Costmap_2d API documentation

⇒ There are two main ways of using this package:

- → Create a Costmap2D object and manage updating it yourself.
- → Use ROS wrapper (Costmap2DROS) for the costmap that manages the map for you, but allows you to get a copy of the underlying Costmap2D object at any time.

★ Costmap2D

① Constructors

```
Costmap2D();
```

{Default Constructor}

```
Costmap2D(const Costmap2D& map);
```

{Copy Constructor}

```
Costmap2D(unsigned int cells_size_x, unsigned int cells_size_y, double resolution,
          double origin_x, double origin_y, unsigned char default_value = 0);
```

{Manually initialized Constructor}

② Copy functions

```
Costmap2D& operator=(const Costmap2D& map);
```

{Copy using '=' operator}

```
bool copyCostmapWindow(const Costmap2D& map, double win_origin_x, double win_origin_y, double win_size_x,
                       double win_size_y);
```

{Copy in a window}

③ Other functions

```
unsigned char getCost(unsigned int mx, unsigned int my) const;
```

{Get cost at particular pixel location in map}

```
void setCost(unsigned int mx, unsigned int my, unsigned char cost);
```

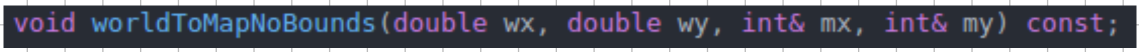
{Set cost at particular pixel location in map}

```
void mapToWorld(unsigned int mx, unsigned int my, double& wx, double& wy) const;
```

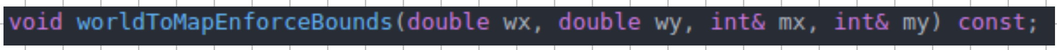
{Map coordinate to world coordinate frame conversion}

```
bool worldToMap(double wx, double wy, unsigned int& mx, unsigned int& my) const;
```

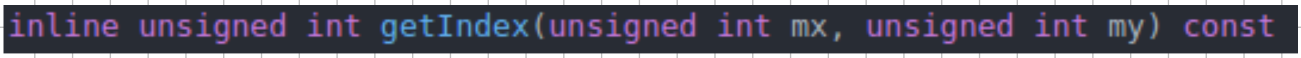
{World coordinate to map coordinate frame conversion



```
void worldToMapNoBounds(double wx, double wy, int& mx, int& my) const;
```

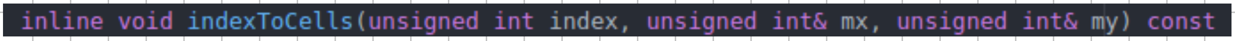


```
void worldToMapEnforceBounds(double wx, double wy, int& mx, int& my) const;
```



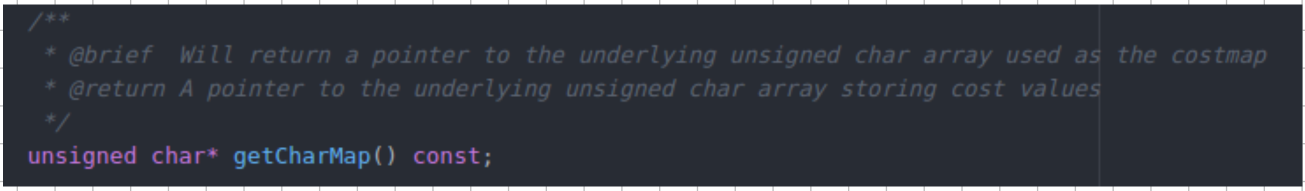
```
inline unsigned int getIndex(unsigned int mx, unsigned int my) const
```

{Given map coordinates compute the associated index}

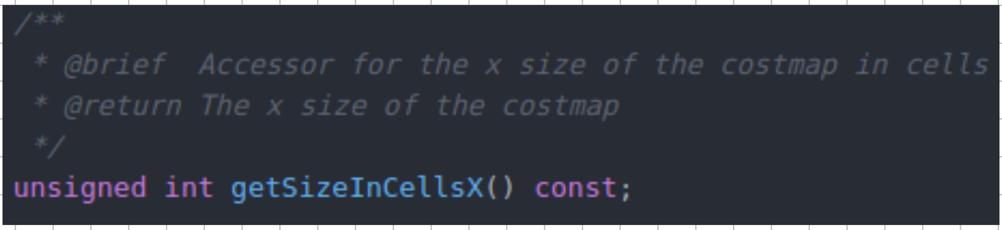


```
inline void indexToCells(unsigned int index, unsigned int& mx, unsigned int& my) const
```

