ActionExekutor

Generated by Doxygen 1.7.6.1

Mon Nov 24 2014 12:11:47

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 exekutor Namespace Reference

Classes

- class ActionExekutor

  A Base class to keep track of all the declared actions and to run a Listener that continuously listens on tuples and spawns the appropriate actionThread() whilst waiting for other tuples whose actions don't affect the actions happening.
- class MoveToExekutor

  The Class that provides the exekutor interface for the "move to" action.
- class MoveToSimpleExekutor

  Simple navigation exekutor using odometry alone.

Typedefs

- typedef std::vector < ActionExekutor * > ActionExekutorPtrVector

  Why not use some convenience typedefs?
- typedef std::vector< std::string > StringVector

Enumerations

- enum TupleType { COMMAND = 0, STATE, PARAMS, RESULT }

  An enumeration for convenience in accessing meta-tuple related structures.
- enum StateValue { COMPLETED = 0, RUNNING, FAILED }

5.1.1 Typedef Documentation

5.1.1.1 typedef std::vector<ActionExekutor*> exekutor::ActionExekutorPtrVector

Why not use some convenience typedefs?

5.1.1.2 typedef std::vector<std::string> exekutor::StringVector

5.1.2 Enumeration Type Documentation

5.1.2.1 enum exekutor::StateValue

Enumerator:

COMPLETED

RUNNING

FAILED

5.1.2.2 enum exekutor::TupleType

An enumeration for convenience in accessing meta-tuple related structures.

Enumerator:

COMMAND

STATE

PARAMS

RESULT

Chapter 6

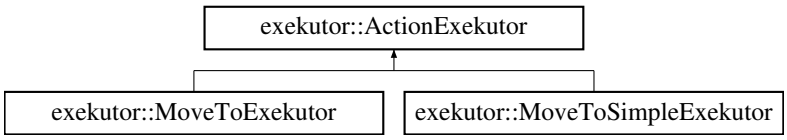
Class Documentation

6.1 exekutor::ActionExekutor Class Reference

A Base class to keep track of all the declared actions and to run a Listener that continuously listens on tuples and spawns the appropriate actionThread() whilst waiting for other tuples whose actions don't affect the actions happening.

`#include <action_exekutor.h>`

Inheritance diagram for exekutor::ActionExekutor:



Public Member Functions

- ActionExekutor (std::string robot_name, std::string a_name)

  Simple constructor that uses the string to initiate all meta tuples related to the action.
- virtual ~ActionExekutor ()

  Destructor.
- std::string getActionName ()

  We haven't found the use of this function till now!
- std::vector< double > extractParams (const char p[])

  Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.
- std::vector< std::string > extractParamStrings (const char p[])

  Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.
- void waitForMyLink ()

This call is specific to the exekutor.

- void cancelWaiting ()

  Cancel waiting on a particular exekutor.

Static Public Member Functions

- static void printAll ()

  Debugging function.

- static void waitForLink ()

  When this function is called, the program starts waiting for links to become available so that it can execute actions.

Public Attributes

- boost::shared_ptr < tf::TransformListener > tf_listener_

  A transform listener pointer.

Static Public Attributes

- static boost::shared_ptr < tf::TransformListener > tf_listener_base_

  A Listener for all exekutors.

Protected Member Functions

- void startAction (int i=ACTION_TIMEOUT)

  The function that makes a call to the actionThread() after doing common tasks.

- void initiateMetaTuples ()

  This function initiates all the peis meta-tuples necessary to use the action.

- void resetMetaTuples ()

  Reset Meta-tuples that were linked previously.

- virtual void actionThread ()=0

  The function that has to be implemented in all classes that derive from ActionExekutor.

- PeisTuple getParamTuple ()

  Convenience function to extract the parameters.

- void setState (StateValue value)

  Set the STATE tuples to the desired values.

- void setResult (std::string result_value)

  Set the result tuple for actions like acquire.

Static Protected Member Functions

- static void addAction (ActionExekutor *_this)

  Add the action to the action_ptr_list.
- static void * waitThread (void *_this_)

  The thread function that actually does the waiting.

