
TangoSpec Documentation

Release 1.1.0final0

TangoSpec development team

March 30, 2015

CONTENTS

1	Getting started	3
1.1	Download & install	3
1.2	Setup a new TangoSpec server	3
1.3	Auto discovery	4
1.4	Spec session reconstruction	5
1.5	Expose a motor	5
1.6	Expose a counter	6
1.7	Expose a variable	6
1.8	Read/Write variables	7
1.9	Run a macro	7
1.10	Move a motor	8
1.11	Count	8
1.12	Listen to output	8
2	TangoSpec API	9
	Index	15

TangoSpec is a [TANGO](#) device server which provides a [TANGO](#) interface to [SPEC](#).

GETTING STARTED

TangoSpec consists of a **TANGO** device server called *TangoSpec*. The device server should contain at least one device of **TANGO** class *Spec*.

All other devices (*SpecMotor*, *SpecCounter*) can be created dynamically on demand by executing commands on the *Spec* device.

This chapter describes how to install, setup, run and customize a new *TangoSpec* server.

1.1 Download & install

1.1.1 Dependencies

TangoSpec **TANGO** device server depends on **PyTango** and **SpecClient_gevent** packages.

1.1.2 ESRF Production environment

For production environment, use the code from the bliss installer package called *TangoSpec* (in Control/Tango/Server).

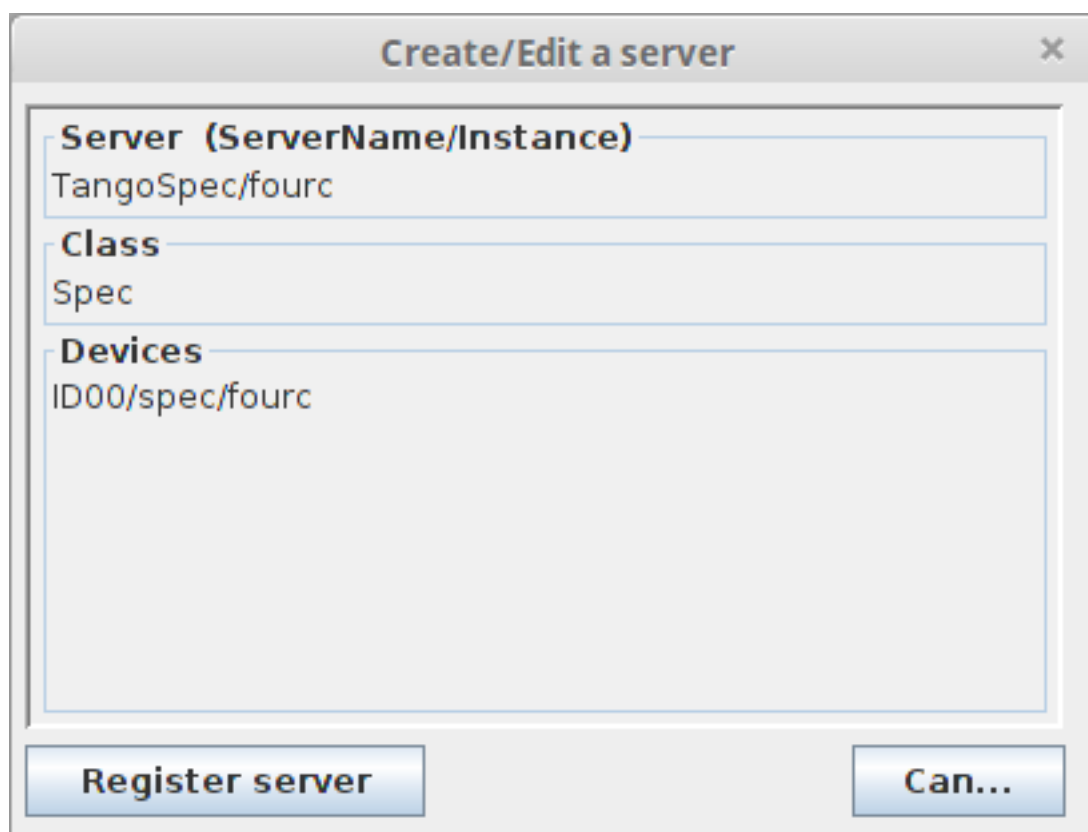
1.1.3 Development environment

For development, you can get the code from ESRF gitlab:

```
$ git clone git@gitlab.esrf.fr:spec/tango-spec-ds.git
```

