# Handshaking

Generated by Doxygen 1.8.12

# Contents

# Chapter 1

# README

IMPORTANT: The project is correctly set up in the XCode project but if for some reason you can't use XCode (like not having a Mac...), you will need to follow these instructions (until I write the appropriate makefile...): To compile the C++ files, you need to add the preprocessor macros: NON_MATLAB_PARSING=1 and MAX_EXT_API_CO↩ NNECTIONS=255; You also need to compile extApi.c and extApiPlatform.

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

# Class Documentation

## 3.1 Robot_VREP Class Reference

### Public Member Functions

- Robot_VREP ()
- Robot_VREP (const int clientId, std::vector< std::string > jointNames)
- ~Robot_VREP ()
- double getAngularPosition (const unsigned int jointID)
- double getAngularVelocity (const unsigned int jointID)
- double getAngularForce (const unsigned int jointID)
- bool setPosition (const unsigned int jointID, const double pos)
- bool setVelocity (const unsigned int jointID, const double speed)
- bool setForce (const unsigned int jointID, const double force)
- bool updateAngularPosition (const unsigned int index=-1)
- bool updateAngularVelocity (const unsigned int index=-1)
- bool updateAngularForce (const unsigned int index=-1)
- bool lockJoint (const int jointHandle)
- void startClock ()
- void updateClock ()
- double getMicroseconds ()
- double getSeconds ()

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 Robot_VREP::Robot_VREP ( )

Robot_VREP Constructor

**See also**

Robot_VREP(const int clientId, std::vector<std::string> jointNames)

#### 3.1.1.2 Robot_VREP::Robot_VREP ( const int *clientId,* std::vector< std::string > *jointNames* )

Robot_VREP Constructor

**Parameters**

| | |
|---|---|
| *clientId* | Id of the VREP client |
| *jointNames* | list of the joint names of the robot |

**See also**

Robot_VREP()

#### 3.1.1.3 Robot_VREP::~Robot_VREP ( )

Robot_VREP Destructor

### 3.1.2 Member Function Documentation

#### 3.1.2.1 double Robot_VREP::getAngularForce ( const unsigned int *jointID* )

Gets the angular force of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to get the force of |

**Returns**

the angular force of the joint in radians (double)

**See also**

updateAngularForce(const unsigned int index)
setForce(const unsigned int jointID, const double force)
getAngularPosition(const unsigned int jointID)
getAngularVelocity(const unsigned int jointID)

#### 3.1.2.2 double Robot_VREP::getAngularPosition ( const unsigned int *jointID* )

Gets the angular position of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to get the position of |

**Returns**

the angular position of the joint in radians (double)

**See also**

updateAngularPosition(const unsigned int index)
setPosition(const unsigned int jointID, const double pos)
getAngularVelocity(const unsigned int jointID)
getAngularForce(const unsigned int jointID)

#### 3.1.2.3 double Robot_VREP::getAngularVelocity ( const unsigned int *jointID* )

Gets the angular velocity of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to get the velocity of |

**Returns**

the angular velocity of the joint in radians (double)

**See also**

updateAngularVelocity(const unsigned int index)
setVelocity(const unsigned int jointID, const double vel)
getAngularPosition(const unsigned int jointID)
getAngularForce(const unsigned int jointID)

#### 3.1.2.4 double Robot_VREP::getMicroseconds ( )

Updates the clock

**Returns**

elapsed time in microseconds (double)

**See also**

startClock()
updateClock()
getSeconds()

#### 3.1.2.5 double Robot_VREP::getSeconds ( )

Updates the clock

**Returns**

elapsed time in seconds (double)

**See also**

startClock()
updateClock()
getMicroseconds()

#### 3.1.2.6 bool Robot_VREP::lockJoint ( const int *jointHandle* )

Is supposed to lock the joint so that it can't move (doesn't seem to be working...)

