# pycram.designators.action_designator

## Contents

# Classes

| | |
|---|---|
| MoveTorsoAction | Action Designator for Moving the torso of the robot up and down |
| SetGripperAction | Set the gripper state of the robot |
| ReleaseAction | Releases an Object from the robot. |
| GripAction | Grip an object with the robot. |
| ParkArmsAction | Park the arms of the robot. |
| PickUpAction | Designator to let the robot pick up an object. |
| PlaceAction | Places an Object at a position using an arm. |
| NavigateAction | Navigates the Robot to a position. |
| TransportAction | Transports an object to a position using an arm |
| LookAtAction | Lets the robot look at a position. |
| DetectAction | Detects an object that fits the object description and returns an object designator describing the object. |
| OpenAction | Opens a container like object |
| CloseAction | Closes a container like object. |
| GraspingAction | Grasps an object described by the given Object Designator description |
| ActionAbstract | Base class for performable performables. |
| MoveTorsoActionPerformable | Move the torso of the robot up and down. |
| SetGripperActionPerformable | Set the gripper state of the robot. |
| ReleaseActionPerformable | Releases an Object from the robot. |
| GripActionPerformable | Grip an object with the robot. |
| ParkArmsActionPerformable | Park the arms of the robot. |
| PickUpActionPerformable | Let the robot pick up an object. |
| PlaceActionPerformable | Places an Object at a position using an arm. |
| NavigateActionPerformable | Navigates the Robot to a position. |
| TransportActionPerformable | Transports an object to a position using an arm |
| LookAtActionPerformable | Lets the robot look at a position. |
| DetectActionPerformable | Detects an object that fits the object description and returns an object designator describing the object. |
| OpenActionPerformable | Opens a container like object |
| CloseActionPerformable | Closes a container like object. |

| | |
|---|---|
| `GraspingActionPerformable` | Grasps an object described by the given Object Designator description |
| `FaceAtPerformable` | Turn the robot chassis such that is faces the `pose` and after that perform a look at action. |
| `MoveAndPickUpPerformable` | Navigate to *standing_position*, then turn towards the object and pick it up. |

# Module Contents

***class*** pycram.designators.action_designator.**MoveTorsoAction**(***positions: typing_extensions.List[float], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[pycram.ontology.ontology.OntologyConceptHolder]] = None***)

Bases: `pycram.designator.ActionDesignatorDescription`

Action Designator for Moving the torso of the robot up and down

**positions**: *typing_extensions.List[float]*

**ground**() → `MoveTorsoActionPerformable`

Creates a performable action designator with the first element from the list of possible torso heights.

**Returns:**

A performable action designator

**__iter__**()

Iterates over all possible values for this designator and returns a performable action designator with the value.

**Returns:**

A performable action designator

***class*** pycram.designators.action_designator.**SetGripperAction**(***grippers: typing_extensions.List[pycram.datastructures.enums.Arms], motions: typing_extensions.List[pycram.datastructures.enums.GripperState], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None***)

Bases: `pycram.designator.ActionDesignatorDescription`

Set the gripper state of the robot

**grippers**: *typing_extensions.List[pycram.datastructures.enums.GripperState]*

**motions**: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**ground**() → `SetGripperActionPerformable`

Default specialized_designators that returns a performable designator with the first element in the grippers and motions list.

**Returns:**

A performable designator

#### __iter__()

Iterates over all possible combinations of grippers and motions

**Returns:**

A performable designator with a combination of gripper and motion

### *class* pycram.designators.action_designator.ReleaseAction(*grippers: typing_extensions.List[pycram.datastructures.enums.Arms], object_designator_description: pycram.designators.object_designator.ObjectDesignatorDescription, resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Releases an Object from the robot.

Note: This action can not be used yet.

#### grippers: *typing_extensions.List[pycram.datastructures.enums.Arms]*

#### object_designator_description

#### ground() → ReleaseActionPerformable

Fill all missing parameters and chose plan to execute.

### *class* pycram.designators.action_designator.GripAction(*grippers: typing_extensions.List[pycram.datastructures.enums.Arms], object_designator_description: pycram.designators.object_designator.ObjectDesignatorDescription, efforts: typing_extensions.List[float], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Grip an object with the robot.

