Job Scheduling

JOBSCHEDULER

API Documentation
March 2015

Contact Information

Software- and Organisations-Service GmbH
Giesebrechtstr. 15
D-10629 Berlin

Telephone +49 (30) 86 47 90-0
Telefax +49 (30) 8 61 33 35
Mail info@sos-berlin.com
Web www.sos-berlin.com

Table of Contents

1 Overview

Supported Languages:

Java	<script language="Java">	Provides jobs in Java.
java:JavaScript	<script language="java:JavaScript">	Provides jobs in JavaScript. Usage of the "Rhino with Beans" implementation.
javax.script:rhino	<script language="javax.script:rhino">	Provides the script language Rhino, that implements the "javax.script" scripting API. Other script languages implementing the "javax.script" scripting API can be used with <script language="javax.script:*language*">.
Spidermonkey (32bit)	<script language="Spidermonkey">	Provides jobs in JavaScript. Usage of the "spidermonkey" Implementation. Only available on 32 bit.
PowerShell	<script language="PowerShell">	Provides jobs in Jobs in PowerShell. Only available on Windows.
VBScript	<script language="VBScript">	Provides jobs in Jobs in VBScript. Only available on Windows.
Perl	<script language="Perl">	Provides jobs in Perl.

Jobs which use the JobScheduler API may be implemented in Java, JavaScript (the Mozilla Spidermonkey implementation) and Perl (Perl 5.8 is supported for Unix and an ActiveState implementation is required for Windows). In addition, JScript, VBScript and Powershell scripting languages are available on Microsoft Windows systems.

Since JobScheduler Version 1.3.10 the javax.script package is supported. Using this plugin different implementations for a lot of script languages such as javascript (Rhino implementation), groovy and python are available. The Spidermonkey implementation for javascript is marked as "deprecated".

Since JobScheduler Version 1.5 the Spidermonkey implementation of javascript is available only on 32 bit. For using javascript on 64 bit choose the "Rhino with Beans" implementation. This implementation is available on 32 bit as well as on 64 bit. The "Rhino with Beans" implementation is an extension to rhino and it replaces the Spidermonkey implementation. The "Rhino with Beans" implementation suits the purpose to port already existing javascript jobs to 64 bit. The usage of both implementations differs slightly, see: Differences between the Spidermonkey and "Rhino with Beans" engines

The following table shows the used engine dependent on the language attribute in <script language="...">

Spidermonkey	Spidermonkey	---
java:JavaScript	Rhino with Beans	Rhino with Beans

In the language attribute the values "Spidermonkey" and "JavaScript" are equivalent (Spidermonkey engine).

Furthermore in the language attribute the values "java:JavaScript", "java:Rhino" and "java:ECMAScript" are equivalent ("Rhino with Beans" engine).

In order to use the API in Powershell, a seperate download and installation of the JobScheduler Powershell Adapter is required. See http://sourceforge.net/apps/mediawiki/jobscheduler/index.php?title=JSApi_Powershell

Jobs are implemented according to the `Job_impl` interface. JobScheduler objects may be accessed using this interface either directly or indirectly.

2 Java API

The following classes are available for Java:

2.1 Error

2.1.1 code

The error code

Syntax: `String` error. **`code`** `()`

2.1.2 is_error

`true`, should an error have occurred

Syntax: `boolean` error. **`is_error`** `()`

2.1.3 text

The error text (with error code)

Syntax: `String` error. **`text`** `()`

2.2 Job

A task can either be waiting in the order queue or be running.

2.2.1 clear_delay_after_error

Resets all delays which have previously been set using `delay_after_error`

Syntax: void spooler_job. **`clear_delay_after_error`** `()`

2.2.2 clear_when_directory_changed

Resets directory notification for all directories which have previously been set using `start_when_directory_changed()`

Syntax: void spooler_job. **clear_when_directory_changed** ()

2.2.3 configuration_directory

Directory for the job configuration file should dynamic configuration from hot folders be used

Syntax: String spooler_job. **configuration_directory** ()

"", when a job does not come from a configuration directory.

2.2.4 delay_after_error

Delays the restart of a job in case of an error

Syntax: void spooler_job.set_**delay_after_error** (int error_steps, double seconds)

Syntax: void spooler_job.set_**delay_after_error** (int error_steps, String hhmm_ss)

Example:

```
spooler_job.set_delay_after_error(  2,  10 );          // A 10 second delay after the
2nd consecutive error
spooler_job.set_delay_after_error(  5, "00:01" );      // One minute delay after the
5th consecutive error
spooler_job.set_delay_after_error( 10, "24:00" );      // A delay of one day after the
10th consecutive error
spooler_job.set_delay_after_error( 20, "STOP" );       // The Job is stopped after the
20th consecutive error
```

Should a (first) error occur whilst a job is being run, the JobScheduler will restart the job immediately.
However, after between two and four consecutive errors, the JobScheduler will wait 10 seconds before restarting the job;

After between five and nine consecutive errors, the job will be restarted after a delay of one minute; After between ten and nineteen errors, the delay is 24 hours.

The job is stopped after the twentieth consecutive error.

A delay can be specified, should a particular number of errors occur in series. In this case the job will be terminated and then restarted after the time specified.

This method call can be repeated for differing numbers of errors. A different delay can be specified for each new method call.

It is possible to set the value of the seconds_or_hhmm_ss parameter to "STOP" in order to restrict the number of (unsuccessful) repetitions of a job. The job then is stopped when the number of consecutive errors specified is reached.

A good position for this call is spooler_init().

See <delay_after_error>.

Parameters:

error_steps	The number of consecutive errors required to initiate the delay
seconds_or_hhmm_ss	The delay after which the job will be rerun

2.2.5 delay_order_after_setback

Delays after an order is setback

Syntax: void spooler_job.set_**delay_order_after_setback** (int setback_count, double seconds)

Syntax: void spooler_job.set_**delay_order_after_setback** (int setback_count, String hhmm_ss)

Example: in javascript

```
spooler_job.delay_order_after_setback( 1 ) = 60;         // for the 1st and 2nd
consecutive setbacks of an order:
               // delay the order 60s.


spooler_job.delay_order_after_setback( 3 ) = "01:00";    // After the 3rd consecutive
setback of an order,
               // the order will be delayed an hour.


spooler_job.max_order_setbacks = 5;                 // The 5th setback sets the order
to the error state
```

A job can delay an order which is currently being carried out with Order.setback(). The order is then positioned at the rear of the order queue for that job and carried out after the specified time limit.

The number of consecutively occurring setbacks for an order is counted. The delay set after a setback can be changed using delay_order_after_setback in the event of consecutively occurring setbacks.

See
<delay_order_after_setback>,

Order.setback(),

Job.max_order_setbacks,

Job_chain.add_job(),

Job.delay_after_error().

Parameters:

setback_count	The number of consecutive errors and therefore setbacks for a job. The setback delay can be varied according to this parameter.
seconds_or_hhmm_ss	Time limit for the setback of the order. After expiry of the time limit, the order is reprocessed in the same job.

2.2.6 folder_path

The directory in which the job is to be found.

Syntax: `String` spooler_job. **`folder_path`** `()`

`""`, when the job does come from the local (`<config configuration_directory="">`) configuration file.

Returns the job part relative to the live directory. The path is to start with a slash ("/") and all path components are to be separated by slashes.

Examples:

- `"/somewhere/excel"` will be returned for the `c:\scheduler\config\live\somewhere\excel\sample.job.xml` job;
- `"/"` returned for the `c:\scheduler\config\live\sample.xml` job and
- `""` (an empty string) returned for a job outside the live directory.

2.2.7 include_path

Value of the `-include-path=` option

Syntax: `String` spooler_job. **`include_path`** `()`

See `-include-path`.

2.2.8 max_order_setbacks

Limits the number of setbacks for an order

Syntax: void spooler_job.`set_` **`max_order_setbacks`** `( int )`

An order state is set to "error" (see `Job_chain_node.error_state`) when it is set back more than the number of times specified here (see `Order.setback()`).

See `Job.delay_order_after_setback` and `<delay_order_after_setback is_maximum="yes">`.

2.2.9 name

The job path beginning without a backslash

Syntax: `String` spooler_job. **`name`** `()`

See `<job name="">`.

2.2.10 order_queue

The job order queue

Syntax: Order_queue spooler_job. **order_queue** ()

Example: in javascript

```
spooler_log.info( 'order=' + ( spooler_job.order_queue ? "yes" : "no" ) );
```

Every job order (<job order="yes">) has an order queue. This queue is filled by the job chain to which the job belongs.

See Job_chain.add_order() , and Job_chain.add_job() .

Returned value:
Order_queue

`null`, should the job have no queue (for <job order="no">).

2.2.11 process_class

The process class

Syntax: Process_class spooler_job. **process_class** ()

See <job process_class=""> .

Returned value:
Process_class

2.2.12 remove

Removes a job

Syntax: void spooler_job. **remove** ()

The job is stopped - i.e. current tasks are terminated and no new ones are started. The job will be removed as soon as no more tasks are running.

Tasks queuing are ignored.

When no job task is running, the remove() function deletes the job immediately.

Job orders (<job order="yes">) cannot be removed.

See <modify_job cmd="remove"> .

2.2.13 start

Creates a new task and places it in the task queue

Syntax: Task spooler_job. **start** (Variable_set variables (optional))

Example:

```
spooler.job( "job_a" ).start();

sos.spooler.Variable_set parameters = spooler.create_variable_set();
parameter.set_var( "my_parameter", "my_value" );
parameter.set_var( "other_parameter", "other_value" );
spooler.job( "job_a" ).start( parameters );
```

The parameters are available to the Task.params task. Two parameters are particularly relevant here:

"spooler_task_name"	gives the task a name which then appears in the status display, e.g. in the web interface.
"spooler_start_after"	specifies a time in seconds (real number), after which the task is to start. The JobScheduler <run_time> is ignored in this case.

See Spooler.create_variable_set(), Spooler.job, Variable_set.value.