**Parameters**

| | |
|---|---|
| *jointHandle* | handle of the joint to lock |

**Returns**

true if the operation succeeded, false otherwise (bool)

#### 3.1.2.7 bool Robot_VREP::setForce ( const unsigned int *jointID,* const double *force* )

Sets the angular force of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to set the force of |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularForce(const unsigned int index)
getAngularForce(const unsigned int jointID)
setPosition(const unsigned int jointID, const double pos)
setVelocity(const unsigned int jointID, const double speed)

#### 3.1.2.8 bool Robot_VREP::setPosition ( const unsigned int *jointID,* const double *pos* )

Sets the angular position of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to set the position of |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularPosition(const unsigned int index)
getAngularPosition(const unsigned int jointID)
setVelocity(const unsigned int jointID, const double speed)
setForce(const unsigned int jointID, const double force)

#### 3.1.2.9 bool Robot_VREP::setVelocity ( const unsigned int *jointID,* const double *speed* )

Sets the angular velocity of a given joint

**Parameters**

| | |
|---|---|
| *jointID* | id of the joint we want to set the velocity of |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularVelocity(const unsigned int index)
getAngularVelocity(const unsigned int jointID)
setPosition(const unsigned int jointID, const double pos)
setForce(const unsigned int jointID, const double force)

#### 3.1.2.10 void Robot_VREP::startClock ( )

Starts the clock

**See also**

updateClock()
getMicroseconds()
getSeconds()

#### 3.1.2.11 bool Robot_VREP::updateAngularForce ( const unsigned int *index* = `-1` )

Updates the angular force of a given joint (Be careful not to call this function more than necessary since it's terribly slow)

**Parameters**

| | |
|---|---|
| *index* | (optional) id of the joint we want to update. If not set, will update every joint |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularPosition(const unsigned int index)
updateAngularVelocity(const unsigned int index)
getAngularForce(const unsigned int jointID)
setForce(const unsigned int jointID, const double force)

#### 3.1.2.12 bool Robot_VREP::updateAngularPosition ( const unsigned int *index* = −1 )

Updates the angular position of a given joint

**Parameters**

| | |
|---|---|
| *index* | (optional) id of the joint we want to update. If not set, will update every joint |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularVelocity(const unsigned int index)
updateAngularForce(const unsigned int index)
getAngularPosition(const unsigned int jointID)
setPosition(const unsigned int jointID, const double pos)

#### 3.1.2.13 bool Robot_VREP::updateAngularVelocity ( const unsigned int *index* = −1 )

Updates the angular velocity of a given joint

**Parameters**

| | |
|---|---|
| *index* | (optional) id of the joint we want to update. If not set, will update every joint |

**Returns**

true if the operation succeeded, false otherwise (bool)

**See also**

updateAngularPosition(const unsigned int index)
updateAngularForce(const unsigned int index)
getAngularVelocity(const unsigned int jointID)
setVelocity(const unsigned int jointID, const double speed)

#### 3.1.2.14 void Robot_VREP::updateClock ( )

Updates the clock

**See also**

startClock()
getMicroseconds()
getSeconds()

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

- Robot_VREP.h
- Robot_VREP.cpp

## 3.2 Simulation Class Reference

### Public Member Functions

- Simulation ()
- Simulation (std::vector< double > variables, std::vector< std::string > jointNames)
- ~Simulation ()
- void start ()
- void init ()
- void run ()
- std::vector< double > step ()
- void saveToFile (const std::string &path)

### 3.2.1 Constructor & Destructor Documentation

#### 3.2.1.1 Simulation::Simulation ( )

Simulation Constructor

**See also**

Simulation(std::vector<double> variables, std::vector<std::string> jointNames)

#### 3.2.1.2 Simulation::Simulation ( std::vector< double > *variables,* std::vector< std::string > *jointNames* )

Simulation Constructor

**Parameters**

| | |
|---|---|
| *variables* | initial values of the variables V1, y1, ss1, V2, y2, ss2 |
| *jointNames* | joint names of the robot |

**See also**

Simulation()

#### 3.2.1.3 Simulation::~Simulation ( )

Simulation Destructor

### 3.2.2 Member Function Documentation

#### 3.2.2.1 void Simulation::init ( )

initializes Simulation: initializes functions, Integrator and updates forces.

#### 3.2.2.2 void Simulation::run ( )

runs whole Simulation (Do NOT use this function with the Grapher! Use step only)

#### 3.2.2.3 void Simulation::saveToFile ( const std::string & *path* )

saves the recorded values into a csv file

**Parameters**

| | |
|---|---|
| *path* | path of the file where the values are saved |

#### 3.2.2.4 void Simulation::start ( )

Simulation start: Connects to remote API server and retrieves sphere informations

#### 3.2.2.5 std::vector< double > Simulation::step ( )

Simulation step

**Returns**

the new variables values for this step (std::vector<double>)

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

- Simulation.h
- Simulation.cpp

# Index