**Variables:**

- **grippers** – The grippers to consider
- **object_designator_description** – The description of objects to consider
- **efforts** – The efforts to consider

Note: This action can not be used yet.

latest

#### grippers: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**object_designator_description:** *pycram.designators.object_designator.ObjectDesignatorDescription*

**efforts:** *typing_extensions.List[float]*

**ground()** → **GripActionPerformable**

Fill all missing parameters and chose plan to execute.

*class* pycram.designators.action_designator.**ParkArmsAction**(*arms: typing_extensions.List[pycram.datastructures.enums.Arms], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: **pycram.designator.ActionDesignatorDescription**

Park the arms of the robot.

**arms:** *typing_extensions.List[pycram.datastructures.enums.Arms]*

**ground()** → **ParkArmsActionPerformable**

Default specialized_designators that returns a performable designator with the first element of the list of possible arms

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**PickUpAction**(*object_designator_description: typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object], arms: typing_extensions.List[pycram.datastructures.enums.Arms], grasps: typing_extensions.List[pycram.datastructures.enums.Grasp], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: **pycram.designator.ActionDesignatorDescription**

Designator to let the robot pick up an object.

**object_designator_description:** *typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object]*

**arms:** *typing_extensions.List[pycram.datastructures.enums.Arms]*

**grasps:** *typing_extensions.List[pycram.datastructures.enums.Grasp]*

latest

**ground()** → **PickUpActionPerformable**

Default specialized_designators, returns a performable designator with the first entries from the lists of possible parameter.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**PlaceAction**(*object_designator_description: typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object], target_locations: typing_extensions.List[pycram.datastructures.pose.Pose], arms: typing_extensions.List[pycram.datastructures.enums.Arms], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Places an Object at a position using an arm.

**object_designator_description**: *typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object]*

**target_locations**: *typing_extensions.List[pycram.datastructures.pose.Pose]*

**arms**: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**ground**() → PlaceActionPerformable

Default specialized_designators that returns a performable designator with the first entries from the list of possible entries.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**NavigateAction**(*target_locations: typing_extensions.List[pycram.datastructures.pose.Pose], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Navigates the Robot to a position.

**target_locations**: *typing_extensions.List[pycram.datastructures.pose.Pose]*

**ground**() → NavigateActionPerformable

Default specialized_designators that returns a performable designator with the first entry of possible target locations.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**TransportAction**(*object_designator_description: typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object], arms: typing_extensions.List[pycram.datastructures.enums.Arms], target_locations: typing_extensions.List[pycram.datastructures.pose.Pose], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Transports an object to a position using an arm

**object_designator_description**: *typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectDesignatorDescription.Object]*

**arms**: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**target_locations**: *typing_extensions.List[pycram.datastructures.pose.Pose]*

**ground**() → TransportActionPerformable

Default specialized_designators that returns a performable designator with the first entries from the lists of possible parameter.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**LookAtAction**(*targets: typing_extensions.List[pycram.datastructures.pose.Pose], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Lets the robot look at a position.

**targets**: *typing_extensions.List[pycram.datastructures.pose.Pose]*

**ground**() → LookAtActionPerformable

Default specialized_designators that returns a performable designator with the first entry in the list of possible targets

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**DetectAction**(*object_designator_description: pycram.designators.object_designator.ObjectDesignatorDescription, resolver=None,*

*ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Detects an object that fits the object description and returns an object designator describing the object.

**object_designator_description:** *pycram.designators.object_designator.ObjectDesignatorDescription*

**ground**() → DetectActionPerformable

Default specialized_designators that returns a performable designator with the resolved object description.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**OpenAction**(*object_designator_description: pycram.designators.object_designator.ObjectPart, arms: typing_extensions.List[pycram.datastructures.enums.Arms], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Opens a container like object

Can currently not be used

**object_designator_description:** *pycram.designators.object_designator.ObjectPart*

**arms:** *typing_extensions.List[pycram.datastructures.enums.Arms]*

**ground**() → OpenActionPerformable

Default specialized_designators that returns a performable designator with the resolved object description and the first entries from the lists of possible parameter.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**CloseAction**(*object_designator_description: pycram.designators.object_designator.ObjectPart, arms: typing_extensions.List[pycram.datastructures.enums.Arms], resolver=None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Closes a container like object.

