# Providing Rule Predicates and Actions

## Actions and Predicates

Actions and Predicates for the rule system are implemented as ActionProvider and PredicateProvider. There are some common build in Provider in the framework. So in most cases these must not be implemented by plugins. But if you need a special provider for a plugin specific action you can implement your own.

## ActionProvider

Let's create a simple action provider that answers the question of life the universe and everything.

Each concrete ActionProvider must be a subclass of ActionProvider. The ActionProvider must implement a `parseAction`-function that is called for every part of the action string of a rule by the framework. If the ActionHandler can handle the input then it should return a info object with a ActionHandler (see below) that can execute the corresponding action.

The returned ActionHandler must be a subclass of ActionHandler and must implement a `executeAction`-function that can be called by the framework.

```
class AnswerActionHandler extends env.actions.ActionHandler

  constructor: (@framework) ->

  executeAction: (simulate) =>
    if simulate
      return Promise.resolve(__("would log 42"))
    else
      env.logger.info "42"
      return Promise.resolve(__("logged 42"))
```

The `executeAction`-function should execute the action if `simulate` is `false`. It should always return a promise that gets fulfilled with a description string, after the action was executed.

Finally you have to register your ActionProvider in the framework.

```
class YourPlugin extends env.plugins.Plugin

  init: (app, @framework, config) =>
    # [...]
    @framework.ruleManager.addActionProvider(new AnswerActionProvider(@fram
ework))
```

For a more complex example, you can take a look at the shell-execute Plugin.

## PredicateProvider

PredicateProvider follow the same pattern as ActionProvider, but the function of a PredicateHandler are different. It follows an Example PredicateProvider that is true if the value of a numeric expression is 42.

```
class AnswerPredicateProvider extends PredicateProvider

  constructor: (@framework) ->

  parsePredicate: (input, context) ->
    result = null

    allVariables = _(@framework.variableManager.getAllVariables()).map( (v)
 => v.name ).valueOf()

    M(input, context)
      .matchNumericExpression(allVariables, (next, leftTokens) =>
        next.match(' is 42', (next)  =>
          result = {
            leftTokens
            match: next.getFullMatch()
          }
        )
      )

    if result?
```

```
      assert Array.isArray result.leftTokens
      return {
        token: result.match
        nextInput: input.substring(result.match.length)
        predicateHandler: new VariablePredicateHandler(
          @framework,
          result.leftTokens
        )
      }
    else
      return null
view rawert Array.isArray result.leftTokens
      return {
        token: result.match
        nextInput: input.substring(result.match.length)
        predicateHandler: new VariablePredicateHandler(
          @framework,
          result.leftTokens
        )
      }
    else
      return null
```

The corresponding PredicateHandler:

```
class AnswerPredicateHandler extends PredicateHandler

    constructor: (@framework, @leftTokens) ->

    setup: ->
      @lastState = null
      @variables = @framework.variableManager.extractVariables(@leftTokens)
      @changeListener = () =>
        @_evaluate().then( (state) =>
          if state isnt @lastState
            @lastState = state
            @emit 'change', state
```

```
        ).catch( (error) =>
          env.logger.error "Error in AnswerPredicateHandler:", error.messag
e
          env.logger.debug error
        )
      for v in @variables
        @framework.variableManager.on("change #{v}", @changeListener)
      super()

    getValue: ->
      if @lastState? then return Q(@lastState)
      else return @_evaluate()

    destroy: ->
      for v in @variables
        @framework.variableManager.removeListener("change #{v}", @changeLis
tener)
      super()
    getType: -> 'state'
    # private helper
    _evaluate: ->
      @framework.variableManager.evaluateNumericExpression(@leftTokens).the
n( (val) =>
        return val is 42
      )
```

and finally the registration:

```
class YourPlugin extends env.plugins.Plugin

  init: (app, @framework, config) =>
    # [...]
    @framework.ruleManager.addPredicateProvider(new AnswerPredicateProvider
(@framework))
```