Protected Attributes

- ros::NodeHandle nh_

  A Nodehandle.
- pthread_t thread_id_

  Id of the thread where this exekutor is running.
- std::string robot_name_

  This member stores the name of the robot.
- std::string action_name_

  This member stores the name of the action.
- std::string tuple_set_ [4]

  Convenience strings to store the Command, State and Parameters meta-keys.

Static Protected Attributes

- static StringVector action_str_list

  This static member has a list of action names of all created objects of supertype - ActionExekutor.
- static int my_peis_id

  Peis ID of the process running the exekutor.
- static ActionExekutorPtrVector action_ptr_list

  This static member has a list of all the created objects of supertype ActionExekutor.

6.1.1 Detailed Description

A Base class to keep track of all the declared actions and to run a Listener that continuously listens on tuples and spawns the appropriate actionThread() whilst waiting for other tuples whose actions don't affect the actions happening.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 exekutor::ActionExekutor::ActionExekutor (std::string *robot_name*, std::string *a_name*)

Simple constructor that uses the string to initiate all meta tuples related to the action.

6.1.2.2 virtual exekutor::ActionExekutor::~ActionExekutor ( ) [virtual]

Destructor.

6.1.3 Member Function Documentation

6.1.3.1 virtual void exekutor::ActionExekutor::actionThread ( ) [protected, pure virtual]

The function that has to be implemented in all classes that derive from ActionExekutor.

This function is called from startAction().

Implemented in exekutor::MoveToExekutor, and exekutor::MoveToSimpleExekutor.

6.1.3.2 static void exekutor::ActionExekutor::addAction (ActionExekutor * *_this*) [inline, static, protected]

Add the action to the action_ptr_list.

6.1.3.3 void exekutor::ActionExekutor::cancelWaiting ( )

Cancel waiting on a particular exekutor.

Useful to cancel out conflicting exekutor instances.

6.1.3.4 std::vector<double> exekutor::ActionExekutor::extractParams (const char *p[]*)

Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.

This function converts the c-string given as argument into double values. The double values in the actual string should be separated by commas or spaces.

6.1.3.5 std::vector<std::string> exekutor::ActionExekutor::extractParamStrings (const char *p[]*)

Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.

This function converts the c-string given as argument into a vector of strings. The strings in the input string should be separated by commas or spaces.

6.1.3.6 std::string exekutor::ActionExekutor::getActionName ( ) [inline]

We haven't found the use of this function till now!

6.1.3.7 PeisTuple exekutor::ActionExekutor::getParamTuple ( ) [protected]

Convenience function to extract the parameters.

6.1.3.8 void exekutor::ActionExekutor::initiateMetaTuples ( ) [protected]

This function initiates all the peis meta-tuples necessary to use the action.

6.1.3.9 static void exekutor::ActionExekutor::printAll ( ) [static]

Debugging function.

6.1.3.10 void exekutor::ActionExekutor::resetMetaTuples ( ) [protected]

Reset Meta-tuples that were linked previously.

6.1.3.11 void exekutor::ActionExekutor::setResult (std::string *result_value*) [inline, protected]

Set the result tuple for actions like acquire.

6.1.3.12 void exekutor::ActionExekutor::setState (StateValue *value*) [inline, protected]

Set the STATE tuples to the desired values.

6.1.3.13 void exekutor::ActionExekutor::startAction (int *i* = ACTION_TIMEOUT) [protected]

The function that makes a call to the actionThread() after doing common tasks.

This takes care of setting the state to RUNNING.

6.1.3.14 static void exekutor::ActionExekutor::waitForLink ( ) [static]

When this function is called, the program starts waiting for links to become available so that it can execute actions.

This is what should be used when creating an executable.

6.1.3.15 void exekutor::ActionExekutor::waitForMyLink ( )

This call is specific to the exekutor.

waitForLink() calls this for every exekutor.

6.1.3.16 static void* exekutor::ActionExekutor::waitThread (void * _this_) [static, protected]

The thread function that actually does the waiting.

One such thread exists for each exekutor running.

6.1.4 Member Data Documentation

6.1.4.1 std::string exekutor::ActionExekutor::action_name_ [protected]

This member stores the name of the action.

Example: MoveTo, FindBlob or FindHuman.