1.2 Setup a new TangoSpec server

Go to jive and select *Edit* → *Create server*. You will get a dialog like the one below:



The screenshot shows a window titled "Create/Edit a server" with a close button (X) in the top right corner. Inside the window, there are three text input fields. The first field is labeled "Server (ServerName/Instance)" and contains the text "TangoSpec/fourc". The second field is labeled "Class" and contains the text "Spec". The third field is labeled "Devices" and contains the text "ID00/spec/fourc". At the bottom of the window, there are two buttons: "Register server" on the left and "Can..." on the right.

The *Server* field should be `TangoSpec/<instance>` where instance is a name at your choice (usually the name of the spec session, ex: `TangoSpec/fourc`).

The *Class* field should be `Spec`.

The *Devices* field should be the **TANGO** device name according to the convention in place at the institute (ex: `ID00/spec/fourc`).

Press *Register server*.

Select the *Server* tab, go to node `TangoSpec/<instance>/Spec/<device name>/properties`. Add a new property called *Spec* by clicking the *New property* button. Set the *Spec* property value to the spec session name (example: `machine01:fourc`).

Optional: By default, *Spec* server will start with auto discovery deactivated. This means that motors and counters will **not** be automatically added. You can changed this behavior by setting a new property called *AutoDiscovery* and setting it to `True` (See [Auto discovery](#))

Now go to the command line and type (replace *fourc* with your server instance):

```
$ TangoSpec fourc
```

1.3 Auto discovery

TangoSpec server can run with auto discovery enabled or disabled.

When auto discovery is enabled, every time the TangoSpec server starts it will synchronize the list of motors and counters with the list provided by spec. All motors and counters from spec will be automatically exposed as TANGO devices.

When auto discovery is disabled, tango motors and counters must be created manually (see [Expose a motor](#) and [Expose a counter](#)).

Auto discovery is disabled by default unless you set the `AutoDiscovery` property of the Spec device has been set to `True`.

Note: When a Spec TANGO server is running, to switch auto discovery mode, you need to change the value of the `AutoDiscovery` **and** execute the `Init` command on the Spec TANGO device to allow changes to take place.

1.4 Spec session reconstruction

It is possible to synchronize the list of TANGO spec motors and counters with the list of motors and counters provided by Spec. To do this, simply execute the `Reconstruct()` command provided by the Spec TANGO device. After executing this command all motors and counters exported by SPEC will be present as TANGO devices. Example:

```
>>> import PyTango
>>> fourc = PyTango.DeviceProxy("ID00/SPEC/fourc")

# tells you the list of existing spec motors
>>> fourc.SpecMotorList
['energy', 'ffsamz', 'ffsamz', 'istopy', 'istopz']

>>> # tells you which spec motors are exposed as tango motors
>>> fourc.MotorList
[]

>>> fourc.Reconstruct()

>>> fourc.MotorList
['energy (ID00/Spec/energy)',
 'ffsamz (ID00/Spec/ffsamz)',
 'ffsamz (ID00/Spec/ffsamz)',
 'istopy (ID00/Spec/istopy)',
 'istopz (ID00/Spec/istopz)']

>>> # now there is a Tango device of class SpecMotor for each motor in the spec session:
>>> energy = PyTango.DeviceProxy("ID00/SPEC/energy")
```

1.5 Expose a motor

Each motor in SPEC can be represented as a TANGO device of TANGO class `SpecMotor`.

When you setup a new *TangoSpec* device server it will not export any of the SPEC motors unless *auto discovery* is enabled.

