and Y axis (blue) */
    axes[0] = sf::Vertex({left, 0.f}, sf::Color::Red);
    axes[1] = sf::Vertex({right, 0.f}, sf::Color::Red);
    axes[2] = sf::Vertex({0.f, top}, sf::Color::Blue);
    axes[3] = sf::Vertex({0.f, bottom}, sf::Color::Blue);

    target.draw(axes);

    /** Draw axis labels on the HUD (default view) */
    target.setView(defaultView);

    const float cellSizeWorld = metersPerCell * pixelsPerMeter;   /** Cell size in pixels */
    const float zoomLevel = defaultView.getSize().x / view.getSize().x;

    /** Clamp text scale between 0.5 and 1.5 for readability */
    float textScale = std::clamp(zoomLevel, 0.5f, 1.5f);
    const unsigned baseFontSize = 12;

    /** Calculate visible cells in current view */
    sf::Vector2f topLeft = window.mapPixelToCoords({0, 0}, view);
    sf::Vector2f bottomRight = window.mapPixelToCoords(
        {static_cast<int>(window.getSize().x), static_cast<int>(window.getSize().y)}, view);

    int firstCellX = static_cast<int>(std::floor(topLeft.x / cellSizeWorld));
    int firstCellY = static_cast<int>(std::floor(topLeft.y / cellSizeWorld));
    int numCols = static_cast<int>((bottomRight.x - topLeft.x) / cellSizeWorld) + 2;
    int numRows = static_cast<int>((bottomRight.y - topLeft.y) / cellSizeWorld) + 2;

    /** Draw labels on X axis every 5 cells */
    for (int i = 0; i <= numCols; ++i) {
        int cellX = firstCellX + i;
        if (cellX % 5 != 0) continue;

        float worldX = cellX * cellSizeWorld;
        sf::Vector2i scr = window.mapCoordsToPixel({worldX, 0.f}, view);
        sf::Vector2f pos(static_cast<float>(scr.x) + 2.f, 4.f);

        sf::Text txt(std::to_string(cellX), font, baseFontSize);
        txt.setFillColor(sf::Color::Yellow);
        txt.setScale(textScale, textScale);
        txt.setPosition(pos);

        sf::FloatRect b = txt.getLocalBounds();
        sf::RectangleShape bg({b.width * textScale, b.height * textScale});
        bg.setPosition(pos);
        bg.setFillColor(sf::Color(0, 0, 0, 180));

        target.draw(bg);
        target.draw(txt);
    }

    /** Draw labels on Y axis every 5 cells */
    for (int i = 0; i <= numRows; ++i) {
        int cellY = firstCellY + i;
        if (cellY % 5 != 0) continue;

        float worldY = cellY * cellSizeWorld;
        sf::Vector2i scr = window.mapCoordsToPixel({0.f, worldY}, view);
        sf::Vector2f pos(4.f, static_cast<float>(scr.y) + 2.f);

        sf::Text txt(std::to_string(cellY), font, baseFontSize);
        txt.setFillColor(sf::Color::Yellow);
        txt.setScale(textScale, textScale);
        txt.setPosition(pos);

        sf::FloatRect b = txt.getLocalBounds();
        sf::RectangleShape bg({b.width * textScale, b.height * textScale});
        bg.setPosition(pos);
        bg.setFillColor(sf::Color(0, 0, 0, 180));

        target.draw(bg);
        target.draw(txt);
    }

    /** Restore original view */
    target.setView(originalView);
}
```

References defaultView, font, fontLoaded, metersPerCell, pixelsPerMeter, view, and window.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.12.3.3 drawGrid()

```
void ViewController::drawGrid (
            sf::RenderTarget & target )
```

Draws the grid lines on the provided render target.

Draws the grid lines centered around (0,0).

Parameters

target	SFML render target (e.g., window).

The grid spans from -mapSizeCells/2 to +mapSizeCells/2 in both directions. Lines are drawn in gray.

Parameters

target	The render target on which to draw the grid.

Vertical lines

Horizontal lines

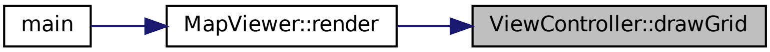
Definition at line 155 of file ViewController.cpp.

```
                                                    {
    float zoom = defaultView.getSize().x / view.getSize().x;
    float cellSize = metersPerCell * pixelsPerMeter;
    float halfWidth = (mapSizeCells / 2) * cellSize;

    sf::VertexArray lines(sf::Lines);

    for (int i = -mapSizeCells / 2; i <= mapSizeCells / 2; ++i) {
        float pos = i * cellSize;
        /** Vertical lines */
        lines.append(sf::Vertex({pos, -halfWidth}, sf::Color(100, 100, 100)));
        lines.append(sf::Vertex({pos, halfWidth}, sf::Color(100, 100, 100)));
        /** Horizontal lines */
        lines.append(sf::Vertex({-halfWidth, pos}, sf::Color(100, 100, 100)));
        lines.append(sf::Vertex({halfWidth, pos}, sf::Color(100, 100, 100)));
    }

    target.draw(lines);
}
```

References defaultView, mapSizeCells, metersPerCell, pixelsPerMeter, and view.

Referenced by MapViewer::render().

Here is the caller graph for this function:



3.12.3.4 getDefaultView()

```
sf::View ViewController::getDefaultView ( ) const  [inline]
```

Returns the default view of the window.

Definition at line 82 of file ViewController.hpp.

```
{ return defaultView; }
```

References defaultView.

3.12.3.5 getHoveredCell()

```
sf::Vector2i ViewController::getHoveredCell (
             int gridSize ) const
```

Returns the grid cell index hovered by the mouse.

Calculates the grid cell indices currently hovered by the mouse.

Parameters

gridSize	Size of the grid in cells.

Converts the mouse position from world coordinates to grid indices, taking into account the size of each cell and the grid origin offset. Returns {-1, -1} if the position is outside the grid.

Parameters

gridSize	The size (number of cells) of the square grid.

Returns

> sf::Vector2i The column and row indices of the hovered cell, or {-1, -1} if invalid.

Convert mouse world coordinates from pixels to meters

Adjust for grid origin being centered (offset)

Compute cell indices

Validate indices are within grid bounds

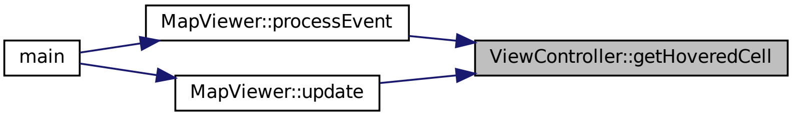
Definition at line 127 of file ViewController.cpp.

```
{
    /** Convert mouse world coordinates from pixels to meters */
    float worldX = mousePosition_W.x / pixelsPerMeter;
    float worldY = mousePosition_W.y / pixelsPerMeter;

    /** Adjust for grid origin being centered (offset) */
    float offset = (gridSize * metersPerCell) / 2.0f;

    /** Compute cell indices */
    int col = static_cast<int>(std::floor((worldX + offset) / metersPerCell));
    int row = static_cast<int>(std::floor((worldY + offset) / metersPerCell));

    /** Validate indices are within grid bounds */
    if (col < 0 || col >= gridSize || row < 0 || row >= gridSize)
        return {-1, -1};

    return {col, row};
}
```

References metersPerCell, mousePosition_W, and pixelsPerMeter.

Referenced by MapViewer::processEvent(), and MapViewer::update().

Here is the caller graph for this function:



3.12.3.6 getMapSizeCells()

```
int ViewController::getMapSizeCells ( ) const  [inline]
```

Returns the total number of grid cells per side.

Definition at line 87 of file ViewController.hpp.

```
87 { return mapSizeCells; }
```

References mapSizeCells.

3.12.3.7 getMetersPerCell()

```
float ViewController::getMetersPerCell ( ) const  [inline]
```

Returns the size of each grid cell in meters.

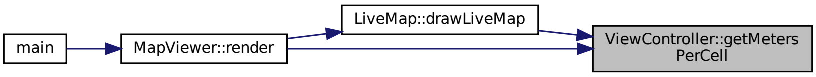
Definition at line 62 of file ViewController.hpp.

```
62 { return metersPerCell; }
```

References metersPerCell.

Referenced by LiveMap::drawLiveMap(), and MapViewer::render().

Here is the caller graph for this function:



3.12.3.8 getMousePixelPosition()

```
sf::Vector2i ViewController::getMousePixelPosition ( ) const  [inline]
```

Returns mouse position in grid coordinates.

Definition at line 72 of file ViewController.hpp.