Returned value:
Task

2.2.14 start_when_directory_changed

Monitors a directory and starts a task should a notification of a change be received

Syntax: void spooler_job. **start_when_directory_changed** (java.io.File directory_path, String filename_pattern (optional))

Syntax: void spooler_job. **start_when_directory_changed** (java.io.File directory_path, String filename_pattern (optional))

Syntax: void spooler_job. **start_when_directory_changed** (String directory_path)

Syntax: void spooler_job. **start_when_directory_changed** (String directory_path, String filename_pattern (optional))

Example: in javascript

```
spooler_job.start_when_directory_changed( "c:/tmp" );

// only relevant for files whose names do not end in "~".
spooler_job.start_when_directory_changed( "c:/tmp", "^.*[^~]$" );
```

Should there not be a task belonging to this job running and a notification be received that a change in the directory being monitored has occurred (that a file has been added, changed or deleted), then this change can be used to prompt the JobScheduler to start a task if the current time falls within that allowed by the <run_time> parameter.

This method can be called a more than once in order to allow the monitoring of a number of directories. A repeat call can also be made to a directory in order to reactivate monitoring - if, for example, it has not been possible to access the directory.

This method call can be coded in the JobScheduler start script or in the `spooler_init()` method. In the latter case, the job must have been started at least once in order for the method call to be carried out. The `<run_time once="yes">` setting should be used for this.

The job should be regularly `<run_time repeat="">` restarted and `<delay_after_error>` set.

The same setting can be made in the XML configuration using the `<start_when_directory_changed>` element.

Parameters:

`directory_path`	the address of the directory being monitored
`filename_pattern`	restricts monitoring to files whose names correspond with the regular expression used.

2.2.15 state_text

Free text for the job state

Syntax: void spooler_job.`set_state_text ( String )`

Example:

```
spooler_job.set_state_text( "Step C succeeded" );
```

The text will be shown in the HTML interface.

2.2.16 title

The job title

Syntax: `String` spooler_job. **`title`** `()`

Example:

```
spooler_log.info( "Job title=" + spooler_job.title() );
```

See `<job title="">`.

2.2.17 wake

Causes a task to be started

Syntax: void spooler_job. **`wake`** `()`

Starts a task, should the job have the `pending` or `stopped` states.

See `Job.start()`.

2.3 Job_chain - job chains for order processing

A job chain is a series of jobs (job chain nodes). Orders (`Order`) proceed along these chains.

Every position in a job chain is assigned a state and a job. When an order is added to the job chain, it is enqueued by the JobScheduler according to the state of the order. The job assigned to this position then carries out the order.

Additionally, each position in a job chain has a successor state and an error state. The JobScheduler changes the state of an order after each job in the job chain has been processed. Should the job step return `(spooler_process) true`, then the JobScheduler sets the succeeding state; otherwise it sets the error state. The order then moves to another position in the job chain as defined by the new state. However, this does not apply when the state is changed during execution with `Order.state`.

A job chain is created using `Spooler.create_job_chain()`; it is filled using `Job_chain.add_job()` and `Job_chain.add_end_state()` and finally made available with `Spooler.add_job_chain()`.

Every node is allocated a unique state. Therefore either `Job_chain.add_job()` or `Job_chain.add_end_state()` must be called once for every state.

Example:

```
Job_chain my_job_chain = spooler.create_job_chain();
my_job_chain.set_name( "JobChain" );

my_job_chain.add_job( "job_100", 100,  200, 999 );
my_job_chain.add_job( "job_200", 200, 1000, 999 );
my_job_chain.add_end_state(  999 );
my_job_chain.add_end_state( 1000 );
spooler.add_job_chain( my_job_chain );
```

2.3.1 add_end_state

Adds the end state to a job chain

Syntax: void job_chain. **`add_end_state`** `( String state )`

This state is not assigned a job. An order that reaches the final state has completed the job chain and will be removed from the chain.

2.3.2 add_job

Adds a job to a job chain

Syntax: void job_chain. **`add_job`** `( String job_name, String input_state, String output_state, String error_state )`

2.3.3 add_or_replace_order

Adds an order to a job chain and replaces any existing order having the same identifier

Syntax: void job_chain. **add_or_replace_order** (Order order)

Should the job chain already contain an order with the same identifier, then this order will be replaced. More accurately: the original order will be deleted and the new one added to the job chain.

As long as an existing order having the same identifier as the new order is being carried out, both orders will be present. However, the original order will have already been deleted from the job chain and database; it is only available to the current task and will completely disappear after it has been completed.

In this case the JobScheduler will wait until the original order has been completed before starting the new one.

See Job_chain.add_order() and Order.remove_from_job_chain()

2.3.4 add_order

Adds an order to a job chain

Syntax: void job_chain. **add_order** (Order order)

Should an order already exist on another job chain, then the JobScheduler removes the order from this other chain.

An order is allocated to the job order queue corresponding to its state, and positioned according to its priority.

The job chain must be specified for the JobScheduler using <job_chain> or Spooler.add_job_chain().

Should an order with the same Order.id already exist in a job chain, then an exception with the error code SCHEDULER-186 is returned. However, see also Job_chain.add_or_replace_order().

Returned value:
Order

2.3.5 name

The name of a job chain

Syntax: void job_chain.set_**name** (String)

Syntax: String job_chain. **name** ()

Example:

```
Job_chain job_chain = spooler.create_job_chain();
job_chain.set_name( "JobChain" );
```

2.3.6 node

The job chain nodes with a given state

Syntax: Job_chain_node job_chain. **node** (String state)

Returned value:
Job_chain_node

2.3.7 order_count

The number of orders in a job chain

Syntax: int job_chain. **order_count** ()

2.3.8 order_queue

= node(state).job().order_queue()

Syntax: Order_queue job_chain. **order_queue** (String state)

Returns the order queue which has a given state.

Returned value:
Order_queue

2.3.9 orders_recoverable

Syntax: void job_chain.set_**orders_recoverable** (boolean)

Syntax: boolean job_chain. **orders_recoverable** ()

See <job_chain orders_recoverable="">.

2.3.10 remove

Job chain deletion

Syntax: void job_chain. **remove** ()

Should orders in a job chain still be being processed (in spooler_process()) when the chain is to be deleted, then the JobScheduler will wait until the last order has been processed before deleting the chain.

Orders remain in the database. Should a new job chain be added which has the same name as a deleted job chain (Spooler.add_job_chain()), then the JobScheduler will reload any orders from the original job chain which have

remained in the database. Note however, that the states of the orders in the new job chain should be the same as those in the original chain at the time of its deletion.

2.3.11 title

Syntax: void job_chain.set_**title** (String)

Syntax: String job_chain. **title** ()

See <job_chain title="">.

2.4 Job_chain_node

A job chain node is assigned a position in a job chain (Job_chain). The following elements make up a job chain node: a state, a job, a successor state and an error state.

A job chain node is created either using Job_chain.add_job() or Job_chain.add_end_state().

2.4.1 action

Stopping or missing out job chain nodes

Syntax: void node.set_**action** (String)

Syntax: String node. **action** ()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "my_job_chain" ).node( 100 );
job_chain_node.set_action( Job_chain_node.ACTION_NEXT_STATE );
```

This option is not possible with distributed job chains.

Possible settings are:

action="process"
: This is the default setting. Orders are carried out.

action="stop"
: Orders are not carried out, they collect in the order queue.

action="next_state"
: Orders are immediately handed over to the next node as specified with next_state.

See also <job_chain_node.modify action="">.

Character string constonants are defined in Java:

- Job_chain_node.ACTION_PROCESS

- Job_chain_node.ACTION_STOP
- Job_chain_node.ACTION_NEXT_STATE

2.4.2 error_node

The next node in a job chain in the event of an error

Syntax: Job_chain_node node.**error_node**()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "error state=" + job_chain_node.error_node().state() );   //
"state=999"
```

Returned value:
Job_chain_node

null, in the event of no error node being defined (the error state has not been specified)

2.4.3 error_state

State of a job chain in event of an error

Syntax: String node.**error_state**()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "error state=" + job_chain_node.error_node().state() );   // "error
state=999"
```

2.4.4 job

The job allocated to a node

Syntax: Job node.**job**()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "job=" + job_chain_node.job().name() );                    //
"job=job_100"
```

Returned value:
Job_

2.4.5 next_node

Returns the next node or null if the current node is assigned the final state.

Syntax: Job_chain_node_ node. **next_node** ()

Returned value:
Job_chain_node_

2.4.6 next_state

The order state in a job chain after successful completion of a job

Syntax: String node. **next_state** ()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "next state=" + job_chain_node.next_state() );            //
"state=200"
```

2.4.7 state

The valid state for a job chain node

Syntax: String node. **state** ()

Example:

```
Job_chain_node job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.info( "state=" + job_chain_node.state() );                       //
"state=100"
```

2.5 Job_impl - Super Class for a Job or the JobScheduler Script

Job methods are called in the following order:

```
spooler_init()
    spooler_open()
        spooler_process()
        spooler_process()
        ...
    spooler_close()
    spooler_on_success() or spooler_on_error()

spooler_exit()
```

None of these methods must be implemented. However, it is usual that at least the spooler_process() method is implemented.

An error during carrying out a job script whilst loading or during spooler_init() causes spooler_on_error(). to be called. The job is then stopped and spooler_exit() called (although spooler_init() has not been called!). The script is then unloaded.

Note that spooler_on_error() must also be able to handle errors which occur during loading or in spooler_init().

Note also that spooler_exit() is called even though spooler_init() has not been called.

2.5.1 spooler

The JobScheduler base object

Syntax: Spooler **spooler**

Example:

```
spooler_log.debug( "The working directory of the JobScheduler is " +
spooler.directory() );
```

Returned value:
Spooler

2.5.2 spooler_close

Task end

Syntax: void **spooler_close** ()

This method is called after a job has been completed. The opposite of this method is spooler_open().

2.5.3 spooler_exit

Destructor

Syntax: void **spooler_exit** ()

Is called as the last method before the script is unloaded. This method can be used, for example, to close a database connection.

2.5.4 spooler_init

Initialization

Syntax: `boolean spooler_init ()`

The JobScheduler calls these methods once before `spooler_open()`. This is analog to `spooler_exit()`. This method is suitable for initializing purposes (e.g. connecting to a database).

Returned value:
`boolean`

`false` ends a task. The JobScheduler continues using the `spooler_exit()` method. When the task is processing an order, then this return value makes the JobScheduler terminate the job with an error. That is, unless a repeated start interval has been set using `Job.delay_after_error`

2.5.5 spooler_job

The job object

Syntax: `Job spooler_job`

Example:

```
spooler_log.info( "The name of this job is " + spooler_job.name() );
```

Returned value:
`Job`

2.5.6 spooler_log

Event logging object

Syntax: `Log spooler_log`

Example:

```
spooler_log.info( "Something has happened" );
```

Returned value:
`Log`

2.5.7 spooler_on_error

Unsuccessful completion of a job

Syntax: void **spooler_on_error** ()

Is called at the end of a job after an error has occurred (after spooler_close() but before spooler_exit()).

2.5.8 spooler_on_success

Successful completion of a job

Syntax: void **spooler_on_success** ()

This method is called by the JobScheduler after spooler_close() and before spooler_exit() ; should no error have occurred.

2.5.9 spooler_open

The Start of a Task

Syntax: boolean **spooler_open** ()

This method is called immediately after spooler_init() . The opposite of this method is spooler_close() .

2.5.10 spooler_process

Job steps or the processing of an order

Syntax: boolean **spooler_process** ()

Processes a job step.

An order driven job stores the current order in Task. order .

The default implementation returns false. The implementation of an order driven job can set the successor state for an order by returning true.

Returned value:
boolean

In the event of standard jobs <job order="no"> : false the JobScheduler ends processing of this job; true> the JobScheduler continues calling the spooler_process() method.

In the event of order driven jobs <job order="yes"> : false the order acquires the error state (s. Job chain node and <job chain node>). true the order acquires the next state or is terminated if the next state is the final state. This, however, does not apply when the state is changed during execution using Order. state .

2.5.11 spooler_task

The task object

Syntax: Task **spooler_task**

Example:

```
spooler_log.info( "The task id is " + spooler_task.id() );
```

Returned value:
Task

2.6 Lock

See also <lock name="">.

Example: in javascript

```
var locks = spooler.locks;
var lock = locks.create_lock();
lock.name = "my_lock";
locks.add_lock( lock );
```

2.6.1 max_non_exclusive

Limitation of non-exclusive allocation

Syntax: void lock.set_**max_non_exclusive** (int)

Syntax: int lock. **max_non_exclusive** ()

The default setting is unlimited (231-1), which means that with <lock.use_exclusive="no"> any number of non-exclusive tasks can be started (but only one exclusive task).
The number cannot be smaller than the number of non-exclusive allocations.

See also <lock max_non_exclusive="">.

2.6.2 name

The lock name

Syntax: void lock.set_**name** (String)

Syntax: String lock. **name** ()

The name can only be set once and cannot be changed.

See also <lock name="">.

2.6.3 remove

Removes a lock

Syntax: void lock. **remove** ()

Example: in javascript

```
spooler.locks.lock( "my_lock" ).remove();
```

A lock can only be removed when it is not active - that is, it has not been allocated to a task and it is not being used by a job (<lock.use>).

See also <lock.remove>.

2.7 Locks

2.7.1 add_lock

Adds a lock to a JobScheduler

Syntax: void locks. **add_lock** (Lock lck)

2.7.2 create_lock

Creates a new lock

Syntax: Lock locks. **create_lock** ()

Returns a new lock Lock. This lock can be added to the JobScheduler using Locks.add_lock().

Returned value:
Lock

2.7.3 lock

Returns a lock

Syntax: Lock locks. **lock** (String lock_name)

An exception will be returned if the lock is unknown.

Returned value:
Lock

2.7.4 lock_or_null

Returns a lock

Syntax: Lock locks. **lock_or_null** (String lock_name)

Returned value:
Lock

null, when the lock is unknown.

2.8 Log - Logging

The spooler_log method can be used in a job or in the JobScheduler start script with the methods described here.
Notification by e-mail

The JobScheduler can send a log file after a task has been completed per e-mail. The following properties define in which cases this should occur.

- Log.mail_on_error,
- Log.mail_on_warning,
- Log.mail_on_process,
- Log.mail_on_success and
- Log.mail_it

Only the end of a task - and not the end of an order - (i.e. spooler_process()) can initiate the sending of e-mails. However, see Task.end() .

The Log.mail method makes the Mail object available, which in turn addresses the mails.

Example:

```
spooler_log.info( "Something for the Log" );

spooler_log.set_mail_on_warning( true );
spooler_log.mail().set_from   ( "scheduler@company.com" );
spooler_log.mail().set_to     ( "admin@company.com" );
spooler_log.mail().set_subject( "Task ended" );
```

2.8.1 debug

Debug message (level -1)

Syntax: void spooler_log. **debug** (String line)

2.8.2 debug1

Debug message (level -1)

Syntax: void spooler_log. **debug1** (String line)

2.8.3 debug2

Debug message (level -2)

Syntax: void spooler_log. **debug2** (String line)

2.8.4 debug3

Debug message (level -3)

Syntax: void spooler_log. **debug3** (String line)

2.8.5 debug4

Debug message (level -4)

Syntax: void spooler_log. **debug4** (String line)

2.8.6 debug5

Debug message (level -5)

Syntax: void spooler_log. **debug5** (String line)

2.8.7 debug6

Debug message (level -6)

Syntax: void spooler_log. **debug6** (String line)

2.8.8 debug7

Debug message (level -7)

Syntax: void spooler_log. **debug7** (String line)

2.8.9 debug8

Debug message (level -8)

Syntax: void spooler_log. **debug8** (String line)

2.8.10 debug9

Debug message (level -9)

Syntax: void spooler_log. **debug9** (String line)

2.8.11 error

Error Message (Level 1)

Syntax: void spooler_log. **error** (String line)

A job stops after a task has ended, should an error message have been written in the task log (spooler_log) and <job stop_on_error="no"> not have been set.

2.8.12 filename

Log file name

Syntax: String spooler_log. **filename** ()

2.8.13 info

Information message (Level 0)

Syntax: void spooler_log. **info** (String line)

2.8.14 last

The last output with the level specified

Syntax: `String` spooler_log. **`last`** `( int level )`

Syntax: `String` spooler_log. **`last`** `( String level )`

2.8.15 last_error_line

The last output line with level 2 (error)

Syntax: `String` spooler_log. **`last_error_line`** `()`

2.8.16 level

Limit protocol level

Syntax: void spooler_log.`set` **`level`** `( int )`

Syntax: `int` spooler_log. **`level`** `()`

Defines the level with which protocol entries should be written. Every protocol entry is given one of the following categories: `error`, `warn`, `info`, `debug1` to `debug9` (`debug1` is the same as `debug`).

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

The `-log-level` option has precedence over this parameter.

The `factory.ini` (section `[job]`, entry `log_level=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_level=…`) setting is overwritten by this parameter.

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`

-1:	debug
0:	info
1:	warn
2:	error

2.8.17 log

Writes in the log file with the specified level.

Syntax: void spooler_log. **log** (int level, String line)

2.8.18 log_file

Adds the content of a file to the log file

Syntax: void spooler_log. **log_file** (java.io.File path)

Syntax: void spooler_log. **log_file** (String path)

Log the content of a file with level 0 (info). An error occurring whilst accessing the file is logged as a warning.

Note that when executed on a remote computer with <process class remote scheduler=""> the file is read from the JobScheduler's file system and not that of the task.

2.8.19 mail

E-mail settings are made in the Mail Object

Syntax: void spooler_log.set_**mail** (Mail)

Syntax: Mail spooler_log. **mail** ()

Returned value:
Mail

2.8.20 mail_it

Force dispatch

Syntax: void spooler_log.set_**mail_it** (boolean)

If this property is set to true, then a log will be sent after a task has ended, independently of the following settings:

Log.mail_on_error__, Log.mail_on_warning__, Log.mail_on_success__, Log.mail_on_process__and Log.mail_on_error_.

2.8.21 mail_on_error

Sends an e-mail should a job error occur. Errors are caused by the Log.error() method or by any exceptions that have not been caught by a job.

Syntax: void spooler_log.set_**mail_on_error** (boolean)

Syntax: boolean spooler_log. **mail_on_error** ()

Content of the e-mail is the error message. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_error=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_error=…) setting is overwritten by this parameter.

Content of the e-mail is the error message. The log file is sent as an attachment.

2.8.22 mail_on_process

Sends an e-mail should a job have successfully processed the number of steps specified. Steps are caused by the spooler_process() methods:

Syntax: void spooler_log.set_**mail_on_process** (int)

Syntax: int spooler_log. **mail_on_process** ()

Causes the task log to be sent when a task has completed at least the specified number of steps - i.e. calls of spooler_process(). Because non-API tasks do not have steps, the JobScheduler counts each task as a single step.

Content of the e-mail is the success message. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_process=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_process=…) setting is overwritten by this parameter.

Content of the e-mail is the success message. The log file is sent as an attachment.

2.8.23 mail_on_success

Sends an e-mail should a job terminate successfully.

Syntax: void spooler_log.set_**mail_on_success** (boolean)

Syntax: boolean spooler_log. **mail_on_success** ()

The success message forms the content of the e-mail. The log file is sent as an attachment.

The factory. ini (section [job] , entry mail_on_success=…) setting is overwritten by this parameter.

The factory. ini (section [spooler] , entry mail_on_success=…) setting is overwritten by this parameter.

The success message forms the content of the e-mail. The log file is sent as an attachment.

2.8.24 mail_on_warning

Sends an e-mail should a job warning occur. Warnings are caused by the Log. warn() method.

Syntax: void spooler_log.set_**mail_on_warning** (boolean)

Syntax: boolean spooler_log. **mail_on_warning** ()

The warning forms the content of the e-mail. The log file is sent as an attachment.

The factory. ini (section [spooler] , entry mail_on_warning=…) setting is overwritten by this parameter.

The warning forms the content of the e-mail. The log file is sent as an attachment.

2.8.25 new_filename

A new name for the log file

Syntax: void spooler_log.set_**new_filename** (String)

Syntax: String spooler_log. **new_filename** ()

Sets the name of the log file. The JobScheduler copies a log into this file after a log has been made. This file is then available to other applications.

2.8.26 start_new_file

Only for the main log file: closes the current log file and starts a new one

Syntax: void spooler_log. **start_new_file** ()

2.8.27 warn

Warning (Level 2)

Syntax: void spooler_log. **warn** (String line)

2.9 Mail - e-mail dispatch

See Log. mail .

2.9.1 add_file

Adds an attachment

Syntax: void mail. **add_file** (String path, String filename_for_mail (optional) , String content_type (optional) , String encoding (optional))

Example:

```
spooler_log.mail().add_file( "c:/tmp/1.txt", "1.txt", "text/plain", "quoted-printable"
);
```

Parameters:

path	path to the file to be appended
filename_for_mail	The file name to appear in the message
content_type	"text/plain" is the preset value.
encoding	e.g. "quoted printable"

2.9.2 add_header_field

Adds a field to the e-mail header

Syntax: void mail. **add_header_field** (String field_name, String value)

2.9.3 bcc

Invisible recipient of a copy of a mail, (*blind carbon copy*)

Syntax: void mail.set_**bcc** (String)

Syntax: String mail. **bcc** ()

Example:

```
spooler_log.mail().set_bcc( "hans@company.com" );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_bcc=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_bcc=…`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

2.9.4 body

Message content

Syntax: void mail.`set_body ( String )`

Syntax: `String` mail. **`body ()`**

Example:

```
spooler_log.mail().set_body( "Job succeeded" );
```

Line feed / carriage return is coded with `\n` (`chr(10)` in VBScript).