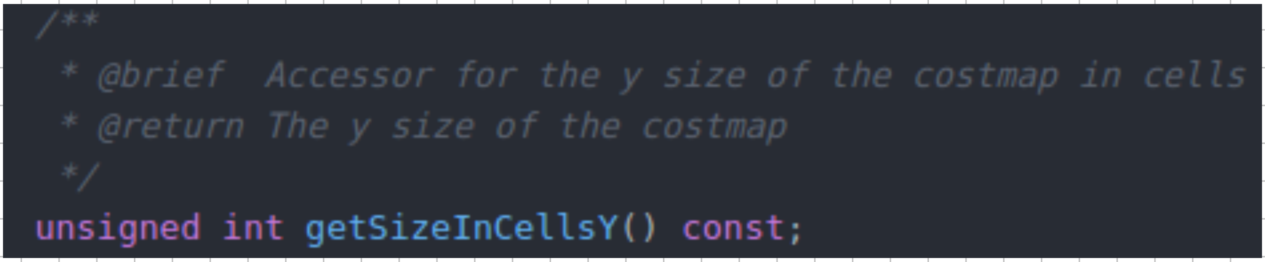
{Given the index compute associated index}



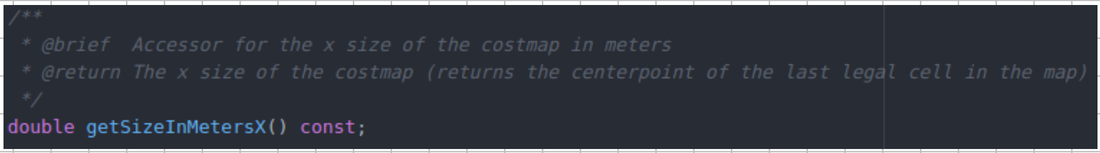
```
/**
 * @brief  Will return a pointer to the underlying unsigned char array used as the costmap
 * @return A pointer to the underlying unsigned char array storing cost values
 */
unsigned char* getCharMap() const;
```



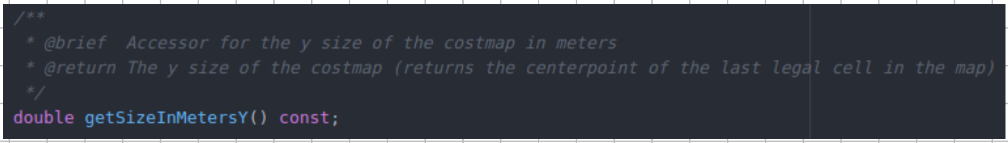
```
/**
 * @brief  Accessor for the x size of the costmap in cells
 * @return The x size of the costmap
 */
unsigned int getSizeInCellsX() const;
```



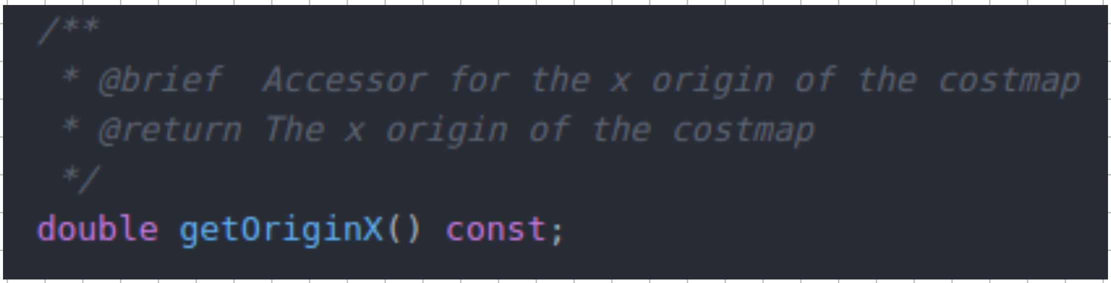
```
/**
 * @brief  Accessor for the y size of the costmap in cells
 * @return The y size of the costmap
 */
unsigned int getSizeInCellsY() const;
```