```
72 { return mousePosition_G; }
```

References mousePosition_G.

3.12.3.9 getMouseWorldPosition()

```
sf::Vector2f ViewController::getMouseWorldPosition ( ) const  [inline]
```

Returns mouse position in world coordinates.

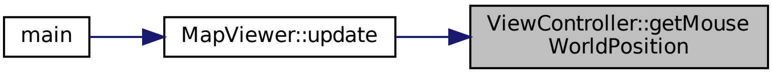
Definition at line 77 of file ViewController.hpp.

```
77 { return mousePosition_W; }
```

References mousePosition_W.

Referenced by MapViewer::update().

Here is the caller graph for this function:



3.12.3.10 getPixelsPerMeter()

```
float ViewController::getPixelsPerMeter ( ) const  [inline]
```

Returns the pixels per meter scale factor.

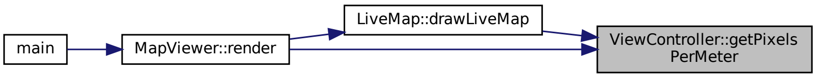
Definition at line 67 of file ViewController.hpp.

```
67 { return pixelsPerMeter; }
```

References pixelsPerMeter.

Referenced by LiveMap::drawLiveMap(), and MapViewer::render().

Here is the caller graph for this function:



3.12.3.11 getView()

```
sf::View ViewController::getView ( ) const
```

Returns the current view.

Gets the current SFML view object.

Returns

The current sf::View being used.

Definition at line 102 of file ViewController.cpp.

```
102                                      {
103     return view;
104 }
```

References view.

3.12.3.12 getZoom()

```
float ViewController::getZoom ( ) const
```

Returns the current zoom level.

Gets the current zoom factor.

The zoom factor is the ratio between the current view size and the default view size.

Returns

Current zoom factor as a float.

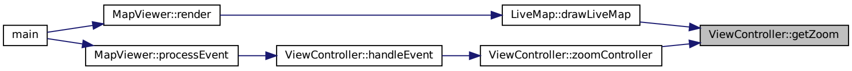
Definition at line 113 of file ViewController.cpp.

```
113                                    {
114     return view.getSize().x / defaultView.getSize().x;
115 }
```

References defaultView, and view.

Referenced by LiveMap::drawLiveMap(), and zoomController().

Here is the caller graph for this function:



3.12.3.13 handleEvent()

```
void ViewController::handleEvent (
             const sf::Event & event )
```

Handles mouse movement events for panning and zooming.

Handles all relevant SFML events.

Parameters

event	The SFML event to process.

Processes mouse wheel zoom, mouse dragging for panning, Escape key for resetting view, and mouse move events to update positions.

Parameters

event	Reference to the SFML event to process.

Calculate delta movement in world coordinates and move the view accordingly

Update mouse position in pixels and corresponding world coordinates

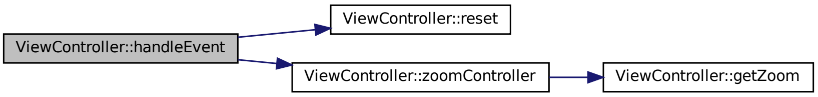
Definition at line 50 of file ViewController.cpp.

```
50                                                        {
51     if (event.type == sf::Event::MouseWheelScrolled) {
52         zoomController(event);
53     }
54     else if (event.type == sf::Event::MouseButtonPressed && event.mouseButton.button == sf::Mouse::Left)
       {
55         dragging = true;
56         dragStart = sf::Mouse::getPosition(window);
57     }
58     else if (event.type == sf::Event::MouseButtonReleased && event.mouseButton.button == sf::Mouse::Left)
       {
59         dragging = false;
60     }
61     else if (event.type == sf::Event::MouseMoved && dragging) {
62         /** Calculate delta movement in world coordinates and move the view accordingly */
63         sf::Vector2i now = sf::Mouse::getPosition(window);
64         sf::Vector2f delta = window.mapPixelToCoords(dragStart) - window.mapPixelToCoords(now);
65         view.move(delta);
66         dragStart = now;
67     }
68     else if (event.type == sf::Event::KeyPressed && event.key.code == sf::Keyboard::Escape) {
69         reset();
70     }
71     else if (event.type == sf::Event::MouseMoved) {
72         /** Update mouse position in pixels and corresponding world coordinates */
73         pixelPos = sf::Mouse::getPosition(window);
74         mousePosition_W = window.mapPixelToCoords(pixelPos, view);
75         mousePosition_G = pixelPos;
76     }
77 }
```

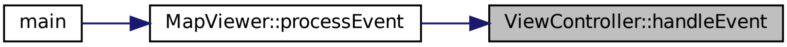
References dragging, dragStart, mousePosition_G, mousePosition_W, pixelPos, reset(), view, window, and zoomController().

Referenced by MapViewer::processEvent().

Here is the call graph for this function:



Here is the caller graph for this function:



3.12.3.14 reset()

```
void ViewController::reset ( )
```

Resets the view to its default state.

Resets the view to the custom default view.

This restores the initial zoom and center set in the constructor.

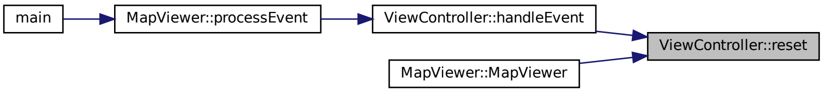
Definition at line 93 of file ViewController.cpp.

```
93                          {
94     view = customDefaultView;
95 }
```

References customDefaultView, and view.

Referenced by handleEvent(), and MapViewer::MapViewer().

Here is the caller graph for this function:



3.12.3.15 windowMousePosition()

```
sf::Vector2i ViewController::windowMousePosition ( ) const  [inline]
```

Returns the mouse position in window pixel coordinates.

Definition at line 57 of file ViewController.hpp.

```
57 { return pixelPos; }
```

References pixelPos.

3.12.3.16 zoomController()

```
void ViewController::zoomController (
            const sf::Event & event )
```

Handles zoom in/out based on user input events.

Controls zooming based on mouse wheel scroll events.

Parameters

event	The SFML event containing zoom information.

Limits zoom to be within min and max zoom levels to avoid excessive zooming in or out.

Parameters

event	The SFML mouse wheel scroll event triggering zoom.

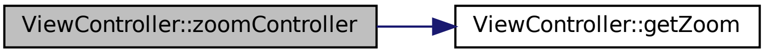
Definition at line 287 of file ViewController.cpp.

```
287                                                         {
288     float factor = (event.mouseWheelScroll.delta > 0) ? (1.f / 1.1f) : 1.1f;
289     float currentZoom = getZoom();
290     float newZoom = currentZoom * factor;
291
292     const float minZoom = 0.1f;
293     const float maxZoom = 1.0f;
294
295     std::cout « "newZoom: " « newZoom « std::endl;
296
297     if (newZoom < minZoom || newZoom > maxZoom) {
298         std::cout « "Zoom out of range [" « minZoom « ", " « maxZoom « "], operation cancelled\n";
299         return;
300     }
301
302     view.zoom(factor);
303 }
```

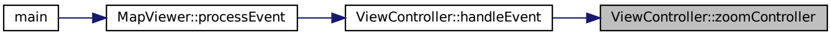
References getZoom(), and view.

Referenced by handleEvent().

Here is the call graph for this function:



Here is the caller graph for this function:



3.12.4 Member Data Documentation

3.12.4.1 cellCoordsValid_

```
bool ViewController::cellCoordsValid_ = false  [private]
```

Indicates if cached cell coordinates are valid.

Definition at line 127 of file ViewController.hpp.

3.12.4.2 cellTopLeft_

```
std::vector<std::vector<sf::Vector2f> > ViewController::cellTopLeft_  [private]
```

Cached coordinates of top-left corners of cells.

Definition at line 126 of file ViewController.hpp.

3.12.4.3 customDefaultView

```
sf::View ViewController::customDefaultView  [private]
```

Custom default view if needed.

Definition at line 109 of file ViewController.hpp.

Referenced by reset(), and ViewController().

3.12.4.4 defaultView

```
sf::View ViewController::defaultView  [private]
```

Default view settings.

Definition at line 108 of file ViewController.hpp.

Referenced by drawAxes(), drawGrid(), getDefaultView(), getZoom(), and ViewController().

3.12.4.5 dragging