2.9.5 cc

Recipient of a copy of a mail, (*carbon copy*)

Syntax: void mail.`set_cc ( String )`

Syntax: `String` mail. **`cc ()`**

Example:

```
spooler_log.mail().set_cc( "hans@company.com" );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_cc=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_cc=…`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

2.9.6 dequeue

Repeated attempts can be made to send messages from the `queue_dir` directory

Syntax: `int` mail. **`dequeue`** `()`

See `Mail.dequeue_log`, `factory.ini` (section `[spooler]`, entry `mail_queue_dir=…`).

Returned value:
`int`

The number of messages sent

2.9.7 dequeue_log

The `dequeue()` log

Syntax: `String` mail. **`dequeue_log`** `()`

Example: in javascript

```
var count = spooler_log.mail.dequeue();
spooler_log.info( count + " messages from mail queue sent" );
spooler_log.info( spooler_log.mail.dequeue_log );
```

See `Mail.dequeue()`.

2.9.8 from

Sender

Syntax: void mail.`set_`**`from`**`( String )`

Syntax: `String` mail. **`from`** `()`

Example:

```
spooler_log.mail().set_from( "scheduler@company.com" );
```

The factory.ini (section [job], entry log_mail_from=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_from=…) setting is overwritten by this parameter.

2.9.9 queue_dir

The directory used for returned e-mails

Syntax: void mail.set_**queue_dir** (String path)

Syntax: String mail. **queue_dir** ()

E-mails which cannot be sent (because, for example, the SMTP server cannot be contacted) are stored in this directory.

In order to send these e-mails later it is necessary to write a job which calls up the Mail.dequeue() method.

This setting is generally made in sos.ini (section [mail], entry queue_dir=…).

Environment variables (e.g. $HOME) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The factory.ini (section [job], entry mail_queue_dir=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_queue_dir=…) setting is overwritten by this parameter.

The sos.ini (section [mail], entry queue_dir=…) setting is overwritten by this parameter.

2.9.10 smtp

The name of the SMTP server

Syntax: void mail.set_**smtp** (String)

Syntax: String mail. **smtp** ()

Example:

```
spooler_log.mail().set_smtp( "mail.company.com" );
```

These settings are generally made using sos.ini (section [mail], entry smtp=...).

smtp=-queue stops e-mails being sent. Instead mails are written into the file specified in queue_dir. See also sos.ini (section [mail], entry queue_only=...).

The factory.ini (section [job], entry smtp=...) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry smtp=...) setting is overwritten by this parameter.

The sos.ini (section [mail], entry smtp=...) setting is overwritten by this parameter.

2.9.11 subject

Subject, *re*

Syntax: void mail.set_**subject** (String)

Syntax: String mail. **subject** ()

Example:

```
spooler_log.mail().set_subject( "Job succeeded" );
```

The factory.ini (section [job], entry log_mail_subject=...) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_subject=...) setting is overwritten by this parameter.

2.9.12 to

Recipient

Syntax: void mail.set_**to** (String)

Syntax: String mail. **to** ()

Example:

```
spooler_log.mail().set_to( "admin@company.com" );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The factory.ini (section [job], entry log_mail_to=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_to=…) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

2.9.13 xslt_stylesheet

The XSLT style sheet for e-mail processing. Before sending an e-mail the JobScheduler creates an XML document containing the e-mail headers, subject and body. The content of these elements can be adjusted or overwritten by an individual XSLT style sheet. This can be used e.g. to create translations of e-mail content. Having processed the XSLT style sheet the JobScheduler sends the resulting content of the XML elements as e-mail.

Syntax: Xslt_stylesheet_ mail. **xslt_stylesheet** ()

Returned value:
Xslt_stylesheet_

The XSLT style sheet as a string

2.9.14 xslt_stylesheet_path

The path and file name of the XSL style sheet for e-mail processing.

Syntax: void mail.set_**xslt_stylesheet_path** (java.io.path path)

Syntax: void mail.set_**xslt_stylesheet_path** (String path)

Example:

```
spooler_log.mail().set_xslt_stylesheet_path( "c:/stylesheets/mail.xslt" );
```

The path to the XSLT style sheet. XSLT style sheets are used by the JobScheduler for the preparation of e-mails. At the time of writing (April 2006) this subject is not documented.

<config mail_xslt_stylesheet="…">

Parameters:

path	The path of the file containing the XSLT style sheet

2.10 Monitor_impl - Using Super Classes for Start Scripts or Jobs

A job can be given a monitor using <monitor>.

A monitor can provide the following methods:

Monitor_impl.spooler_task_before()
Before starting a task - can prevent a task from being started.

Monitor_impl.spooler_task_after()
After a task has been completed.

Monitor_impl.spooler_process_before()
Before spooler_process() - this method can stop spooler_process() from being called.

Monitor_impl.spooler_process_after()
After spooler_process() - can be used to change its return value.

2.10.1 spooler

The JobScheduler Object

Syntax: Spooler **spooler**

Example:

```
spooler_log.debug( "The working directory of the JobScheduler is " +
spooler.directory() );
```

Is the same object as spooler in the Job_impl class.

Returned value:
Spooler

2.10.2 spooler_job

The Job Object

Syntax: Job **spooler_job**

Example:

```
spooler_log.info( "The name of this job is " + spooler_job.name() );
```

Is the same object as spooler_job in the Job_impl class.

Returned value:
Job

2.10.3 spooler_log

Writing Log Files

Syntax: `Log` **`spooler_log`**

Example:

```
spooler_log.info( "Something has happened" );
```

Is the same object as `spooler_log` in the `Job_impl` class.

Returned value:
`Log`

2.10.4 spooler_process_after

After `spooler_process()`

Syntax: `boolean` **`spooler_process_after`** `( boolean spooler_process_result )`

Example:

```
public boolean spooler_task_after( boolean spooler_process_result )  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    spooler_log.info( "spooler_process() didn't throw an exception and delivered " +
spooler_process_result );
    return spooler_process_result;   // Unchanged result
}
```

The JobScheduler calls this method after `spooler_process()` has been carried out.

Parameters:

`spooler_process_result`	The return value from the `spooler_process()` is set to `false`, should `spooler_process()` have ended with an exception.

Returned value:
`boolean`

Replaces the return value from the `spooler_process()` method or `false`, should `spooler_process()` have ended with an error.

2.10.5 spooler_process_before

Before `spooler_process()`

Syntax: `boolean` **`spooler_process_before`** `()`

Example:

```
public boolean spooler_process_before()  throws Exception
{
    spooler_log.info( "SPOOLER_PROCESS_BEFORE()" );
    return true;   // spooler_process() will be executed
}
```

Example:

```
public boolean spooler_process_before()  throws Exception
{
    boolean continue_with_spooler_process = true;

    if( !are_needed_ressources_available() )
    {
        spooler_task.order().setback();
        continue_with_spooler_process = false;
    }

    return continue_with_spooler_process;
}
```

This method is called by the JobScheduler before each call of spooler_process() .

Returned value:
boolean

false prevents further calls to spooler_process() . The JobScheduler continues as though false had been returned by spooler_process() false.

2.10.6 spooler_task

The Task Object

Syntax: Task **spooler_task**

Example:

```
spooler_log.info( "The task id is " + spooler_task.id() );
```

Is the same object as spooler_task in the Job_impl class.

Returned value:
Task

2.10.7 spooler_task_after

After Completing a Task

Syntax: void **spooler_task_after** ()

Example:

```
public void spooler_task_after()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_AFTER()" );
}
```

This method is called by the JobScheduler after a task has been completed.

2.10.8 spooler_task_before

Before Starting a Task

Syntax: `boolean` **spooler_task_before** ()

Example:

```
public boolean spooler_task_before()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    return true;   // Task will be started
  //return false;  // Task will not be started
}
```

This method is called by the JobScheduler before a task is loaded.

Returned value:
`boolean`

`false` does not allow a task to start and `Monitor_impl.spooler_task_after()` will not be called.

2.11 Order - Order

See JobScheduler Documentation, `Spooler.create_order()`, `Job_chain.add_order()`, `Task.order`.
File order

A file order is an order with for which the `scheduler_file_path` parameter has been set: `Order.params`. `Variable_set.value()`.

See JobScheduler Documentation.

Example: An Order with a simple Payload, in javascript

```
// Create order:
{
    var order = spooler.create_order();
    order.id         = 1234;
    order.title      = "This is my order";
    order.state_text = "This is my state text";
    order.payload    = "This is my payload";
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    spooler_log.info( "order.payload=" + order.payload );
    return true;
}
```

Example: Creating an Order with a Variable_set as a Payload, in javascript

```
// Create order:
{
    var variable_set = spooler.create_variable_set();
    variable_set.value( "param_one" ) = "11111";
    variable_set.value( "param_two" ) = "22222";

    var order = spooler.create_order();
    order.id      = 1234;
    order.payload = variable_set;
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    var variable_set = order.payload;
    spooler_log.info( "param_one=" + variable_set.value( "param_one" ) );
    spooler_log.info( "param_two=" + variable_set.value( "param_two" ) );
    return true;
}
```

2.11.1 at

The order start time

Syntax: void order.`set_at ( String| DATE )`

Example:

```
order.set_at( "now+60" );                                        // set_at( String )
order.set_at( new Date( new Date().getTime() + 60 * 1000 ) );  // set_at(
java.util.Date )
spooler.job_chain( "my_job_chain" ).add_order( order );
```

Used to set the start time before an order is added to an order queue. The following can be specified as a string:

- `"now"`
- `"yyyy-mm-dd HH:MM[:SS]"`
- `"now + HH:MM[:SS]"`
- `"now + seconds"`

This setting changes start times set by `Order.run_time` or `Order.setback()`.

See `<add_order at="">`.

