untangle::actuator

Generated by Doxygen 1.8.13

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Module Documentation

3.1 namespace untangle: functions

Functions

- template<typename actionT , typename ... Actions>
  auto untangle::connect (actionT &A1, Actions &... An)

  Creates an actuator holding an initial list of actions.
- template<typename classT , typename T , typename actionT = std::function<T>>
  static actionT untangle::bind (const std::shared_ptr< classT > &obj, T classT::*method)

  Binding to a class function member.
- template<typename classT , typename T , typename actionT = std::function<T>>
  static actionT untangle::bind (classT *obj, T classT::*method)

  Binding to a class method.

3.1.1 Detailed Description

3.1.2 Function Documentation

3.1.2.1 connect()

```
template<typename actionT , typename ...  Actions>
auto untangle::connect (
            actionT & A1,
            Actions &...  An )
```

Creates an actuator holding an initial list of actions.

Parameters

A1..An	Any number of actions. They are specified as std::function<...>.

Returns

An actuator.

Example:

```
void rotate_shapes(std::vector<shape*>& shapes, int angle)
{
  for (const auto& s : shapes)
  {
    s->rotate(angle);
  }
}


  // How to use actuator instead of polymorphism.

  std::shared_ptr<triangle> t(new triangle);
  std::shared_ptr<circle> c(new circle);
  std::shared_ptr<square> s(new square);

  // using polymorphism
  std::vector<shape*> shapes;
  shapes.push_back(t.get());
  shapes.push_back(c.get());
  shapes.push_back(s.get());

  std::cout << "using polymorphism\n" << std::endl;
  rotate_shapes(shapes, 10);

  // using actuator
  auto action1 = untangle::bind(t, &triangle::rotate);
  auto action2 = untangle::bind(c, &circle::rotate);
  auto action3 = untangle::bind(s, &square::rotate);

  auto actuator_rotate = untangle::connect(action1, action2, action3);
  std::cout << "\nusing actuator\n" << std::endl;
  actuator_rotate(20);
```

3.1.2.2 bind() [1/2]

```
template<typename classT , typename T , typename actionT = std::function<T>>
static actionT untangle::bind (
              const std::shared_ptr< classT > & obj,
              T classT::* method )  [static]
```

Binding to a class function member.

It returns an std::function(lambda) that wraps the function member. It may be used to provide an action for connect() or actuator::add().

Remarks

It requires a shared pointer to the class type. This shared pointer is captured internally in a lambda, and it can be checked if the shared object is valid. Therefore it is safe to use actions provided by this binding inside an actuator.

Parameters

obj	- Class object.
*classT::*method*	- Pointer to function member. It is specified as &<class type>=""">::<function member>="">

Returns

actionT - A std::function that wraps the pointer to function member.

Remarks

If the class object gets invalid, invoking this binding will throw an exception of type invalid_action.

3.1.2.3 bind() [2/2]

```
template<typename classT , typename T , typename actionT = std::function<T>>
static actionT untangle::bind (
            classT * obj,
            T classT::* method )  [static]
```

Binding to a class method.

Attention

It is not safe to use this binding to provide actions to an actuator.The class object is provided through a pointer type. This pointer is captured internally in a lambda, so it can not be checked if it gets null.

Remarks

It is provided for conveniency of use: within a class it is safe to create bindings through **this** pointer.

Parameters

obj	- Pointer to class.
classT::∗method	- Pointer to function member. It is specified as &<class type>="">::<function member>="">

Returns

actionT - A std::function that wraps the pointer to function member.

Chapter 4

Class Documentation

4.1 untangle::actuator< actionT > Struct Template Reference

An actuator is a function object that can trigger a dynamic list of actions (of type std::function<...>).

`#include <actuator.h>`

Public Types

- using actionsT = std::list< actionT * >
  Actions container type.
- using resultsT = std::vector< typename resultT::type >
  Results container type.

Public Member Functions

- actuator & operator= (const actuator &other)
  Assignment operator.
- template<typename ... Args>
  void operator() (Args &&... args)
  The call operator.
- template<typename ... Args>
  void invokeAction (std::string name, Args &&... args)
  Invokes one single action associated with a key.
- void add (actionT *action)
  Add an action to the actions list.
- void add (std::string name, actionT *action)
  Add action to the actions map associated with a name.
- void remove (const actionT *action)
  Remove an action from the actions list.
- void remove (const std::string &name)
  Remove an action from actions map.
- bool is_connected ()
  Check if this actuator is "connected" with other actions.
- bool has_action (std::string name)
  Check if there is certain named action.

Public Attributes

- actionsT actions
  Actions list.
- mapActionsT mapActions
  Actions list.
- resultsT results
  Actions return values list.

4.1.1 Detailed Description

template<typename actionT>
struct untangle::actuator< actionT >

An actuator is a function object that can trigger a dynamic list of actions (of type std::function<...>).

Remarks

An actuator object can be constructed with an initial list of actions by connect().

Template Parameters

actionT	Action type. It is specified as std::function<...>.

4.1.2 Member Typedef Documentation

4.1.2.1 actionsT

```
template<typename actionT>
using untangle::actuator< actionT >::actionsT = std::list<actionT*>
```

Actions container type.