```
bool ViewController::dragging = false  [private]
```

True if currently dragging/panning.

Definition at line 111 of file ViewController.hpp.

Referenced by handleEvent().

3.12.4.6 dragStart

```
sf::Vector2i ViewController::dragStart  [private]
```

Mouse position where dragging started.

Definition at line 112 of file ViewController.hpp.

Referenced by handleEvent().

3.12.4.7 font

```
sf::Font ViewController::font  [private]
```

Font used for axis numbering.

Definition at line 123 of file ViewController.hpp.

Referenced by drawAxes(), and ViewController().

3.12.4.8 fontLoaded

```
bool ViewController::fontLoaded = false  [private]
```

Flag indicating font load success.

Definition at line 124 of file ViewController.hpp.

Referenced by drawAxes(), and ViewController().

3.12.4.9 mapSizeCells

```
const int ViewController::mapSizeCells = 1000  [private]
```

Number of cells per side in the map.

Definition at line 121 of file ViewController.hpp.

Referenced by drawGrid(), and getMapSizeCells().

3.12.4.10 metersPerCell

```
float ViewController::metersPerCell  [private]
```

Size of each grid cell in meters.

Definition at line 119 of file ViewController.hpp.

Referenced by drawAxes(), drawGrid(), getHoveredCell(), and getMetersPerCell().

3.12.4.11 mousePosition_G

```
sf::Vector2i ViewController::mousePosition_G  [private]
```

Mouse position in grid coordinates.

Definition at line 116 of file ViewController.hpp.

Referenced by getMousePixelPosition(), and handleEvent().

3.12.4.12 mousePosition_W

```
sf::Vector2f ViewController::mousePosition_W  [private]
```

Mouse position in world coordinates.

Definition at line 115 of file ViewController.hpp.

Referenced by getHoveredCell(), getMouseWorldPosition(), and handleEvent().

3.12.4.13 pixelPos

```
sf::Vector2i ViewController::pixelPos  [private]
```

Mouse position in window pixels.

Definition at line 117 of file ViewController.hpp.

Referenced by handleEvent(), and windowMousePosition().

3.12.4.14 pixelsPerMeter

```
float ViewController::pixelsPerMeter  [private]
```

Conversion factor from meters to pixels.

Definition at line 120 of file ViewController.hpp.

Referenced by drawAxes(), drawGrid(), getHoveredCell(), and getPixelsPerMeter().

3.12.4.15 view

```
sf::View& ViewController::view  [private]
```

Reference to the controlled view.

Definition at line 107 of file ViewController.hpp.

Referenced by applyView(), drawAxes(), drawGrid(), getView(), getZoom(), handleEvent(), reset(), ViewController(), and zoomController().

3.12.4.16 window

```
sf::RenderWindow& ViewController::window  [private]
```

Reference to the SFML window.

Definition at line 106 of file ViewController.hpp.

Referenced by applyView(), drawAxes(), handleEvent(), and ViewController().

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

- include/artgslam_vsc/ViewController.hpp
- src/ViewController.cpp

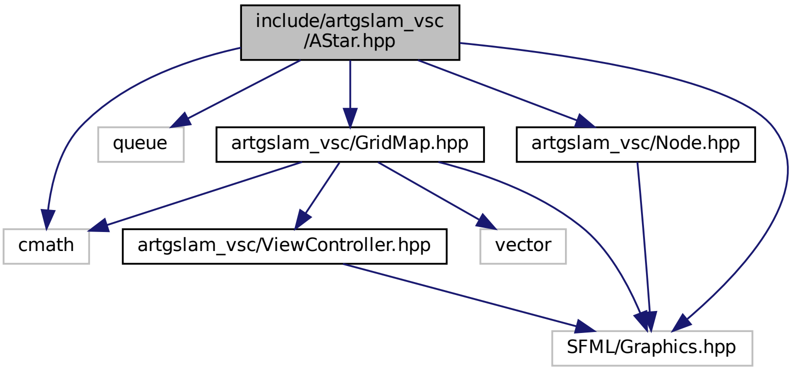
Chapter 4

File Documentation

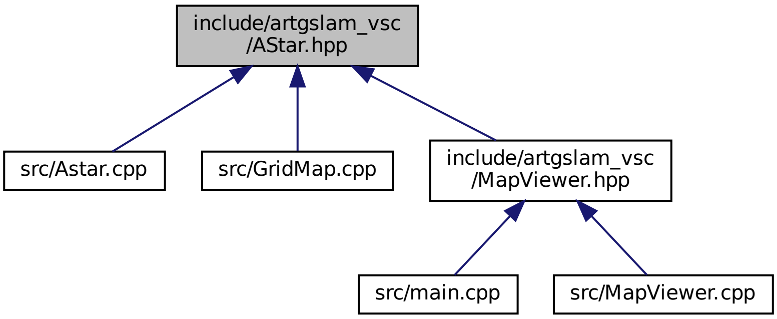
4.1 include/artgslam_vsc/AStar.hpp File Reference

```
#include <cmath>
#include <queue>
#include <SFML/Graphics.hpp>
#include "artgslam_vsc/GridMap.hpp"
#include "artgslam_vsc/Node.hpp"
```

Include dependency graph for AStar.hpp:



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



Classes

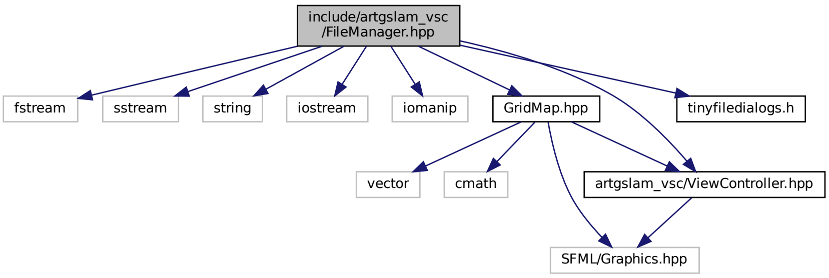
- class AStar

  This class handles the A∗ algorithm for path planning. It receives start and goal coordinates from the GridMap, performs the algorithm step-by-step, and supports animation and path retrieval.

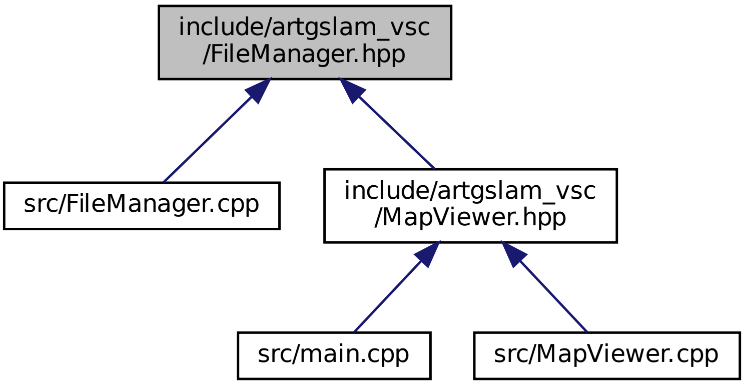
4.2 include/artgslam_vsc/FileManager.hpp File Reference

```
#include <fstream>
#include <sstream>
#include <string>
#include <iostream>
#include <iomanip>
#include "GridMap.hpp"
#include "ViewController.hpp"
#include "tinyfiledialogs.h"
```

Include dependency graph for FileManager.hpp:



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



Classes

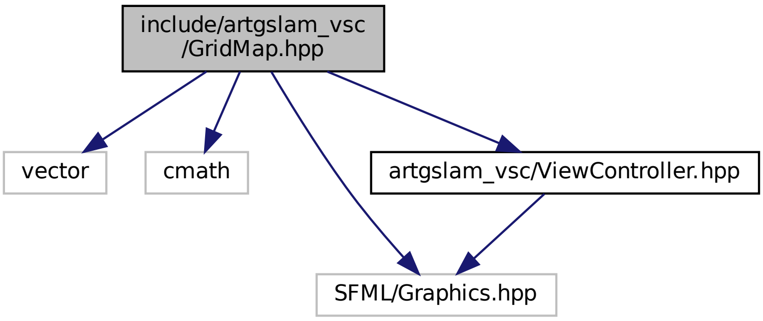
- class FileManager

  Manages file input/output operations for grid map visualization.

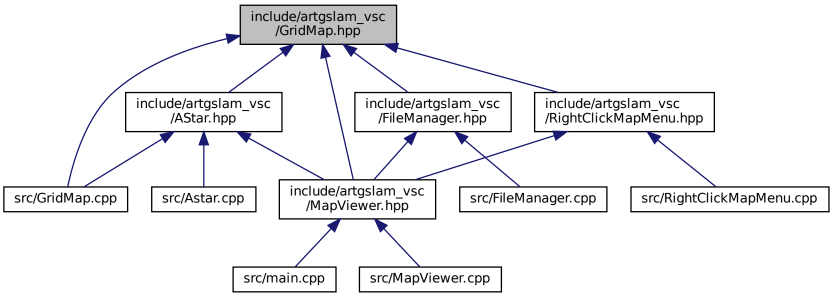
4.3 include/artgslam_vsc/GridMap.hpp File Reference

