



orkhestra: Configuration and User Reference
Version 1

CML00041-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com
Copyright © 2014 by Code Magus Limited
All rights reserved



November 30, 2018

Contents

1	Introduction	4
1.1	Overview	4
2	Environmental Variables	5
3	Configuration	6
3.1	Command interface	6
3.2	Command Elements	6
3.2.1	Comments	6
3.2.2	Reserved Words	7
3.2.3	Identifiers	7
3.2.4	Strings	7
3.2.5	Filenames	8
3.2.6	Integers	8
3.2.7	Environment Variables	8
3.3	Command Syntax and Semantics	9
3.3.1	General Command Syntax	9
3.3.2	Command Options	11
3.3.3	Alter Command	13
3.3.4	Cancel Command	18
3.3.5	Close Command	18
3.3.6	Connect Command	18
3.3.7	Define Command	19
3.3.8	Display command	21
3.3.9	Exit Command	32
3.3.10	Flush Command	32
3.3.11	Help Command	33
3.3.12	Load Command	33
3.3.13	Mark Command	33
3.3.14	Open Command	34
3.3.15	Send Command	35
3.3.16	Set and Reset Commands	36
3.3.17	Shutdown Command	36
3.3.18	Start Command	36
3.3.19	Stop Command	37
3.3.20	Switch Command	38
4	Dashboard Metrics	39
4.1	Introduction	39
4.2	Elements	39
4.2.1	Comments	39
4.2.2	Reserved Words	40
4.2.3	Identifiers	40
4.2.4	Strings	40
4.2.5	Integers	40

4.3	Metric Definition	41
5	State Machine definition	45
5.1	Introduction	45
5.2	Elements	45
5.2.1	Comments	45
5.2.2	Reserved Words	47
5.2.3	Identifiers	47
5.2.4	Strings	47
5.2.5	Integers	47
5.3	Machine Definition	48
5.3.1	Preamble	49
5.3.2	Declarations	52
5.3.3	State Definition	56
5.4	Internal Functions	58
5.4.1	Choose Function	58
5.4.2	Start Timer Function	59
5.4.3	Cancel Timer Function	60
5.4.4	Start Machine Function	61
6	Remote Control Programs	62
6.1	Overview	62
6.2	Clock Synchronisation	63
6.3	Local Agent	63
6.3.1	Operation	64
6.3.2	Shutdown	65
6.3.3	Error Shutdown	65
6.4	Remote Agent	65
6.4.1	Synopsis	66
6.4.2	Operation	66
6.4.3	Shutdown	66
6.4.4	Error Shutdown	66
7	Agent Configuration File	67
7.1	Elements	67
7.1.1	Comments	67
7.1.2	Reserved Words	67
7.1.3	Identifiers	67
7.1.4	Strings	68
7.1.5	Environment Variables	68
7.2	Syntax and Semantics	68
7.2.1	Remote Agent definition	69
A	Sample State machine: orksample.mch	72

B Sample Agent Configuration File: `orkhestra_agent.cfg`

75

1 Introduction

1.1 Overview

orkhestra loads a defined external protocol control program and interacts with it (Refer to *orkhestra: Control Program API Reference Version 1[1]*) under the control of a *State Machine* (see section 5 on page 45), during which it accumulates or records metrics about the complete process. It also implements a command interface through which the metrics and definitions may be viewed or execution of the *orkhestra* environment may be dynamically reconfigured.

orkhestra is very well suited, but not limited, to network testing and/or simulation. As a simplistic example, for a network implementation, when the control program receives a network transaction, it supplies the relevant input to the *State Machine* and the *State Machine* transitions to a new state. The control program then receives a new output from the *State Machine*, and consequently sends a response to the original transaction. This continues until the state machine reaches a final state or is shut down by command.

Refer to figure 1 on page 4 for an overview of the *orkhestra* environment.

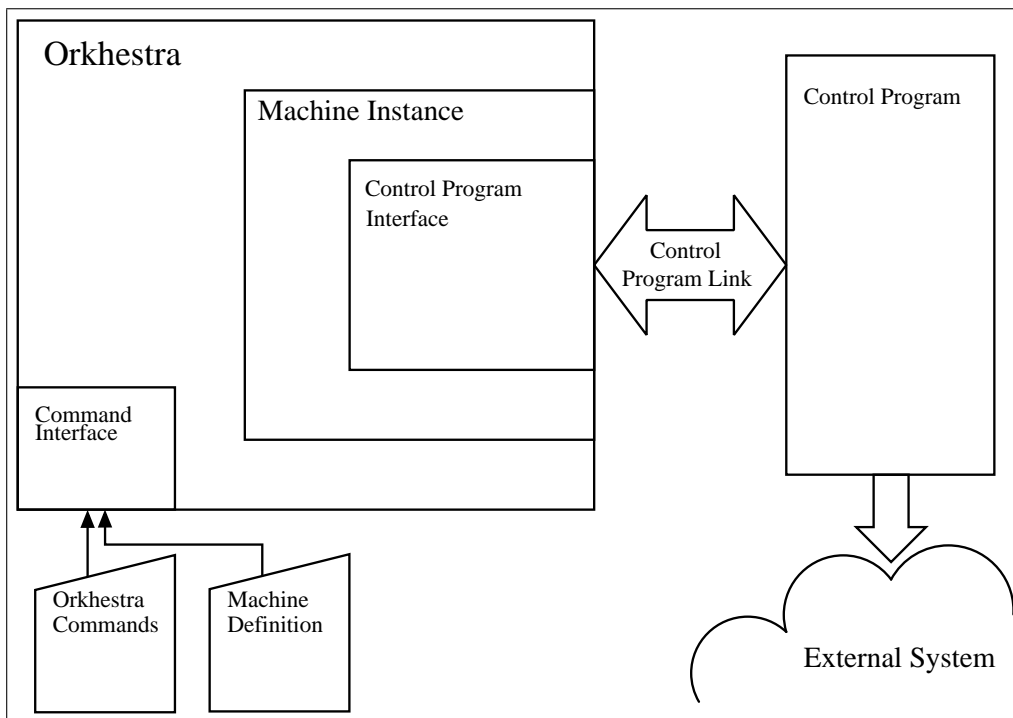


Figure 1: Main Process Flow of the *orkhestra* Environment.

2 Environmental Variables

The following environmental variables are required by *orkhestra*:

- CODEMAGUS_HOME

Specify the path for the Codemagus software installation..

3 Configuration

The configuration of control programs, state machines and parameters are done via commands to *orkhestra*. The metrics and definitions may also be viewed or execution of the *orkhestra* environment may be dynamically reconfigured via commands. This section describes the *orkhestra* command interface and commands.

3.1 Command interface

orkhestra is configured from commands via its command interface. There are various ways to present commands to the command interface. They are:

- **Standard input**
When *orkhestra* is run in the fore ground, the user directly communicates with the *orkhestra* command interface via the terminal. Note that if *orkhestra* is started in the background immediately; this channel is no longer available; even if *orkhestra* is subsequently brought back to the foreground.
- **Network**
A TCP/IP interface to the *orkhestra* command port. Note the TCP/IP port must have been configured (see 3.3.14 on page 34).
- **Command File.**
A file containing commands. If an unrecognised input is passed to *orkhestra*, it will assume that this could be the name of a file containing commands. It will attempt to open this file and process it.
- **Command Line Parameter**
A single command at start up, for example:

```
orkhestra -c ods_qa_cmd
```


ods_qa_cmd is a file containing commands for *orkhestra*.

3.2 Command Elements

The elements of the commands to *orkhestra* comprise reserved words, identifiers, string literals, comments and integers. The commands are free format and white spaces have no grammatical meaning except where they might appear within string literals.

3.2.1 Comments

Comments are introduced by using a hash ('#') and continue up to the end of the current input line.

Examples:

```
# File: xml_ncacrag_qa.cmd
#
# ncacrag control program parameters.
#
```

3.2.2 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of commands. The reserved words are:

all	alter	cancel	class
close	cmdinterface	cmdparser	constant
copy	copies	control_program	connect
dashboard	define	delay	deviation
display	distribution	echo	exception
exit	shutdown	flush	group
help	instance	instances	input
load	log	machine	max
mean	metric	metric-group	min
minmax	mu	open	on
option	options	output	path
parameter	port	refresh	repeat
repeat_command	reset	rule	send
set	start	state	stats
statistic	statistics	statsonall	stop
switch	transition	transitions	value
verbose	weight	weights	weight_distribution

Table 1: *orkhestra* reserved words

3.2.3 Identifiers

An *Identifier* is case sensitive, it starts with a letter which can be followed by any number of letters, digits or the under-score character.

Examples:

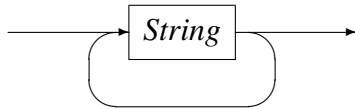
```
think_time ncacrag_123 RecordStaffArrgmntDet
```

3.2.4 Strings

Strings are:

- any sequence of characters (except double quotes and the newline character) enclosed by double quotes.
- any sequence of characters (except single quotes and the newline character) enclosed by single quotes.

Strings cannot span source text lines, but they may be concatenated:

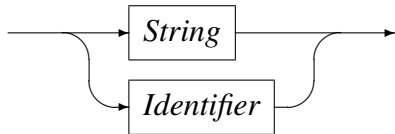


Examples:

```
"GigabitEthernet0/0 In Octets"  
'$Revision: 1.2 $'
```

3.2.5 Filenames

Filename



A *Filename* is usually written as a *String* but may also be an *Identifier*. Most importantly a *Filename* must conform to any constraints of the underlying file system.

3.2.6 Integers

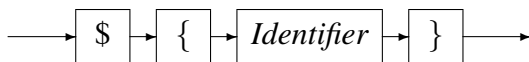
A *Integer* consists of a nonempty sequence of decimal digits ‘0’ through ‘9’.

Examples:

```
1234  
0
```

3.2.7 Environment Variables

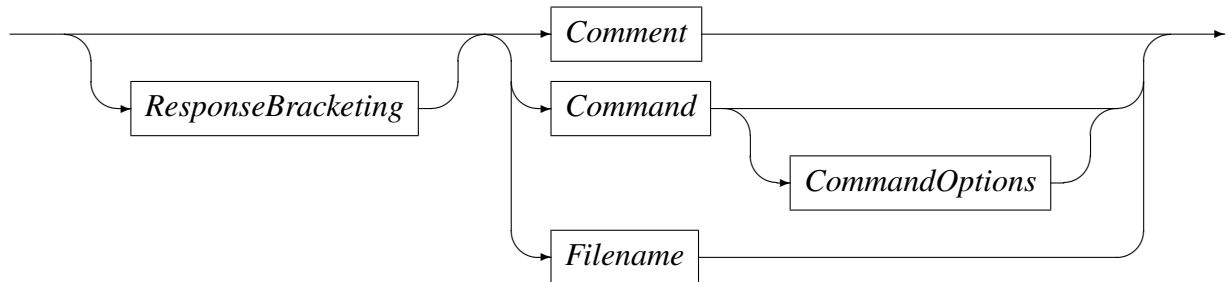
EnvironmentVariable



Environment variables are expanded to their value when encountered in command input text.

3.3 Command Syntax and Semantics

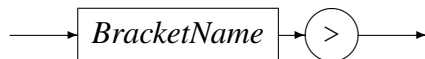
3.3.1 General Command Syntax



Input to the command processor is either:

- A *Comment*. The whole line is ignored by the command processor, see subsection 3.2.1 on page 6.
- A *Command*, optionally followed by command options.
- A *Filename*. If the input is not recognised, the command processor will assumed that this could be the name of a file containing commands, it will attempt to open this file and process it.

ResponseBracketing



BracketName can be any character except the ‘>’ character. The response to the *orkhes-tra* command will be preceded by ‘.begin BracketName;’ and followed by a new-line and ‘.end BracketName;’.

Example:

Display a machine value with BracketName ‘Disp think_time’:

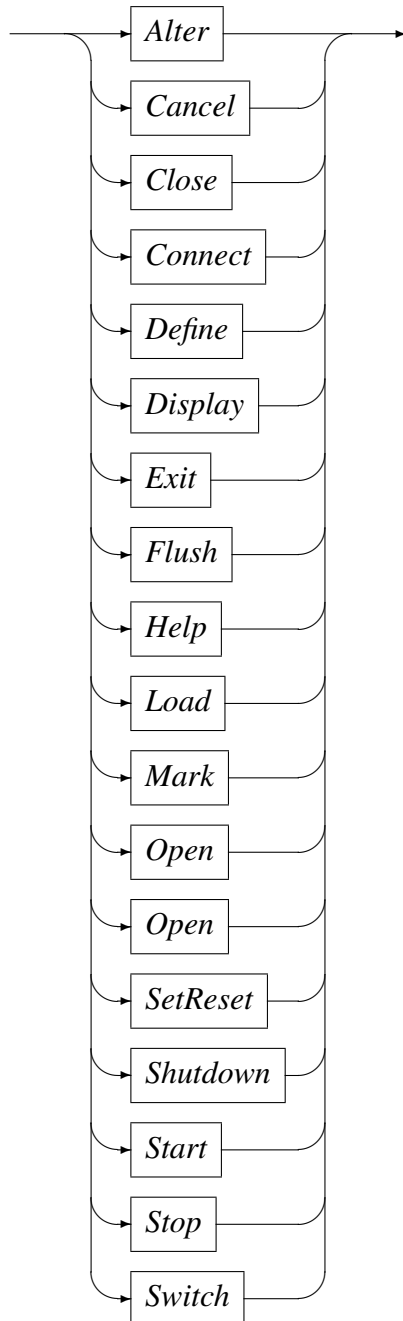
```

Disp think_time> display value think_time(machine(posdev) group(auths))

.begin Disp think_time;
value think_time
(
  machine(posdev)
  group(auths)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(exponential) min(10000) max(3000000) mu(0))
);

.end Disp think_time;
  
```

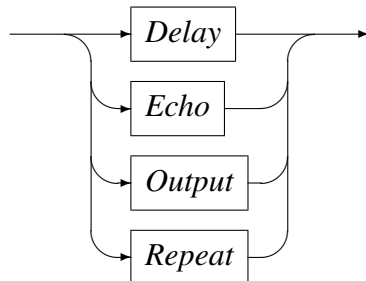
command



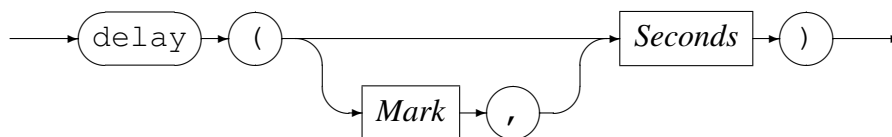
All commands can be followed by zero or more of the command options. These options affect the way in which the command is executed; for example by delaying the command or causing it to be repeated at intervals.

3.3.2 Command Options

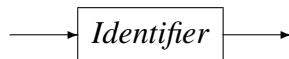
CommandOptions



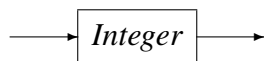
Delay



Mark



Seconds



Delay the execution of a command by *Seconds* or delay the execution of a command as per the time specified in *Mark* plus *Seconds*. Refer to 3.3.13 on page 33 for information about the *Mark* specification.

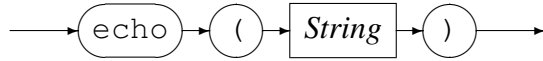
Examples:

Start machine `bici` after 10 seconds:

```
start machine bici delay(10)
```

As another example, assume it is required to run a machine and increment the number of running instances every 5 minutes on a 5 minute boundary. This allows for set time periods from a specific start point. In the example below the next 5 minute boundary is calculated using the *mark* command and associated with the label `MT300A`. The machine is altered to a various number of instances at specific time boundaries relative to the time specified in `MT300A`.

```
mark time MT300A next(300)
alter machine m1 (group(g1) instances(100)) delay(MT300A,0)
alter machine m1 (group(g1) instances(200)) delay(MT300A,300)
alter machine m1 (group(g1) instances(300)) delay(MT300A,600)
alter machine m1 (group(g1) instances(400)) delay(MT300A,900)
alter machine m1 (group(g1) instances(500)) delay(MT300A,1200)
```

Echo

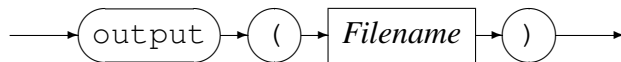
Causes the output of the associated command to be preceded with ‘begin_echo=String;’ and ended with ‘{end_echo=String;’, where *String* is the *String* specified on the echo option.

Example:

Display a machine *value* with *echo* ‘Disp think_time’:

```
display value think_time(machine(posdev) group(auths)) echo("Disp think_time")

begin_echo="Disp think_time";
value think_time
(
  machine(posdev)
  group(auths)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(exponential) min(10000) max(3000000) mu(0))
);
end_echo="Disp think_time";
```

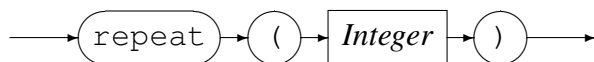
Output

Redirect the output responses for the associated command to the file specified by *Filename* (see subsection 3.2.5 on page 8). If the file exists, the response will be appended to the file, otherwise the file will be created.

Example:

Write the command response to the file `${STATSPATH}/tmsods.stats.500.txt`.

```
display statistics machine bsbic group(bsbic) reset(minmax)
  repeat(60) output("${STATSPATH}/bsbiciso_cc.stats.raw")
```

Repeat

Repeat the command every *Integer* seconds.