Can currently not be used

**object_designator_description**: *pycram.designators.object_designator.ObjectPart*

**arms**: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**ground**() → CloseActionPerformable

Default specialized_designators that returns a performable designator with the resolved object designator and the first entry from the list of possible arms.

**Returns:**

A performable designator

*class* pycram.designators.action_designator.**GraspingAction**(*arms: typing_extensions.List[pycram.datastructures.enums.Arms], object_description: typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription, pycram.designators.object_designator.ObjectPart], resolver: typing_extensions.Callable = None, ontology_concept_holders: typing_extensions.Optional[typing_extensions.List[owlready2.Thing]] = None*)

Bases: pycram.designator.ActionDesignatorDescription

Grasps an object described by the given Object Designator description

**arms**: *typing_extensions.List[pycram.datastructures.enums.Arms]*

**object_description**: *pycram.designators.object_designator.ObjectDesignatorDescription*

**ground**() → GraspingActionPerformable

Default specialized_designators that takes the first element from the list of arms and the first solution for the object designator description ond returns it.

**Returns:**

A performable action designator that contains specific arguments

*class* pycram.designators.action_designator.**ActionAbstract**

Bases: pycram.designator.ActionDesignatorDescription.Action, abc.ABC

Base class for performable performables.

**orm_class**: *typing_extensions.Type[pycram.orm.action_designator.Action]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

*abstract* **perform**() → None

Perform the action.

Will be overwritten by each action.

to_sql() → pycram.orm.action_designator.Action

Convert this action to its ORM equivalent.

Needs to be overwritten by an action if it didn't overwrite the orm_class attribute with its ORM equivalent.

**Returns:**

An instance of the ORM equivalent of the action with the parameters set

insert(*session: sqlalchemy.orm.Session*, ***kwargs*) → pycram.orm.action_designator.Action

Insert this action into the database.

Needs to be overwritten by an action if the action has attributes that do not exist in the orm class equivalent. In that case, the attributes need to be inserted into the session manually.

**Parameters:**

- **session** – Session with a database that is used to add and commit the objects
- **kwargs** – Possible extra keyword arguments

**Returns:**

The completely instanced ORM action that was inserted into the database

*class* pycram.designators.action_designator.MoveTorsoActionPerformable

Bases: ActionAbstract

Move the torso of the robot up and down.

position: *float*

Target position of the torso joint

orm_class: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

perform() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.SetGripperActionPerformable

Bases: ActionAbstract

Set the gripper state of the robot.

gripper: *pycram.datastructures.enums.Arms*

The gripper that should be set

latest

motion: *pycram.datastructures.enums.GripperState*

The motion that should be set on the gripper

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**ReleaseActionPerformable**

Bases: ActionAbstract

Releases an Object from the robot.

Note: This action can not ve used yet.

**gripper**: *pycram.datastructures.enums.Arms*

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

*abstract* **perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**GripActionPerformable**

Bases: ActionAbstract

Grip an object with the robot.

Note: This action can not be used yet.

**gripper**: *pycram.datastructures.enums.Arms*

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

**effort**: *float*

*abstract* **perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**ParkArmsActionPerformable**

Bases: ActionAbstract

Park the arms of the robot.

**arm**: *pycram.datastructures.enums.Arms*

Entry from the enum for which arm should be parked

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**PickUpActionPerformable**

Bases: ActionAbstract

Let the robot pick up an object.

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

Object designator describing the object that should be picked up

**arm**: *pycram.datastructures.enums.Arms*

The arm that should be used for pick up

**grasp**: *pycram.datastructures.enums.Grasp*

The grasp that should be used. For example, ‘left’ or ‘right’

**object_at_execution**: *typing_extensions.Optional[pycram.designators.object_designator.ObjectDesignatorDescription.Object]*

The object at the time this Action got created. It is used to be a static, information holding entity. It is not updated when the BulletWorld object is changed.

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**__post_init__**()

latest

**perform**() → None

Perform the action.