```
#include <vector>
#include <cmath>
#include <SFML/Graphics.hpp>
#include "artgslam_vsc/ViewController.hpp"
```

Include dependency graph for GridMap.hpp:



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



Classes

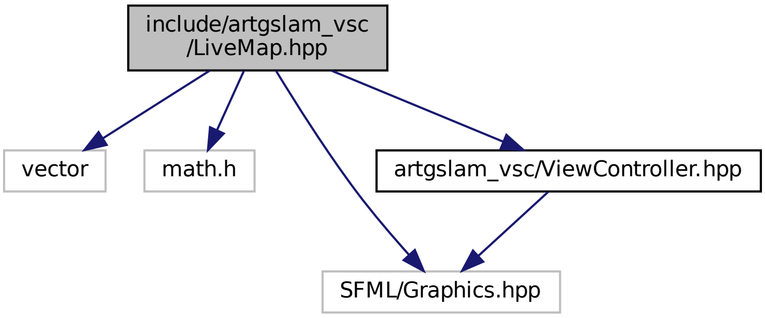
- class GridMap

  Handles the creation and management of map data.

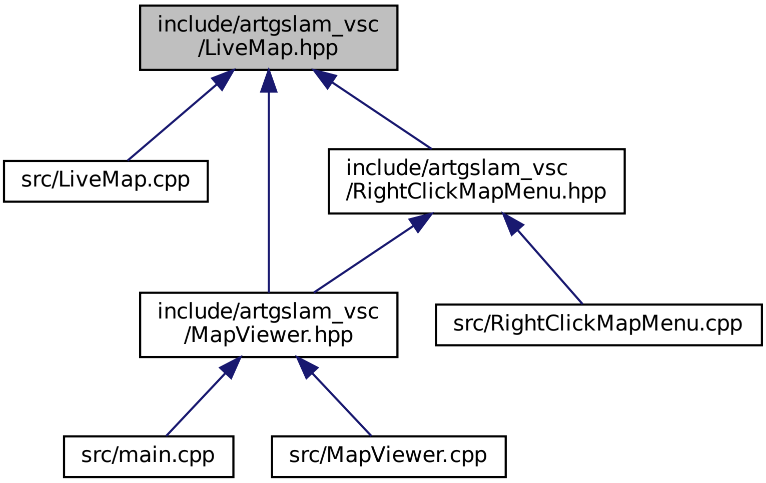
4.4 include/artgslam_vsc/LiveMap.hpp File Reference

```
#include <vector>
#include <math.h>
#include <SFML/Graphics.hpp>
#include "artgslam_vsc/ViewController.hpp"
```

Include dependency graph for LiveMap.hpp:



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



Classes

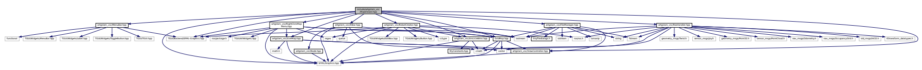
- class LiveMap

  This class manages live map creation in "live mode", by dynamically updating the map from incoming ROS data (e.g., from RosHandler). It builds an occupancy grid in real time and draws it using a ViewController.

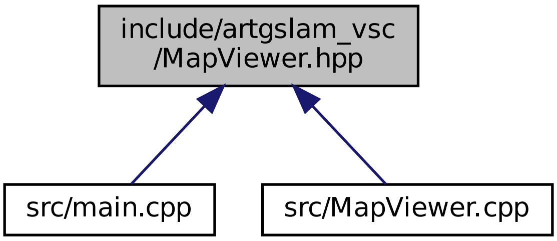
4.5 include/artgslam_vsc/MapViewer.hpp File Reference

```
#include "artgslam_vsc/MenuBar.hpp"
#include "artgslam_vsc/FileManager.hpp"
#include "artgslam_vsc/GridMap.hpp"
#include "artgslam_vsc/RosHandler.hpp"
#include "artgslam_vsc/ViewController.hpp"
#include "artgslam_vsc/RobotCreator.hpp"
#include "artgslam_vsc/UnicicleWmr.hpp"
#include "artgslam_vsc/LiveMap.hpp"
#include "artgslam_vsc/RightClickMapMenu.hpp"
#include "artgslam_vsc/AStar.hpp"
#include <SFML/Graphics.hpp>
#include <TGUI/Backend/SFML-Graphics.hpp>
```

Include dependency graph for MapViewer.hpp:



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



Classes

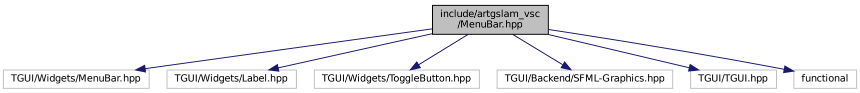
- class MapViewer

  This is the main class. It manages mouse/keyboard events and GUI integration. It also coordinates the rendering and simulation components of the application.

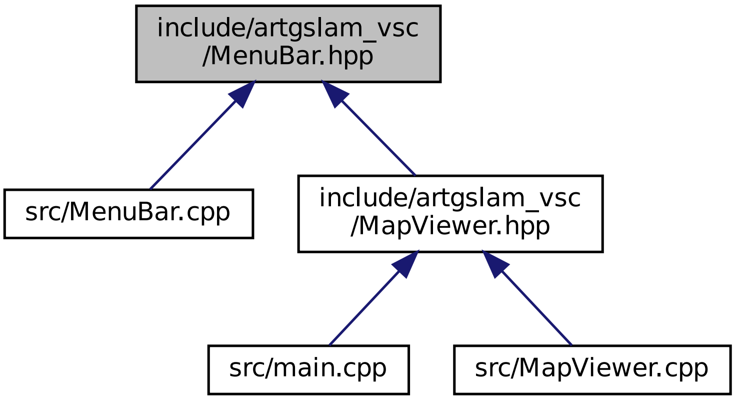
4.6 include/artgslam_vsc/MenuBar.hpp File Reference