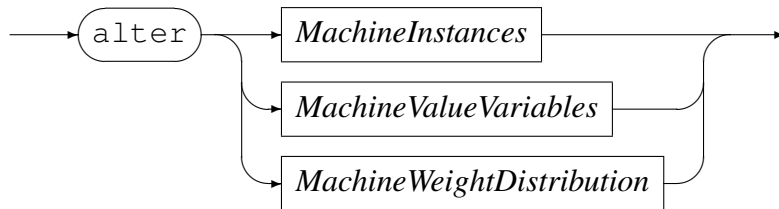
Example:

Write the command response to the file `${STATSPATH}/tmsods.stats.500.txt` and repeat the command every 30 seconds.

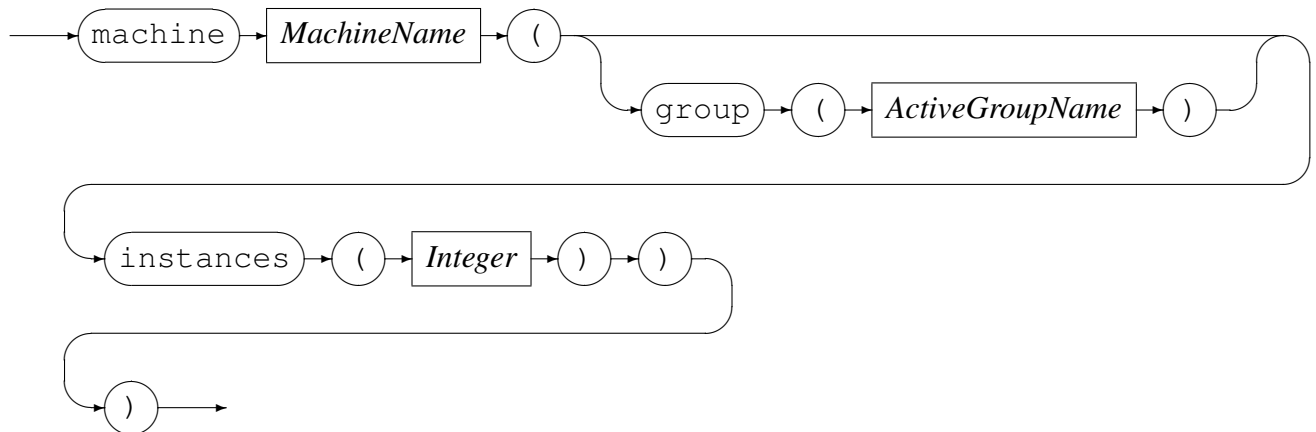
```
display statistics machine tmsods group(auths) reset(minmax)
  output("${STATSPATH}/tmsods.stats.500.txt")
  repeat(30)
```

3.3.3 Alter Command

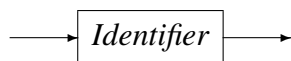
The *alter* command is used to alter machine *instances*, *values* and *weight* distributions.



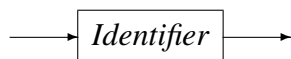
MachineInstances



MachineName



ActiveGroupName



Alter the number of instances:

- For a defined machine, this is the number of instances to start when starting this machine.
- For an active group, dynamically alter the instances in this group.

If the number of machines is altered to zero, then the machine is effectively suspended and no transition metrics are generated.

Examples:

- Alter the number of instances for the defined machine `posdev` to 100:

```
alter machine posdev (instances(100))
```

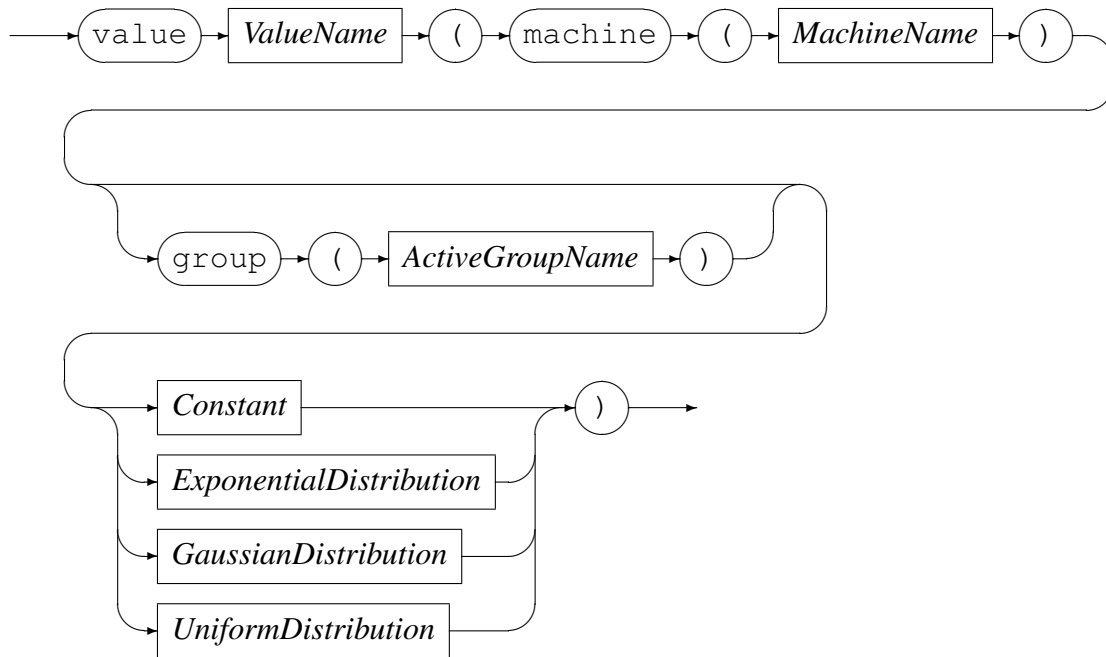
```
Response: Machine posdev instances changed from 1 to 100;
```

- Alter the number of instances for the active machine `posdev`, group `auths` to 222:

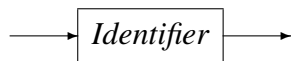
```
alter machine posdev (group(auths) instances(222))
```

```
Response: Machine posdev group(auths) instances changed from 100 to 222;
```

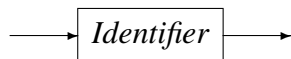
MachineValueVariables



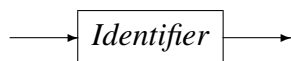
MachineName



ActiveGroupName



ValueName

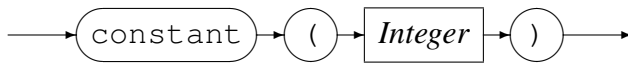


Alter the parameters of a machines *value* variable:

- For a defined machine, this is what a machine will be starting with.
- For an active group, dynamically alter the parameters of the *value*.

A *value* is used by the *State Machine* for the *start_timer()* function, and can have be any one of the following:

- *Constant*



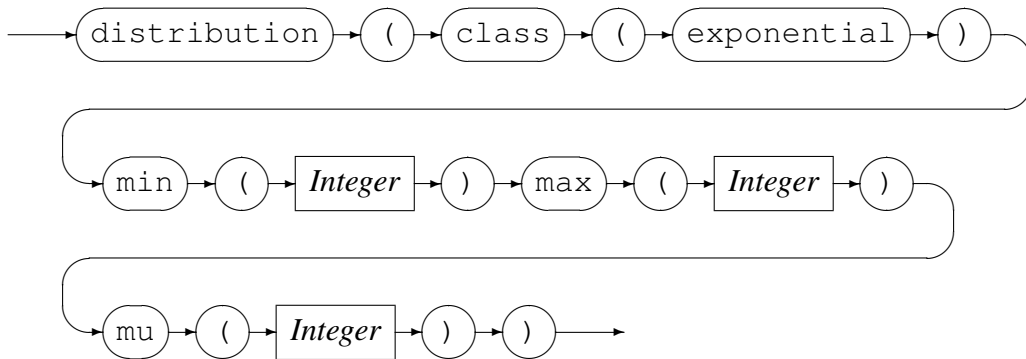
A (Constant) is specified in milliseconds.

Example:

Alter the value of `think_time` for the active machine `posdev`, group `auths` to 5 seconds:

```
alter value think_time(machine(posdev) group(auths) constant(5000))
value think_time
(
  machine(posdev)
  group(auths)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  constant(5000)
);
```

- *ExponentialDistribution*

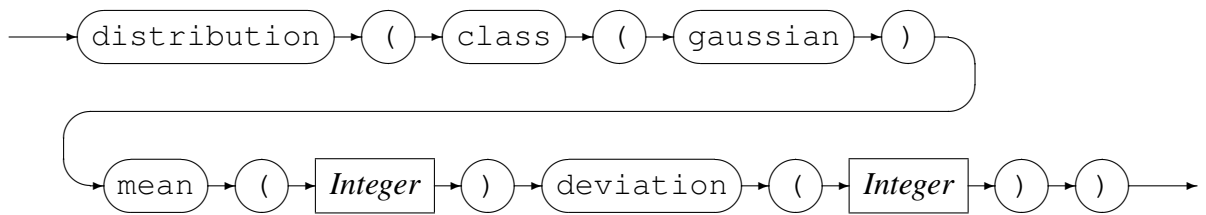


Example:

Alter the value of `think_time` for the active machine `posdev`, group `auths` to a exponential distribution:

```
alter value think_time(machine(posdev) distribution(class(exponential)
  min(10000) max(3000000) mu(100000)))
value think_time
(
  machine(posdev)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(exponential) min(10000) max(3000000) mu(100000))
);
```

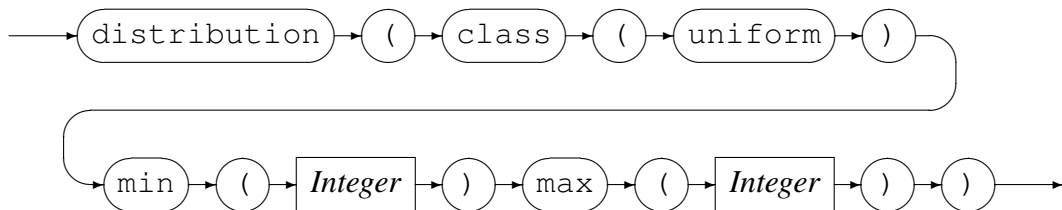
- *GaussianDistribution*

**Example:**

Alter the value of think_time for the active machine posdev, group auths to a gaussian distribution:

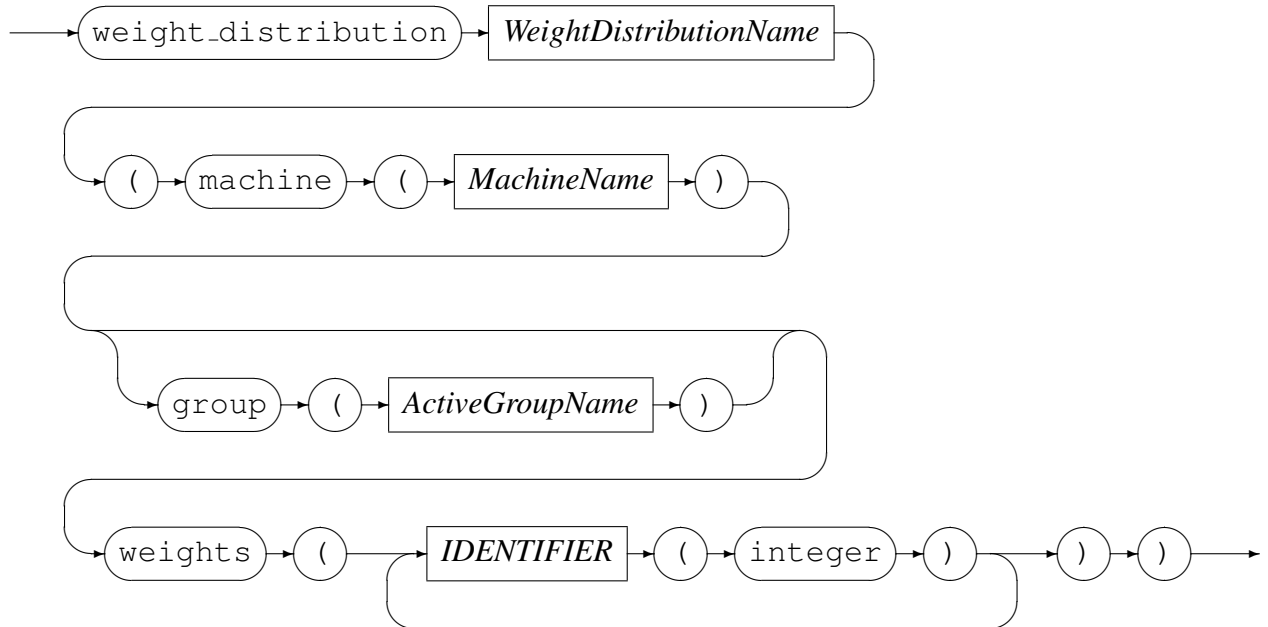
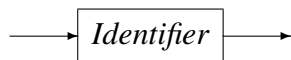
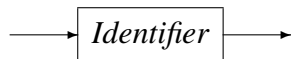
```
alter value think_time(machine(posdev) distribution(class(gaussian)
    mean(10000) deviation(2000)))
value think_time
(
  machine(posdev)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(gaussian) mean(10000) deviation(2000))
);
```

- *UniformDistribution*

**Example:**

Alter the value of think_time for the active machine posdev, group auths to a uniform distribution:

```
alter value think_time(machine(posdev) distribution(
    class(uniform) min(90000) max(110000)))
value think_time
(
  machine(posdev)
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(uniform) min(90000) max(110000))
);
```

MachineWeightDistribution*MachineName**ActiveGroupName**WeightDistributionName*

Alter the parameters of a machine's machine weight distribution:

- For a defined machine, this is what a machine will be starting with.
- For an active group, dynamically alter the parameters of the weight distribution.

A weight distribution is used by the *State Machine* for the `choose()` function.

Example:

Alter the weight distribution `what_transaction` for the active machine `posdev`, group `auths`:

```

alter weight_distribution what_transaction(machine(posdev) group(auths)
    weights(credit_card(4) debit_card(1)))
weight_distribution what_transaction
(
    machine(posdev)
    group(auths)
  
```

```

title("Transaction profile")
description("Ratio between the various transactions")
weight(logon_only(0) credit_card(4) debit_card(1) download_bin(0)
  download_new_hotcard(0) download_software(0) )
);

```

3.3.4 Cancel Command



Cancel a repeat command. The number of the command to cancel can be found by displaying the repeat commands (see section 3.3.8 on page 21).

Example:

Firstly display all the repeat commands and then delete the repeat of display control_program:

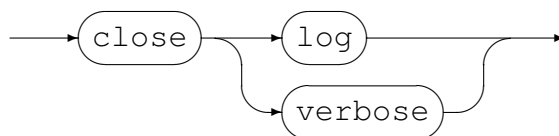
```

display repeat_command
Response:  1: display instances(machine(posdev) group(auths) repeat(10)
           output("test1.txt");
Response:  2: display control_program all repeat(11);

cancel repeat_command 2
Response: Repeat command number 2 deleted;

```

3.3.5 Close Command



Close the current open log or verbose file.

Example:

Close the current log file:

```

close log
Response: Closed output log file;

```

3.3.6 Connect Command



Connect to a GUI dashboard with the given IP address and port number.

Example:

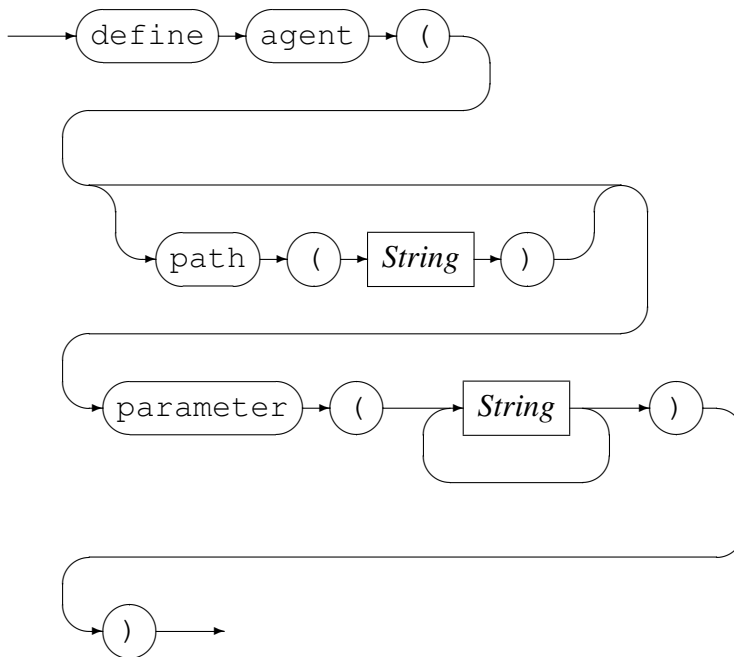
Connect to a *DashBoard* server on the same machine listening on port 62345:

```
connect dashboard localhost:62345
Response: Connection requested;
```

3.3.7 Define Command

This command is used for:

- Defining a agent.



Define a agent for running control programs remotely, see section 3.3.7 on page 19.