6.1.4.2 ActionExekutorPtrVector exekutor::ActionExekutor::action_ptr_list [static, protected]

This static member has a list of all the created objects of supertype ActionExekutor.

6.1.4.3 StringVector exekutor::ActionExekutor::action_str_list [static, protected]

This static member has a list of action names of all created objects of supertype ActionExekutor.

6.1.4.4 int exekutor::ActionExekutor::my_peis_id [static, protected]

Peis ID of the process running the exekutor.

6.1.4.5 ros::NodeHandle exekutor::ActionExekutor::nh_ [protected]

A Nodehandle.

This ensures that ROS stays alive as long as the last exekutor is alive.

6.1.4.6 std::string exekutor::ActionExekutor::robot_name_ [protected]

This member stores the name of the robot.

Example: Doro1, Oro1, Coro1, Turtlebot342.

6.1.4.7 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_

A transform listener pointer.

In practive we create one static Listener. Actually the original tf_listener_base_ can be used everywhere in code. To avoid too much rewriting, I have left this as such.

6.1.4.8 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_base_ [static]

A Listener for all exekutors.

All the tf_listener_ pointers point to this only.

6.1.4.9 pthread_t exekutor::ActionExekutor::thread_id_ [protected]

Id of the thread where this exekutor is running.

6.1.4.10 std::string exekutor::ActionExekutor::tuple_set_[4] [protected]

Convenience strings to store the Command, State and Parameters meta-keys.

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

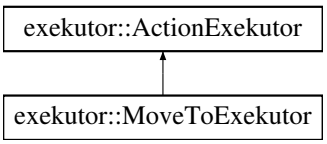
- /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/action_exekutor.h

6.2 exekutor::MoveToExekutor Class Reference

The Class that provides the exekutor interface for the "move to" action.

`#include <move_to_exekutor.h>`

Inheritance diagram for exekutor::MoveToExekutor:



Public Member Functions

- MoveToExekutor (std::string robot_name, std::string move_to_name)

  A constructor that takes in the name of the robot and the name of the action.
- virtual ~MoveToExekutor ()

  Destructor.
- std::string getActionName ()

  We haven't found the use of this function till now!

- std::vector< double > extractParams (const char p[])

  Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.
- std::vector< std::string > extractParamStrings (const char p[])

  Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.
- void waitForMyLink ()

  This call is specific to the exekutor.
- void cancelWaiting ()

  Cancel waiting on a particular exekutor.

Static Public Member Functions

- static void printAll ()

  Debugging function.
- static void waitForLink ()

  When this function is called, the program starts waiting for links to become available so that it can execute actions.

Public Attributes

- boost::shared_ptr < tf::TransformListener > tf_listener_

  A transform listener pointer.

Static Public Attributes

- static boost::shared_ptr < tf::TransformListener > tf_listener_base_

  A Listener for all exekutors.

Protected Member Functions

- virtual void actionThread ()

  This function creates a client to the move_base action.
- void startAction (int i=ACTION_TIMEOUT)

  The function that makes a call to the actionThread() after doing common tasks.
- void initiateMetaTuples ()

  This function initiates all the peis meta-tuples necessary to use the action.
- void resetMetaTuples ()

  Reset Meta-tuples that were linked previously.
- PeisTuple getParamTuple ()

  Convenience function to extract the parameters.
- void setState (StateValue value)

  Set the STATE tuples to the desired values.

- void setResult (std::string result_value)

  Set the result tuple for actions like acquire.

Static Protected Member Functions

- static void addAction (ActionExekutor *_this)

  Add the action to the action_ptr_list.
- static void * waitThread (void *_this_)

  The thread function that actually does the waiting.

Protected Attributes

- actionlib::SimpleActionClient < move_base_msgs::MoveBaseAction > mb_client_

  The action client (simple) to a 'move_base' action.
- ros::NodeHandle nh_

  A Nodehandle.
- pthread_t thread_id_

  Id of the thread where this exekutor is running.
- std::string robot_name_

  This member stores the name of the robot.
- std::string action_name_

  This member stores the name of the action.
- std::string tuple_set_ [4]

  Convenience strings to store the Command, State and Parameters meta-keys.