```
#include <TGUI/Widgets/MenuBar.hpp>
#include <TGUI/Widgets/Label.hpp>
#include <TGUI/Widgets/ToggleButton.hpp>
#include <TGUI/Backend/SFML-Graphics.hpp>
#include <TGUI/TGUI.hpp>
#include <functional>
```

Include dependency graph for MenuBar.hpp:



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



Classes

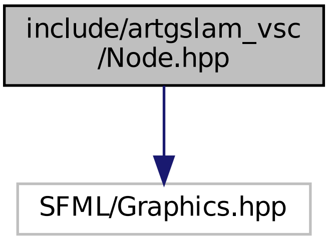
- class MenuBar

  Manages the menu bar located at the bottom of the screen. Provides options for file handling, view controls, robot creation, and live mode toggle.

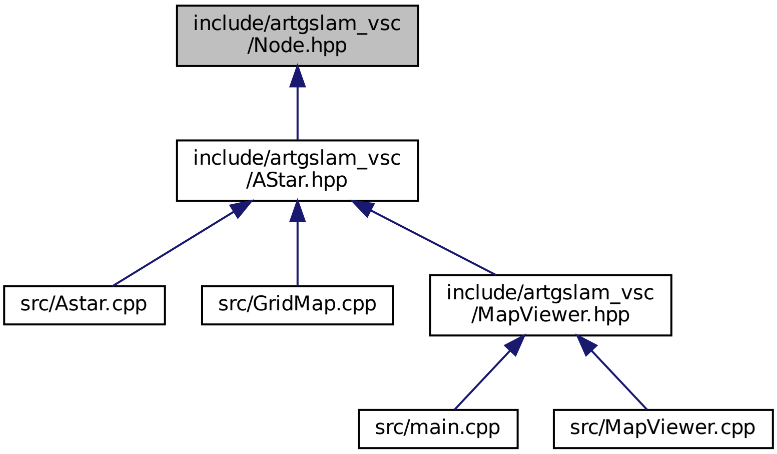
4.7 include/artgslam_vsc/Node.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Include dependency graph for Node.hpp:



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



Classes

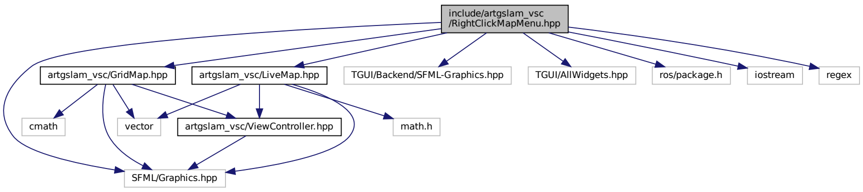
- class Node

  Represents a node in a grid used for path planning algorithms (e.g., A∗). Each node corresponds to a cell and holds all relevant data for cost calculation and path reconstruction.

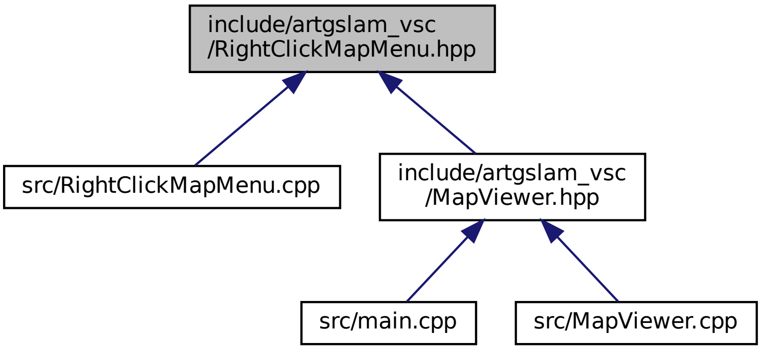
4.8 include/artgslam_vsc/RightClickMapMenu.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include <TGUI/Backend/SFML-Graphics.hpp>
#include <TGUI/AllWidgets.hpp>
#include <ros/package.h>
#include <iostream>
#include <regex>
#include "artgslam_vsc/GridMap.hpp"
#include "artgslam_vsc/LiveMap.hpp"
```

Include dependency graph for RightClickMapMenu.hpp:



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



Classes

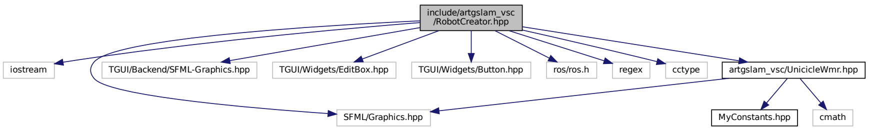
- class RightClickMapMenu

  Manages a contextual right-click menu displayed on top of a map. Allows users to set the start and goal positions or clear the grid through TGUI buttons.

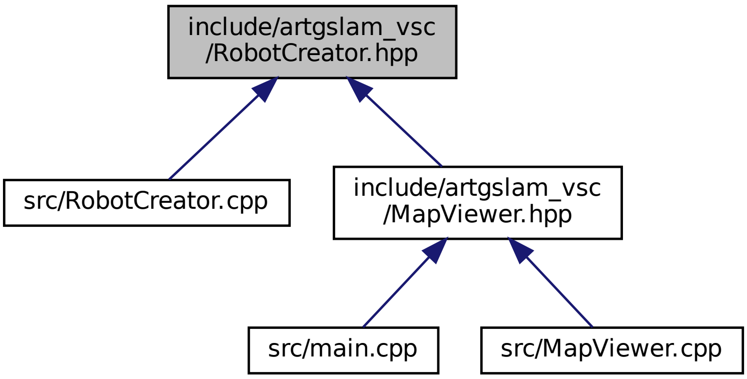
4.9 include/artgslam_vsc/RobotCreator.hpp File Reference

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include <TGUI/Backend/SFML-Graphics.hpp>
#include <TGUI/Widgets/EditBox.hpp>
#include <TGUI/Widgets/Button.hpp>
#include <ros/ros.h>
#include <regex>
#include <cctype>
#include "artgslam_vsc/UnicicleWmr.hpp"
```

Include dependency graph for RobotCreator.hpp:



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



Classes

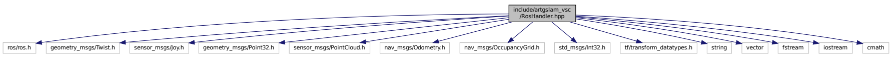
- class RobotCreator

  Manages a GUI window for creating and configuring a Unicycle Wheeled Mobile Robot (WMR). Allows user input for robot parameters like width, height, and color, then applies these to the robot model.

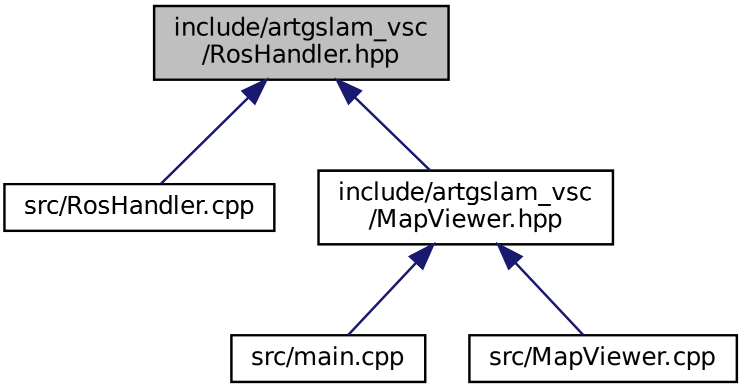
4.10 include/artgslam_vsc/RosHandler.hpp File Reference

```
#include <ros/ros.h>
#include <geometry_msgs/Twist.h>
#include <sensor_msgs/Joy.h>
#include <geometry_msgs/Point32.h>
#include <sensor_msgs/PointCloud.h>
#include <nav_msgs/Odometry.h>
#include <nav_msgs/OccupancyGrid.h>
#include <std_msgs/Int32.h>
#include <tf/transform_datatypes.h>
#include <string>
#include <vector>
#include <fstream>
#include <iostream>
#include <cmath>
```

Include dependency graph for RosHandler.hpp:



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



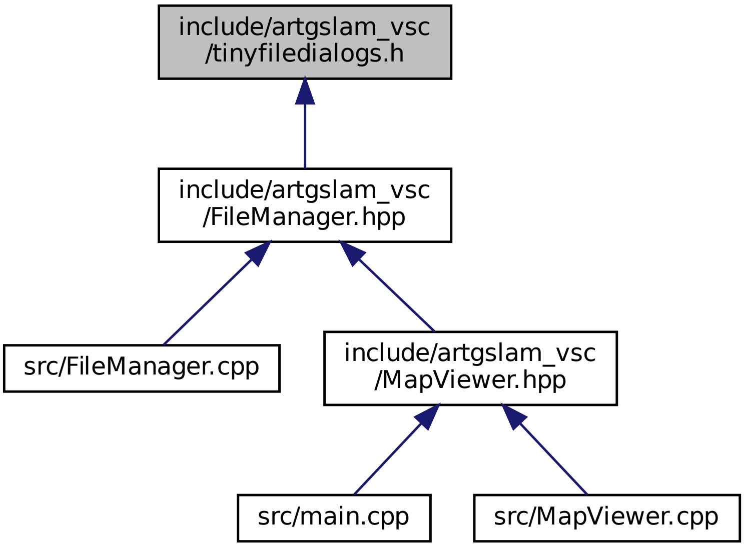
Classes

- class RosHandler

  Handles ROS communication:

4.11 include/artgslam_vsc/tinyfiledialogs.h File Reference

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



Functions

- const char ∗ tinyfd_getGlobalChar (char const ∗aCharVariableName)
- int tinyfd_getGlobalInt (char const ∗aIntVariableName)
- int tinyfd_setGlobalInt (char const ∗aIntVariableName, int aValue)
- void tinyfd_beep (void)
- int tinyfd_notifyPopup (char const ∗aTitle, char const ∗aMessage, char const ∗aIconType)
- int tinyfd_messageBox (char const ∗aTitle, char const ∗aMessage, char const ∗aDialogType, char const ∗a←IconType, int aDefaultButton)
- char ∗ tinyfd_inputBox (char const ∗aTitle, char const ∗aMessage, char const ∗aDefaultInput)
- char ∗ tinyfd_saveFileDialog (char const ∗aTitle, char const ∗aDefaultPathAndOrFile, int aNumOfFilter←Patterns, char const ∗const ∗aFilterPatterns, char const ∗aSingleFilterDescription)
- char ∗ tinyfd_openFileDialog (char const ∗aTitle, char const ∗aDefaultPathAndOrFile, int aNumOfFilter←Patterns, char const ∗const ∗aFilterPatterns, char const ∗aSingleFilterDescription, int aAllowMultipleSelects)
- char ∗ tinyfd_selectFolderDialog (char const ∗aTitle, char const ∗aDefaultPath)
- char ∗ tinyfd_colorChooser (char const ∗aTitle, char const ∗aDefaultHexRGB, unsigned char const aDefault←RGB[3], unsigned char aoResultRGB[3])

Variables

- char tinyfd_version [8]
- char tinyfd_needs [ ]
- int tinyfd_verbose
- int tinyfd_silent
- int tinyfd_allowCursesDialogs

  Curses dialogs are difficult to use and counter-intuitive.
- int tinyfd_forceConsole
- char tinyfd_response [1024]

4.11.1 Function Documentation

4.11.1.1 tinyfd_beep()