To export a SPEC motor to spec just execute the TANGO command `AddMotor()` on the *Spec* device. This can be done in Jive or from a python shell:

```
>>> import PyTango
>>> fourc = PyTango.DeviceProxy("ID00/SPEC/fourc")
>>> fourc.SpecMotorList
energy
ffsamz
ffsamz
istopy
istopz

>>> # creates a SpecMotor called 'ID00/SPEC/energy' and with alias 'energy'
>>> fourc.addMotor(["energy"])
```

```
>>> energy = PyTango.DeviceProxy("energy") # or PyTango.DeviceProxy("ID00/SPEC/energy")

>>> # creates a SpecMotor called 'a/b/ffsamy' and with alias 'ffsamy'
>>> fourc.addMotor(["theta", "a/b/ffsamy"])
>>> theta = PyTango.DeviceProxy("ffsamy") # or PyTango.DeviceProxy("a/b/ffsamy")

>>> # creates a SpecMotor called 'a/b/istopy' and with alias 'spec_istopy'
>>> fourc.addMotor(["istopy", "a/b/istopy", "spec_istopy"])
>>> phi = PyTango.DeviceProxy("spec_istopy") # or PyTango.DeviceProxy("a/b/istopy")
```

1.6 Expose a counter

Each counter in **SPEC** can be represented as a **TANGO** device of **TANGO** class *SpecCounter*.

When you setup a new *TangoSpec* device server it will not export any of the **SPEC** counters unless *auto discovery* is enabled.

To export a **SPEC** counter to spec just execute the **TANGO** command *AddCounter()* on the *TangoSpec* device. This can be done in Jive or from a python shell:

```
>>> import PyTango
>>> fourc = PyTango.DeviceProxy("ID00/SPEC/fourc")
>>> fourc.SpecCounterList
sec
mon
det
c1
c2
c3

>>> # creates a SpecCounter called 'ID00/SPEC/sec' and with alias 'sec'
>>>
>>> fourc.addCounter(["sec"])
>>> sec = PyTango.DeviceProxy("sec") # or PyTango.DeviceProxy("ID00/SPEC/sec")

>>> # creates a SpecCounter called 'a/b/sec' and with alias 'sec'
>>>
>>> fourc.addCounter(["sec", "a/b/sec"])
>>> theta = PyTango.DeviceProxy("sec") # or PyTango.DeviceProxy("a/b/sec")

>>> # creates a SpecCounter called 'a/b/det' and with alias 'spec_det'
>>>
>>> fourc.addCounter(["det", "a/b/det", "spec_det"])
>>> phi = PyTango.DeviceProxy("specdet") # or PyTango.DeviceProxy("a/b/det")
```

1.7 Expose a variable

SPEC variables can be exported to **TANGO** as dynamic attributes in the *TangoSpec* device.

To expose an existing **SPEC** variable to **TANGO** just execute the **TANGO** command *AddVariable()* on the *TangoSpec* device.

As a result, a new attribute with the same name as the **SPEC** variable name will be created in the *TangoSpec* device.

Example how to expose a **SPEC** variable called *FF_DIR*:

```
>>> import PyTango
>>> fourc = PyTango.DeviceProxy("ID00/SPEC/Fourc")
```

```
>>> # expose a variable called 'FF_DIR'
>>> fourc.AddVariable("FF_DIR")
```

Note: Spec sessions can contain literally thousands of variables. For this reason neither the *auto discovery* nor the *Reconstruct()* command will expose spec variables automatically to TANGO

1.8 Read/Write variables

The new TANGO attribute will a read-write scalar string. In order to be able to represent proper data types the string is encoded in json format. In order to read the value of a SPEC variable you must first decode it from json. Fortunately, json is a well known format. Example how to read the value of a previously exposed (see chapter above) SPEC variable called *FF_DIR* (the variable is an associative array):

```
>>> import json
>>> FF_DIR = json.loads(fourc.FF_DIR)
>>> FF_DIR
{'config': u'/users/homer/Fourc/config',
 u'data': u'/users/homer/Fourc/data',
 u'sample': u'niquel'}

>>> type(FF_DIR)
dict
```

Notice that the value of *FF_DIR* is **not** a string but an actual dictionary.

To write a new value into a SPEC variable the opposite operation needs to be performed. Example:

```
>>> FF_DIR = dict(config="/tmp/config", data="/tmp/data", sample="copper")
>>> fourc.FF_DIR = json.dumps(FF_DIR)
```

1.9 Run a macro

To run a macro use the *ExecuteCmd()* command. Example:

```
>>> fourc.ExecuteCmd("wa")
```

(nothing will be shown because you are not listening to SPEC output. See *Listen to output*)

Quick macros can be ran using this synchronous method. Macros that take a long time (ex: ascan) will block the client and eventually a timeout exception will be raised (default timeout is 3s).