2.11.2 end_state

The state that should be reached when an order has been successfully completed

Syntax: void order.`set_`**`end_state`**`( String )`

Syntax: `String` order. **`end_state`** `()`

When an order has its own `end_state` other than `""` then it is considered to be completed after the job allocated to this end state has been completed and before the order otherwise leaves this state (see `<job_chain_node>` for example to continue to another job which usually comprises a part of the job chain).

The state specified has to reference a valid state of a job node in the job chain.

2.11.3 id

Order Identification

Syntax: void order.`set_`**`id`**`( String )`

Syntax: `String` order. **`id`** `()`

Every order has an identifier. This identifier must be unique within a job chain or job order queue. It should also correspond to the data being processed. Normally database record keys are used.

When an `id` is not set, then the JobScheduler automatically allocates one using `Job_chain.add_order()`.

2.11.4 job_chain

The job chain containing an order

Syntax: `Job_chain` order. **`job_chain`** `()`

Returned value:
`Job_chain`

2.11.5 job_chain_node

The job chain nodes which correspond with the order state

Syntax: _Job_chain_node_ order. **job_chain_node** ()

Returned value:
Job_chain_node_

2.11.6 log

Order log

Syntax: _Log_ order. **log** ()

Example:

```
spooler_task.order.log.info( "Only for order log, not for task log" );
spooler_log.info( "For both order log and task log" );
```

Returned value:
Log_

2.11.7 params

The order parameters

Syntax: void order.set_**params** (Variable_set_)

Syntax: _Variable_set_ order. **params** ()

`params` is held in Order.payload_, the latter cannot, therefore, be used together with `params`.

See <add_order>_.

Returned value:
Variable_set_

2.11.8 payload

Load - an order parameter.

Syntax: void order.set_**payload** (Object payload)

Syntax: Object order. **payload** ()

Instead of this property, the use of Order.params is recommended, which corresponds to (Variable_set) order.payload.

In addition to Order.id which identifies an order, this field can be used for other information.

See Order.params and Order.xml_payload.

Parameters:

payload May be a string or a Variable_set.

Returned value:

Object

May be a string or a Variable_set.

2.11.9 payload_is_type

Checks the payload COM-Type

Syntax: boolean order. **payload_is_type** (String type_name)

Parameters:

type_name "Spooler.Variable_set","Hostware.Dyn_obj" or "Hostware.Record".

2.11.10 priority

Orders with a higher priority are processed first

Syntax: void order.set_**priority** (int)

Syntax: int order. **priority** ()

2.11.11 remove_from_job_chain

Syntax: void order. **remove_from_job_chain** ()

Note that when an order has just been started by a task, then the Order.job_chain property will still return the job chain from which the order has just been removed, using this call, even when "remove_from_job_chain" has been carried out. It is only when the execution has been ended that this method returns null. (other than when the order has just been added to a job chain). This ensures that the job_chain property remains stable whilst a task is being executed.

2.11.12 run_time

<run_time> is used to periodically repeat an order

Syntax: Run_time order. **run_time** ()

Example: in javascript

```
order.run_time.xml = "<run_time><at at='2006-05-23 11:43:00'/></run_time>";
```

See <run_time>.

The <modify_order at="now"> command causes an order which is waiting because of run_time to start immediately.

Returned value:
Run_time

2.11.13 setback

Delays an order back for a period of time

Syntax: void order. **setback** ()

An order will be delayed and repeated after the period of time specified in either <delay_order_after_setback> or Job.delay_order_after_setback. When the job is repeated, only the spooler_process() job function is repeated. If the order.setback() function is called from spooler_process(), then the retrun value from spooler_process() will have no effect. .

An order counts the number of times this method is called in sequence. This count is then used by <delay_order_after_setback>. It is set to 0, when spooler_process() is completed without <delay_order_after_setback> being called. All counters are set to 0 when the JobScheduler is started.

The <modify_order at="now"> command causes a blocked order to start immediately.

2.11.14 setback_count

How many times the order is setting back?

Syntax: int order. **setback_count** ()

see also <delay_order_after_setback>.

2.11.15 state

The order state

Syntax: void order.set_**state** (String)

Syntax: `String order.` **`state`** `()`

When an order is in a job chain, then its state must correspond with one of the states of the job chain.

Whilst an order is being processed by a job the following state, as defined in the job chain (`<job_chain_node next_state="">`) has no effect. Similarly, the return values from `spooler_process()` and `Monitor_impl.spooler_process_after()` are meaningless. This means that with `Order.state` the following state for a job can be set as required.

An order is added to the job order queue which is corresponding to its state. See `<job_chain_node>`. The execution by this job will be delayed until the job currently carrying out the order has been completed.

2.11.16 state_text

Free text for the order state

Syntax: void order.`set_`**`state_text`** `( String )`

Syntax: `String order.` **`state_text`** `()`

This text is shown on the HTML interface.

For non-API jobs the JobScheduler fills this field with the first line from stdout, up to a maximum of 100 characters.

2.11.17 string_next_start_time

The next start time of an order when `<run_time>` is being used

Syntax: `String order.` **`string_next_start_time`** `()`

Returned value:
`String`

`"yyyy-mm-dd HH:MM:SS.MMM"` or `"now"` or `"never"`.

2.11.18 suspended

Suspended order

Syntax: void order.`set_`**`suspended`** `( boolean )`

Syntax: `boolean order.` **`suspended`** `()`

A suspended order will not be executed.

When an order is being carried out by a task when it is suspended, then the `spooler_process()` step will be completed and the order allocated the successor state before being suspended.

This means that an order can be set to an end state, which stops it from being removed. The JobScheduler can remove such an order only when it is not suspended - i.e. `order.suspended=false`).

A suspended order with the end state can be allocated a different state corresponding to a job node in the job chain. This is effected by using `Order.state`. In this case the order remains suspended.

2.11.19 title

Optionally a title can be allocated to an order that will show up in the HTML interface and in the logs.

Syntax: void order.`set_title ( String )`

Syntax: `String` order. **`title ()`**

2.11.20 web_service

The web service to which an order has been allocated

Syntax: `Web_service` order. **`web_service ()`**

When an order has not been allocated to a web service, then this call returns the `SCHEDULER-240` error.

See also `Order.web_service_or_null`.

Returned value:
`Web_service`

2.11.21 web_service_operation

The web service operation to which an order has been allocated

Syntax: `Web_service_operation` order. **`web_service_operation ()`**

Example:

```
public boolean spooler_process()  throws Exception
{
    Order                 order                 = spooler_task.order();
    Web_service_operation web_service_operation = order.web_service_operation();
    Web_service_request   request               = web_service_operation.request();

    // Decode request data
    String request_string = new String( request.binary_content(),
request.charset_name() );


    process request_string ...;


    String                response_string = "This is my response";
    String                charset_name    = "UTF-8";
    ByteArrayOutputStream byos            = new ByteArrayOutputStream();

    // Encode response data
    Writer writer = new OutputStreamWriter( byos, charset_name );
    writer.write( response_string );
    writer.close();

    // Respond
    Web_service_response response = web_service_operation.response();

    response.set_content_type( "text/plain" );
    response.set_charset_name( charset_name );
    response.set_binary_content( byos.toByteArray() );
    response.send();

    // Web service operation has finished

    return true;
}
```

See <web_service>, Web_service_operation and Order.web_service_operation_or_null,

Returned value:
Web_service_operation

2.11.22 web_service_operation_or_null

The web service operation to which an order has been allocated, or `null`

Syntax: Web_service_operation order. **web_service_operation_or_null** ()

See Order.web_service_operation, Web_service_operation and <web_service>.

Returned value:
Web_service_operation

2.11.23 web_service_or_null

The web service to which an order has been allocated, or `null`.

Syntax: _Web_service_ order. **web_service_or_null** ()

See also Order. web_service_.

Returned value:
Web_service_

2.11.24 xml

Order in XML: <order>...</order>

Syntax: `String` order. **xml** ()

Returned value:
`String`

See <order>

2.11.25 xml_payload

XML payload - an order parameter.

Syntax: void order.`set_`**xml_payload** `( String xml )`

Syntax: `String` order. **xml_payload** ()

This property can include an XML document (in addition to the Order. params property).

<xml_payload> contains the XML document root element (instead of it being in `#PCDATA` coded form).

2.12 Order_queue - The order queue for an order controlled job

An order controlled job (<job order="yes"> has an order queue, which is filled by the orders to be processed by a job. The orders are sorted according to their priority and the time at which they enter the queue.

Processing means that the JobScheduler calls the spooler_process() method for a task. This method can access the order using the Task. order property. Should the `spooler_process()` end without an error (i.e. without any exceptions), then the JobScheduler removes the order from the order queue. If the order is in a job chain then it is moved to the next position in the chain.

2.12.1 length

The number of orders in the order queue

Syntax: `int q. length ()`

2.13 Process_class

See also <process_class name="">.

Example: in javascript

```
var process_classs = spooler.process_classs;
var process_class = process_classs.create_process_class();
process_class.name = "my_process_class";
process_classs.add_process_class( process_class );
```

2.13.1 max_processes

The maximum number of processes that are executed in parallel

Syntax: void process_class.`set_max_processes ( int )`

Syntax: `int` process_class. `max_processes ()`

Should more tasks have to be started than allowed by this setting, then these tasks starts would be delayed until processes become freed. The default setting is 10.

See also <process_class max_processes="">.

2.13.2 name

The process class name

Syntax: void process_class.`set_name ( String )`

Syntax: `String` process_class. `name ()`

The name can only be set once and may not be changed.

See also <process_class name="">.

2.13.3 remote_scheduler

The address of the remote JobScheduler, which is to execute a process

Syntax: void process_class.set_**remote_scheduler** (String)

Syntax: String process_class. **remote_scheduler** ()

Example: in javascript

```
spooler.process_classes.process_class( "my_process_class" ).remote_scheduler =
"host:4444";
```

See also <process_class remote_scheduler="">.

Parameters:

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

Returned value:
String

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

2.13.4 remove

Removal of the process class

Syntax: void process_class. **remove** ()

Example: in javascript

```
spooler.process_classs.process_class( "my_process_class" ).remove();
```

The JobScheduler delays deletion of the process class as long as tasks are still running. No new tasks will be started before the class is deleted.

See also <process_class.remove>.

2.14 Process_classes

2.14.1 add_process_class

Adds a process class to the JobScheduler

Syntax: void process_classs. **add_process_class** (Process_class pc)

2.14.2 create_process_class

Creates a new process class

Syntax: Process_class process_classs. **create_process_class** ()

Returnds a new Process_class. This class can be made added to the JobScheduler using Process_classes.add_process_class().