```
void tinyfd_beep (
            void  )
```

4.11.1.2 tinyfd_colorChooser()

```
char* tinyfd_colorChooser (
            char const * aTitle,
            char const * aDefaultHexRGB,
            unsigned char const aDefaultRGB[3],
            unsigned char aoResultRGB[3] )
```

4.11.1.3 tinyfd_getGlobalChar()

```
const char* tinyfd_getGlobalChar (
             char const * aCharVariableName )
```

4.11.1.4 tinyfd_getGlobalInt()

```
int tinyfd_getGlobalInt (
             char const * aIntVariableName )
```

4.11.1.5 tinyfd_inputBox()

```
char* tinyfd_inputBox (
             char const * aTitle,
             char const * aMessage,
             char const * aDefaultInput )
```

4.11.1.6 tinyfd_messageBox()

```
int tinyfd_messageBox (
             char const * aTitle,
             char const * aMessage,
             char const * aDialogType,
             char const * aIconType,
             int aDefaultButton )
```

4.11.1.7 tinyfd_notifyPopup()

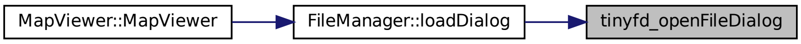
```
int tinyfd_notifyPopup (
             char const * aTitle,
             char const * aMessage,
             char const * aIconType )
```

4.11.1.8 tinyfd_openFileDialog()

```
char* tinyfd_openFileDialog (
            char const * aTitle,
            char const * aDefaultPathAndOrFile,
            int aNumOfFilterPatterns,
            char const *const * aFilterPatterns,
            char const * aSingleFilterDescription,
            int aAllowMultipleSelects )
```

Referenced by FileManager::loadDialog().

Here is the caller graph for this function:

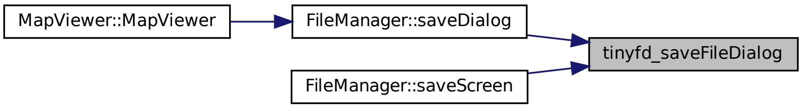


4.11.1.9 tinyfd_saveFileDialog()

```
char* tinyfd_saveFileDialog (
            char const * aTitle,
            char const * aDefaultPathAndOrFile,
            int aNumOfFilterPatterns,
            char const *const * aFilterPatterns,
            char const * aSingleFilterDescription )
```

Referenced by FileManager::saveDialog(), and FileManager::saveScreen().

Here is the caller graph for this function:



4.11.1.10 tinyfd_selectFolderDialog()

```
char* tinyfd_selectFolderDialog (
            char const * aTitle,
            char const * aDefaultPath )
```

4.11.1.11 tinyfd_setGlobalInt()

```
int tinyfd_setGlobalInt (
             char const * aIntVariableName,
             int aValue )
```

4.11.2 Variable Documentation

4.11.2.1 tinyfd_allowCursesDialogs

```
int tinyfd_allowCursesDialogs
```

Curses dialogs are difficult to use and counter-intuitive.

On windows they are only ascii and still uses the unix backslash !

4.11.2.2 tinyfd_forceConsole

```
int tinyfd_forceConsole
```

4.11.2.3 tinyfd_needs

```
char tinyfd_needs[]
```

4.11.2.4 tinyfd_response

```
char tinyfd_response[1024]
```

4.11.2.5 tinyfd_silent

```
int tinyfd_silent
```

4.11.2.6 tinyfd_verbose

```
int tinyfd_verbose
```

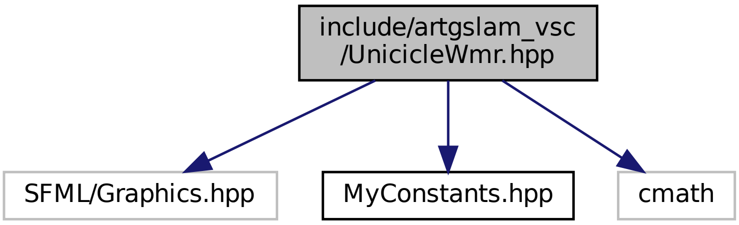
4.11.2.7 tinyfd_version

```
char tinyfd_version[8]
```

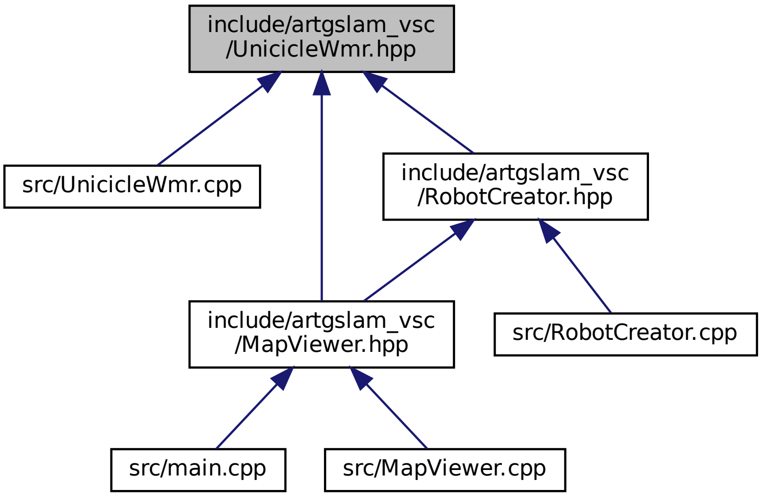
4.12 include/artgslam_vsc/UnicicleWmr.hpp File Reference

```
#include <SFML/Graphics.hpp>
#include "MyConstants.hpp"
#include <cmath>
```

Include dependency graph for UnicicleWmr.hpp:



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



Classes

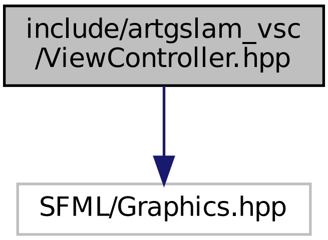
- class UnicicleWmr

  Simulates a unicycle model robot with position, velocity, and rendering support.

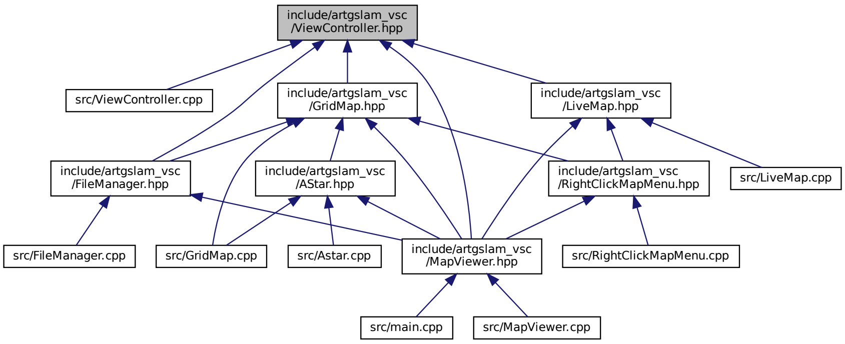
4.13 include/artgslam_vsc/ViewController.hpp File Reference