To run long macros there are two options:

1.9.1 Run macro asynchronously

Tell the TANGO server to start executing the macro asynchronously allowing you to do other stuff while the macro is running. For this use the command *ExecuteCmdA()*.

If you are interested you can monitor if the macro as finished (*IsReplyArrived()* command) and optionally get the result of it's execution (*GetReply()*). Example:

```
>>> ascan_id = fourc.ExecuteCmd("ascan phi 0 90 100 1.0")
>>> # do my stuff while the ascan is running...

>>> while not fourc.IsReplyArrived(ascan_id):
...     # do more stuff

>>> ascan_result = fourc.GetReply(ascan_id)
```

Note: `GetReply()` will block until the command finishes.

1.9.2 Run macro synchronously

If you want to be blocked until the macro finishes: First, configure the DeviceProxy timeout to a long time and then execute the macro using the `ExecuteCmd()` command:

```
>>> fourc.set_timeout_millis(1000*60*60*24*7) # a week
>>> ascan_result = fourc.ExecuteCmd("ascan phi 0 90 100 1.0")
```

Just make sure the ascan takes less than a week ;-)

1.10 Move a motor

Todo

write Move a motor chapter

1.11 Count

Todo

write Count chapter

1.12 Listen to output

Todo

write list to output chapter

TANGOSPEC API

A **TANGO** device server which provides a **TANGO** interface to **SPEC**.

`TangoSpec.run(**kwargs)`
Runs the Spec device server

class `TangoSpec.Spec(*args, **kwargs)`
Bases: `PyTango.server.Device`

A **TANGO** device for **SPEC** based on `SpecClient`.

Spec

TANGO device property containing spec session name (examples: `localhost:spec`, `mach101:fourc`)

AutoDiscovery

TANGO device property (bool) describing if auto discovery is enabled or disabled (see: *Auto discovery*). Default value is `False`.

OutputBufferMaxLength

TANGO device property (int) describing the output history buffer maximum length (in number of output lines). Default is 1000 lines.

SpecMotorList

TANGO attribute containing the list of all **SPEC** motors

SpecCounterList

TANGO attribute containing the list of all **SPEC** counters

MotorList

TANGO attribute containing the list of **SPEC** motors exported to **TANGO**

CounterList

TANGO attribute containing the list of **SPEC** counters exported to **TANGO**

VariableList

TANGO attribute containing the list of **SPEC** variables exported to **TANGO**

Output

TANGO attribute which reports **SPEC** console output (output/tty variable)

ExecuteCmd(*args, **kwargs)

Execute a **SPEC** command synchronously. Use `ExecuteCmdA()` instead if you intend to run commands that take some time.

Parameters `command` (*str*) – the command to be executed (ex: `"wa"`)

ExecuteCmdA(*args, **kwargs)

Execute a **SPEC** command asynchronously.

Parameters `command` (*str*) – the command to be executed (ex: `"ascan energy 0.1 10 20 0.1"`)

Returns an identifier for the command.

Return type `int`

GetReply (**args*, ***kwargs*)

Returns the reply of the `SPEC` command given by the `cmd_id`, previously requested through `ExecuteCmdA()`. It waits if the command is not finished

Parameters `cmd_id` (*int*) – command identifier

Returns the reply for the requested command

Return type `str`

IsReplyArrived (**args*, ***kwargs*)

Determines if a command executed previously with the given `cmd_id` is finished.

Parameters `cmd_id` (*int*) – command identifier

Returns True if the command response as arrived or False otherwise

Return type `bool`

AbortCmd (**args*, ***kwargs*)

Aborts the command in execution given by the `cmd_id`, previously requested through `ExecuteCmdA()`.

Parameters `cmd_id` (*int*) – command identifier

AddVariable (**args*, ***kwargs*)

Export a `SPEC` variable to Tango by adding a new attribute to this device with the same name as the variable.

Parameters `variable_name` (*str*) – `SPEC` variable name to be exported as a `TANGO` attribute

Throws `PyTango.DevFailed` If the variable is already exposed in this `TANGO` DS.

RemoveVariable (**args*, ***kwargs*)

Unexposes the given variable from this device.

Parameters `variable_name` (*str*) – the name of the `SPEC` variable to be removed

Throws `PyTango.DevFailed` If the variable is not exposed in this `TANGO` DS

AddMotor (**args*, ***kwargs*)

Adds a new `SpecMotor` to this DS.