Returned value:
Process_class

2.14.3 process_class

Returns a process class

Syntax: Process_class process_classs. **process_class** (String process_class_name)

An exception will occur if the process class is not known.

Returned value:
Process_class

2.14.4 process_class_or_null

Returns a process class

Syntax: Process_class process_classs. **process_class_or_null** (String process_class_name)

Returned value:
Process_class

null, when the process class is not known.

2.15 Run_time - Managing Time Slots and Starting Times

See <run_time>, Order.Schedule.

Example: in javascript

```
var order = spooler_task.order;

// Repeat order daily at 15:00
order.run_time.xml = "<run_time><period single_start='15:00'/></run_time>";
```

2.15.1 schedule

`<schedule>`

Syntax: Schedule run_time. **schedule** ()

Returned value:
Schedule

2.15.2 xml

`<run_time>`

Syntax: void run_time.set_**xml** (String)

Discards the current setting and resets `Run_time`.

Parameters:

XML document as a string

2.16 Schedule - Runtime

See <schedule>, <run_time>, Spooler.schedule, Run_time.

Example: in javascript

```
spooler.schedule( "my_schedule" ).xml = "<schedule><period single_start='15:00'/><
/schedule>";
```

2.16.1 xml

`<schedule>`

Syntax: void schedule.set_**xml** (String)

Syntax: String schedule. **xml** ()

Deletes the previous setting and resets `Schedule`.

Parameters:

XML document as a string

Returned value:
String

XML document as a string

2.17 Spooler

There is only one class for this object: spooler_.

2.17.1 abort_immediately

Aborts the JobScheduler immediately

Syntax: void spooler. **abort_immediately** ()

Stops the JobScheduler immediately. Jobs do not have the possibility of reacting.

The JobScheduler kills all tasks and the processes that were started using the Task. create_subprocess() method. The JobScheduler also kills processes for which a process ID has been stored using the Task. add_pid() method.

See <modify_spooler cmd="abort_immediately"> and JobScheduler Documentation.

2.17.2 abort_immediately_and_restart

Aborts the JobScheduler immediately and then restarts it.

Syntax: void spooler. **abort_immediately_and_restart** ()

Similar to the Spooler. abort_immediately() method, only that the JobScheduler restarts itself after aborting. It reuses the command line parameters to do this.

See <modify_spooler cmd="abort_immediately_and_restart"> and JobScheduler Documentation.

2.17.3 add_job_chain

Syntax: void spooler. **add_job_chain** (Job_chain chain)

Job_chain. orders_recoverable_=true causes the JobScheduler to load the orders for a job chain from the database.

See Spooler. create_job_chain() _. and <job_chain>_.

2.17.4 configuration_directory

Path of the Configuration Directory with hot folders

Syntax: `String` spooler. **`configuration_directory`** `()`

```
<config configuration_directory="…">
```

2.17.5 create_job_chain

Syntax: `Job_chain` spooler. **`create_job_chain`** `()`

Returns a new `Job_chain` object. This job chain can be added to the JobScheduler using `Spooler.add_job_chain()` after it has been filled with jobs.

See `<job_chain>`.

Returned value:
`Job_chain`

2.17.6 create_order

Syntax: `Order` spooler. **`create_order`** `()`

Creates a new order. This order can be assigned to a job chain using the `Job_chain.add_order()` method.

Returned value:
`Order`

2.17.7 create_variable_set

Syntax: `Variable_set` spooler. **`create_variable_set`** `()`

Returned value:
`Variable_set`

2.17.8 create_xslt_stylesheet

Syntax: `Xslt_stylesheet` spooler. **`create_xslt_stylesheet`** `()`

Syntax: `Xslt_stylesheet` spooler. **`create_xslt_stylesheet`** `( java.io.path path )`

Syntax: `Xslt_stylesheet` spooler. **`create_xslt_stylesheet`** `( String path )`

Parameters:

`xml` Creates an XSLT style sheet as an XML string.

Returned value:
`Xslt_stylesheet Xslt_stylesheet Xslt_stylesheet`

2.17.9 db_history_table_name

The name of the database table used for the job history

Syntax: `String` spooler. **`db_history_table_name`** `()`

See also `Spooler. db_history_table_name()`

The `factory. ini` (section `[ spooler]`, entry `db_history_table=…`) setting is overwritten by this parameter.

2.17.10 db_name

The database path

Syntax: `String` spooler. **`db_name`** `()`

The database connection string for the history. Should no value be specified here, then the files will be saved in .csv format. See `factory. ini` (section `[ spooler]`, entry `history_file=…`).

A simple file name ending in `. mdb` (e.g. `scheduler. mdb`) can also be specified here when the JobScheduler is running on Windows. The JobScheduler then uses a Microsoft MS Access database of this name, which is located in the protocol directory (see the option `-log-dir`). Should such a database not exist, then the JobScheduler will create this database.

The JobScheduler automatically creates the tables necessary for this database.

The `factory. ini` (section `[ spooler]`, entry `db=…`) setting is overwritten by this parameter.

2.17.11 db_order_history_table_name

The name of the order history database table

Syntax: `String` spooler. **`db_order_history_table_name`** `()`

See also Spooler.db_order_history_table_name()

The factory.ini (section [spooler], entry db_order_history_table=…) setting is overwritten by this parameter.

2.17.12 db_orders_table_name

The name of the database table used for orders

Syntax: String spooler. **db_orders_table_name** ()

See also Spooler.db_orders_table_name()

The factory.ini (section [spooler], entry db_orders_table=…) setting is overwritten by this parameter.

2.17.13 db_tasks_table_name

The name of the task database table

Syntax: String spooler. **db_tasks_table_name** ()

See also Spooler.db_tasks_table_name()

The factory.ini (section [spooler], entry db_tasks_table=…) setting is overwritten by this parameter.

2.17.14 db_variables_table_name

The name of the database table used by the JobScheduler for internal variables

Syntax: String spooler. **db_variables_table_name** ()

The JobScheduler records internal counters, for example, the ID of the next free task, in this database table.

See also Spooler.db_variables_table_name()

The factory.ini (section [spooler], entry db_variables_table=…) setting is overwritten by this parameter.

2.17.15 directory

The working directory of the JobScheduler on starting

Syntax: `String` spooler. **`directory`** `()`

Changes the Working Directory.

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

The `-cd` option has precedence over this parameter.

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

Returned value:
`String`

The directory ends on Unix with "/" and on Windows with "\".

2.17.16 execute_xml

Carries out XML commands

Syntax: `String` spooler. **`execute_xml`** `( String xml )`

Example: in javascript

```
spooler_log.info( spooler.execute_xml( "<show_state/>" ) );
```

Errors are returned as XML `<ERROR>` replies.

Parameters:

`xml` See JobScheduler Documentation.

Returned value:
`String`

Returns the answer to a command in XML format.

2.17.17 hostname

The name of the computer on which the JobScheduler is running.

Syntax: `String` spooler. **`hostname`** `()`

2.17.18 id

The value of the command line -id= setting

Syntax: String spooler. **id** ()

The JobScheduler only selects elements in the XML configuration whose spooler_id attributes are either empty or set to the value given here.

When the JobScheduler ID is not specified here, then the JobScheduler ignores the spooler_id= XML attribute and selects all the elements in the XML configuration.

See, for example, <config>.

The -id option has precedence over this parameter.

The factory.ini (section [spooler], entry id=…) setting is overwritten by this parameter.

2.17.19 include_path

Returns the command line setting -include-path=.

Syntax: String spooler. **include_path** ()

The directory of the files which are to be included by the <include> element.

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

Environment variables (e.g. $HOME) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The -include-path option has precedence over this parameter.

The factory.ini (section [spooler], entry include_path=…) setting is overwritten by this parameter.

<config include_path="…">

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

2.17.20 ini_path

The value of the -ini= option (the name of the factory.ini file)

Syntax: String spooler. **ini_path** ()

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

See -ini , JobScheduler Documentation

2.17.21 is_service

Syntax: boolean spooler. **is_service** ()

Returned value:
boolean

is true, when the JobScheduler is running as a service (on Windows) or as a daemon (on Unix).

2.17.22 job

Returns a job

Syntax: Job spooler. **job** (String job_name)

An exception is returned should the job name not be known.

Returned value:
Job

2.17.23 job_chain

Returns a job chain

Syntax: Job_chain spooler. **job_chain** (String name)

Should the name of the job chain not be known, then the JobScheduler returns an exception.

Returned value:
Job_chain

2.17.24 job_chain_exists

Syntax: boolean spooler. **job_chain_exists** (String name)

2.17.25 let_run_terminate_and_restart

Syntax: void spooler. **let_run_terminate_and_restart** ()

The JobScheduler ends all tasks (by calling the Job_impl method) as soon as all orders have been completed and then stops itself. It will then be restarted under the same command line parameters.

See <modify_spooler cmd="let_run_terminate_and_restart"> and JobScheduler Documentation.

2.17.26 locks

Returns the locks

Syntax: Locks spooler. **locks** ()

Returned value:
Locks

2.17.27 log

The main log

Syntax: Log spooler. **log** ()

spooler_log() is usually used for this property.

Returned value:
Log

2.17.28 log_dir

Protocol directory

Syntax: String spooler. **log_dir** ()

The directory in which the JobScheduler writes log files.

log_dir= *stderr allows the JobScheduler to write log files to the standard output (stderr, normally the screen) .

A task executed on a remote JobScheduler (<process class remote_scheduler="">) returns the value for the remote Scheduler.

The -log-dir option has precedence over this parameter.

The factory.ini (section [spooler], entry log_dir=…) setting is overwritten by this parameter.

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

2.17.29 param

The command line option -param=

Syntax: String spooler. **param** ()

Free text. This parameter can be read using spooler. param.

The -param option has precedence over this parameter.

The factory. ini (section [spooler], entry param=…) setting is overwritten by this parameter.

2.17.30 process_classes

Returns the process classes

Syntax: Process_classes spooler. **process_classes** ()

Returned value:
Process_classes

2.17.31 schedule

Returns the Schedule with the name specified or null

Syntax: Schedule spooler. **schedule** (String path)

Returned value:
Schedule

2.17.32 supervisor_client

Returns the Supervisor_client or null

Syntax: Supervisor_client spooler. **supervisor_client** ()

Returned value:
Supervisor_client

2.17.33 tcp_port

Port for HTTP and TCP commands for the JobScheduler

Syntax: `int` spooler. **`tcp_port`** `()`

The JobScheduler can accept commands via a TCP port whilst it is running. The number of this port is set here - depending on the operating system - with a number between 2048 and 65535. The default value is 4444.