Remarks

The elements stored are of pointer type, that is required to implement the remove() operation. std::function supports only equality operator for nullptr (two std::function(s) can not compare).

4.1.2.2 resultsT

```
template<typename actionT>
using untangle::actuator< actionT >::resultsT = std::vector<typename resultT::type>
```

Results container type.

It holds the return values of the actions that have a non void return type. Upon the actuator invocation, the returns can be extracted from results.

4.1.3 Member Function Documentation

4.1.3.1 operator=()

```
template<typename actionT>
actuator& untangle::actuator< actionT >::operator= (
            const actuator< actionT > & other )  [inline]
```

Assignment operator.

Example:

```
std::shared_ptr<triangle> t(new triangle);
std::shared_ptr<circle> c(new circle);
std::shared_ptr<square> s(new square);

auto action1 = untangle::bind(t, &triangle::rotate);
auto action2 = untangle::bind(c, &circle::rotate);
auto action3 = untangle::bind(s, &square::rotate);

auto actuator_rotate = untangle::connect(action1, action2, action3);

std::cout << "assign to an actuator member of class triangle\n" << std::endl;
t->actuator_rotate = actuator_rotate;
t->actuator_rotate(30);
```

4.1.3.2 operator()()

```
template<typename actionT>
template<typename ...  Args>
void untangle::actuator< actionT >::operator() (
            Args &&...  args )  [inline]
```

The call operator.

Actions in the actuator::actions list are triggered by invoking the call operator.

Parameters

args	- Arguments list must match the action arity.

4.1.3.3 invokeAction()

```
template<typename actionT>
template<typename ...  Args>
void untangle::actuator< actionT >::invokeAction (
```

```
            std::string name,
            Args &&... args )  [inline]
```

Invokes one single action associated with a key.

Parameters

name	- Key associated with the action.
args	- Arguments list must match the action arity.

4.1.3.4 add() [1/2]

```
template<typename actionT>
void untangle::actuator< actionT >::add (
            actionT * action )  [inline]
```

Add an action to the actions list.

Parameters

action	- Action to be added.

Example:

```
std::shared_ptr<triangle> t(new triangle);
std::shared_ptr<circle> c(new circle);
std::shared_ptr<square> s(new square);

auto action1 = untangle::bind(t, &triangle::rotate);
auto action2 = untangle::bind(c, &circle::rotate);
auto action3 = untangle::bind(s, &square::rotate);

auto actuator_rotate = untangle::connect(action1, action2, action3);

std::cout << "\nadd an action\n" << std::endl;
actuator_rotate.add(&action1);
actuator_rotate(40);
```

4.1.3.5 add() [2/2]

```
template<typename actionT>
void untangle::actuator< actionT >::add (
            std::string name,
            actionT * action )  [inline]
```

Add action to the actions map associated with a name.

Parameters

name	- Name of the action.
action	- Action to be added.

4.1.3.6 remove() [1/2]

```
template<typename actionT>
void untangle::actuator< actionT >::remove (
            const actionT * action )  [inline]
```

Remove an action from the actions list.

An invalid action (empty std::function) is implicitly removed when operator()() is invoked.

Parameters

action	- Action to be removed.

Example:

```
std::shared_ptr<triangle> t(new triangle);
std::shared_ptr<circle> c(new circle);
std::shared_ptr<square> s(new square);

auto action1 = untangle::bind(t, &triangle::rotate);
auto action2 = untangle::bind(c, &circle::rotate);
auto action3 = untangle::bind(s, &square::rotate);

auto actuator_rotate = untangle::connect(action1, action2, action3);

std::cout << "\nadd an action\n" << std::endl;
actuator_rotate.add(&action1);
actuator_rotate(40);
```

4.1.3.7 remove() [2/2]

```
template<typename actionT>
void untangle::actuator< actionT >::remove (
            const std::string & name )  [inline]
```

Remove an action from actions map.

Parameters

name	- Name of the action to remove.

4.1.3.8 is_connected()

```
template<typename actionT>
bool untangle::actuator< actionT >::is_connected ( )  [inline]
```

Check if this actuator is "connected" with other actions.

Returns

true - if the actuator::actions list is not empty.
false - if the actuator::actions list is empty.

4.1.3.9 has_action()

```
template<typename actionT>
bool untangle::actuator< actionT >::has_action (
            std::string name )  [inline]
```

Check if there is certain named action.

Parameters

name	- Name associated with the action.

Returns

true - if name can be found in actuator::mapActions
false - if name can not be found in actuator::mapActions

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

- actuator.h

4.2 untangle::invalid_action Struct Reference

Invalid action exception.

```
#include <actuator.h>
```

Public Member Functions

- invalid_action (const std::string &text)

  Construct a new invalid action object.

Public Attributes

- std::string what

  It holds the message text.

4.2.1 Detailed Description

Invalid action exception.

Remarks

An action may be provided as a binding to a class function member, by using bind(). When the class object gets invalid, invoking the action will raise an exception of this type.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 invalid_action()

```
untangle::invalid_action::invalid_action (
            const std::string & text )  [inline]
```

Construct a new invalid action object.

Parameters

text	- A message text, describing the reason of this exception.

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

- actuator.h

Index