Parameters `motor_info` (*sequence<str>*) – sequence of strings with the following syntax: `spec_motor_name` [, `tango_device_name` [, `tango_alias_name`]]

Examples:

```
spec = PyTango.DeviceProxy("ID00/spec/fourc")
spec.AddMotor(("th",))
spec.AddMotor(("tth", "ID00/fourc/tth", "theta2"))
```

spec_motor_name name of the spec motor to export to `TANGO`

tango_device_name optional tango name to give to the new `TANGO` motor device [default: `<tangospec_domain>/<tangospec_family>/<spec_motor_name>`]

tango_alias_name optional alias to give to the new tango motor device [default: `<spec_motor_name>`]. Note: if the alias exists it will **not** be overwritten.

Throws `PyTango.DevFailed` If `SPEC` motor does not exist or if motor is already exported

RemoveMotor (**args*, ***kwargs*)

Removes the given `SpecMotor` from this DS.

Parameters `motor_name` (*str*) – `SPEC` motor name to be removed

Examples:

```
spec = PyTango.DeviceProxy("ID00/spec/fourc")
spec.RemoveMotor("th")
```

Throws PyTango.DevFailed If motor does not exist

AddCounter (*args, **kwargs)

Adds a new SpecCounter to this DS.

Parameters **counter_info** (*sequence<str>*) – sequence of strings with the following syntax: spec_counter_name [, tango_device_name [, tango_alias_name]]

Examples:

```
spec = PyTango.DeviceProxy("ID00/spec/fourc")
spec.AddCounter(("sec",))
spec.AddCounter(("det", "ID00/fourc/detector", "detector"))
```

spec_counter_name name of the spec counter to export to TANGO

tango_device_name optional tango name to give to the new TANGO counter device [default: <tangospec_domain>/<tangospec_family>/<spec_counter_name>]

tango_alias_name optional alias to give to the new tango counter device [default: <spec_counter_name>]. Note: if the alias exists it will **not** be overwritten.

Throws PyTango.DevFailed If SPEC counter does not exist or if counter is already exported

RemoveCounter (*args, **kwargs)

Removes the given SpecCounter from this DS.

Parameters **counter_name** (*str*) – SPEC counter name to be removed

Examples:

```
spec = PyTango.DeviceProxy("ID00/spec/fourc")
spec.RemoveCounter("th")
```

Throws PyTango.DevFailed If counter does not exist

Reconstruct (*args, **kwargs)

Exposes to Tango all counters and motors that were found in SPEC.

class TangoSpec.SpecMotor (*cl, name*)

Bases: PyTango.server.Device

A TANGO SPEC motor device based on SpecClient.

SpecMotor

TANGO device property containing the spec motor mnemonic (examples: th, localhost:spec::chi, mach101:fourc::phi). The full name is only required if running the TangoSpec DS without a Spec manager device.

Position

TANGO attribute for the motor user position. Setting a value on this attribute will move the motor to the specified value.

State

TANGO attribute for the motor state.

- INIT - motor initialization phase (startup or through Init command)

- ON - motor is enabled and stopped.
- MOVING - motor is moving
- ALARM - motor limit switch is active or position in the alarm range
- FAULT - connection to [SPEC](#) motor lost

Status

[TANGO](#) attribute for the motor status.

DialPosition

[TANGO](#) attribute for the motor dial position.

Sign

[TANGO](#) attribute for the motor sign.

Offset

[TANGO](#) attribute for the motor offset.

AcceletationTime

[TANGO](#) attribute for the motor acceleration time (s).

Backlash

[TANGO](#) attribute for the motor backlash.

StepSize

[TANGO](#) attribute for the current step size (used by the StepDown and StepUp commands).

Limit_Switches

[TANGO](#) attribute for the motor limit switches (home, upper, lower).

Init ()

Initializes the [TANGO](#) motor

Stop (*args, **kwargs)

Stop the motor (allowing deceleration time)

Abort (*args, **kwargs)

Stop the motor immediately

Move (*args, **kwargs)

Move the motor to the given absolute position

Parameters `abs_position` (*float*) – absolute destination position

MoveRelative (*args, **kwargs)

Move the motor by the given displacement.