```
#include <SFML/Graphics.hpp>
```

Include dependency graph for ViewController.hpp:



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



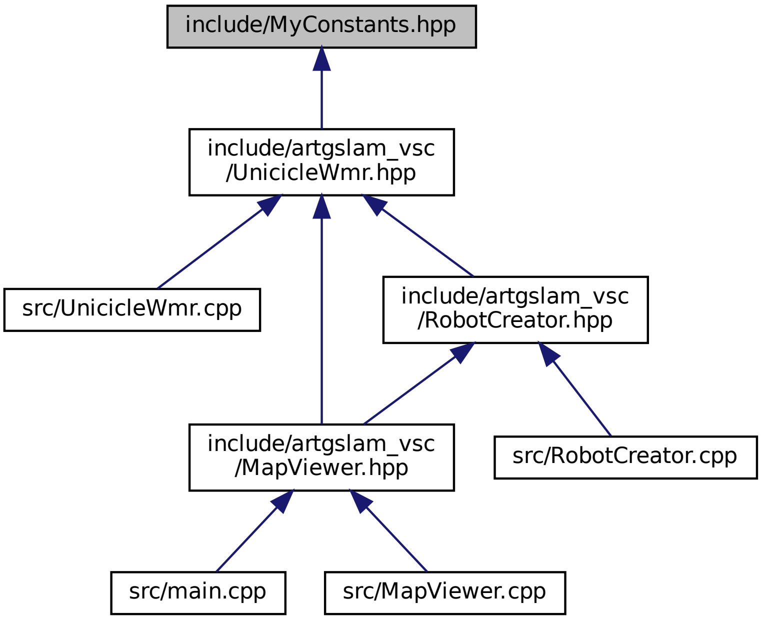
Classes

- class ViewController

  Controls view, zoom, and panning.

4.14 include/MyConstants.hpp File Reference

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



Variables

- const double PI = 3.14159265358979323846

4.14.1 Variable Documentation

4.14.1.1 PI

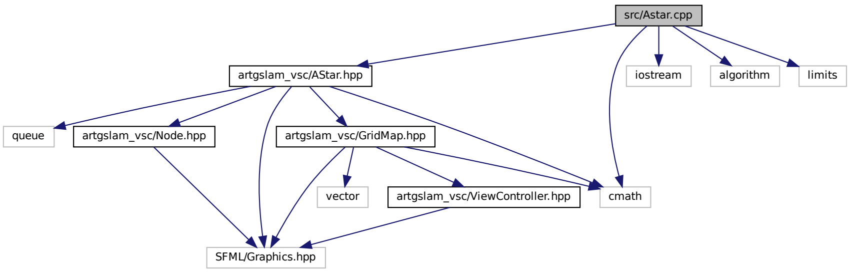
```
const double PI = 3.14159265358979323846
```

Definition at line 2 of file MyConstants.hpp.

4.15 src/Astar.cpp File Reference

```
#include "artgslam_vsc/AStar.hpp"
#include <iostream>
#include <cmath>
#include <algorithm>
#include <limits>
```

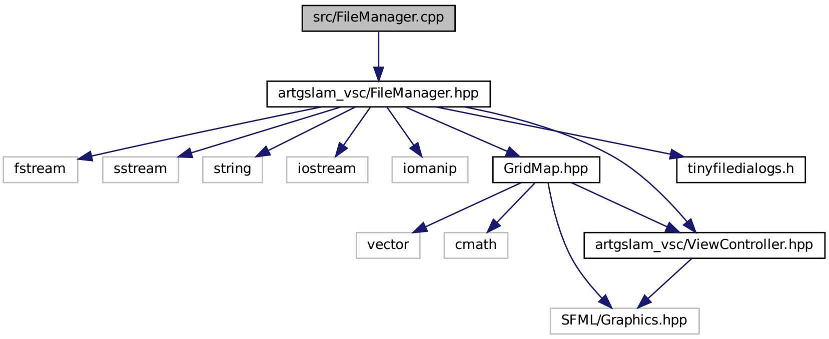
Include dependency graph for Astar.cpp:



4.16 src/FileManager.cpp File Reference

```
#include "artgslam_vsc/FileManager.hpp"
```

Include dependency graph for FileManager.cpp:

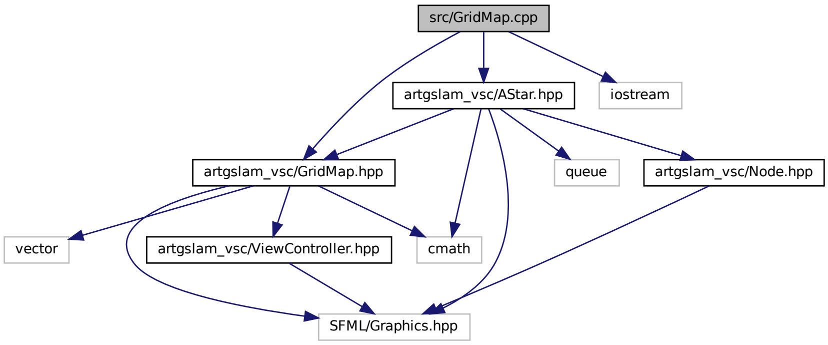


4.17 src/GridMap.cpp File Reference

```
#include "artgslam_vsc/GridMap.hpp"
#include "artgslam_vsc/AStar.hpp"
```

```
#include <iostream>
```

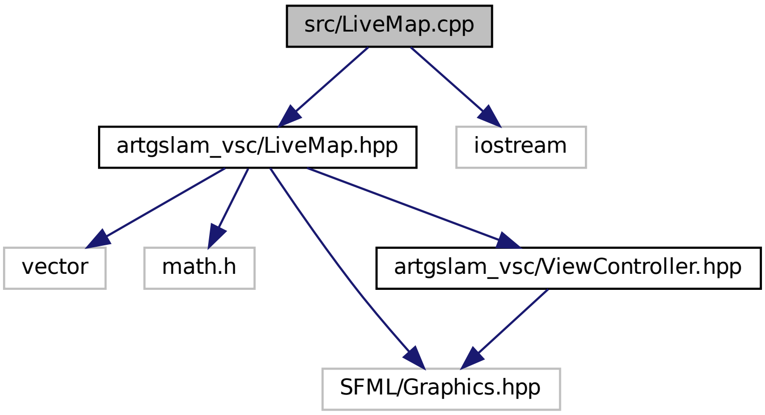
Include dependency graph for GridMap.cpp:



4.18 src/LiveMap.cpp File Reference

```
#include <artgslam_vsc/LiveMap.hpp>
#include <iostream>
```

Include dependency graph for LiveMap.cpp:



4.19 src/main.cpp File Reference

```
#include <ros/ros.h>
#include <SFML/Graphics.hpp>
#include "artgslam_vsc/MapViewer.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main (int argc, char ∗∗argv)

  Main entry point for the ARTG SLAM visualization node.

4.19.1 Function Documentation

4.19.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Main entry point for the ARTG SLAM visualization node.

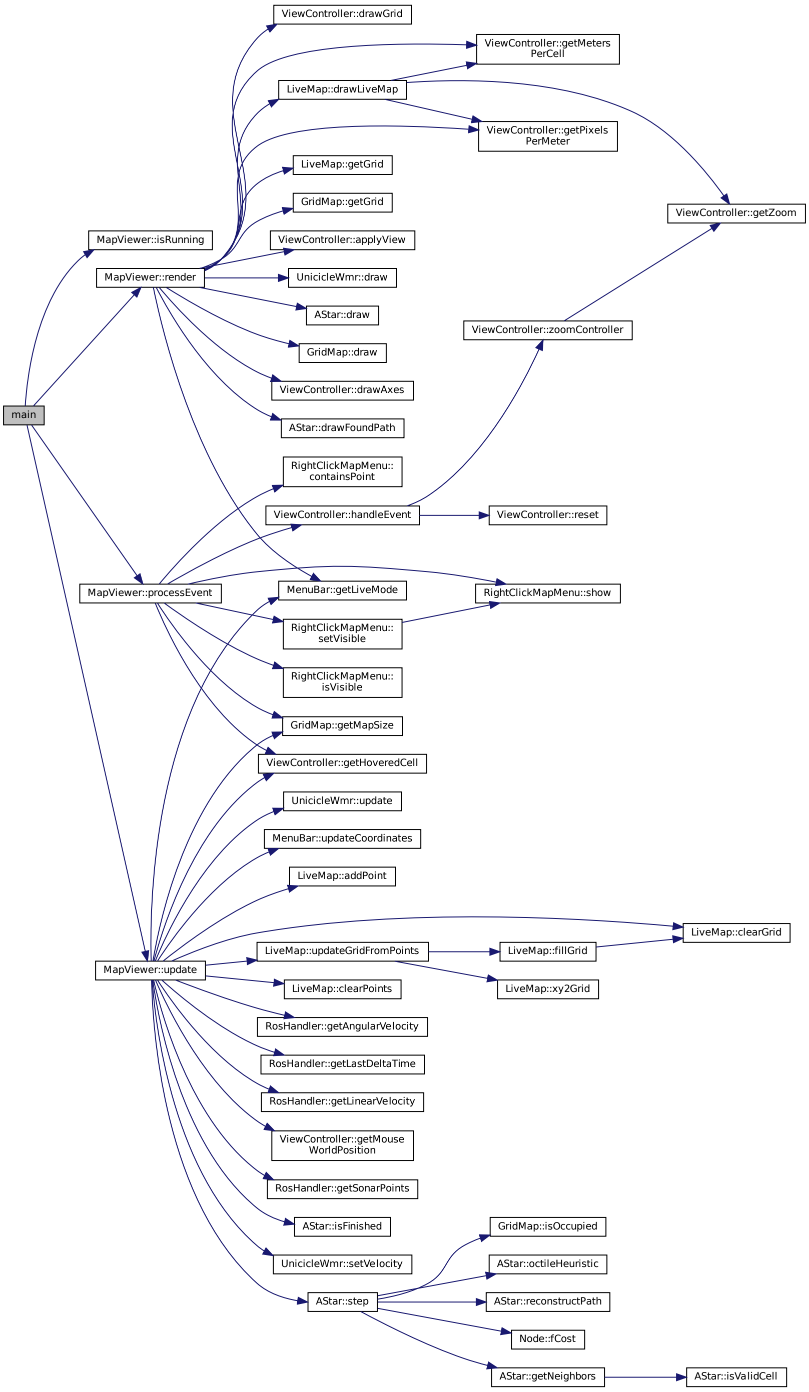
Initializes ROS and SFML window, then runs the visualization loop.