Static Protected Attributes

- static StringVector action_str_list

  This static member has a list of action names of all created objects of supertype ActionExekutor.
- static int my_peis_id

  Peis ID of the process running the exekutor.
- static ActionExekutorPtrVector action_ptr_list

  This static member has a list of all the created objects of supertype ActionExekutor.

6.2.1 Detailed Description

The Class that provides the exekutor interface for the "move to" action.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 exekutor::MoveToExekutor::MoveToExekutor (std::string *robot_name*, std::string *move_to_name*)

A constructor that takes in the name of the robot and the name of the action.

6.2.2.2 virtual exekutor::MoveToExekutor::~MoveToExekutor ( ) [virtual]

Destructor.

6.2.3 Member Function Documentation

6.2.3.1 virtual void exekutor::MoveToExekutor::actionThread ( ) [protected, virtual]

This function creates a client to the move_base action.

The implementation of actionThread() function. This is called from ActionExekutor's startActionThread(). This is where we create the client to the move_base action.

Implements exekutor::ActionExekutor.

6.2.3.2 static void exekutor::ActionExekutor::addAction (ActionExekutor * *_this*) [inline, static, protected, inherited]

Add the action to the action_ptr_list.

6.2.3.3 void exekutor::ActionExekutor::cancelWaiting ( ) [inherited]

Cancel waiting on a particular exekutor.

Useful to cancel out conflicting exekutor instances.

6.2.3.4 std::vector<double> exekutor::ActionExekutor::extractParams (const char *p[]*) [inherited]

Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.

This function converts the c-string given as argument into double values. The double values in the actual string should be separated by commas or spaces.

6.2.3.5 std::vector<std::string> exekutor::ActionExekutor::extractParamStrings (const char *p[]*) [inherited]

Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.

This function converts the c-string given as argument into a vector of strings. The strings in the input string should be separated by commas or spaces.

6.2.3.6 std::string exekutor::ActionExekutor::getActionName ( ) [inline, inherited]

We haven't found the use of this function till now!

6.2.3.7 PeisTuple exekutor::ActionExekutor::getParamTuple ( ) [protected, inherited]

Convenience function to extract the parameters.

6.2.3.8 void exekutor::ActionExekutor::initiateMetaTuples ( ) [protected, inherited]

This function initiates all the peis meta-tuples necessary to use the action.

6.2.3.9 static void exekutor::ActionExekutor::printAll ( ) [static, inherited]

Debugging function.

6.2.3.10 void exekutor::ActionExekutor::resetMetaTuples ( ) [protected, inherited]

Reset Meta-tuples that were linked previously.

6.2.3.11 void exekutor::ActionExekutor::setResult (std::string *result_value*) [inline, protected, inherited]

Set the result tuple for actions like acquire.

6.2.3.12 void exekutor::ActionExekutor::setState (StateValue *value*) [inline, protected, inherited]

Set the STATE tuples to the desired values.

6.2.3.13 void exekutor::ActionExekutor::startAction (int *i* = ACTION_TIMEOUT) [protected, inherited]

The function that makes a call to the actionThread() after doing common tasks.

This takes care of setting the state to RUNNING.

6.2.3.14 static void exekutor::ActionExekutor::waitForLink ( ) [static, inherited]

When this function is called, the program starts waiting for links to become available so that it can execute actions.

This is what should be used when creating an executable.

6.2.3.15 void exekutor::ActionExekutor::waitForMyLink ( ) [inherited]

This call is specific to the exekutor.

waitForLink() calls this for every exekutor.

6.2.3.16 static void* exekutor::ActionExekutor::waitThread (void * *_this_*) [static, protected, inherited]

The thread function that actually does the waiting.

One such thread exists for each exekutor running.

6.2.4 Member Data Documentation

6.2.4.1 std::string exekutor::ActionExekutor::action_name_ [protected, inherited]

This member stores the name of the action.

Example: MoveTo, FindBlob or FindHuman.

6.2.4.2 ActionExekutorPtrVector exekutor::ActionExekutor::action_ptr_list [static, protected, inherited]

This static member has a list of all the created objects of supertype ActionExekutor.

6.2.4.3 StringVector exekutor::ActionExekutor::action_str_list [static, protected, inherited]

This static member has a list of action names of all created objects of supertype ActionExekutor.