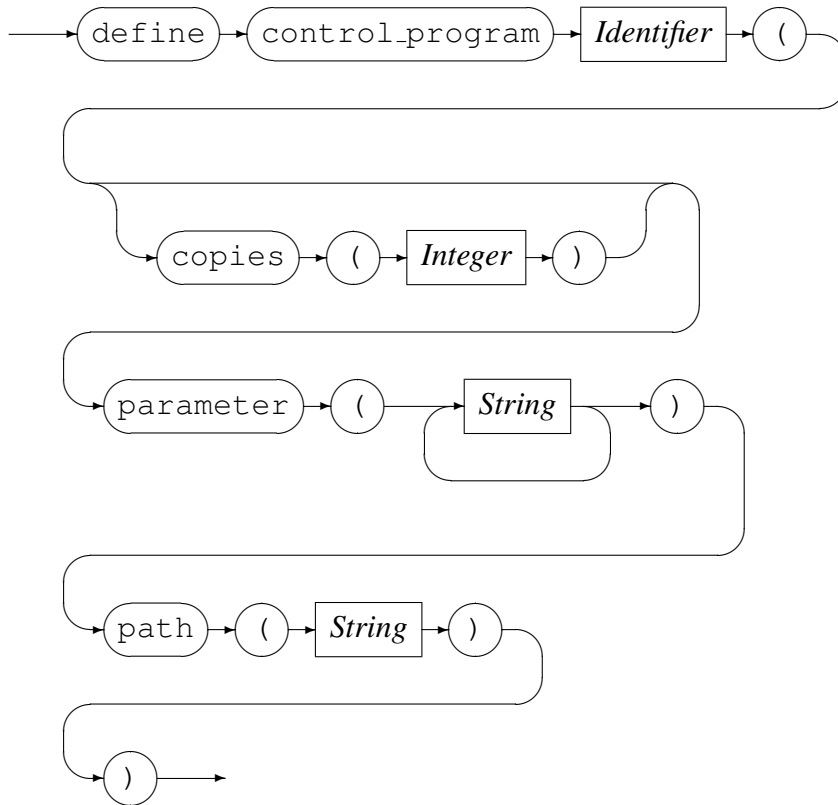
path is optional and is the agent program name. This name can be specified as a fully qualified *path* or just the name, in which case the environment variable CODEMAGUS_HOME will be used to determinate the *path*. If *path* is not specified, the agent program name defaults to *orkagent1*. The *parameter* will be passed to the control program via the command line when it is invoked.

Example:

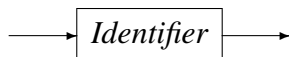
define an agent with a fully qualified path name:

```
define agent (
  path ("${HOME}/bin/orkagent1")
  parameter(
    "-v "
    "--config=${SCRIPTS}/lagent.cfg "
  )
)
```

- Defining a control program.



ControlProgramName



Define a *orkhestra* control program for processing the outputs from a *State Machine* and sending inputs to a *State Machine*.

copies is the number of instances to start when the control program is activated. the default is one copy. The *parameter* will be passed to the control program via the command line when it is invoked. *path* is the fully qualified name of the control program.

Example:

Define control program *posdev*, to be started with two copies:

```

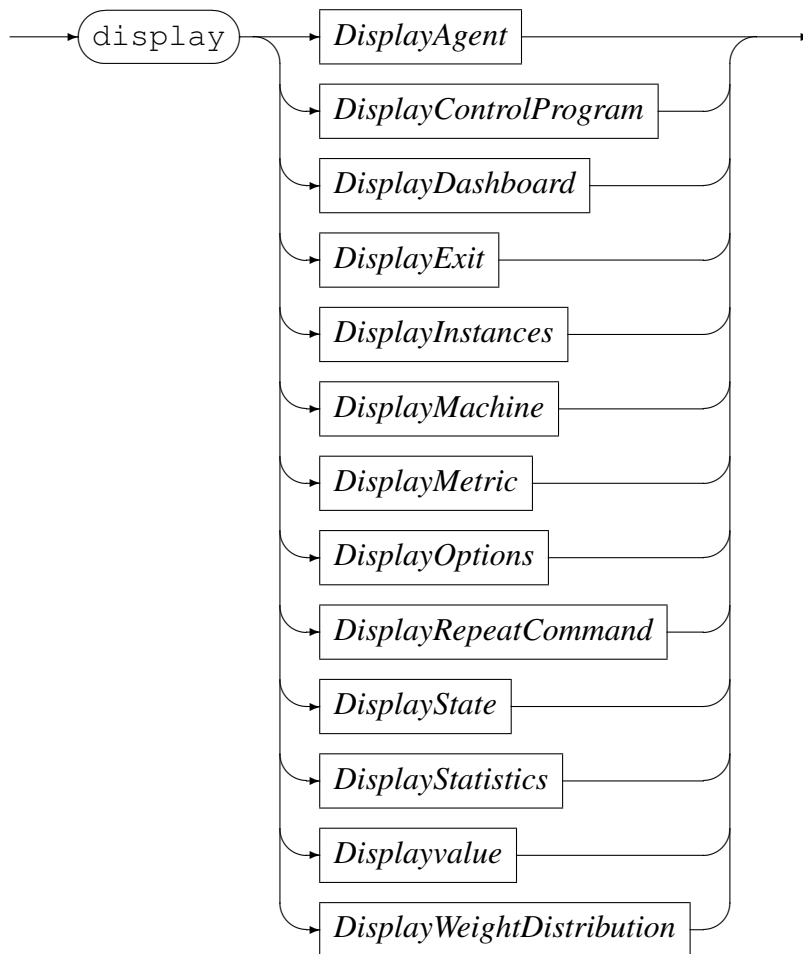
define control_program posdev (
  copies (2)
  path ("${HOME}/bin/posdcp")
  parameter (
    "--host-addr-file=${CONFIGS}/test_pos_host.txt "
    "--objtype=${POSFORMATS}/objtypes/pos_device.objtypes "
    "--mtemplate=${POSTEST}/logs/pos_device_msg.bin "
    "--merchant=${POSTEST}/scripts/merchants_qa.csv "
  )
)
  
```

```

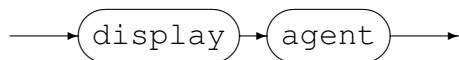
    "--pcard=${CONFIGS}/test_plastics.txt "
    "--hsm-port=7171 --hsm-name=10.57.24.23 "
    "--log=${LOGSPATH}/posdcp " )
)
Response: Control program posdev defined;

```

3.3.8 Display command



DisplayAgent



Display the definition and status of a *orkhestra* control program agent.

Example:

```

display agent
agent lagent
(

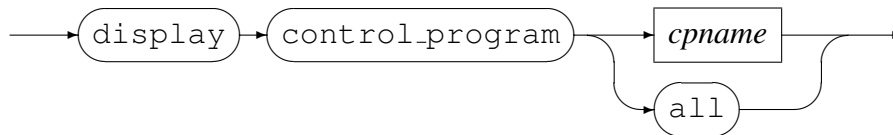
```

```

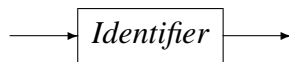
Copies(1) status(inactive)
path("../orkagent1")
Parameter(
    "-v "
    "--config=${SCRIPTS}/lagent.cfg ")
);

```

DisplayControlProgram



cpname



Display the definition and status of a control program.

Example:

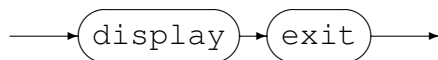
Display the definition and status of the control program `posdev`:

```

display control_program posdev
control_program posdev
(
    copies(1) status(active) pid(3662)
    path("/home/jan/bin/posdcp")
    parameter(
        "--host-addr-file=/home/jan/serfboard/test_configs/test_pos_host.txt "
        "--objtype=/home/jan/CodeMagus/POSFormats/objtypes/pos_device.objtypes "
        "--mtemplate=/home/jan/CodeMagus/POSTest/tdata/logs/pos_device_msg.bin"
        "--merchant=/home/jan/CodeMagus/POSTest/tdata/scripts/merchants_qa.csv"
        "--pcard=/home/jan/serfboard/test_configs/test_plastics.txt "
        "--hsm-port=7171 "
        "--hsm-name=10.57.24.23 "
        "--log=/home/jan/test_pos/logs/posdcp ")
);

```

DisplayExit



Displays exception exit conditions in *orkhestra*.

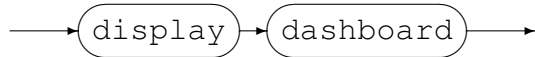
Example:

Display the defined `exit` conditions:

```

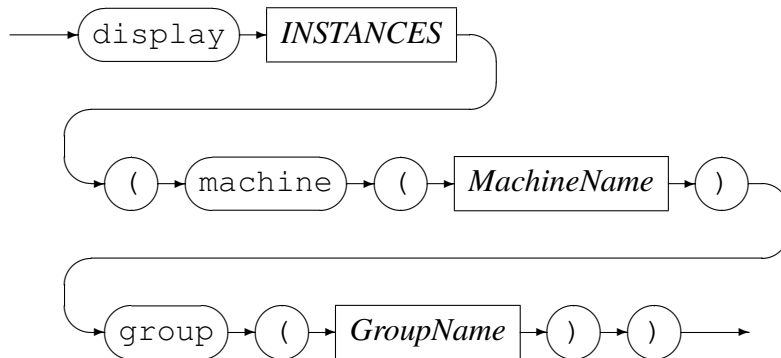
display exit
    exit exception(machine=posdev);
    exit exception(control_program=posdev);

```

DisplayDashboard

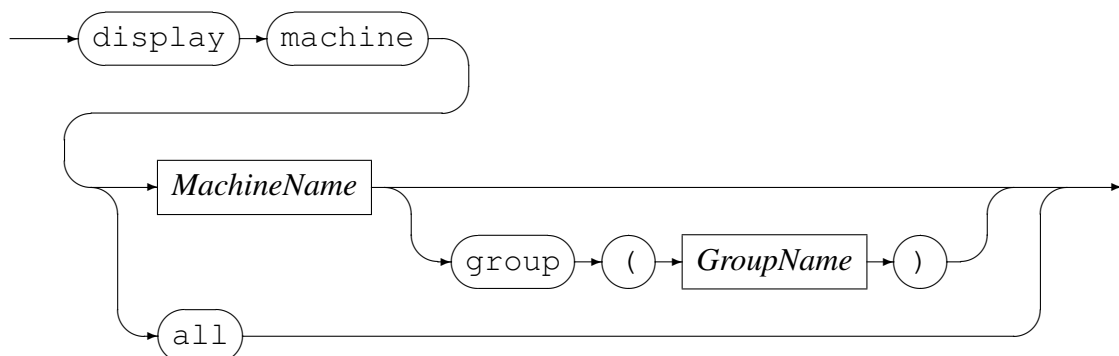
Displays the *DashBoard* connection status, for example:

```
display dashboard
Response: Connected IP: 127.0.0.1 port: 62345;
```

DisplayInstances

Display a snap shot of the number of instances within the various states of an active machine, for example:

```
display instances(machine(posdev) group(auths))
instances 200
(
  machine(posdev) group(auths)
  device_idle (173)
  ready_for_transacting (19)
  send_cc_completion (8)
);
```

DisplayMachine

Display the definition and status of a *State Machine*.

Examples:

- Display the definition and status of the *State Machine* posdev:

```
display machine posdev

machine posdev
(
  status(default)
  instances(1)
  control_program(posdev)
  initial_state(startup)
  target("Stratus")
  creator("Jan Vlok")
  modified("Jan Vlok")
  date(2008-01-16T10:51:18)
  notes("None")
  description("POS stress")
  value timeout_value
  (
    title("Response time out")
    description("Time in milliseconds to wait for a response message")
    constant(10000)
  )
  weight_distribution what_transaction
  (
    title("Transaction profile")
    description("Ratio between the various transactions")
    weight(logon_only(0) credit_card(1) debit_card(1) download_bin(0)
      download_new_hotcard(0) download_software(0) )
  )
  .
  .
  .
);
```

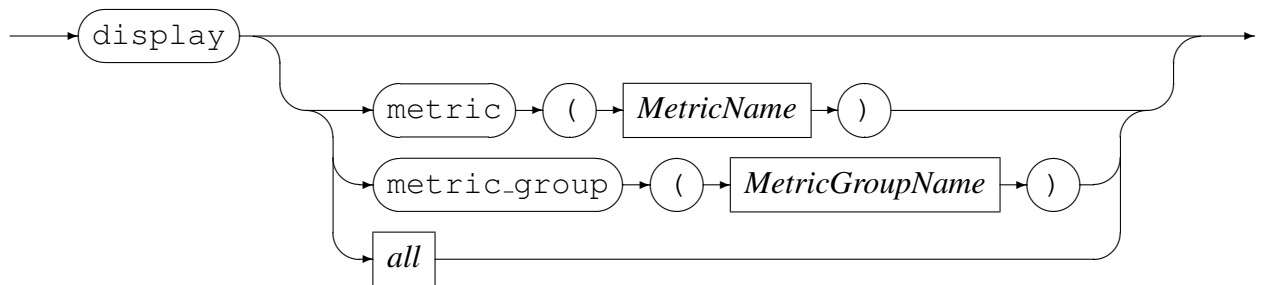
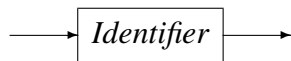
- Display the status of the active *State Machine* posdev, group auths:

```
display machine posdev group(auths)
machine posdev
(
  group(auths) status(active)
  instances(100)
  control_program(posdev)
  initial_state(startup)
  value timeout_value
  (
    title("Response time out")
    description("Time in milliseconds to wait for a response message")
    constant(10000)
  )
  weight_distribution what_transaction
  (
    title("Transaction profile")
    description("Ratio between the various transactions")
```

```

weight(logon_only(0) credit_card(1) debit_card(1) download_bin(0)
download_new_hotcard(0) download_software(0) )
.
.
.
);

```

DisplayMetric*MetricName**MetricGroupName*

This command displays information about metrics that may be sent to a *Dashboard*. It displays the full metric definition and status when the request is for a named metric, for *all* and *metric_group* a brief summary of the metrics are displayed. A metric destined for a *Dashboard* can have one of three states:

1. Active. The named metric is being generated and sent to the *Dashboard*.
2. Queued. The metric is defined, but not being sent to the *Dashboard*; possibly because the *Dashboard* is not running yet and can not be connected to.
3. Not specified. This is the default when the `status` keyword is omitted. The metric is defined in *orchestra* but not requested to be sent to the *Dashboard*.

Examples:

- Display the *Dashboard* metric Timeouts:

```

display metric Timeouts
metric Timeouts
(
  group (Test_POS)
  refresh (10)
  machine (posdev)
  title ("Request timeouts")
  description ("Request timeouts")
)

```

```

type(count)
sum
(
  wait_logon_reply.timer_expire(msg_timed_out),
  retry_1_wait_logon_reply.timer_expire(msg_timed_out),
  retry_2_wait_logon_reply.timer_expire(msg_timed_out),
  retry_3_wait_logon_reply.timer_expire(msg_timed_out),
  wait_cc_auth_reply.timer_expire(msg_timed_out),
  wait_cc_confirmation.timer_expire(msg_timed_out),
  wait_db_ses_key_reply.timer_expire(msg_timed_out),
  wait_db_auth_reply.timer_expire(msg_timed_out),
  wait_db_confirmation.timer_expire(msg_timed_out)
)
);

```

- Display all the defined *Dashboard* metrics:

```

display metric all
metric Sessions (group(Test_POS)
  title("Sessions offered")
);
metric Connection_Retry_1 (group(Test_POS)
  title("First connection retry")
  status(Queued)
);
.
.
.
metric Timeouts (group(Test_POS)
  title("Request timeouts")
  status(Active)
);

```

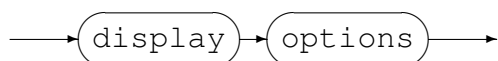
- Display all the metrics belonging to the group Test_POS:

```

display metric_group Test_POS
Metric group Test_POS:
  Connection_Retries
    is "Connection retries"
  Error_disconnect
    is "Unexpected circuit disconnects"
  Timeouts
    is "Request timeouts"
;

```

DisplayOptions



Display *orkhestra* verbose output settings for example:

```

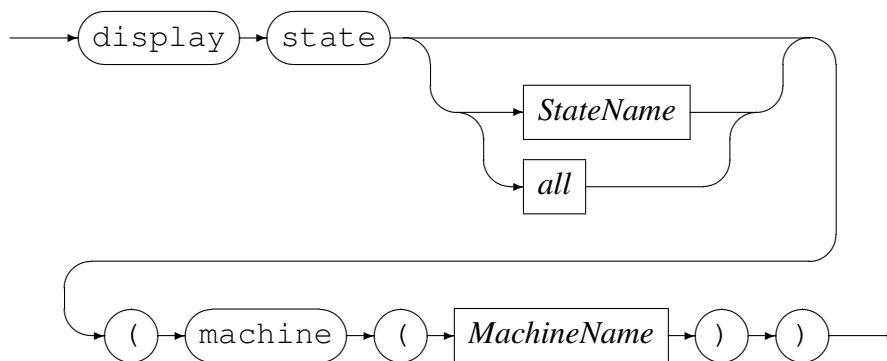
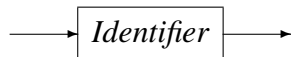
display options
  verbose:
  exceptions OFF

```

DisplayRepeat

Display the current active repeat commands for example:

```
display repeat_command
Response:  1: display instances(machine(posdev) group(auths) repeat(10)
           output("test1.txt");
Response:  2: display control_program all repeat(240);
```

DisplayState*StateName**MachineName*

Display the state definitions of a *State Machine*.