Definition at line 9 of file main.cpp.

```
{
    // Initialize ROS node with a descriptive name
    ros::init(argc, argv, "artgslam_vsc_node");

    // Create an SFML window for rendering the visualizer
    sf::RenderWindow window(sf::VideoMode(1024, 768), "ARTG SLAM Visualizer");
    window.setFramerateLimit(60);  // Cap frame rate at 60 FPS for smooth rendering

    // Instantiate MapViewer, which handles:
    //  - Map rendering and overlays
    //  - Simulation and live data updates
    //  - ROS topic subscriptions and publishing
    //  - User input and camera control
    MapViewer mapViewer(window);

    // Main loop: run while ROS is active and window is open
    while (ros::ok() && mapViewer.isRunning())
    {
        // Process user events (keyboard, mouse, window events)
        mapViewer.processEvent();

        // Update application state (map updates, robot pose, live mode, etc.)
        mapViewer.update();

        // Render the current frame to the window
        mapViewer.render();

        // Handle incoming ROS messages and callbacks
        ros::spinOnce();
    }

    return 0;
}
```

References MapViewer::isRunning(), MapViewer::processEvent(), MapViewer::render(), and MapViewer::update().

Here is the call graph for this function:

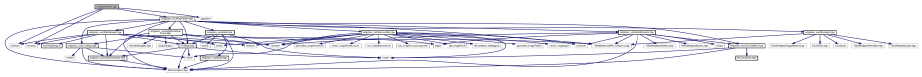


4.20 src/MapViewer.cpp File Reference

```
#include "artgslam_vsc/MapViewer.hpp"
#include <sstream>
#include <iomanip>
```

```
#include <algorithm>
```

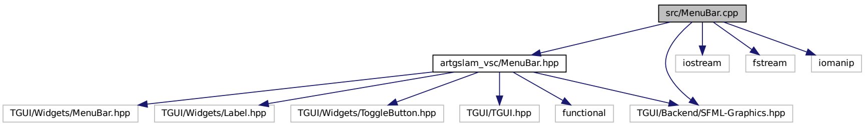
Include dependency graph for MapViewer.cpp:



4.21 src/MenuBar.cpp File Reference

```
#include "artgslam_vsc/MenuBar.hpp"
#include <TGUI/Backend/SFML-Graphics.hpp>
#include <iostream>
#include <fstream>
#include <iomanip>
```

Include dependency graph for MenuBar.cpp:

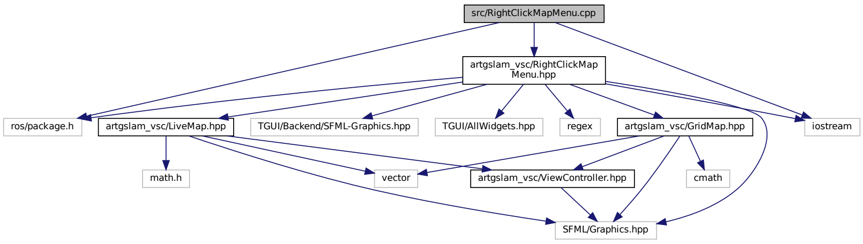


4.22 src/MyConstants.cpp File Reference

4.23 src/RightClickMapMenu.cpp File Reference

```
#include "artgslam_vsc/RightClickMapMenu.hpp"
#include <ros/package.h>
#include <iostream>
```

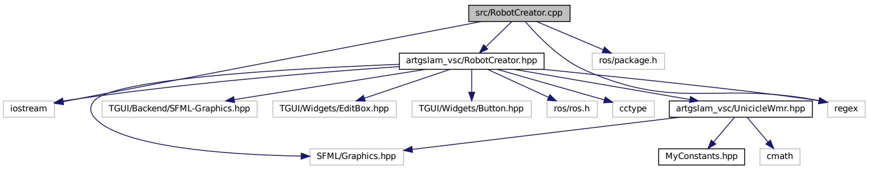
Include dependency graph for RightClickMapMenu.cpp:



4.24 src/RobotCreator.cpp File Reference

```
#include "artgslam_vsc/RobotCreator.hpp"
#include <ros/package.h>
#include <iostream>
#include <regex>
```

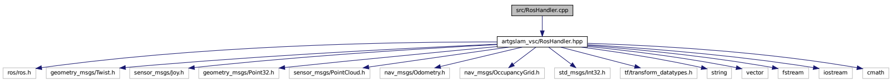
Include dependency graph for RobotCreator.cpp:



4.25 src/RosHandler.cpp File Reference

```
#include "artgslam_vsc/RosHandler.hpp"
```

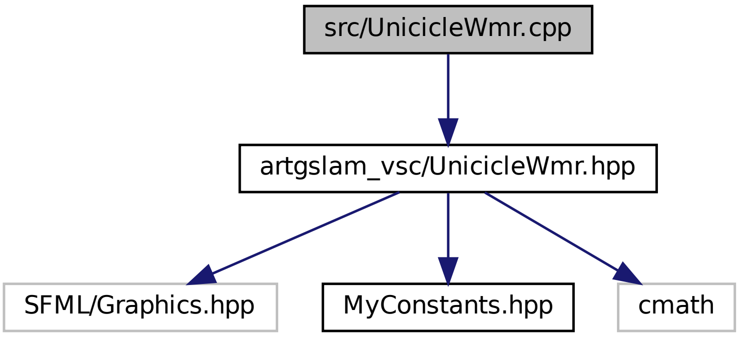
Include dependency graph for RosHandler.cpp:



4.26 src/UnicicleWmr.cpp File Reference

```
#include "artgslam_vsc/UnicicleWmr.hpp"
```

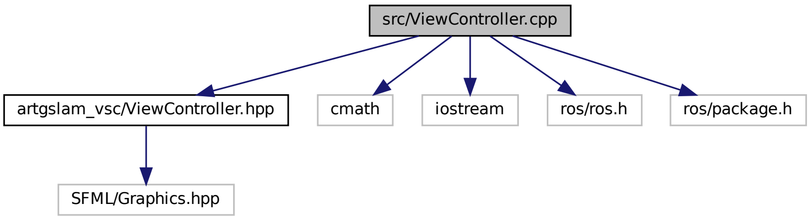
Include dependency graph for UnicicleWmr.cpp:



4.27 src/ViewController.cpp File Reference

```
#include "artgslam_vsc/ViewController.hpp"
#include <cmath>
#include <iostream>
#include <ros/ros.h>
#include <ros/package.h>
```

Include dependency graph for ViewController.cpp:


src/ViewController.cpp
artgslam_vsc/ViewController.hpp
cmath
iostream
ros/ros.h
ros/package.h
SFML/Graphics.hpp