Will be overwritten by each action.

#### to_sql() → pycram.orm.action_designator.Action

Convert this action to its ORM equivalent.

Needs to be overwritten by an action if it didn't overwrite the orm_class attribute with its ORM equivalent.

**Returns:**

An instance of the ORM equivalent of the action with the parameters set

#### insert(*session: sqlalchemy.orm.Session*, ***kwargs*) → pycram.orm.action_designator.Action

Insert this action into the database.

Needs to be overwritten by an action if the action has attributes that do not exist in the orm class equivalent. In that case, the attributes need to be inserted into the session manually.

**Parameters:**

- **session** – Session with a database that is used to add and commit the objects
- **kwargs** – Possible extra keyword arguments

**Returns:**

The completely instanced ORM action that was inserted into the database

### *class* pycram.designators.action_designator.PlaceActionPerformable

Bases: ActionAbstract

Places an Object at a position using an arm.

#### object_designator: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

Object designator describing the object that should be place

#### arm: *pycram.datastructures.enums.Arms*

Arm that is currently holding the object

#### target_location: *pycram.datastructures.pose.Pose*

Pose in the world at which the object should be placed

#### orm_class: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

#### perform() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**NavigateActionPerformable**

Bases: ActionAbstract

Navigates the Robot to a position.

**target_location**: *pycram.datastructures.pose.Pose*

Location to which the robot should be navigated

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**TransportActionPerformable**

Bases: ActionAbstract

Transports an object to a position using an arm

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

Object designator describing the object that should be transported.

**arm**: *pycram.datastructures.enums.Arms*

Arm that should be used

**target_location**: *pycram.datastructures.pose.Pose*

Target Location to which the object should be transported

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.



*class* pycram.designators.action_designator.**LookAtActionPerformable**

Bases: ActionAbstract

Lets the robot look at a position.

**target**: *pycram.datastructures.pose.Pose*

Position at which the robot should look, given as 6D pose

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**DetectActionPerformable**

Bases: ActionAbstract

Detects an object that fits the object description and returns an object designator describing the object.

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

Object designator loosely describing the object, e.g. only type.

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**OpenActionPerformable**

Bases: ActionAbstract

Opens a container like object

**object_designator**: *pycram.designators.object_designator.ObjectPart.Object*

Object designator describing the object that should be opened

**arm**: *pycram.datastructures.enums.Arms*

Arm that should be used for opening the container

latest

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**CloseActionPerformable**

Bases: ActionAbstract

Closes a container like object.

**object_designator**: *pycram.designators.object_designator.ObjectPart.Object*

Object designator describing the object that should be closed

**arm**: *pycram.datastructures.enums.Arms*

Arm that should be used for closing

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**GraspingActionPerformable**

Bases: ActionAbstract

Grasps an object described by the given Object Designator description

**arm**: *pycram.datastructures.enums.Arms*

The arm that should be used to grasp

**object_desig**: *typing_extensions.Union[pycram.designators.object_designator.ObjectDesignatorDescription.Object, pycram.designators.object_designator.ObjectPart.Object]*

Object Designator for the object that should be grasped

**orm_class**: *typing_extensions.Type[ActionAbstract]*

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**FaceAtPerformable**

Bases: ActionAbstract

Turn the robot chassis such that is faces the `pose` and after that perform a look at action.

**pose**: *pycram.datastructures.pose.Pose*

The pose to face

**orm_class**

The ORM class that is used to insert this action into the database. Must be overwritten by every action in order to be able to insert the action into the database.

**perform**() → None

Perform the action.

Will be overwritten by each action.

*class* pycram.designators.action_designator.**MoveAndPickUpPerformable**

Bases: ActionAbstract

Navigate to *standing_position*, then turn towards the object and pick it up.

**standing_position**: *pycram.datastructures.pose.Pose*

The pose to stand before trying to pick up the object

**object_designator**: *pycram.designators.object_designator.ObjectDesignatorDescription.Object*

The object to pick up

**arm**: *pycram.datastructures.enums.Arms*

The arm to use

**grasp**: *pycram.datastructures.enums.Grasp*

The grasp to use

**perform**()

Perform the action.

Will be overwritten by each action.