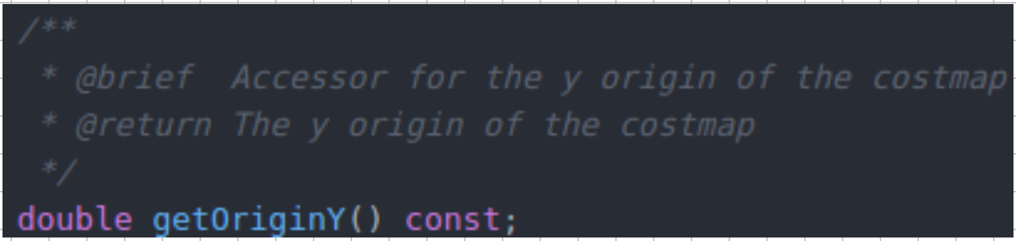
```
/**
 * @brief  Accessor for the x size of the costmap in meters
 * @return The x size of the costmap (returns the centerpoint of the last legal cell in the map)
 */
double getSizeInMetersX() const;
```



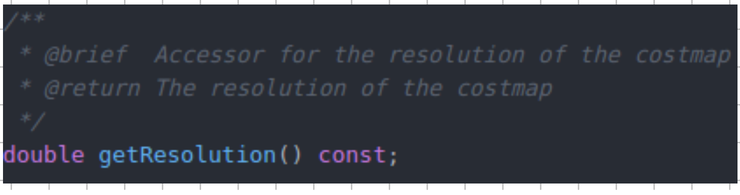
```
/**
 * @brief  Accessor for the y size of the costmap in meters
 * @return The y size of the costmap (returns the centerpoint of the last legal cell in the map)
 */
double getSizeInMetersY() const;
```



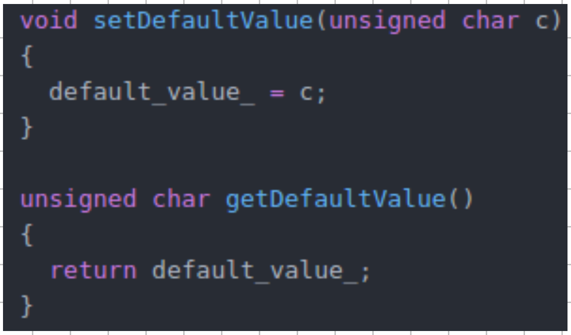
```
/**
 * @brief  Accessor for the x origin of the costmap
 * @return The x origin of the costmap
 */
double getOriginX() const;
```



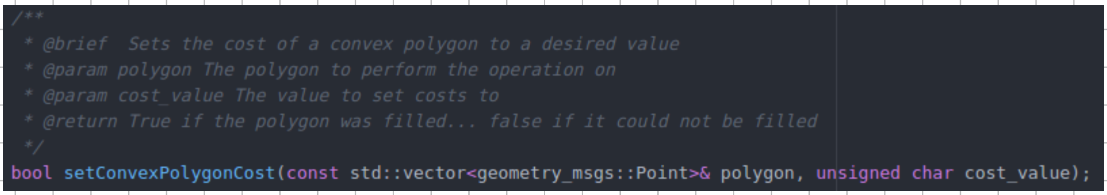
```
/**
 * @brief  Accessor for the y origin of the costmap
 * @return The y origin of the costmap
 */
double getOriginY() const;
```



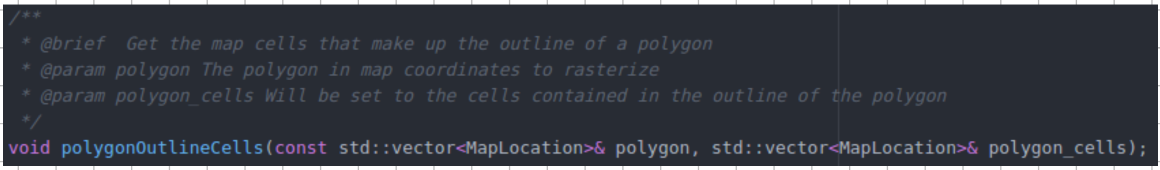
```
/**
 * @brief  Accessor for the resolution of the costmap
 * @return The resolution of the costmap
 */
double getResolution() const;
```



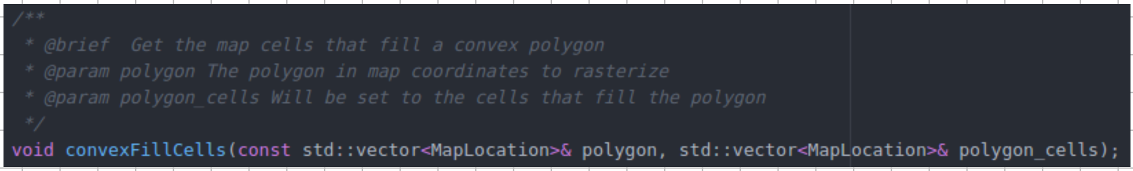
```cpp
void setDefaultValue(unsigned char c)
{
  default_value_ = c;
}

unsigned char getDefaultValue()
{
  return default_value_;
}
```



