Robot World Protocol Requirements

The client sends request messages to the server and receives response messages from the server. Every request must be replied to by a response.

Request Message

A request is a JSON string with the following structure.

Structure of a Request Message

```
{
  "robot": "name of the robot",                          ①
  "command": "name of the command",                      ②
  "arguments": [ value1, value2, ...]                    ③
}
```

① Mandatory. String: the name of the robot.

② Mandatory. String: the name of a command. Must be one of `launch`, `look`, `forward`, `back`, `turn`, `repair`, `reload`, `mine`, `fire`, `state`. Must be lower-case.

③ Mandatory. Array: arguments for the command with `value1` being the first argument, `value2` being the second, etc. The type and meaning of the arguments are specific for the command. If there are no arguments, then the array is empty.

See the Command Specifications below for more details.

Response Message

A response is a JSON string with the following structure.

Structure of a Response Message

```
{
  "result": "result of the command",                     ①
  "data": {                                              ②
    "key1": value1,                                      ③
    "key2": value2,
    ...
  },
  "state: {                                              ④
    "position": [x,y],                                   ⑤
    "direction": "direction that the robot is facing",   ⑥
    "shields": number of hits that can be absorbed,      ⑦
    "shots": number of shots left,                       ⑧
    "status": "operational status that the robot is in"     ⑨
   }
}
```

① Mandatory. String: The result of trying to execute the command. If it was successfully processed by the world, then the value is `OK` otherwise `ERROR`.

② Mandatory. Map: Every response has `data` key.

③ Key-Value: The `data` consists of one or more key-value pairs that is unique for the response for the specific command.

④ Optional. Map: A response has a `state` map only if the `result` value is `OK`.

⑤ Array: The coordinates of the current position of the robot as an array with two elements.

⑥ String: This is the direction that the robot is currently facing, being one of `NORTH`, `SOUTH`, `EAST`, `WEST`.

⑦ Integer: Representing the number of hits that the shields can still absorb.

⑧ Integer: Represents the number of shots left in the gun.

⑨ String: The operational mode in which the robot is in at the moment being one of `RELOADING`, `REPAIRING`, `SETMINE`, `NORMAL`, `DEAD`.

- `RELOADING` - the robot is reloading weapons
- `REPAIRING` - the robot is repairing shields
- `SETMINE` - the robot is setting a mine
- `NORMAL` - the robot is waiting for the next command
- `DEAD` - the robot is dead and is no longer in the world.

See the Command Specifications below for more details.

General Responses

The following are general response messages for badly formed requests.

Server could not parse the request arguments

In the situation where the server cannot parse the arguments for a command in a request message then the following response must be returned.

```
{
  "result": "ERROR",
  "data": {
    message: "Could not parse arguments"
  }
}
```

Server does not support the command

In the situation where the server does not support the command requested, the following response must be returned.

```
{
  "result": "ERROR",
  "data": {
    message: "Unsupported command"
  }
}
```

Command Specifications

The following are specifications for request and response messages for each command. Some keys are left out for brevity and replaced with `…`. This means that you should fill in the missing values as appropriate.

Launch

Launch a robot into the world.

Request

```
{
  "robot": ...,
  "command": "launch",
  "arguments": [ kind,                                    ①
                 maximum shield strength,                 ②
                 maximum shots                            ③
               ]
}
```

① String: The name of the kind of robot that is being launched. This is your uniquely programmed model of robot.

② Integer: The configured maximum shield strength of the robot in hits before it can be killed.

③ Integer: The maximum number of shots that the robot can take before reloading.

Response: Successful

```
{
  "result": "OK",
  "data": {
    position: [x,y],                                      ①
    "visibility": steps                                   ②
    "reload": seconds,                                    ③
    "repair": seconds,                                    ④
    "mine": seconds,                                      ⑤
    "shields": hits                                       ⑥
  },
  "state": {
    "position": [x,y],                                    ⑦
    "direction: "NORTH",                                  ⑧
    "shields": hits,                                      ⑨
    "shots": shots,                                       ⑩
    "status": "NORMAL"                                    ⑪
   }
}
```

① Array: the coordinates of the position of the robot

② Integer: The world's configured visibility in number of steps.

③ Integer: The number of seconds that it takes to reload weapons to the maximum number of shots.

④ Integer: The number of seconds that it takes to repair shields to the maximum.

⑤ Integer: The number of seconds that it takes to set a mine.

⑥ Integer: The maximum strength of shields

⑦ Array: The coordinates of the position of the robot. This should be the same as in the `data position` key.

⑧ String: The robot will always face north when launched.

⑨ Integer: The robot starts with maximum shield strength for which it was configured

⑩ Integer: The robot starts with maximum number of shots for which it was configured

⑪ The robot starts off in `NORMAL` operational mode

Response: No free location

```
{
  "result": "ERROR",
  "data": {
    "message": "No more space in this world"
  }
}
```

Response: Name already taken

```
{
  "result": "ERROR",
  "data": {
    "message": "Too many of you in this world"
  }
}
```

State

Ask the world for the state of the robot.

Request

```
{
  "robot": ...,
  "command": "state",
  "arguments": []
}
```

Response: Successful

```
{
  "state": {                                                 ①
  ...
  }
}
```

① The state information is returned as describe in Response Message.

Look

Look around in the world.