6.2.4.4 actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> exekutor::MoveToExekutor::mb_client_ [protected]

The action client (simple) to a 'move_base' action.

This stays alive for as long as the exekutor instance is alive.

6.2.4.5 int exekutor::ActionExekutor::my_peis_id [static, protected, inherited]

Peis ID of the process running the exekutor.

6.2.4.6 ros::NodeHandle exekutor::ActionExekutor::nh_ [protected, inherited]

A Nodehandle.

This ensures that ROS stays alive as long as the last exekutor is alive.

6.2.4.7 std::string exekutor::ActionExekutor::robot_name_ [protected, inherited]

This member stores the name of the robot.

Example: Doro1, Oro1, Coro1, Turtlebot342.

6.2.4.8 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_ [inherited]

A transform listener pointer.

In practive we create one static Listener. Actually the original tf_listener_base_ can be used everywhere in code. To avoid too much rewriting, I have left this as such.

6.2.4.9 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_base_ [static, inherited]

A Listener for all exekutors.

All the tf_listener_ pointers point to this only.

6.2.4.10 pthread_t exekutor::ActionExekutor::thread_id_ [protected, inherited]

Id of the thread where this exekutor is running.

6.2.4.11 std::string exekutor::ActionExekutor::tuple_set_[4] [protected, inherited]

Convenience strings to store the Command, State and Parameters meta-keys.

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

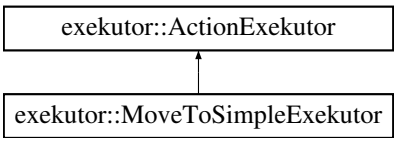
- /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/move_to_exekutor.h

6.3 exekutor::MoveToSimpleExekutor Class Reference

Simple navigation exekutor using odometry alone.

`#include <move_to_simple_exekutor.h>`

Inheritance diagram for exekutor::MoveToSimpleExekutor:



Public Member Functions

- MoveToSimpleExekutor (std::string robot_name, std::string action_name)

  Constructor.
- virtual ~MoveToSimpleExekutor ()

  Destructor.
- std::string getActionName ()

  We haven't found the use of this function till now!
- std::vector< double > extractParams (const char p[])

  Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.
- std::vector< std::string > extractParamStrings (const char p[])

  Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.
- void waitForMyLink ()

  This call is specific to the exekutor.
- void cancelWaiting ()

  Cancel waiting on a particular exekutor.

Static Public Member Functions

- static void printAll ()

  Debugging function.
- static void waitForLink ()

  When this function is called, the program starts waiting for links to become available so that it can execute actions.

Public Attributes

- boost::shared_ptr < tf::TransformListener > tf_listener_

  A transform listener pointer.

Static Public Attributes

- static boost::shared_ptr < tf::TransformListener > tf_listener_base_

  A Listener for all exekutors.

Protected Member Functions

- void actionThread ()

  This function creates a client to the move_to_simple action.
- void startAction (int i=ACTION_TIMEOUT)

  The function that makes a call to the actionThread() after doing common tasks.
- void initiateMetaTuples ()

  This function initiates all the peis meta-tuples necessary to use the action.
- void resetMetaTuples ()

  Reset Meta-tuples that were linked previously.
- PeisTuple getParamTuple ()

  Convenience function to extract the parameters.
- void setState (StateValue value)

  Set the STATE tuples to the desired values.
- void setResult (std::string result_value)

  Set the result tuple for actions like acquire.

Static Protected Member Functions

- static void addAction (ActionExekutor *_this)

  Add the action to the action_ptr_list.
- static void * waitThread (void *_this_)

  The thread function that actually does the waiting.

Protected Attributes

- actionlib::SimpleActionClient < simple_service::MoveToSimpleAction > move_to_simple_client_

  A simple_action_client to the 'move_to_simple' action server.
- ros::NodeHandle nh_

  A Nodehandle.
- pthread_t thread_id_

  Id of the thread where this exekutor is running.
- std::string robot_name_

  This member stores the name of the robot.
- std::string action_name_

  This member stores the name of the action.
- std::string tuple_set_ [4]

  Convenience strings to store the Command, State and Parameters meta-keys.