Example:

Display the state definition of `wait_download_packet_reply` of the defined *State Machine* `posdev`:

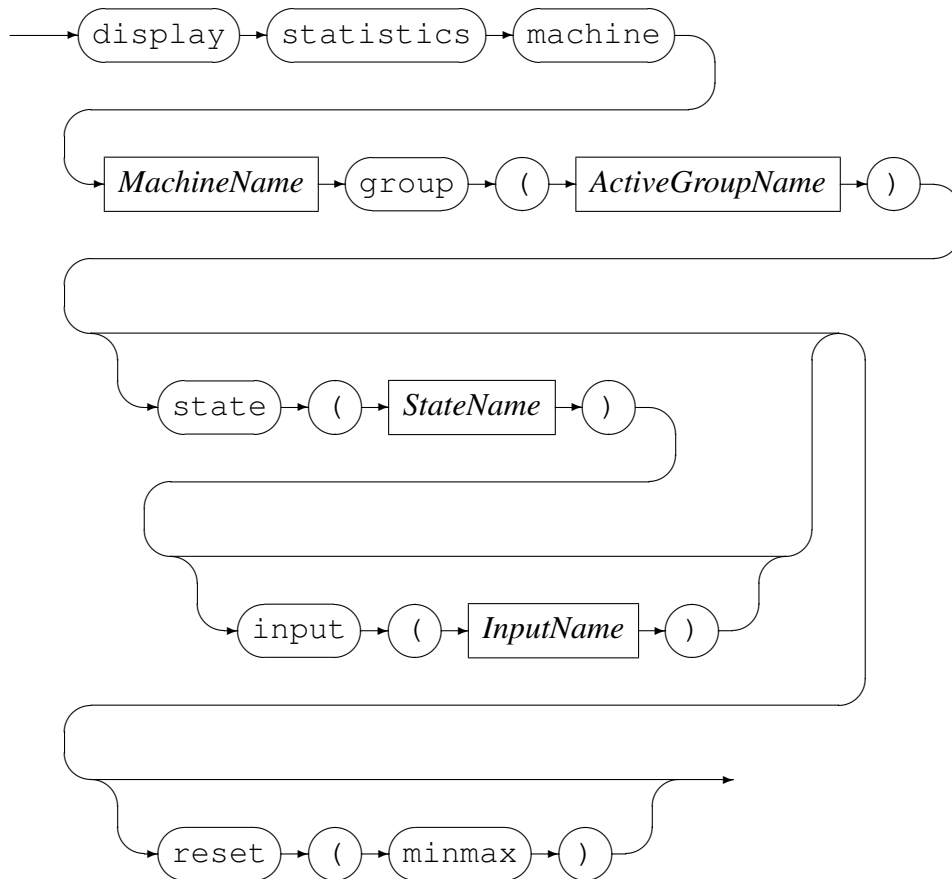
```
display state wait_download_packet_reply (machine(posdev))
state=wait_download_packet_reply
  Input=POS_REMOTE_DOWNLOAD_PACKET_REPLY
  Action=cancel_timer(msg_timed_out)
  Action=start_timer(ready_to_send, sw_down_latency)
To state=send_download_packet_request
;
state=wait_download_packet_reply
  Input=POS_REMOTE_DOWNLOAD_PACKET_REPLY_LAST
  Action=cancel_timer(msg_timed_out)
  Action=start_timer(ok_to_disconnect, disconnect_delay)
To state=do_disconnect
```

```

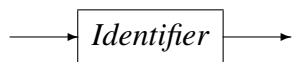
;
state=wait_dwnload_packet_reply
  Input=disconnect
  Action=cancel_timer(msg_timed_out)
  Action=start_timer(device_ready, think_time)
To state=device_idle
;
state=wait_dwnload_packet_reply
  Input=timer_expire(msg_timed_out)
  Action=start_timer(ready_to_send, sw_down_latency)
To state=send_dwnload_packet_request
;

```

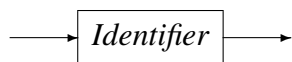
DisplayStatistics



MachineName



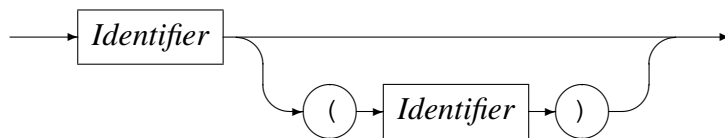
ActiveGroupName



StateName



InputName



Display the transition metrics for an active machine. If *state* and *input* are omitted, all the metrics for the machine will be displayed. If *reset* of *minmax* is requested, the minimum and maximum metrics will be reset.

Example:

Write all the metrics for an active machine to a file every sixty seconds:

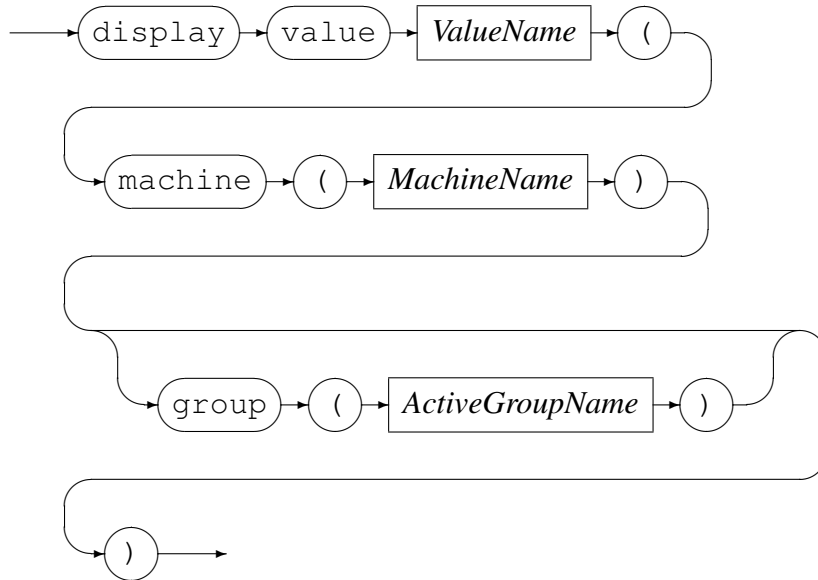
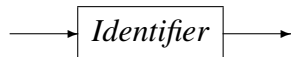
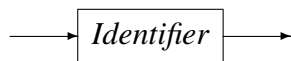
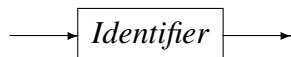
```
display machine posdev group(auths) reset(minmax) \
  repeat(60) output("${HOME}/card/combine/x25_auth.stats.txt")
```

Display metrics for a specific transition, every 30 seconds.

```
display statistics machine tmsods group(auths) state(device_idle)
  input(timer_expire(device_ready)) repeat(30)
```

Response:

```
statistic machine=amsods group=a instances=200 time=1157601808.642343
  elapsed=17.863820
  {state=device_idle input=timer_expire(device_ready)
    count=12 sum_response=168673 sq_response=2388681423 min=11585.028
    max=15996.190}
;
```

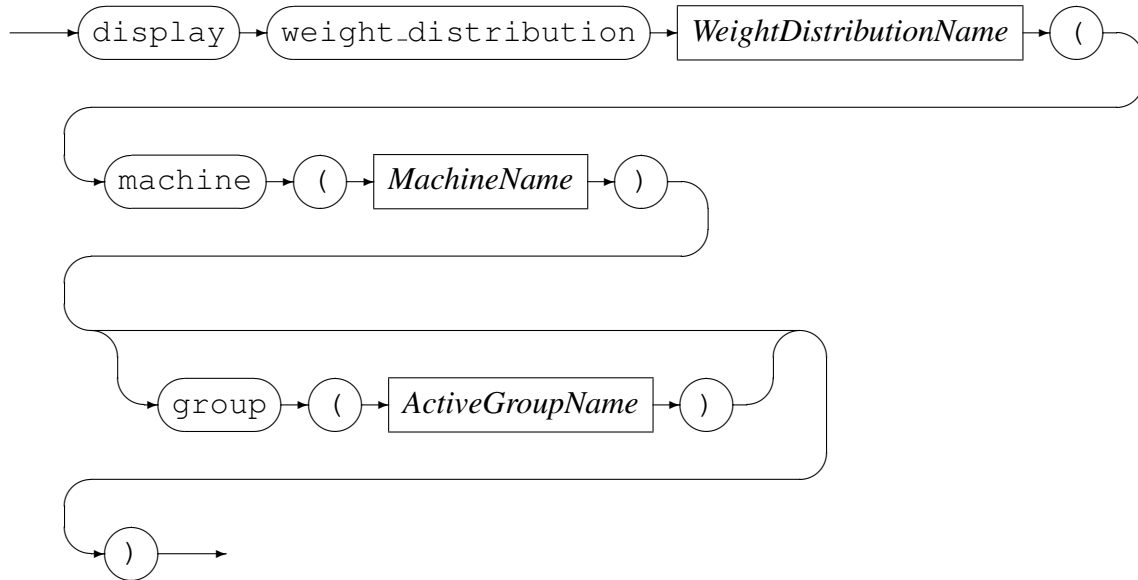
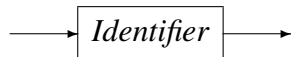
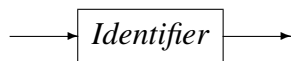
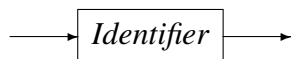
DisplayValue*MachineName**ActiveGroupName**ValueName*

Display a *value* variable of a *State Machine*.

Example:

Display the value of `cc_auth_r_latency` defined in the active *State Machine* `posdev`,
group `auths`:

```
display value cc_auth_r_latency (machine(posdev) group(auths))
value cc_auth_r_latency
(
  machine(posdev)
  group(auths)
  title("CC auth req latency")
  description("Time in milliseconds to delay, before sending a Credit "
    "card authorisation request")
  distribution(class(uniform) min(1900) max(2400))
);
```

DisplayWeightDistribution*MachineName**ActiveGroupName**WeightDistributionName*

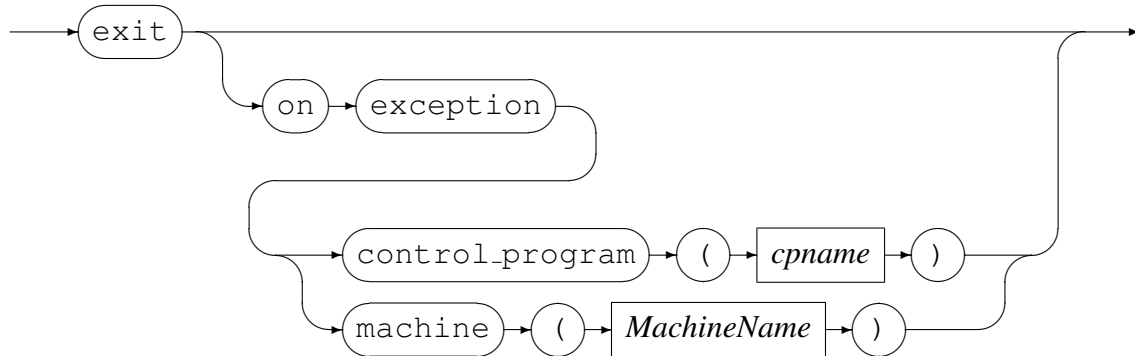
Display a *weight distribution* in a *State Machine*.

Example:

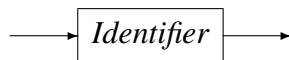
Display the weight distribution what_transaction defined in the active *State Machine* posdev, group auths:

```
display weight_distribution what_transaction (machine(posdev) group(auths))
weight_distribution what_transaction
(
  machine(posdev)
  group(auths)
  title("Transaction profile")
  description("Ratio between the various transactions")
  weight(logon_only(0) credit_card(1) debit_card(1) download_bin(0)
  download_new_hotcard(0) download_software(0) )
);
```

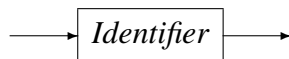

3.3.9 Exit Command



cpname



MachineName



Set exit conditions for *orkhestra*:

- `exit` with no parameters terminates the TCP/IP command session to *orkhestra*.
- Terminate *orkhestra* when a control program abnormally terminates.
- Terminate *orkhestra* when a active machine instance abnormally terminates.

Example:

Terminate *orkhestra* when either the control program `pos` or an active instance of machine `pos` abnormally terminates:

```

exit on exception(control_program(pos))
Response: Exit condition set;
exit on exception(machine(pos))
Response: Exit condition set;
  
```

3.3.10 Flush Command

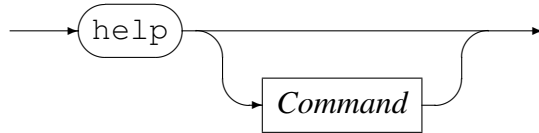


Forces a write on the file, for example to flush the current log:

```

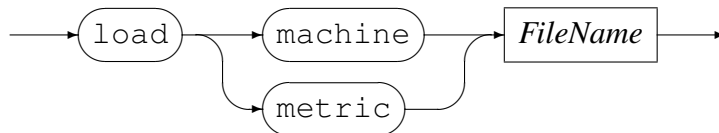
flush log
Response: Flushed log file SAMPLE_20080704_0711.log;
  
```

3.3.11 Help Command



Help on *orkhestra* commands.

3.3.12 Load Command



- *machine* - Load a state machine's definition from the file specified by *FileName*.
- *metric* - Load *DashBoard* metrics from the file specified by *FileName*.

Example:

- Load the *State Machine* defined in the configuration file `pos_device_circuit.mch`:

```
load machine "pos_device_circuit.mch"
Response: Parsing Machine definition file pos_device_circuit.mch;
Response: Machine poscir defined;
```

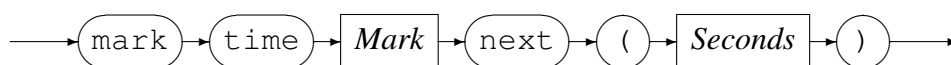
- Load the *DashBoard* metrics that are defined in the configuration file

```
$CONFIGS/test_db.metrics:
```

- *machine* -

```
load metric "${CONFIGS}/test_db.metrics"
Response: Parsing metric definition file ../test_db.metrics;
Response: Created metric Sessions;
Response: Created metric Connection_Retries;
Response: Created metric Connection_Refused;
Response: Created metric logon_OKAY;
Response: Created metric logon_DENY;
Response: Created metric logon_BAD;
```

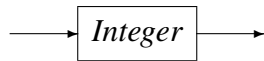
3.3.13 Mark Command



Mark



Seconds



- *Mark* is the label associated with a future time that falls on the next time boundary with modulus *seconds*.

Example:

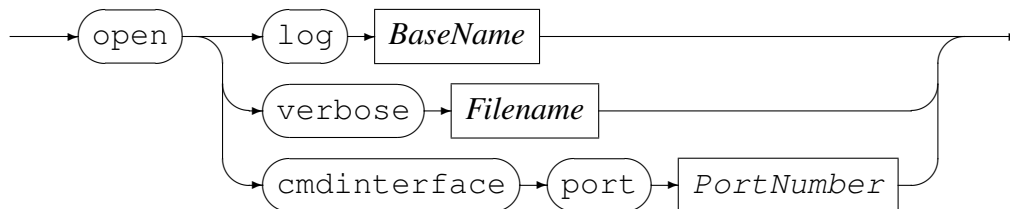
- Mark a time in the future that falls on the next 5 minute boundary and associate it with the label `STARTTIME`.

```
mark time STARTTIME next(300)
```

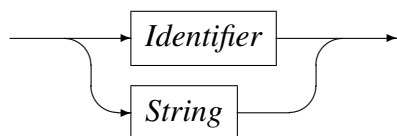
```
Response: Mark time STARTTIME at Fri Nov 30 09:50:00 2018 (+285 seconds);
```

In the above example it can be seen that the next 5 minute boundary was 285 seconds away. See section 3.3.2 on page 11 for an example of using the *Mark* command.

3.3.14 Open Command



basename



For opening:

- An output log file - date, time and 'log' will be appended to the *BaseName*, for example:

```
open log orchestra_pos
```

```
Response: Opened output log file;
```

The log file name will be something like `orchestra_pos_20060925_0933.log`

- An output *verbose* file - all the verbose output from *orchestra* will be written to this file.

- A passive TCP/IP connection, on which connections will be accepted for the command interface of *orkhestra*.

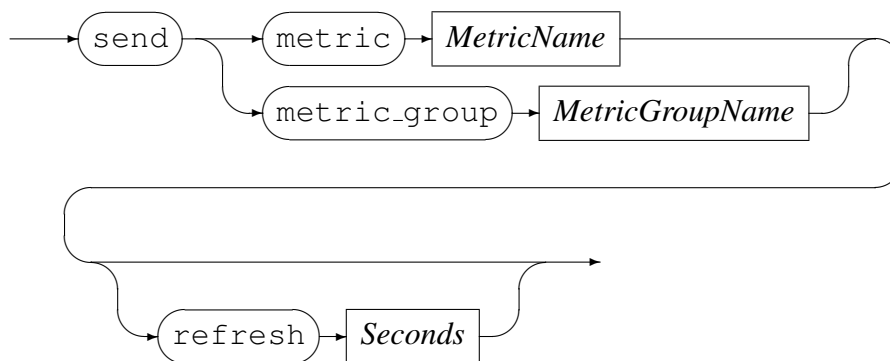
Example:

Open a command interface that will listen on port 61100 for connections to *orkhestra* command interface:

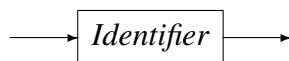
```
open cmdinterface port 61100
```

```
Response: Command interface listen on port 61100;
```

3.3.15 Send Command



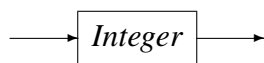
MetricName



MetricGroupName



Seconds



Queue:

- the metric for sending to the *Dashboard*.
- all the metrics in metric group for sending to the *Dashboard*.