The JobScheduler operates a HTTP/HTML server on the same port, enabling it to be reached using a web browser - e.g. via http://localhost:4444.

The JobScheduler does not respond to the `tcp_port=0` default setting either with TCP or HTTP protocols. This setting can therefore be used to block a JobScheduler from being accessed - for example via TCP.

The `-tcp-port` option has precedence over this parameter.

`<config tcp_port="…">`

Returned value:
`int`

0, when no port is open.

2.17.34 terminate

The proper ending of the JobScheduler and all related tasks

Syntax: void spooler. **`terminate`** `( int timeout (optional) , boolean restart (optional) , boolean all_schedulers (optional) , boolean continue_exclusive_operation (optional) )`

Ends all tasks (by calling the `spooler_close()`) method and terminates the JobScheduler.

Should a time limit be specified, then the JobScheduler ends all processes still running after this limit has expired. (Typical processes are tasks which have remained too long in a method call such as `spooler_process()`.)

See `<modify_spooler cmd="terminate">` and JobScheduler Documentation.

Parameters:

`timeout`	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.
`restart`	`restart=true` allows the JobScheduler to restart after ending.
`all_schedulers`	`all_schedulers=true` ends all the JobSchedulers belonging to a cluster (see `-exclusive`). This may take a minute.
`continue_exclusive_operation`	`continue_exclusive_operation=true` causes another JobScheduler in the Cluster to take become active (see `-exclusive`).

2.17.35 terminate_and_restart

Correctly terminates the JobScheduler and all tasks before restarting

Syntax: void spooler. **terminate_and_restart** (int timeout (optional))

Similar to the Spooler.terminate() method, but the JobScheduler restarts itself.

See <modify_spooler cmd="terminate_and_restart"> and JobScheduler Documentation.

Parameters:

timeout	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.

2.17.36 udp_port

Port for UDP commands for the JobScheduler

Syntax: int spooler. **udp_port** ()

The JobScheduler can also accept UDP commands addressed to the port specified in this setting. Note that a UDP command must fit in a message and that the JobScheduler does not answer UDP commands.

The default value of udp_port=0 does not allow the JobScheduler to open a UDP port.

The -udp-port option has precedence over this parameter.

<config udp_port="…">

Returned value:
int

0, when no port is open.

2.17.37 var

Allows access to variables defined in the JobScheduler start script

Syntax: void spooler.set_**var** (String name, String)

Syntax: String spooler. **var** (String name)

The variables are used by all JobScheduler job implementations.

2.17.38 variables

The JobScheduler variables as a `Variable_set`

Syntax: `Variable_set` spooler. **`variables`** `()`

The variables can be set in the configuration file using `<config>`.

Returned value:
`Variable_set`

2.18 Spooler_program - Debugging Jobs in Java

Starts the JobScheduler using Java, so that jobs written in Java can be debugged (e.g. using Eclipse). See Javadoc for information about the methods.

The JobScheduler is started as a Windows application and not as a console program. Output to `stderr` is lost - standard output is shown in Eclipse. `-log-dir` shows no output.

See JobScheduler Documentation.

Example:

```
C:\>java -Djava.library.path=… -classpath …\sos.spooler.jar sos.spooler.Spooler_program
 configuration.scheduler -log-dir=c:\tmp\scheduler
```

Should the location of the `scheduler.dll` not be specified in `%PATH%` then it may be set using `-Djava.library.path=…`.

2.19 Subprocess

A subprocess is a process which can be started using either `Task.create_subprocess()` or `Subprocess.start()`.

Example: system() - the Simple Execution of a Command, in javascript

```
exit_code = my_system( "backup /" );

function system( cmd, timeout )
{
    var subprocess = spooler_task.create_subprocess();

    try
    {
        if( timeout )  subprocess.timeout = timeout;
        subprocess.start( cmd );
        subprocess.wait_for_termination();
        return subprocess.exit_code;
    }
    finally
    {
        subprocess.close();
    }
}
```

Example: in javascript

```
var subprocess = spooler_task.create_subprocess();

subprocess.environment( "test1" ) = "one";
subprocess.environment( "test2" ) = "two";
subprocess.ignore_error = true;

subprocess.start( "sleep 20" );

spooler_log.info( "pid=" + subprocess.pid );
subprocess.timeout = 10;

spooler_log.info( "wait_for_termination ..." );
var ok = subprocess.wait_for_termination( 10 );
spooler_log.info( "wait_for_termination ok=" + ok );

if( subprocess.terminated )
{
    spooler_log.info( "exit code=" + subprocess.exit_code );
    spooler_log.info( "termination signal=" + subprocess.termination_signal );
}
```

2.19.1 close

Frees system resources

Syntax: void subprocess. **`close`** `()`

This method should only be called in language with a garbage collector (Java, JavaScript). In all other cases the task ends immediately.

Should this method have been called in a language with a garbage collector, then the `Subprocess` is no longer usable.

2.19.2 env

Environment Variables as Variable_sets

Syntax: Variable_set subprocess. **env** ()

Example: in javascript

```
var subprocess = spooler_task.create_subprocess();
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
subprocess.wait_for_termination();
```

Returns a Variable_set for the environment variables.

Initially the environment is filled by the environment variables from the calling process. Environment variables can be removed in that they are set to "". Calling Subprocess.start() hands over environment variables to the subprocess.

Note that the names of environment variables are case sensitive on UNIX systems.

Changes made to environment variables after the start of a subprocess have no effect. This is also true for environment variables changed by the process.

This object cannot be handed over to other objects - it is a part of the task process, whereas the majority of other objects are part of the JobScheduler process.

Returned value:
Variable_set

2.19.3 environment

Environment variables

Syntax: void subprocess.set_**environment** (String name, String value)

Example:

```
// The following two statements have the same effect
subprocess.set_environment( "my_variable", "my_value" )
subprocess.env().set_value( "my_variable" ) = "my_value"
```

Variables set here are handed over to a new subprocess together with any other environment variables belonging to the process.

Note that the names of environment variables are case sensitive on UNIX systems.

See also Subprocess.env.

2.19.4 exit_code

Syntax: int subprocess. **exit_code** ()

Is only called after Subprocess.terminated == true.

2.19.5 ignore_error

Prevents that a job is stopped, should exit_code != 0.

Syntax: void subprocess.set_**ignore_error** (boolean)

Syntax: boolean subprocess. **ignore_error** ()

Prevents a job from being stopped, when at the end of a task the subprocess ends with Subprocess.exit_code != 0.

Should a task not wait for the end of a subprocess with the Subprocess.wait_for_termination method, then the JobScheduler waits at the end of the task for the end of any subprocesses. In this case the job is stopped with an error when a subprocess ends with Subprocess.exit_code != 0.

This may be avoided using ignore_error.

2.19.6 ignore_signal

Prevents a job from being stopped when the task is stopped with a UNIX signal.

Syntax: void subprocess.set_**ignore_signal** (int)

Syntax: int subprocess. **ignore_signal** ()

This property does not work on Windows systems, as this system does not support signals.

2.19.7 kill

Stops a subprocess

Syntax: void subprocess. **kill** (int signal (optional))

Parameters:

signal Only on UNIX systems: The kill() signal. 0 is interpreted here as 9 (SIGKILL, immediate ending).

2.19.8 own_process_group

Subprocesses as a Process Group

Syntax: void subprocess.set_**own_process_group** (boolean)

Syntax: boolean subprocess. **own_process_group** ()

Only available for UNIX systems.

The default setting can be made using factory.ini (section [spooler], entry subprocess.own_process_group=…).

own_process_group allows a subprocess to run in its own process group, by executing the setpgid(0,0) system call. When the JobScheduler then stops the subprocess, then it stops the complete process group.

2.19.9 pid

Process identification

Syntax: int subprocess.**pid**()

2.19.10 priority

Process Priority

Syntax: void subprocess.set_**priority**(int)

Syntax: int subprocess.**priority**()

Example: in javascript

```
spooler_task.priority = +5;     // UNIX: reduce the priority a little
```

UNIX: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 "idle", 6 "below_normal", 8 "normal", 10 "above_normal" and 13 "high" (other values are rounded down). See also Task.priority_class.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Subprocess.priority_class. See also Task.priority.

2.19.11 priority_class

Priority Class

Syntax: void subprocess.set_**priority_class**(String)

Syntax: String subprocess.**priority_class**()

Example: in javascript

```
subprocess.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and UNIX Systems:

Priority Class	Windows	UNIX
"idle"	4	16
"below_normal"	6	6
"normal"	8	0
"above_normal"	10	-6
"high"	13	-16

Note that when it is not possible to set a priority for a task - for example, because of inappropriate permissions - then this must not cause an error. On the other hand, an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Subprocess.priority, Task.priority_class and Microsoft® Windows® Scheduling Priorities.

2.19.12 start

Starts the process

Syntax: void subprocess.**start** (String command_line)

Syntax: void subprocess.**start** (String filename_and_arguments)

Windows immediately detects whether the program cannot be executed. In this case the method returns an error.

On UNIX systems the Subprocess.exit_code property is set to 99. Before this is done, the end of the process must be waited on with Subprocess.wait_for_termination().

Shell operators such as | , && and > are not interpreted. The /bin/sh or c:\windows\system32\cmd.exe programs must be used to do this. (Note that the actual paths will depend on the installation.)

This process is started on UNIX systems using execvp() and with CreateProcess() on Windows systems.

2.19.13 terminated

Syntax: boolean subprocess.**terminated** ()

Verifies that a process has ended. Should the process in question have ended, then the Subprocess.exit_code and Subprocess.termination_signal classes may be called.

2.19.14 termination_signal

Signal with which a process (only on UNIX systems) ends

Syntax: `int` subprocess. **`termination_signal`** `()`

Is only called, after `Subprocess.terminated_ == true`.

2.19.15 timeout

Time limit for a subprocess

Syntax: void subprocess.`set_`**`timeout`** `( double seconds )`

After the time allowed, the JobScheduler stops the subprocess (UNIX: with `SIGKILL`).

This time limit does not apply to processes running on remote computers with `<process_class remote_scheduler="">_`.

2.19.16 wait_for_termination

Syntax: void subprocess. **`wait_for_termination`** `()`

Syntax: `boolean` subprocess. **`wait_for_termination`** `( double seconds )`

Parameters:

`seconds`	Waiting time. Should this parameter not be specified, then the call will take place after the subprocess has ended.

Returned value:
`boolean`

`true`, after a subprocess has ended.
`false`, should the subprocess continue beyond the waiting time.

2.20 Supervisor_client

This object is returned by `Spooler.supervisor_client_`.

Example: in javascript