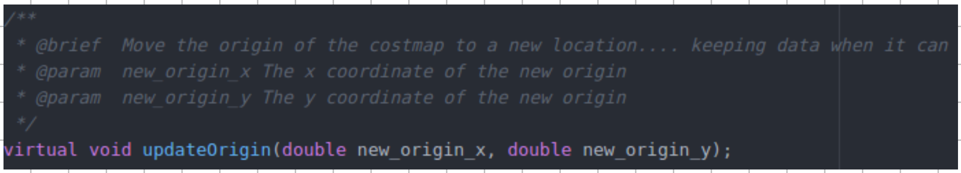
```cpp
/**
 * @brief  Sets the cost of a convex polygon to a desired value
 * @param polygon The polygon to perform the operation on
 * @param cost_value The value to set costs to
 * @return True if the polygon was filled... false if it could not be filled
 */
bool setConvexPolygonCost(const std::vector<geometry_msgs::Point>& polygon, unsigned char cost_value);
```



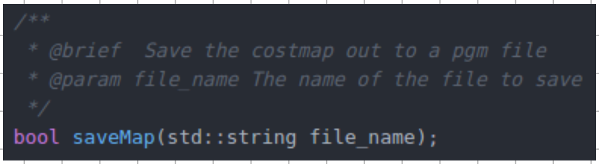
```cpp
/**
 * @brief  Get the map cells that make up the outline of a polygon
 * @param polygon The polygon in map coordinates to rasterize
 * @param polygon_cells Will be set to the cells contained in the outline of the polygon
 */
void polygonOutlineCells(const std::vector<MapLocation>& polygon, std::vector<MapLocation>& polygon_cells);
```



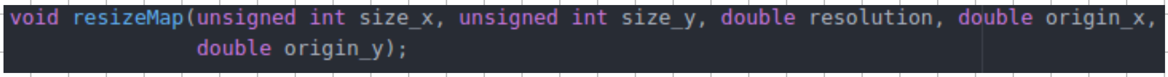
```cpp
/**
 * @brief  Get the map cells that fill a convex polygon
 * @param polygon The polygon in map coordinates to rasterize
 * @param polygon_cells Will be set to the cells that fill the polygon
 */
void convexFillCells(const std::vector<MapLocation>& polygon, std::vector<MapLocation>& polygon_cells);
```



```cpp
/**
 * @brief  Move the origin of the costmap to a new location.... keeping data when it can
 * @param  new_origin_x The x coordinate of the new origin
 * @param  new_origin_y The y coordinate of the new origin
 */
virtual void updateOrigin(double new_origin_x, double new_origin_y);
```

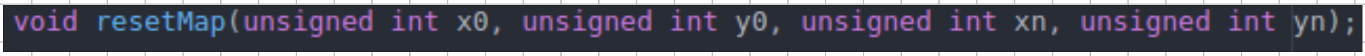


```cpp
/**
 * @brief  Save the costmap out to a pgm file
 * @param file_name The name of the file to save
 */
bool saveMap(std::string file_name);
```



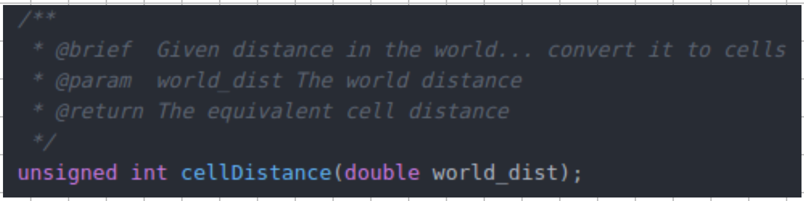
```cpp
void resizeMap(unsigned int size_x, unsigned int size_y, double resolution, double origin_x,
               double origin_y);
```

{To resize the map}



```cpp
void resetMap(unsigned int x0, unsigned int y0, unsigned int xn, unsigned int yn);
```

{To reset a position of map
to default value}



```cpp
/**
 * @brief  Given distance in the world... convert it to cells
 * @param  world_dist The world distance
 * @return The equivalent cell distance
 */
unsigned int cellDistance(double world_dist);
```