If the refresh rate is not specified, it defaults to what was defined for the metric when it was loaded.

Example:

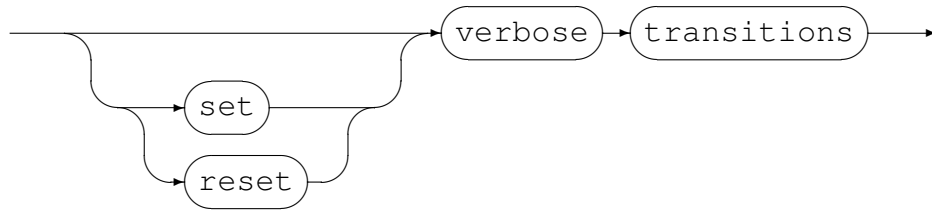
Queue all the metrics defined in group CodeMagus for sending to *Dashboard*:

```
send metric group CodeMagus
```

```
Response: 8 metrics queued;
```

```
send metric Posdev_Instances refresh 10
Response: Metric queued;
```

3.3.16 Set and Reset Commands

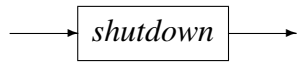


Set or Reset verbose of all *State Machine* transitions.



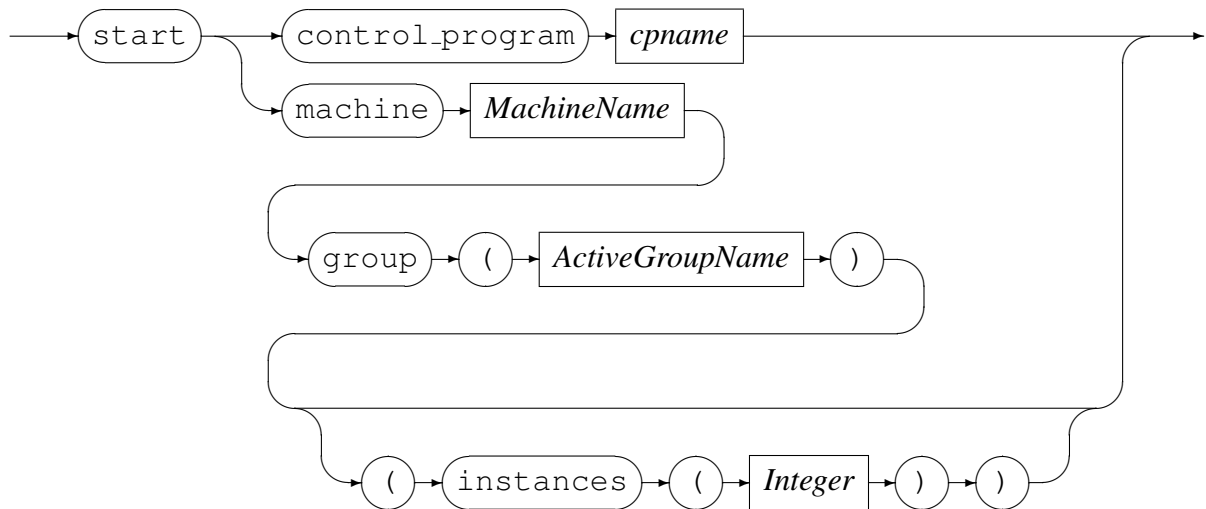
Set the maximum number of instances that may be set.

3.3.17 Shutdown Command



Terminates *orkhestra*:

3.3.18 Start Command



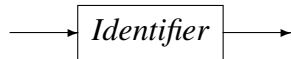
cpname



MachineName



ActiveGroupName



Start a control program or an active machine. Note if instances are not specified, the number of instances in the machine definition will be used.

Examples:

- Start the control program *posdev*:

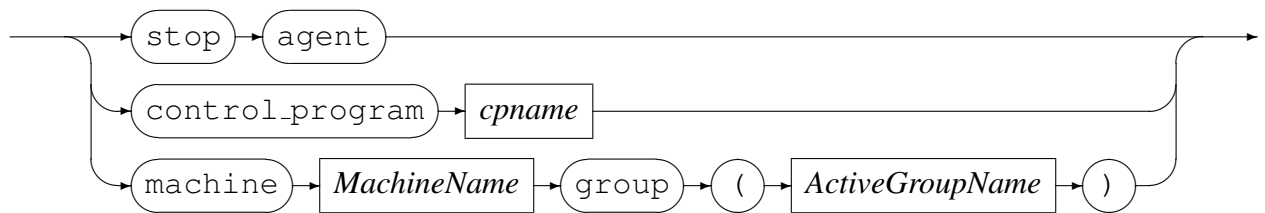
```
start control_program posdev
Response: Control program posdev started with pid 5697;
```

- Alter the number of instances of the *State Machine* *posdev* to 100 and then start it as active group *auths*:

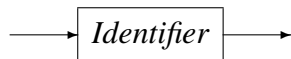
```
alter machine posdev (instances(100))
Response: Machine posdev instances changed from 0 to 100;
```

```
start machine posdev group(auths)
Response: Machine posdev.auths started with 100 instances;
```

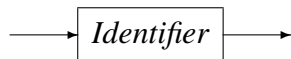
3.3.19 Stop Command



cpname



MachineName



Stop a agent, control program or an active machine.

Examples:

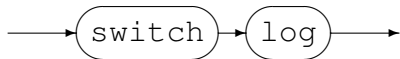
- Stop *State Machine* posdev, active group auths:

```
stop machine posdev group(auths)
delete_mch(): Group posdev.auths stopped
delete_mch(): Group posdev.auths deleted
Response: Machine posdev.auths stopped;
```

- Stop the control program posdev:

```
stop control_program posdev
Response: Control program posdev stopped;
cp_terminated(): Control program posdev terminated
orkhestra(): Control program posdev (pid = 5697) exited with code 0
```

3.3.20 Switch Command



Closes the current log file and opens a new one, for example:

```
switch log
Response: Closed output log file;
Response: Opened output log file;
```

4 Dashboard Metrics

4.1 Introduction

DashBoard metrics are defined in a configuration file, that only contain the definition for these metrics. They are loaded into *orkhestra* using the `load` command; see section 3.3.12 on page 33.

These metrics are send to the *DashBoard* if the following conditions are satisfied:

- The connection to the *DashBoard* as requested by using the `connect` command is established; see section 3.3.6 on page 18.
- The *State Machine* they refer to is active.
- The metric has been queued for sending; see section 3.3.15 on page 35.

4.2 Elements

The elements for defining *DashBoard* metrics comprise reserved words, identifiers, string literals, comments and integers. The definitions are free format and white spaces have no grammatical meaning except where they might appear within string literals.

4.2.1 Comments

Comments are introduced in two ways:

- Using double minus sign (`'--'`) and continue up to the end of the current input line.
- Using the left brace (`'{'`) and continue up to and including the next right brace (`'}'`). Comments can span lines and can contain any characters except the right brace (`'}'`), which would end the comment. Consequently, comments cannot be nested.

Examples:

```
-- File: atmams.metric
--
-- Defining of the common dashboard metrics for the ATM AMS stress test.
--
```


4.2.2 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of commands. The reserved words are:

description	group	input	instances
list	machine	metric	refresh
state	sum	title	type

Table 2: *State Machine* reserved words

4.2.3 Identifiers

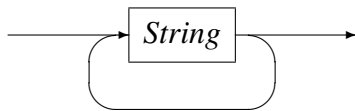
An *Identifier* is case sensitive, it starts with a letter which can be followed by any number of letters, digits or the under-score character.

Examples:

```
Connected wait_connection connect
```

4.2.4 Strings

A *String* is any sequence of printable (or keyboard) characters enclosed in double quotes. The enclosing double quote may not appear within the *String*, neither the newline character (i.e. strings cannot span source text lines), but they may be concatenated:



Examples:

```
description("Before an ATM can start transacting, "
            "it must be connected.")
```

4.2.5 Integers

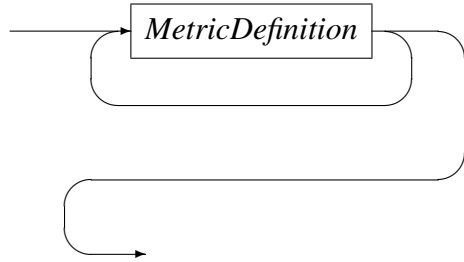
An *Integer* consists of a nonempty sequence of decimal digits ‘0’ through ‘9’.

Examples:

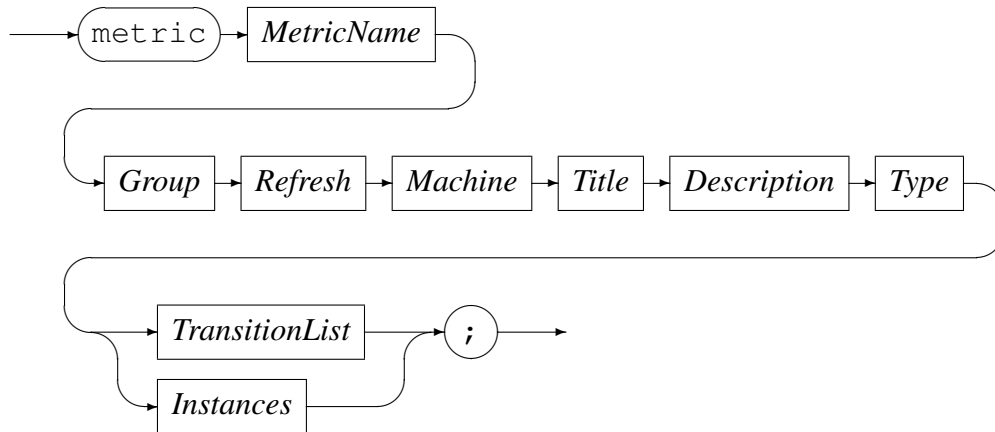
```
1234
0
```

4.3 Metric Definition

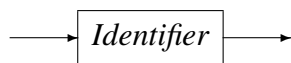
This section describes the definition of a metric that is to be sent to the *DashBoard* server.



MetricDefinition

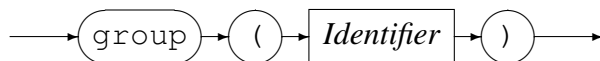


MetricName



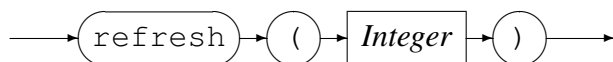
MetricName identifies the metric.

Group



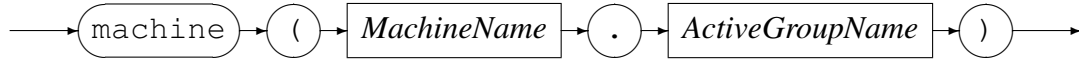
Assigns the metric to a *DashBoard* metric group.

Refresh



Assigns the refresh rate (in seconds) to the metric.

Machine



MachineName

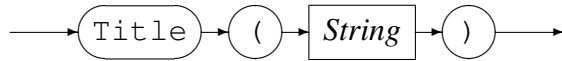


ActiveGroupName



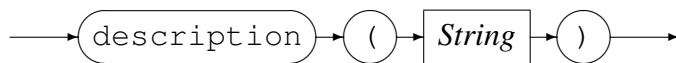
MachineName and ActiveGroupName identifies the *State Machine* that this metric belongs to.

Title



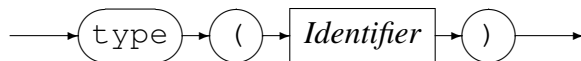
Assigns a title to the metric which is formally part of the description of it.

Description



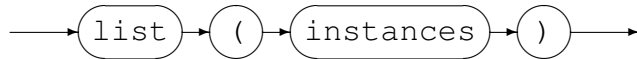
This provides a mechanism for assigning an comment to a metric which is formally part of the description of it.

Type



Defines the metric type.

Instances

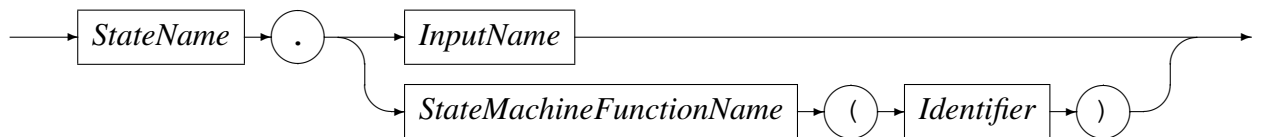


List a snapshot of the requested machine's instances.

TransitionList



Transition



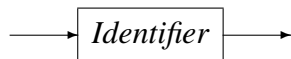
StateName



InputName



StateMachineFunctionName



Summation of the metrics in the listed *State Machine* transitions.

Examples:

- Define metric `logon_OKAY` in the *Dashboard* group `CodeMagus` to be send to the *Dashboard* server every ten seconds for the active *State Machine* `posdev.auths`. This metric is a summation of the transitions from the *State Machine*'s input `POS_LOGON_REPLY`.

```
metric logon_OKAY
  group (CodeMagus)
  refresh (10)
  machine (posdev.auths)
  title ("POS device logon accepted")
  description ("POS device logon accepted")
  type (logon_OKAY)
  sum
    (
      wait_logon_reply.POS_LOGON_REPLY,
      retry_1_wait_logon_reply.POS_LOGON_REPLY,
      retry_2_wait_logon_reply.POS_LOGON_REPLY,
      retry_3_wait_logon_reply.POS_LOGON_REPLY
    )
  ;
```

- Define metric `Posdev_Instances` in the *Dashboard* group `CodeMagus` to be send to the *Dashboard* server every ten seconds for the active *State Machine* `posdev.auths`. This metric is a snapshot of instances count in the various states of the *State Machine*.

```
metric Posdev_Instances
  group (CodeMagus)
  refresh (10)
  machine (posdev.auths)
  title ("Posdev instances")
  description ("Snapshot of instances count in the various states")
  type (Posdev_Instances)
  list (instances)
  ;
```

- Define metric `Sessions` in the *DashBoard* group `CodeMagus` to the *DashBoard* server to be send every ten seconds for the active *State Machine* `posdev.auths`. This metric is a summation of the transitions from the *State Machine*'s input `timer_expire(device_ready)`.

```
metric Sessions
  group(CodeMagus)
  refresh(10)
  machine(posdev.auths)
  title("Sessions offered")
  description("Sessions offered")
  type(Sessions)
  sum
    (
      device_idle.timer_expire(device_ready)
    )
  ;
```

5 *State Machine* definition

5.1 Introduction

The *State Machine* consists of a finite number of states and transitions. One of these states is always the current status of the machine; i.e. transitions caused by inputs from the time the *State Machine* system starts to the present time lead to the current status.

A transition indicates a change from one state to another and is described by an input that would need to be fulfilled to enable the transition. For each transition there are optional actions that can be performed before entering the next state. An action is an external output and/or internal *orkhestra* functions.

To summarise, the *State Machine* can be described as:

- An initial state, in which the machine is in at start-up.
- A set of possible input events.
- A transition which includes a set of possible actions (output and/or *orkhestra* functions) that result from the input.
- A new state that results from the transition.
- A final state, in which the machine is considered complete.

At machine start the initial state is automatically set to first state defined and the input that will be triggered is `startup`.

A machine has a specific final state defined, the name of which is `final`. The final state is one in which no transitions lead out of.

The *orkhestra* internal functions return control immediately. An example is to trigger an input, either immediately or at some time in future. Refer to section 5.4 on page 58 for a full description of these functions.

5.2 Elements

The elements for defining a *State Machine* comprise reserved words, identifiers, string literals, comments and integers. The definitions are free format and white spaces have no grammatical meaning except where they might appear within string literals.

5.2.1 Comments

Comments are introduced in two ways:

- Using double minus sign ('--') and continue up to the end of the current input line.
- Using the left brace ('{') and continue up to and including the next right brace ('}'). Comments can span lines and can contain any characters except the right brace ('}') which would end the comment. Consequently, comments cannot be nested.

Examples:

```
-- File: atmams.mch
--
-- ATM AMS creditcard stress.
--
```

5.2.2 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of commands. The reserved words are:

begin	by	class	constant
control_program	constant	created	date
description	deviation	distribution	end
exponential	gaussian	machine	max
mean	min	modified	mu
notes	target	title	uniform
value	weight	weights	
machine			

Table 3: *State Machine* reserved words

5.2.3 Identifiers

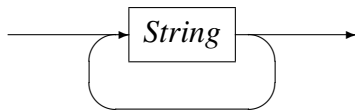
An *Identifier* is case sensitive, it starts with a letter which can be followed by any number of letters, digits or the under-score character.

Examples:

```
Connected wait_connection connect
```

5.2.4 Strings

A *String* is any sequence of printable (or keyboard) characters enclosed in double quotes. The enclosing double quote may not appear within the *String*, neither the newline character (i.e. strings cannot span source text lines), but they may be concatenated:



Examples:

```
description("Time in milliseconds to delay, before sending "
           "a request")
```

5.2.5 Integers

A *Integer* consists of a nonempty sequence of decimal digits ‘0’ through ‘9’.