Parameters `rel_position` (*float*) – displacement

StepUp (*args, **kwargs)

Move the motor up by the currently configured step size

StepDown (*args, **kwargs)

Move the motor down by the currently configured step size

class `TangoSpec.SpecCounter` (*cl, name*)

Bases: `PyTango.server.Device`

A TANGO SPEC counter device based on SpecClient.

SpecCounter

TANGO device property containing the spec counter mnemonic (examples: `sec`, `localhost:spec::det`, `mach101:fourc::mon`). The full name is only required if running the TangoSpec DS without a Spec manager device.

State

TANGO attribute for the counter state.

- INIT - counter initialization phase (startup or through Init command)
- ON - counter is enabled and stopped.
- RUNNIG - counter is counting
- ALARM - counter value in the alarm range
- FAULT - connection to **SPEC** counter lost

Status

TANGO attribute for the counter status.

Value

TANGO attribute for the counter value.

Init ()

Initializes the **TANGO** counter

Count (*args, **kwargs)

Count by the specified time (s)

Parameters `count_time` – count time (s)

Stop (*args, **kwargs)

Stop counting

setEnabled (*args, **kwargs)

Enable/Disable counter

Parameters `enabled` (*bool*) – enable or disable

A

Abort() (TangoSpec.SpecMotor method), 12
 AbortCmd() (TangoSpec.Spec method), 10
 AccelationTime (TangoSpec.SpecMotor attribute), 12
 AddCounter() (TangoSpec.Spec method), 11
 AddMotor() (TangoSpec.Spec method), 10
 AddVariable() (TangoSpec.Spec method), 10
 AutoDiscovery (TangoSpec.Spec attribute), 9

B

Backlash (TangoSpec.SpecMotor attribute), 12

C

Count() (TangoSpec.SpecCounter method), 13
 CounterList (TangoSpec.Spec attribute), 9

D

DialPosition (TangoSpec.SpecMotor attribute), 12

E

ExecuteCmd() (TangoSpec.Spec method), 9
 ExecuteCmdA() (TangoSpec.Spec method), 9

G

GetReply() (TangoSpec.Spec method), 10

I

Init() (TangoSpec.SpecCounter method), 13
 Init() (TangoSpec.SpecMotor method), 12
 IsReplyArrived() (TangoSpec.Spec method), 10

L

Limit_Switches (TangoSpec.SpecMotor attribute), 12

M

MotorList (TangoSpec.Spec attribute), 9
 Move() (TangoSpec.SpecMotor method), 12
 MoveRelative() (TangoSpec.SpecMotor method), 12

O

Offset (TangoSpec.SpecMotor attribute), 12
 Output (TangoSpec.Spec attribute), 9
 OutputBufferMaxLength (TangoSpec.Spec attribute), 9

P

Position (TangoSpec.SpecMotor attribute), 11

R

Reconstruct() (TangoSpec.Spec method), 11
 RemoveCounter() (TangoSpec.Spec method), 11
 RemoveMotor() (TangoSpec.Spec method), 10
 RemoveVariable() (TangoSpec.Spec method), 10
 run() (in module TangoSpec), 9

S

setEnabled() (TangoSpec.SpecCounter method), 13
 Sign (TangoSpec.SpecMotor attribute), 12
 Spec (class in TangoSpec), 9
 Spec (TangoSpec.Spec attribute), 9
 SpecCounter (class in TangoSpec), 12
 SpecCounter (TangoSpec.SpecCounter attribute), 12
 SpecCounterList (TangoSpec.Spec attribute), 9
 SpecMotor (class in TangoSpec), 11
 SpecMotor (TangoSpec.SpecMotor attribute), 11
 SpecMotorList (TangoSpec.Spec attribute), 9
 State (TangoSpec.SpecCounter attribute), 13
 State (TangoSpec.SpecMotor attribute), 11
 Status (TangoSpec.SpecCounter attribute), 13
 Status (TangoSpec.SpecMotor attribute), 12
 StepDown() (TangoSpec.SpecMotor method), 12
 StepSize (TangoSpec.SpecMotor attribute), 12
 StepUp() (TangoSpec.SpecMotor method), 12
 Stop() (TangoSpec.SpecCounter method), 13
 Stop() (TangoSpec.SpecMotor method), 12

T

TangoSpec (module), 9

V

Value (TangoSpec.SpecCounter attribute), 13
 VariableList (TangoSpec.Spec attribute), 9