Request

```
{
  "robot": ...,
  "command": "look",
  "arguments": []
}
```

Response: Successful

```
{
  "result": "OK",
  "data": {
    "objects": [                                             ①
        {                                                    ②
            "direction": ...,                                ③
            "type": type of object,                          ④
            "distance": steps                                ⑤
        },
        { ... },                                             ⑥
        ...
    ]
  },
  "state": { ... }                                           ⑦
}
```

① Array: The data contains a list of objects.

② Map: Each object is a map of key-value pairs.

③ String: The direction in which the object can be found, being one of `NORTH`, `SOUTH`, `EAST`, `` `WEST ``

④ String: The type of object seen. The following object types must be supported. Note that it is possible to have other types of objects as programmed in the server world.

- `OBSTACLE` - the object is an obstacle
- `PIT` - the object is a bottomless pit
- `MINE` - the object is a mine
- `ROBOT` - the object is a robot
- `EDGE` - the object is the boundary of the world.

⑤ Integer: How far the object is from the robot in steps.

⑥ The next object in the array.

⑦ The state of the robot

Movement

Move forward or backward in the world

Request

```
{
  "robot": ...,
  "command": "forward" or "back"                              ①
  "arguments": [steps]                                        ②
}
```

① String: The command is one of `forward` or `back`.

② Integer: number of steps to move either forward or back.

Response: Successful - the robot was able to take the number of steps

```
{
  "result": "OK",
  "data": {
    "message": "Done"
  },
  "state": { ... }                                            ①
}
```

① The state should be updated with the new position.

Response: Obstacle/Robot in the way

```
{
  "result": "OK",
  "data": {
    "message": "Obstructed"
  },
  "state": {
    "position": [x,y],                                    ①
    ...
  }
}
```

① The position is set to the coordinates of the last possible step the robot was able to take before it was obstructed.

Response: Fell into a pit

```
{
  "result": "OK",
  "data": {
    "message": "Fell"
  },
  "state": {
    "status": "DEAD"                                      ①
    ...
  }
}
```

① The operational state is dead and the world has removed the robot.

Response: Stepped on a mine and survived

```
{
  "result": "OK",
  "data": {
    "message": "Mine"
  },
  "state": {
    "position": [x,y]                                     ①
    "status": "NORMAL",                                   ②
    "shields": hits,                                      ③
    ...
  }
}
```

① This is the position of the robot on the exploded mine.

② The robot is still in normal operation.

③ The number of hits left for its shield is returned.

Response: Stepped on a mine and died The robot stepped on a mine and its shields were not strong enough to sustain the damage from the mine.

```
{
  "result": "OK",
  "data": {
    "message": "Mine"
  },
  "state": {
    "status": "DEAD",                                   ①
    ...
  }
}
```

① The robot is dead.

NOTE

When a robot takes hit from stepping on a mine, then the server must send a message to the robot that was hit using the response message of the State message.

Turn

Turn left or right

Request

```
{
  "robot": ...,
  "command": "turn",
  "arguments": ["left" or "right"]                     ①
}
```

① String: Which way the robot must turn, either "left" or "right".

Response: Successful

```
{
  "result": "OK",
  "data": {
    "message": "Done"
  },
  "state": {
    "direction": direction,                             ①
    ...
  }
}
```

① String: The direction that the robot is now facing, being one of `NORTH`, `SOUTH`, `EAST`, `WEST`.

Repair

Instruct the robot to repair its shields

Request

```
{
  "robot": ...,
  "command": "repair",
  "arguments": []
}
```

Response: Successful

```
{
  "result": "OK",
  "data": {
    "message": "Done"
  },
  "state": {
    "status": "REPAIR"
    ...
  }
}
```

Reload

Instruct the robot to reload its weapons.

Request

```
{
  "robot": ...,
  "command": "reload",
  "arguments": []
}
```

Response: Successful

```
{
  "result": "OK",
  "data": {
    "message": "Done"
  },
  "state": {
    "status": "RELOAD"
    ...
  }
}
```

Mine

Instruct the robot to set a mine.

Request

```
{
  "robot": ...,
  "command": "mine",
  "arguments": []
}
```

Response: Successful

```
{
  "result": "OK",
  "data": {
    "message": "Done"
  },
  "state": {
    "status": "SETMINE"
    ...
  }
}
```

Note that the robot must move forward one step after it has set the mine. If it cannot move forward one step, then it will step on its own mine and take hits.

NOTE

When a robot takes hit from stepping on a mine, then the server must send a message to the robot that was hit using the response message of the State message.

Fire

Instruct the robot to fire its gun.

Request

```
{
  "robot": ...,
  "command": "fire",
  "arguments": []
}
```

Response: Hit another robot

```
{
  "result": "OK",
  "data": {
    "message": "Hit",
    "distance": steps,                                    ①
    "robot": name                                              ②
    "state": { ... }                                           ③
  },
  "state": {
    "shots":  shots                                            ④
  }
}
```

① Integer: The distance of the robot that was hit from this robot’s current position.

② String: The name of the robot that was hit.

③ Map: The state information of the robot that was hit as per the Response Message specification.

④ Integer: The number of shots left is updated.

NOTE

When a robot takes hit from another robot, the server must send a message to the robot that was hit using the response message of the State message.

Response: Miss

```
{
  "result": "OK",
  "data": {
    "message": "Miss"
  },
  "state": {
    "shots":  shots                                            ①
  }
}
```

① Integer: The number of shots left is updated.