Static Protected Attributes

- static StringVector action_str_list

  This static member has a list of action names of all created objects of supertype ActionExekutor.
- static int my_peis_id

  Peis ID of the process running the exekutor.
- static ActionExekutorPtrVector action_ptr_list

  This static member has a list of all the created objects of supertype ActionExekutor.

6.3.1 Detailed Description

Simple navigation exekutor using odometry alone.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 exekutor::MoveToSimpleExekutor::MoveToSimpleExekutor (std::string *robot_name*, std::string *action_name*)

Constructor.

6.3.2.2 virtual exekutor::MoveToSimpleExekutor::~MoveToSimpleExekutor ( ) [virtual]

Destructor.

6.3.3 Member Function Documentation

6.3.3.1 void exekutor::MoveToSimpleExekutor::actionThread ( ) [protected, virtual]

This function creates a client to the move_to_simple action.

The implementation of actionThread() function. This is called from ActionExekutor's startActionThread(). This is where we create the client to the move_to_simple action.

Implements exekutor::ActionExekutor.

6.3.3.2 static void exekutor::ActionExekutor::addAction (ActionExekutor * *_this*) [inline, static, protected, inherited]

Add the action to the action_ptr_list.

6.3.3.3 void exekutor::ActionExekutor::cancelWaiting ( ) [inherited]

Cancel waiting on a particular exekutor.

Useful to cancel out conflicting exekutor instances.

6.3.3.4 std::vector<double> exekutor::ActionExekutor::extractParams (const char *p[]*) [inherited]

Converts an input string that consists of multiple double values separated by commas or spaces into a vector of doubles.

This function converts the c-string given as argument into double values. The double values in the actual string should be separated by commas or spaces.

6.3.3.5 std::vector<std::string> exekutor::ActionExekutor::extractParamStrings (const char *p[]*) [inherited]

Converts an input string that consists of multiple strings separated by commas or spaces into a vector of strings.

This function converts the c-string given as argument into a vector of strings. The strings in the input string should be separated by commas or spaces.

6.3.3.6 std::string exekutor::ActionExekutor::getActionName ( ) [inline, inherited]

We haven't found the use of this function till now!

6.3.3.7 PeisTuple exekutor::ActionExekutor::getParamTuple ( ) [protected, inherited]

Convenience function to extract the parameters.

6.3.3.8 void exekutor::ActionExekutor::initiateMetaTuples ( ) [protected, inherited]

This function initiates all the peis meta-tuples necessary to use the action.

6.3.3.9 static void exekutor::ActionExekutor::printAll ( ) [static, inherited]

Debugging function.

6.3.3.10 void exekutor::ActionExekutor::resetMetaTuples ( ) [protected, inherited]

Reset Meta-tuples that were linked previously.

6.3.3.11 void exekutor::ActionExekutor::setResult (std::string *result_value*) [inline, protected, inherited]

Set the result tuple for actions like acquire.

6.3.3.12 void exekutor::ActionExekutor::setState (StateValue *value*) [inline, protected, inherited]

Set the STATE tuples to the desired values.

6.3.3.13 void exekutor::ActionExekutor::startAction (int *i* = ACTION_TIMEOUT) [protected, inherited]

The function that makes a call to the actionThread() after doing common tasks.

This takes care of setting the state to RUNNING.

6.3.3.14 static void exekutor::ActionExekutor::waitForLink ( ) [static, inherited]

When this function is called, the program starts waiting for links to become available so that it can execute actions.

This is what should be used when creating an executable.

6.3.3.15 void exekutor::ActionExekutor::waitForMyLink ( ) [inherited]

This call is specific to the exekutor.

waitForLink() calls this for every exekutor.

6.3.3.16 static void* exekutor::ActionExekutor::waitThread (void * *_this_*) [static, protected, inherited]

The thread function that actually does the waiting.

One such thread exists for each exekutor running.

6.3.4 Member Data Documentation

6.3.4.1 std::string exekutor::ActionExekutor::action_name_ [protected, inherited]

This member stores the name of the action.

Example: MoveTo, FindBlob or FindHuman.