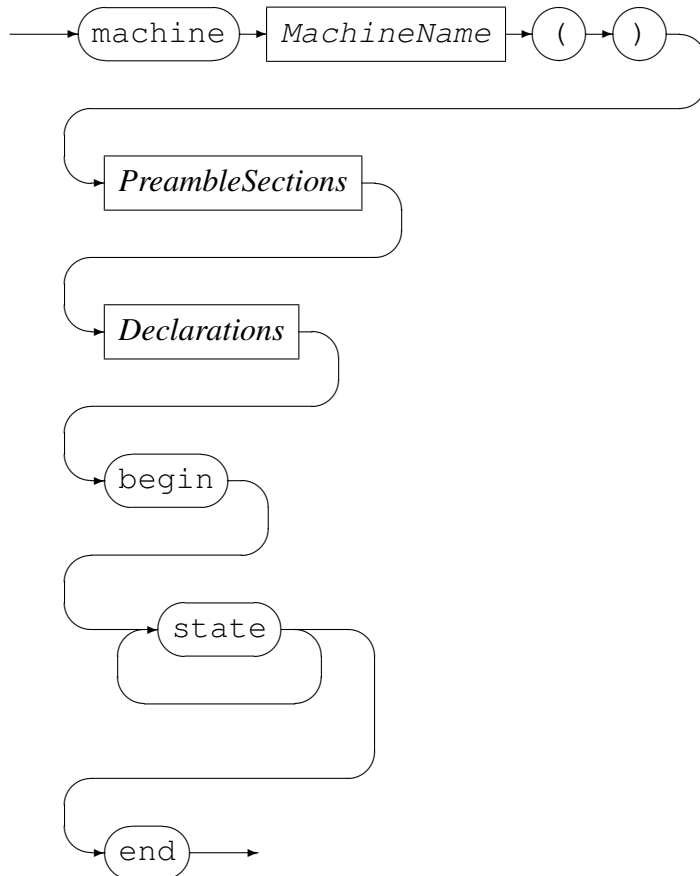
Examples:

```
1234
0
```

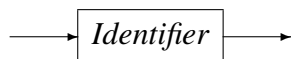
5.3 Machine Definition

The *State Machine* definition starts by defining the name of the current machine. This identifies the machine once loaded by *orkhestra*. Following the name is the preamble sections and then the state definitions, which is enclosed by the begin and end keywords; see figure 2 on page 49.

See appendix A on page 72 for an example of a very basic *State Machine* definition.



MachineName



```
-- An example of a simple state machine.
--
machine orksample();

-- Mandatory preamble.

created by ("Mr Sample");
description("Orkhestra sample control program");
date("2008-01-16T10:51:18");
target("Demonstration");
control_program (orksample);
control_program ods;

-- Optional preamble.
notes("Bla bla");
modified by (Mr Sample2);

-- Declarations

value timeout_value
    ( ... );
weight_distribution what_transaction
    ( ... );
.
.
.

-- States - transitions

begin
State 1
    .
    .
    .
state n

-- End of State Machine definition.

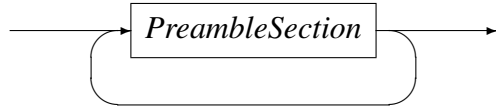
end
```

Figure 2: Example of a State Machine Definition

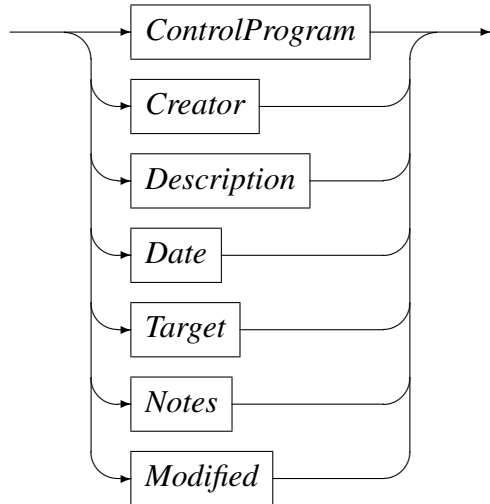
5.3.1 Preamble

The *Preamble* provides certain documentation regarding the *State Machine*. A *Preamble* comprises a number of sections some of which are mandatory and some of which are optional.

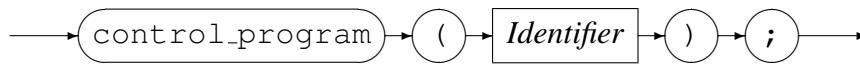
Preamble



PreambleSection



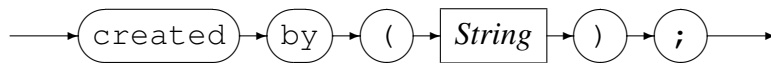
ControlProgram



Defines the control program that will be used by this machine. This control program must be defined to *orkhestra* and started before the *State Machine* can be started.

The *ControlProgram* preamble section is mandatory.

Creator



The *Creator* preamble section is mandatory and Identifies the author.

Description



The *Description* section provides a mechanism for assigning an comment to the *State Machine* which is formally part of the description of it.

The *Description* preamble section is mandatory.

Date



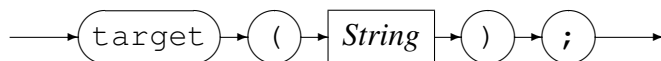
The Date section is provided so that a date can be associated with the *State Machine*. This date is interpreted as the date the *State Machine* was created. `ISODate` has the *ISO* date and time format:

```
yyyy-mm-ddThh:mm:ss
```

Where the portion before the T-character is the date and the portion after the T character is the time stamp. In the date portion, yyyy is the four digit year, mm is two digit the month number, and dd is the two digit day of the month. In the format of the time stamp, the hh is the hour of the day according to the twenty four hour clock format, mm is the two digit minutes passed the hour and ss is the two digits passed the minute

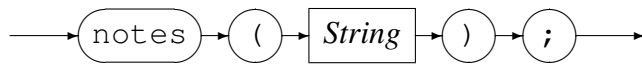
The Date preamble section is mandatory.

Target



The Target preamble section is a mandatory comment field which indicates the target system under test to which this *State Machine* applies.

Notes



The Notes section is designed so that any additional commentary can be include as part of the *State Machine*. For example, if the source are being version controlled through a CVS repository, then you might choose to described the CVS entries as note strings.

Modified



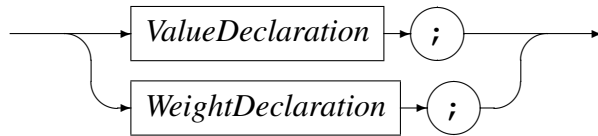
The Modified optional preamble section is provided as a means by which anyone modifying the *State Machine* can record the name of the user who modified the *State Machine*. A Notes optional preamble section can be used to record the details of the modification

Example preamble sections:

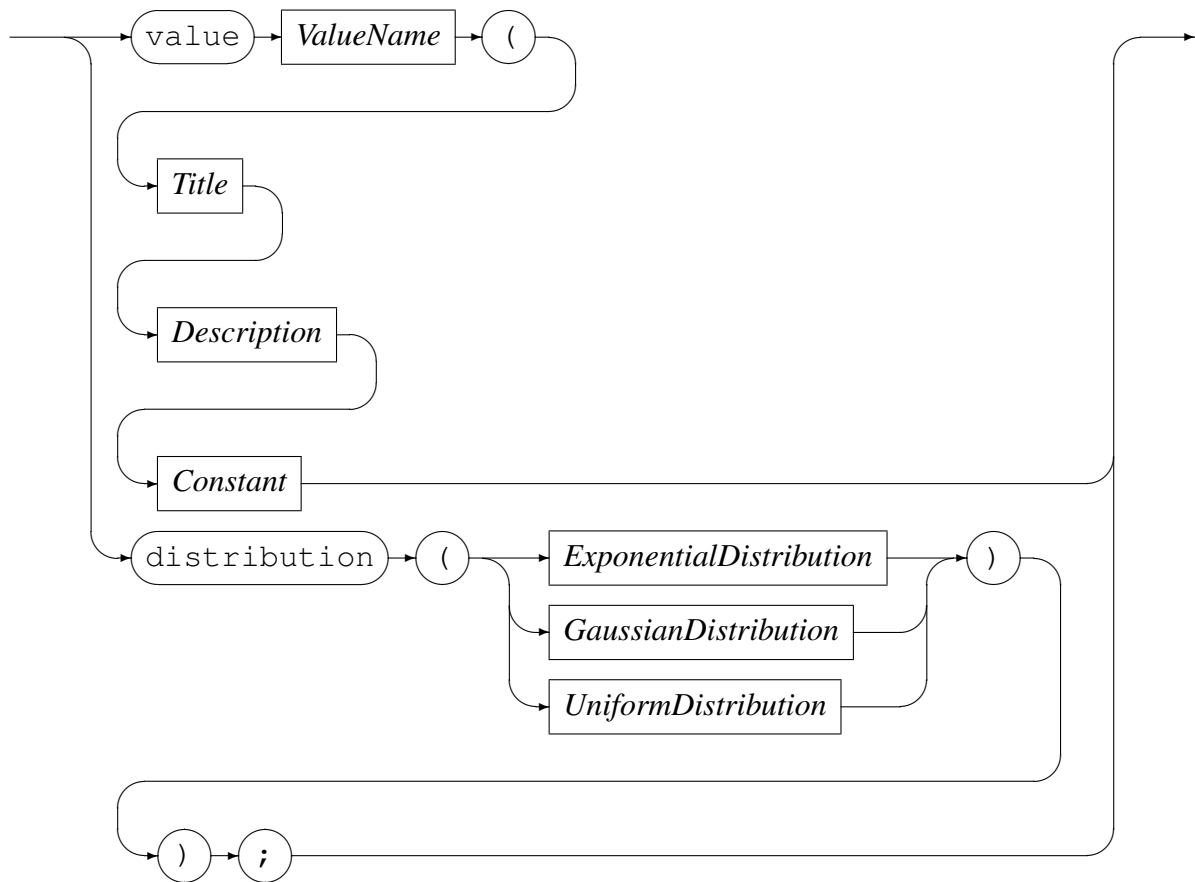
```
created by ("Jan Vlok");
description("Orchestra sample control program");
date("2008-01-16T10:51:18");
target("Demonstration");
control_program (orksamle);
```

5.3.2 Declarations

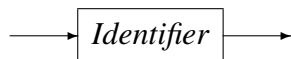
This subsections describe the declarations of values and weight distributions that are used for defining the states of the *State Machine*.



ValueDeclaration

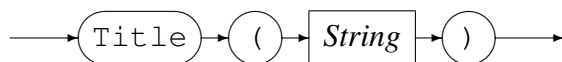


ValueName



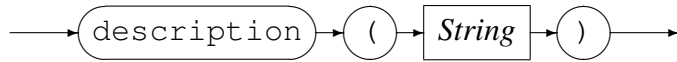
Define a value. A value is used by the *orchestra* internal function `start_timer()`; see section 5.4.2 on page 59.

Title



Assigns a title to the value which is formally part of the description of it.

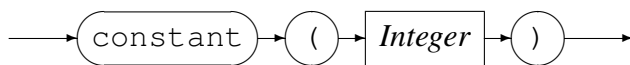
Description



This provides a mechanism for assigning an comment to a *State Machine* value which is formally part of the description of it.

A value can be any one of the following:

- Constant



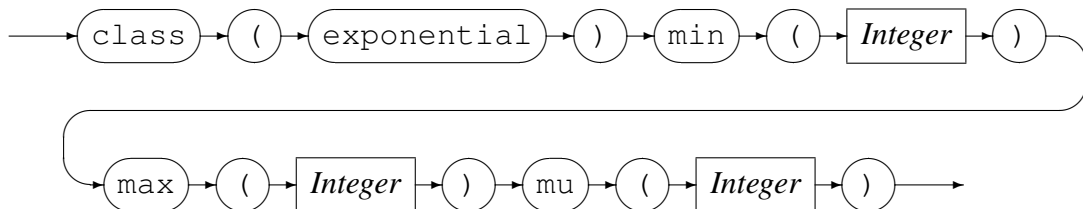
A *Constant* is specified in milliseconds.

Example:

Define the *value* `timeout_value` as a constant with a value of 50 seconds:

```
value timeout_value
(
  title("Response time out")
  description("Time in milliseconds to wait for a response message")
  constant(50000)
);
```

- ExponentialDistribution



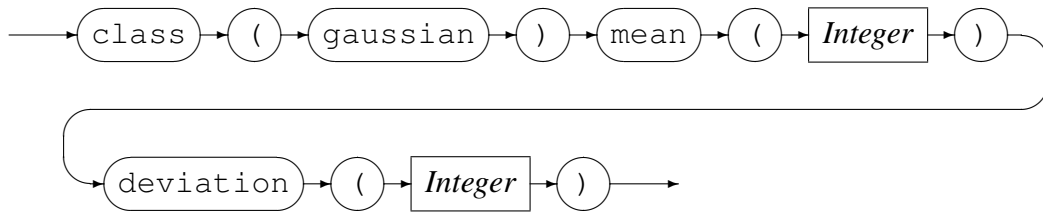
min, *max* and *mu* are specified in milliseconds.

Example:

Define the *value* `think_time` as an exponential distribution with *min* as 10 seconds, *max* as 3000 seconds and *mu* as 100 seconds:

```
value think_time
(
  title("Device Idle time")
  description("Time in milliseconds an instance will be in the idle state")
  distribution(class(exponential) min(10000) max(3000000) mu(100000))
);
```

- GaussianDistribution



mean and *deviation* are specified in milliseconds.

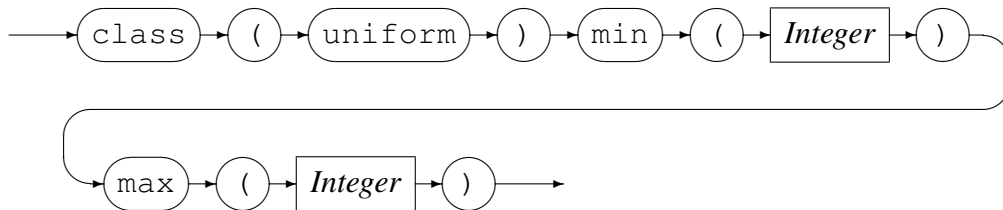
Example:

Define the *value* `sw_down_latency` as a gaussian distribution with *mean* as 2 seconds and *deviation* as 500 milliseconds:

```
value sw_down_latency
(
  title("SW download latency")
  description("Time in milliseconds to delay, before sending the "
             "next packet request")
  distribution(class(gaussian) mean(2000) deviation(500))
);
```

- **UniformDistribution**

UniformDistribution



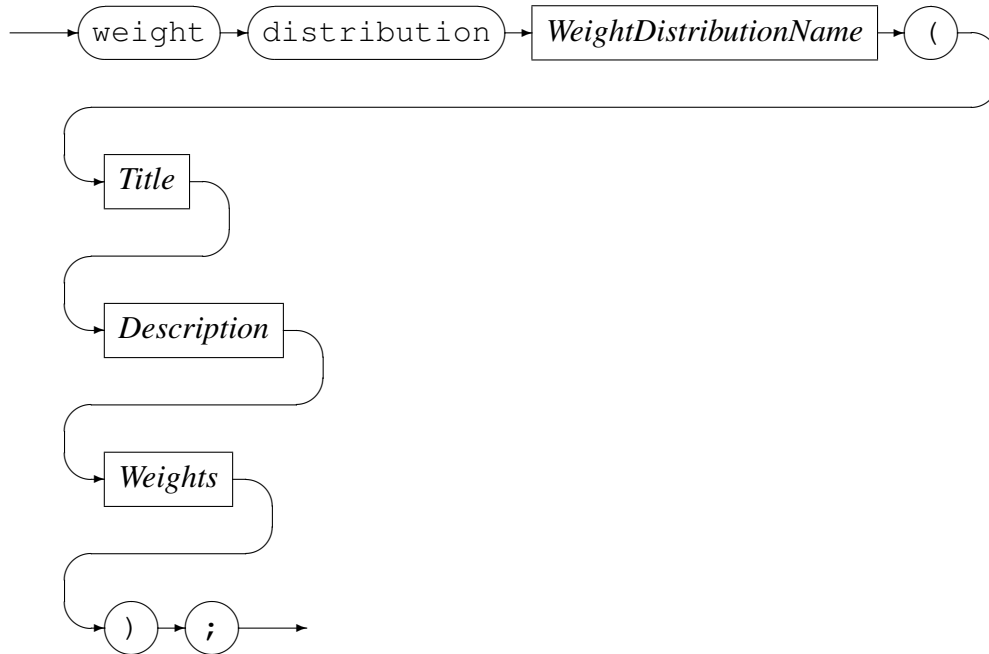
min and *max* are specified in milliseconds.

Example:

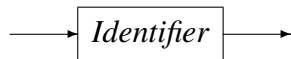
Define the *value* `logon_latency` as an uniform distribution with *min* as 100 milliseconds and *max* as 4600 milliseconds:

```
value logon_latency
(
  title("Logon latency")
  description("Time in milliseconds to delay, before sending a logon request")
  distribution(class(uniform) min(200) max(4600))
);
```

WeightDeclaration

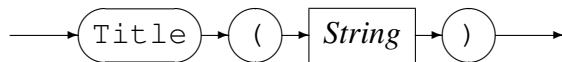


WeightDistributionName



Define a weight distribution. A weight distribution is used by the *orchestra* internal function `choose ()`; see section 5.4.1 on page 58.

Title



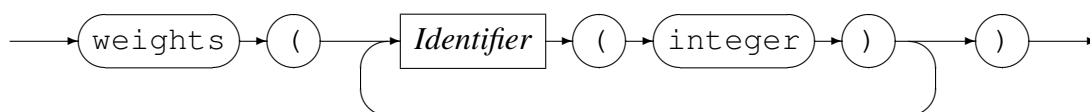
Assigns a title to the `weight` distribution which is formally part of the description of it.

Description



This provides a mechanism for assigning an comment to a *State Machine* weight distribution which is formally part of the description of it.

Weights



Define the `weights` within the distribution with their relevant weights.

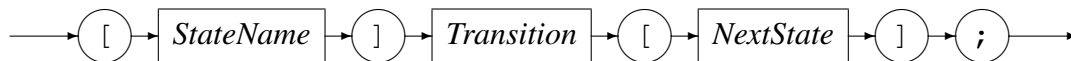
Example:

Define the weight distribution `what_transaction` to reflect the desired transaction profile:

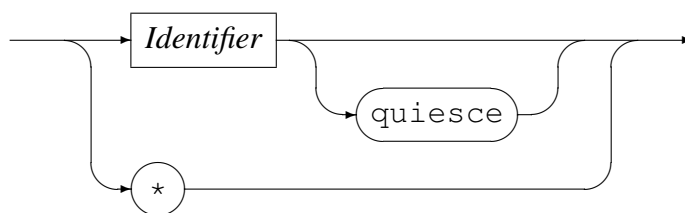
```
weight distribution what_transaction
(
  title("Transaction profile")
  description("Ratio between the various transactions")
  weights
  (
    logon_only(1)
    credit_card(50)
    debit_card(50)
    download_bin(1)
    download_new_hotcard(1)
    download_software(1)
  )
);
```

5.3.3 State Definition

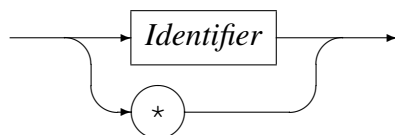
This sub section describes the definition of the states of a *State Machine*. A *state* definition defines the transition out of the current state to a new state.



StateName



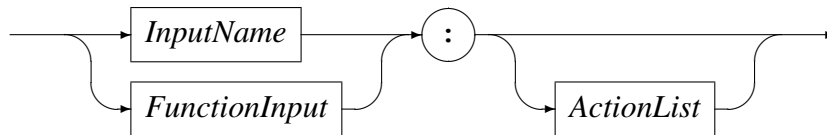
NextState



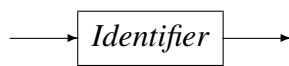
If the state name is ‘*’ the transition will apply to all the states in the *State Machine* that do not have this specified transition. In other words a ‘*’ defined transition acts as a default transition. If the new state name is ‘*’ then the transition does not change the state. In other words the new state is the current state and this is a short form of writing the name again.

If the key word *quiesce* are defined, this state is eligible for deleting an instance when the *State Machine* is quiescing the instances. When instances are shrunk and there are no quiesce states defined - the excess instances are immediately deleted. When there are quiesce states defined, an instance will be deleted when it transitions to a quiesce state, this continues until the desired number of instances are reached.

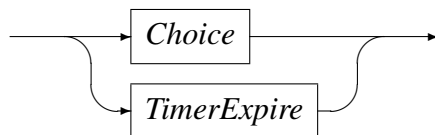
Transition



InputName



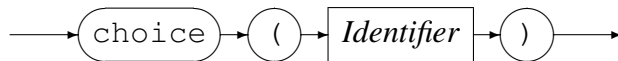
FunctionInput



A transition is defined by an input that needs to be fulfilled followed by optional actions in order for the *State Machine* to reach a new state.

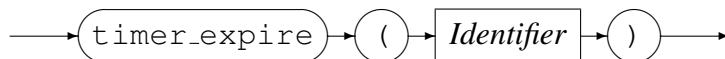
An Input is either an external input name or an input that is generated due to a previous invocation of an internal *orkhestra* function as defined in section 5.4 on page 58.

Choice



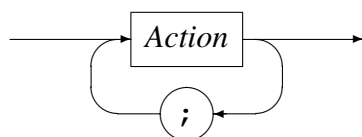
A *Choice* names an input generated in a previous state by the `choose()` function. See section 5.4.1 on page 58.

TimerExpire

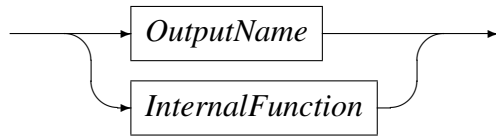


A timer expiration names an input generated in a previous state by the `start_timer()` function; see section 5.4.2 on page 59.

ActionList



An action list is a semi-colon separated list of one or more individual actions.

Action

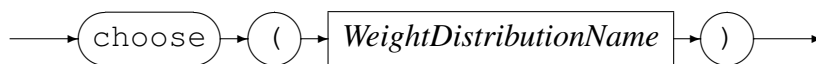
An action is an external output or an internal *orchestra* function. Within a transition there may be no more than one external output specified, but any number of *orchestra* internal functions may be specified. See section 5.4 on page 58 for the definition of the internal *orchestra* functions.

5.4 Internal Functions

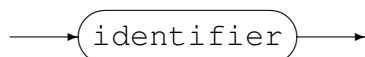
This section describes the internal *State Machine* functions. The following functions are available:

- *choose()*
Enables the selection of any given number of choices of a future internal input biased by the associated weighting definition. The future input is matched in the new state by `choice()` immediately (see section 5.3.3 on page 57).
- *start_timer()*
Start a timer in order to trigger a future internal input when a given time limit expired. The future input is matched in the new state by `timer_expire()` on expiry (see section 5.3.3 on page 57).
- *cancel_timer()*
Cancel a previous timer event.
- *start_machine()* Start one or more instances of a *State Machine*.

5.4.1 Choose Function



WeightDistributionName



Enables the selection of any given number of choices of next internal input name with each selection having a relative weight associated with it. This weighting determines the relative likeliness of a selection being made. Once a selection is made by *orchestra* the selected name is triggered as a future input event to *orchestra*.

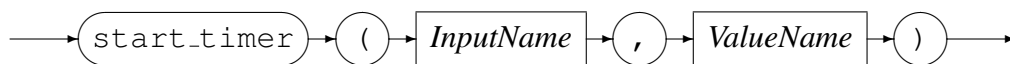
See Weight declaration in section 5.3.2 on page 54.

Example:

A transaction can be either reversed, ignored or accepted. State machine extract:

```
weight distribution TW
(
  title("Reversal profile")
  description("Ratio for reversal of transactions")
  weights
  (
    reverse(5)
    continue(25)
    ignore(10)
  )
);
.
.
.
[transaction_01]
  TRAN01_name:
  choose(TW);
[transaction_01_choice]
;
-- What was the choice? do we need to reverse, ignore or continue with
-- the transaction?

[transaction_01_choice]
  choice(reverse): -- Need to send the reversal
  TRAN01_REVERSE; -- Send the reversal to the control program
[device_idle]
;
[transaction_01_choice]
  choice(ignore): -- OK, continue with the sequence.
  TRAN01_DROP; -- Send new output to CP - DROP transaction.
[device_idle]
;
[transaction_01_choice]
  choice(continue): -- OK, continue with the sequence.
  TRAN01_OK; -- Next message in the sequence.
[wait_tran01_resp]
;
;
```

5.4.2 Start Timer Function

InputName



ValueName



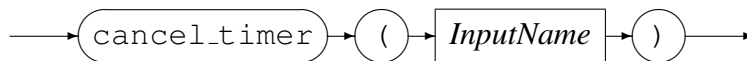
Start a timer event. When the timer expires, the input, as specified on the request will be triggered. The value used must be defined in the preamble section; (see section 5.3.2 on page 52). Time values are in milliseconds.

Example:

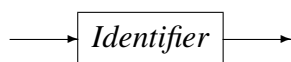
Define a transition out of state *initial* to a state of *device_idle*. On receiving the input *startup* the *State Machine* transitions to the new state and will execute the internal *orkhestra* function *start_timer*.

```
-- At start-up this is the initial state
-- and the default input is startup.
--
[initial]
  startup:
    start_timer(device_ready,think_time);
[device_idle]
;
```

5.4.3 Cancel Timer Function



InputName



Cancel a previously started timer.

Example:

On receiving the input *BICISO_0110_1* in state *wait_m100_resp* the timer *m100_timeout* must be cancelled. This timer was set in state *start_seq*. State machine extract:

```
-- The choice has been made, so get on with it.
--
[start_seq]
  choice(send_100):
    BICISO_0100_1;
    start_timer(m100_timeout,msg_timeout);
[wait_m100_resp]
;
```

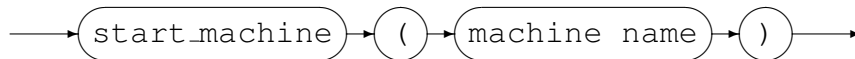
```
--
-- Received the '0100' message response.
-- Cancel the outstanding timer.
--
```

```

[wait_m100_resp]
    BICISO_0110_1:
        cancel_timer(m100_timeout);
[send_220]
;
--
-- Timed out waiting for the '0100' message response.
-- Go back to the idle state.
--
[wait_m100_resp]
    timer_expire(m100_timeout):
        start_timer(device_ready,think_time);
[device_idle]
;

```

5.4.4 Start Machine Function



Start a *State Machine* with the number of instances in the machine definition in *orkhes-tra*.

Example:

Start a machine with the name of NEWTRAN, after receiving the input LOGON_TRAN:

```

[wait_logon_response]
    LOGON_TRAN:
        start_machine(NEWTRAN);
[echo]
;

```

6 Remote Control Programs

6.1 Overview

Orkhestra have the ability to run remote control programs. This is done in order for a test to be able to use the computing power of machines and in such a way that makes the ramp up of instances scalable for all practical purposes (that is to achieve tens of thousands of instances, possibly up to a hundred thousand).

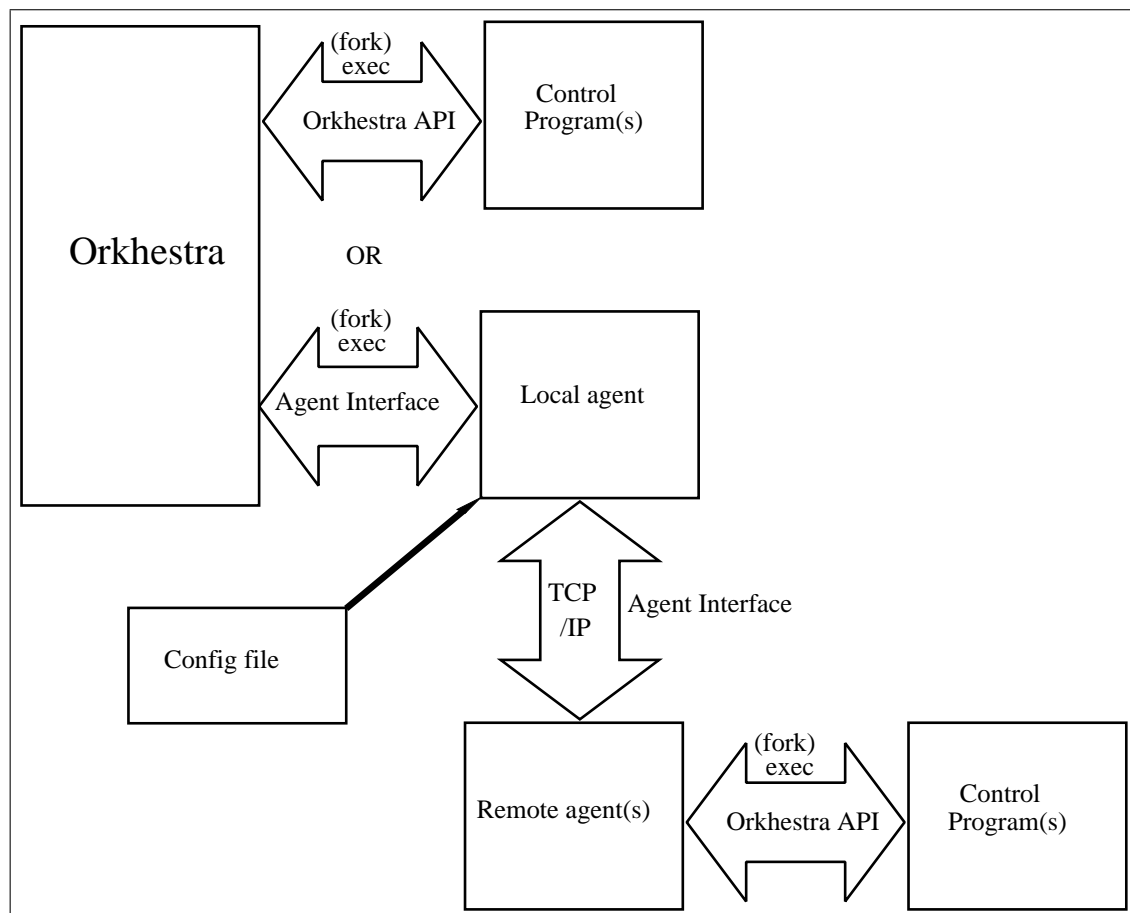


Figure 3: Relationship of orkhestra, the agents and the control programs

Note:

Interfaces:

- *orkhestra* API - see section ?? on page ??.
This interface is between the control programs and either *orkhestra* or the remote agent.
- Agent Interface
This interface is between:

- *orkhestra* and the local agent.
- local agent and remote agents

Note:

- *orkhestra* will either run the control programs directly or through agents, a mix of the two scenarios is not permitted.

6.2 Clock Synchronisation

There is a need for clock synchronisation between the local and remote agents so that *orkhestra* can be aware of any actual event times on the remote machines with respect to the local machine's clock. This drift will be determined by the having the local and remote agent determine the differences in clock values by the following procedure: Once communication between the local and remote agent has been established a clock diff between the local and remote machines will be determined by the local machine sending a message to the remote machine and taken the local clock value at the time the message is sent, and then taking the local clock value when the reply is received. The remote machine will simply respond to the message by replying with the remote clock value. The two clock values recorded on the local machine will be used to determine the response time of the remote machine by taking subtracting the message send clock time from the message response clock time.

The above step are performed 100 times, and the smallest response time message is used to determine the clock drift between the two machines. The clock drift is determined by subtracting the remote clock value in the reply message corresponding to the shortest response time from the clock value record when the message was sent from the local machine to the remote machine; and then subtracting half the response time for this shortest response time message.

The local agent will adjust the time stamp in all the control program messages destined to *orkhestra* with the relative time difference.

6.3 Local Agent

The local agent is *orkhestra*'s interface to the remote control programs and is totally transparent to the *orkhestra State Machine*. The local agent load balances the traffic from the *State Machine* to the various remote agents.

The *orkhestra* commands applicable to the local agent are:

- 'define agent' : Defines the local agent, see section [3.3.7](#) on page [19](#).
- 'start agent' : Starts the local agent, see section [3.3.8](#) on page [21](#).

- ‘stop agent’: Stops the local agent, see section 3.3.19 on page 37.

On starting the local agent, *orkhestra* passes its parameters as command line options. The local agent has the following options:

- ‘-c|--config’ Specifies the configuration file for the agents, see section 7 on page 67.
- ‘-v|--verbose’ When specified, the local agent operates in a verbose manner.

6.3.1 Operation

On startup the local agent initialisation steps are:

1. Parse the agent configuration file as specified by the command line option ‘-c|--config’. If any errors are encountered in parsing the configuration file the local agent will do an error shutdown, see section 6.3.3 on page 65.
2. Establish connection with all the configured remote agents, if unable to do so, it will do an error shutdown, see section 6.3.3 on page 65.
3. Send each remote agent its configuration.
4. When acknowledge of successfully initialisation is received from an remote agent, do the clock synchronisation with him.
5. When all the remote agents is up and running, the local agent notifies *orkhestra* that it is ready for running the control programs remotely.

Messages from *orkhestra* are dealt with as follows:

- *State Machine* outputs:
A *State Machine* message is related to an instance in the *State Machine*. The first time the local agent encounters an instance, it allocates it on a round robin basis to the next remote agent that is running the designated control program for this instance. Once an instance is allocated to a remote agent, forward it to that agent.
- control program control: Forward to the all the remote agents that are configured for the control program designated in the message.

Messages from the remote agents are dealt with as follows:

- *State Machine* inputs:
Adjust the time stamp in the messages with the relative time difference and forward to *orkhestra*.
- control program control:
Forward to *orkhestra*.

6.3.2 Shutdown

The local agent will shutdown when requested so by *orkhestra*, this is done by *orkhestra* closing the connection to the local agent. The local agent before terminating:

- Close the log file, if open.
- Close all connections to the remote agents, this will result in them shutting down.

See section [6.3.3](#) on page [65](#) for exceptions and errors.

6.3.3 Error Shutdown

When the agent encounters a severe error, that needs human interaction, it will notify *orkhestra* of this with a `ORKAGENT_FAILED` message, informing *orkhestra* as to the reason for this. On receiving this, *orkhestra* will shutdown the local agent by closing the connection to it.

The following constitutes a severe error:

- Configuration file errors.
- Unable to connect to all the remote agents.
- Receiving a start up message for an unknown control program, that is not defined.
- Input from *orkhestra* out of context. This constitutes a bug of some sorts, that needs to be fixed.
- Closure of a remote agent's connection.
- Receiving a `ORKAGENT_FAILED` message from a remote agent.
- Unable to start a control program for any reason - the remote agent that could not start the control program will send a `ORKAGENT_FAILED` message.