6.3.4.2 ActionExekutorPtrVector exekutor::ActionExekutor::action_ptr_list [static, protected, inherited]

This static member has a list of all the created objects of supertype ActionExekutor.

6.3.4.3 StringVector exekutor::ActionExekutor::action_str_list [static, protected, inherited]

This static member has a list of action names of all created objects of supertype ActionExekutor.

6.3.4.4 actionlib::SimpleActionClient<simple_service::MoveToSimpleAction> exekutor::MoveToSimpleExekutor::move_to_simple_client_ [protected]

A simple_action_client to the 'move_to_simple' action server.

6.3.4.5 int exekutor::ActionExekutor::my_peis_id [static, protected, inherited]

Peis ID of the process running the exekutor.

6.3.4.6 ros::NodeHandle exekutor::ActionExekutor::nh_ [protected, inherited]

A Nodehandle.

This ensures that ROS stays alive as long as the last exekutor is alive.

6.3.4.7 std::string exekutor::ActionExekutor::robot_name_ [protected, inherited]

This member stores the name of the robot.

Example: Doro1, Oro1, Coro1, Turtlebot342.

6.3.4.8 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_ [inherited]

A transform listener pointer.

In practive we create one static Listener. Actually the original tf_listener_base_ can be used everywhere in code. To avoid too much rewriting, I have left this as such.

6.3.4.9 boost::shared_ptr<tf::TransformListener> exekutor::ActionExekutor::tf_listener_base_ [static, inherited]

A Listener for all exekutors.

All the tf_listener_ pointers point to this only.

6.3.4.10 pthread_t exekutor::ActionExekutor::thread_id_ [protected, inherited]

Id of the thread where this exekutor is running.

6.3.4.11 std::string exekutor::ActionExekutor::tuple_set_[4] [protected, inherited]

Convenience strings to store the Command, State and Parameters meta-keys.

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

- /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/move_to_simple_exekutor.h

Chapter 7

File Documentation

7.1 /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/action_exekutor.h File Reference

```
#include <string> #include <vector> #include <unistd.h>
#include <stdlib.h> #include <iostream> #include <pthread.h>
#include <string.h> #include <ros/ros.h> #include
<tf/tf.h> #include <tf/transform_listener.h> #include
<cam_interface/cam_interface.h> #include <peiskernel/peiskernel.h>
#include <peiskernel/peiskernel_mt.h>
```

Classes

- class exekutor::ActionExekutor

  A Base class to keep track of all the declared actions and to run a Listener that continuously listens on tuples and spawns the appropriate actionThread() whilst waiting for other tuples whose actions don't affect the actions happening.

Namespaces

- namespace exekutor

Defines

- #define ACTION_TIMEOUT 10

  The macro that defines when an action should time-out.

Typedefs

- typedef std::vector < ActionExekutor * > exekutor::ActionExekutorPtrVector

Why not use some convenience typedefs?

- typedef std::vector< std::string > exekutor::StringVector

Enumerations

- enum exekutor::TupleType { exekutor::COMMAND = 0, exekutor::STATE, exekutor::PARAMS, exekutor::RESULT }

  An enumeration for convenience in accessing meta-tuple related structures.
- enum exekutor::StateValue { exekutor::COMPLETED = 0, exekutor::RUNNING, exekutor::FAILED }

7.1.1 Define Documentation

7.1.1.1 #define ACTION_TIMEOUT 10

The macro that defines when an action should time-out.

7.2 /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/move_to_exekutor.h File Reference

`#include "exekutor/action_exekutor.h" #include <move_base_msgs/MoveBaseAction.h> #include <actionlib/client/simple_action_client.h>`

Classes

- class exekutor::MoveToExekutor

  The Class that provides the exekutor interface for the "move to" action.

Namespaces

- namespace exekutor

7.3 /home/ace/chittaranjan_ros/catkin_ws3/src/exekutor/action_exekutor/include/exekutor/move_to_simple_exekutor.h File Reference

`#include <exekutor/action_exekutor.h> #include <actionlib/client/simple_action_client.h> #include <simple_service/MoveToSimpleAction.h>`

Classes

- class exekutor::MoveToSimpleExekutor

  Simple navigation exekutor using odometry alone.

Namespaces

- namespace exekutor