6.4 Remote Agent

A remote agent at startup do not have any configuration, other than the TCP/IP port to listen on on for a connection from a *orkhestra*'s local agent. Once a local agent has connected, the local agent will send the applicable configuration details from the agents configuration file (section [7](#) on page [67](#)). The remote agent is now ready to run control programs and route the relevant traffic from/to *orkhestra* and the control programs. When the remote program shutdown, it reverts to its initial state, waiting for a connection, see section [6.4.3](#) on page [66](#).

6.4.1 Synopsis

```
Code Magus Limited Orkhestra V2.0: build 2009-11-19-17.27.27
[orkagentr] $Id: orkagentr.c,v 1.1.1.1 2011/06/13 12:06:24 janvlok Exp $
Copyright (c) 2009 by Code Magus Limited. All rights reserved.
[Contact: stephen@codemagus.com].
Usage: orkagentr [OPTION...]
  -p, --port=<port>      Port to listen on
  -v, --verbose          Verbose output
```

```
Help options:
  -?, --help             Show this help message
  --usage                Display brief usage message
```

Where:

- ‘-p|--port’ Specifies the port number to listen on for a connection from the local agent.
- ‘-v|--verbose’ When specified, the remote agent operates in a verbose manner.

6.4.2 Operation

6.4.3 Shutdown

Shutdown is logical and not termination of the program: reset to start up state, waiting for a local agent to connect.

The remote agent will shutdown when the local agent closes the `socket` connection.

- Close the log file, if open.
- Reset, clear everything and wait for a connection from the local agent.

6.4.4 Error Shutdown

When the remote agent encounters a severe error, that needs human interaction, it will send a `ORKAGENT_FAILED` message, using the structure `orkagent_info_msg` to *orkhestra* via the local agent describing the reason. On receiving this, *orkhestra* will shutdown the local agent by closing the connection `pipes` to the local agent.

The following constitutes a severe error:

- Input out of context. This constitutes a bug of some sorts, that needs to be fixed.
- Closure of the local agent’s connection.
- Unable to start a control program for any reason.

7 Agent Configuration File

This section describe the configuration file required by the local agent (section 6.3 on page 63). It optionally specifies a log file for the local agent and describes the definition and configuration of the remote agents.

7.1 Elements

The elements for defining the configuration file comprise reserved words, identifiers, string literals, comments and integers. The definitions are free format and white spaces have no grammatical meaning except where they might appear within string literals.

7.1.1 Comments

Comments are introduced by using double minus sign ('--') and continue up to the end of the current input line.

Examples:

```
-- File: agent.cfg
--
-- Configuration for running control programs orksample and orksample2
-- remotely:
--   orksamle is to run on both blackbox and codemagus.
--   orksamle2 only run on blackbox.
-- The control program names as specified here are as per orchestra's
-- configuration.
```

7.1.2 Reserved Words

Reserved words have a special meaning in terms of directing the parsing of commands. The reserved words are:

agent	host	port	setenv
control	copies	log	program
end	open	programs	

Table 4: Agent configuration file reserved words

7.1.3 Identifiers

An *Identifier* is case sensitive, it starts with a letter which can be followed by any number of letters, digits or the under-score character.

Examples:

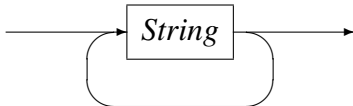
```
orksample orksample_2
```

7.1.4 Strings

Strings are:

- any sequence of characters (except double quotes and the newline character) enclosed by double quotes.
- any sequence of characters (except single quotes and the newline character) enclosed by single quotes.

Strings cannot span source text lines, but they may be concatenated:

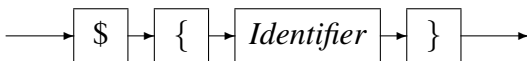


Examples:

```
open log "text(${LOGSPATH}/local_agent_D${DATE_YYYYMMDD}_T${TIME_HHMMSS}"
        ".txt,mode=w)";
```

7.1.5 Environment Variables

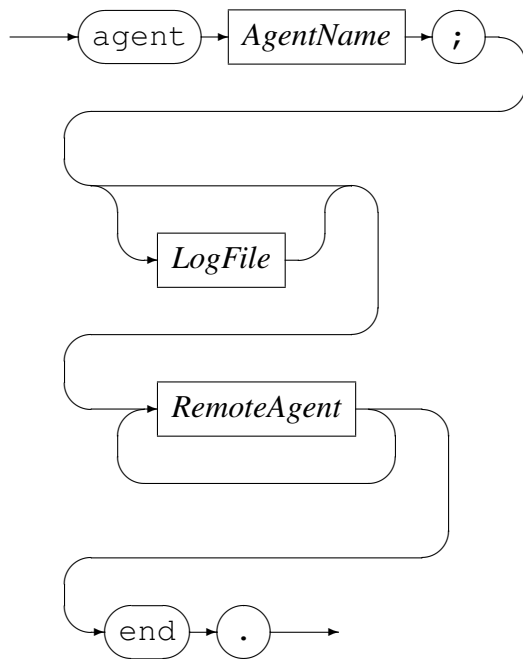
EnvironmentVariable



Environment variables are expanded to their value when encountered in command input text.

7.2 Syntax and Semantics

The configuration file starts by defining a name for the local agent, followed by an optional log file for the local agent and then the definition of remote agents.



AgentName

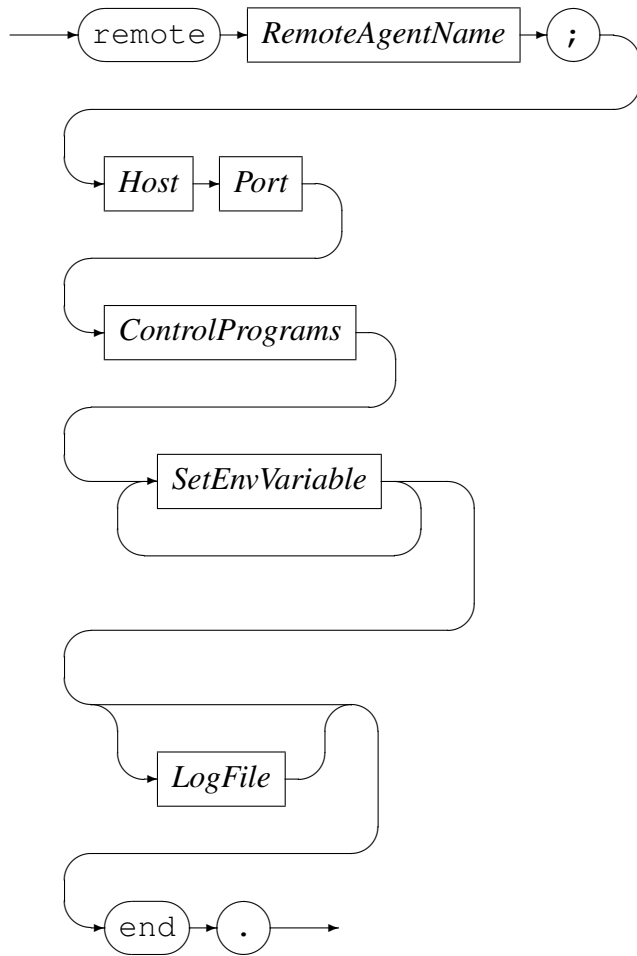


AgentName identifies the name of the local agent, this name is used for agent message identification in the log file and trace messages. The definition of *LogFile* is describe in the next section, see section 7.2.1 on page 69. appendix B on page 75.

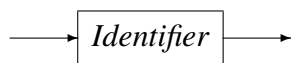
See appendix B on page 75 for an example of a sample configuration file.

7.2.1 Remote Agent definition

The definition of a remote agent starts with the key word `remote` and is terminated with `end`;



RemoteAgentName

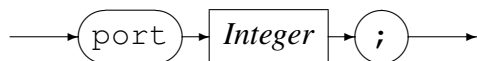


RemoteAgentName identifies the name of the remote agent.

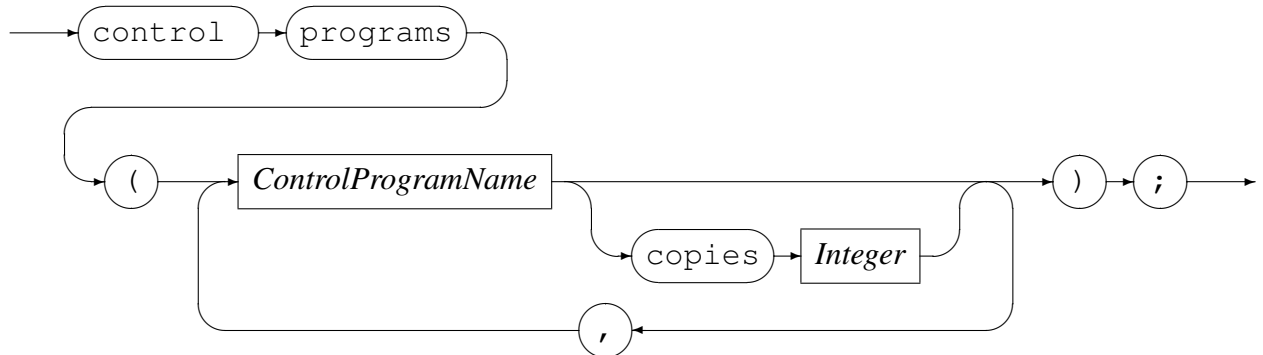
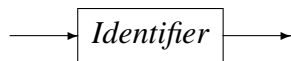
Host



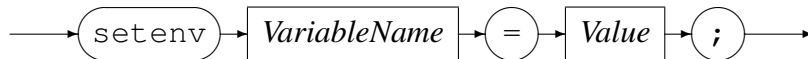
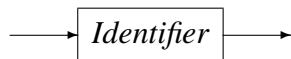
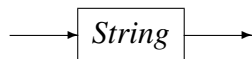
Port



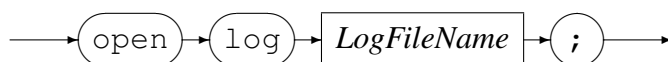
Host and *Port* specifies the remote agent's TCP / IP address. *IPAddress* can be specified as a host name or by using the Internet notation of dots and numbers.

ControlPrograms*ControlProgramName*

This specifies a comma separated list of one or more *orkhestra* control programs that this remote agent is to run, on request from *orkhestra*. If *copies* is specified for a control program then the remote agent will start that many copies of that control program; This overrides the number of copies requested from *orkhestra*.

SetEnvVariable*VariableName**Value*

The remote agent sets the environment variable *VariableName* to *Value*. This is done before any control programs are started.

LogFile*LogFileName*

Open a log file with the name as specified by *LogFileName* ;

A Sample State machine: *orksample.mch*

```
{
This is a very simple machine:
```

At start-up the first transition for a state machine instance is to the idle state, and after it transacts, it is back to the idle state. Before entering the idle state an idle timer will be set, for the duration a instance (device) spends in the idle state. The value definition 'think_time' is used for the idle time.

Once the timer expires:
Output 'connect' for a connection request to the control program and wait for input as to the outcome. If the input 'disconnect', set the idle timer and back to the idle state.

Input 'connect' tells us that the instance is connected, and just to demonstrate the use of the choose() function, for some instances a canned message (GENERIC_REQUEST) are requested to be send, an for others a immediate 'disconnect' are outputted, which eventually leads back to the idle state. The relevant weights for this are defined by the weight distribution 'what_to_do'.

When outputting the 'GENERIC_REQUEST', a timeout timer is set, using the value definition 'timeout_value'.

Waiting for the response, we need to cater for:

- . The time out timer that was set expired - output 'disconnect' and via waiting for the 'disconnect' input, back to the idle state.
- . Input 'disconnect' - set the idle time and back to idle state.
- . Input 'GENERIC_RESPONSE', output 'disconnect' and via waiting for the 'disconnect' input, back to the idle state.

```
}
machine orksample();
```

```
    created by ("Jan Vlok");
    description("orkhestra sample control program.");
    date("2008-01-16T10:51:18");
    target("Demonstration");
    control_program (orksample);
    modified by("Jan Vlok");
```

```
value timeout_value
(
    title("Response time out")
    description("Time in milliseconds to wait for a response message")
    constant(50000)
);
```

```
value think_time
(
    title("Device Idle time")
```

A SAMPLE STATE MACHINE: *ORKSAMPLE.MCH*

```
description("Time in milliseconds an instance will be in the idle state")
distribution(class(exponential) min(10000) max(3000000) mu(100000))
);

weight distribution what_to_do
(
  title("Disconnect or continue")
  description("Some instances we want to disconnect, before sending a message")
  weights(disconnect(1) continue(2))
);

begin

-- This wild card state [*] is the default for a disconnect in any state,
-- it will only, if a state do not have a transition for disconnect.
-- If we get a disconnect in any state, there is not much we can or want to
-- do about it.
[*]
  disconnect:
    start_timer(device_ready, think_time);
[device_idle]
;

-- The first state defined will be by default initial state, so at
-- start up of an instance, this is the initial state, and the default.
-- input is startup.
[startup]
  startup:
    start_timer(device_ready,think_time);
[device_idle]
;

-- Instance is ready for the next message, once the timer has expired.
-- Get it connected.
--
[device_idle]
  timer_expire(device_ready):
    connect;
[wait_connection]
;

-- Waiting for for the termination of a connection, a disconnect
-- was requested.
[wait_disconnect]
  disconnect:
    start_timer(device_ready,think_time);
[device_idle]
;

-- Just to demonstrate the choose() action:
-- Some connection we are going to terminate, without sending
-- a message, using the what_to_do weight distribution,
```

```
-- that is to say, once we have a connection established.
[wait_connection]
    connect:
        choose(what_to_do);
[choice_what_to_do]
;

-- OK disconnect
[choice_what_to_do]
    choice(disconnect):
        disconnect;
[wait_disconnect]
;

-- Continue to send a message
[choice_what_to_do]
    choice(continue):
        GENERIC_REQUEST;
        start_timer(msg_timed_out,timeout_value);
[wait_response]
;

-- Bummer - Bad parameters give to the control program?
--
[wait_connection]
    connect_error:
[final]
;

-- Wait for the response
[wait_response]
    GENERIC_RESPONSE:
        cancel_timer(msg_timed_out);
        disconnect;
[wait_disconnect]
;

[wait_response]
    timer_expire(msg_timed_out):
        disconnect;
[wait_disconnect]
;

[wait_response]
    disconnect:
        cancel_timer(msg_timed_out);
        start_timer(device_ready,think_time);
[device_idle]
;

end
```

B Sample Agent Configuration File: *orkhestra_agent.cfg*

```
-- File: orkhestra_agent.cfg
--
-- Configuration for running control programs orksample and orksample2
-- remotely:
--   orksamle is to run on both blackbox and codemagus.
--   orksamle2 only run on blackbox.
-- The control program names as specified here are as per orkhestra's
-- configuration.
--
-- Author: Jan Vlok.
--
-- Copyright (c) 2009 Code Magus Limited. All rights reserved.
--
-- $Author: janvlok $
-- $Date: 2016/04/07 07:59:12 $
-- $Id: orkhestra_agent.cfg,v 1.3 2016/04/07 07:59:12 janvlok Exp $
-- $Source: /home/cvs/cvsroot/orkhestra/orkhestra_agent.cfg,v $
-- $Revision: 1.3 $
-- $State: Exp $
--
-- $Log: orkhestra_agent.cfg,v $
-- Revision 1.3 2016/04/07 07:59:12  janvlok
-- Change references to orkhestra
--
-- Revision 1.2 2015/04/23 14:31:01  janvlok
-- Enhanced remote agent to start control programs with copies specified in
-- the agent configuration file, overriding orkhestra's request for copies to
-- start with.
--
-- Revision 1.1 2011/06/15 10:15:32  janvlok
-- Initial import of documentation
--
agent orhestra_agentr;

    open log "text(${LOGSPATH}/local_agent_D${DATE_YYYYMMDD}_T${TIME_HHMMSS}"
        ".txt,mode=w)";

remote agent blackbox
    host blackbox.africa.nedcor.net;
    port 22221;
    control programs (orksamle copies 16,orksamle2);
    setenv CODEMAGUS_SOURCE="${HOME}/dev";
    setenv LOGSPATH="${CODEMAGUS_SOURCE}/orkhestra/test-agent/logs";
    setenv SCRIPTS="${CODEMAGUS_SOURCE}/orksamle/scripts";
    setenv CODEMAGUS_MSGLEVEL="VERBOSE";
    open log "text(${LOGSPATH}/blackbox_agent_D${DATE_YYYYMMDD}_T"
        "${TIME_HHMMSS}.txt,mode=w)";

end;
```

```
remote agent codemagus
  host codemagus.it.nednet.co.za;
  port 22222;
  control programs (orksample copies 3);
  setenv CODEMAGUS_SOURCE="${HOME}/dev";
  setenv LOGSPATH="${CODEMAGUS_SOURCE}/orkhestra/test-agent/logs";
  setenv SCRIPTS="${CODEMAGUS_SOURCE}/orksample/scripts";
  setenv CODEMAGUS_MSGLEVEL="NOTSET";
  open log "text (${LOGSPATH}/codemagus_agent_D${DATE_YYYYMMDD}_T"
    "${TIME_HHMMSS}.txt,mode=w)";
end;

end.
```

References

- [1] orkhestra: Control Program API Reference Version 1. CML Document CML00084-01, Code Magus Limited, June 2011. [PDF](#).