```
var supervisor_hostname = spooler.supervisor_client.hostname;
```

2.20.1 hostname

The name or IPnumber of the host computer on which the suupervising JobScheduler is running

Syntax: `String` supervisor_client. **`hostname`** `()`

See also `<config supervisor="">`.

2.20.2 tcp_port

the TCP port of the supervisor

Syntax: `int` supervisor_client. **`tcp_port`** `()`

See also `<config supervisor="">`.

2.21 Task

A task is an instance of a job which is currently running.

A task can either be waiting in a job queue or being carried out.

A task is implemented using `Job_impl`.

2.21.1 add_pid

Makes an independent, temporary process known to the JobScheduler

Syntax: void spooler_task. **`add_pid`** `( int pid )`

Syntax: void spooler_task. **`add_pid`** `( int pid, double timeout_seconds )`

This call is used to restrict the time allowed for processes that have been launched by a task. The JobScheduler ends all independent processes still running at the end of a task.

A log entry is made each time the JobScheduler stops a process. This does not affect the state of a task.

The `<kill_task>` method stops all processes for which the `add_pid()` method has been called.

A process group ID can be handed over on Unix systems as a negative pid. `kill` then stops the complete process group.

This time limit does not apply for processes being run on remote computers with `<process_class remote_scheduler="">`.

2.21.2 call_me_again_when_locks_available

Repeats spooler_open() or spooler_process() as soon as locks become available

Syntax: void spooler_task. **call_me_again_when_locks_available** ()

Causes the JobScheduler to repeat a call of spooler_open() or spooler_process(), after an unsuccessful Task.try_hold_lock() or Task.try_hold_lock_non_exclusive() as soon as the locks required are available. The JobScheduler then repeats the call once it holds the locks, so that the first call (i.e. spooler_open()) will be successful.

After this call, true/false values returned by spooler_open() or spooler_process() has no effect. The JobScheduler leaves the state of the Task.order unchanged.

2.21.3 changed_directories

The directory in which the change which started a task occurred

Syntax: String spooler_task. **changed_directories** ()

See Job.start_when_directory_changed(), Task.trigger_files.

Returned value:
String

Directory names are to be separated using a semicolon.

"", should no change have occurred in a directory.

2.21.4 create_subprocess

Starts a monitored subprocess

Syntax: Subprocess spooler_task. **create_subprocess** ()

Syntax: Subprocess spooler_task. **create_subprocess** (String command_line)

Syntax: Subprocess spooler_task. **create_subprocess** (String filename_and_arguments)

Returned value:
Subprocess Subprocess Subprocess

2.21.5 delay_spooler_process

Delays the next call of spooler_process()

Syntax: void spooler_task.set_**delay_spooler_process** (double)

Syntax: void spooler_task.set_**delay_spooler_process** (String hhmm_ss)

Only functions in spooler_process().

2.21.6 end

Ends a task

Syntax: void spooler_task. **end** ()

The JobScheduler no longer calls the spooler_process() method. Instead the spooler_close() method is called.

This method call can be used at the end of a task to trigger sending a task log. See Log.

2.21.7 error

Sets an error and stops the current job

Syntax: void spooler_task.set_**error** (String)

Syntax: Error spooler_task. **error** ()

This method call returns the last error which has occurred with the current task. Should no error have occurred, an Error object is returned, with the is_error property set to false.

An error message can also be written in the task log file using Log. error()

Returned value:
String Error

2.21.8 exit_code

Exit-Code

Syntax: void spooler_task.set_**exit_code** (int)

Syntax: int spooler_task. **exit_code** ()

Example: in javascript

```
spooler_log.error( "This call of spooler_log.error() sets the exit code to 1" );
spooler_task.exit_code = 0;   // Reset the exit code
```

The initial exit-code value is 0 - this is changed to 1 should an error occur. Note that an error is defined here as occurring when the JobScheduler writes a line in the task log containing "[ERROR] ":

- calling the Log. error() method;
- setting the Task. error property;
- the script returns an exception.

The job can then set the Task.exit_code property - e.g. in the spooler_on_error() method.

The exit code resulting from an operating system process executing a task is not relevant here and, in contrast to jobs with <process> or <script language="shell">, is not automatically handed over to this property.

The exit code determines the commands to be subsequently carried out. See <job> <commands on_exit_code="" > for more information.

The exit codes have no influence for API jobs on whether or not a job is stopped (a task error message causes jobs to be stopped).

2.21.9 history_field

A field in the task history

Syntax: void spooler_task.set_**history_field** (String name, String value)

Example: in javascript

```
spooler_task.history_field( "extra" ) = 4711;
```

The database table (see factory.ini (section [spooler], entry db_history_table=…)) must have a column with this name and have been declared in the factory.ini (section [job], entry history_columns=…) file.

2.21.10 id

The task identifier

Syntax: int spooler_task. **id** ()

The unique numerical identifier of every task run by a JobScheduler.

2.21.11 job

The job which a task belongs to

Syntax: Job spooler_task. **job** ()

Returned value:
Job

2.21.12 order

The current order

Syntax: Order spooler_task. **order** ()

Example:

```
Order order = spooler_task.order();

spooler_log.info( "order.id=" + order.id() + ", order.title=" + order.title() );
```

Returned value:
Order

null, should no order exist.

2.21.13 params

The task parameters

Syntax: Variable_set spooler_task. **params** ()

Example:

```
String value = spooler_task.params().var( "parameter3" );
```

Example:

```
Variable_set parameters = spooler_task.params();
if( parameters.count() > 0 )  spooler_log.info( "Parameters given" );

String value1 = parameters.var( "parameter1" );   // "", should the variable not exist

String value1 = parameters.var( "parameter1" );   // "", wenn die Variable nicht
vorhanden ist
String value2 = parameters.var( "parameter2" );
```

A task can have parameters. These parameters can be set using:

- <params> in the <job> element in the configuration file;
- Job.start() and
- <start_job>.

Returned value:
Variable_set

!= null

2.21.14 priority

Priority of the Current Task

Syntax: void spooler_task.set_**priority** (int)

Syntax: `int` spooler_task. **`priority`** `()`

Example: in javascript

```
spooler_task.priority = +5;     // Unix: reduce the priority a little
```

Unix: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 `"idle"`, 6 `"below_normal"`, 8 `"normal"`, 10 `"above_normal"` and 13 `"high"` (other values are rounded down). See also `Task.priority_class`.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with `Task.priority_class`.

2.21.15 priority_class

Priority Class of the Current Class

Syntax: void spooler_task.`set`**`_priority_class`** `( String )`

Syntax: `String` spooler_task. **`priority_class`** `()`

Example: in javascript

```
spooler_task.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and Unix Systems:

Priority Class	Windows	Unix
`"idle"`	4	16
`"below_normal"`	6	6
`"normal"`	8	0
`"above_normal"`	10	-6
`"high"`	13	-16

Note that an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also `Task.priority`, `Subprocess.priority_class` and Microsoft® Windows® Scheduling Priorities.

2.21.16 remove_pid

The opposite to `add_pid()`

Syntax: void spooler_task. **remove_pid** (int pid)

An error does not occur when the pid has not been added using Task.

See Task. add_pid( ).

2.21.17 repeat

Restarts a task after the specified time

Syntax: void spooler_task.set_**repeat** (double)

(This method actually belongs to the Job class and has nothing to do with the task currently being processed.)

Should there be no task belonging to the current job running after the time specified has expired, then the JobScheduler starts a new task. Note that the <run_time> element is considered here, and that the <period repeat=""> attribute may be temporarily ignored.

Job. delay_after_error has priority, should a task return an error.

2.21.18 stderr_path

The path to the file in which stderr task output is captured

Syntax: String spooler_task. **stderr_path** ()

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
String

"", should a task not run in a separate <process_classes> process.

2.21.19 stderr_text

Text written to stderr up to this point by the process that was started by the task.

Syntax: String spooler_task. **stderr_text** ()

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
String

"", should the task not have been started in a separate process <process_classes>.

2.21.20 stdout_path

The path of the file in which `stdout` task output is captured

Syntax: `String` spooler_task. **`stdout_path`** `()`

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`String`

`""`, should a task not run in a separate `<process_classes>` process.

2.21.21 stdout_text

Text written to `stdout` up to this point by the process that was started by the task.

Syntax: `String` spooler_task. **`stdout_text`** `()`

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`String`

`""`, should a task not run in a separate `<process_classes>` process.

2.21.22 trigger_files

File paths in folders monitored with regex

Syntax: `String` spooler_task. **`trigger_files`** `()`

Returns the file paths from monitored directories (`Job.start_when_directory_changed()` or `<start_when_directory_changed>`) at the time a task is started. Only applies to directories for which a regular expression has been defined (`regex`).

The paths are taken from the addresses defined in `Job.start_when_directory_changed()` or `<start_when_directory_changed>` and combined with the file names.

The non-API `<process>` and `<script language="shell">` jobs make the content of `Task.trigger_files` available to the `SCHEDULER_TASK_TRIGGER_FILES` environment variable.

See `Job.start_when_directory_changed()` and `Task.changed_directories()`.

Returned value:
`String`

The file paths are separated by semicolons.

`""` otherwise

2.21.23 try_hold_lock

Try to hold a lock

Syntax: `boolean` spooler_task. **`try_hold_lock`** `( String lock_path )`

Example: in javascript

```
function spooler_process()
{
    var result = false;

    if( spooler_task.try_hold_lock( "Georgien" )  &&
        spooler_task.try_hold_lock_non_exlusive( "Venezuela" ) )
    {
        // Task is holding the two locks. Insert processing code here.
        result = ...
    }
    else
    {
        spooler_task.call_me_again_when_locks_available();
    }

    return result;
}
```

`try_lock_hold()` attempts to retain the lock specified (`Lock`), and can be called in:

- `spooler_open()`: the lock is held for the task being carried out and will be freed after the task has been completed,
- `spooler_process()`: the lock is only held for the job step currently being carried out and will be given up after the step has been completed - i.e. after leaving `spooler_process()`.

When the lock is not available and calling this method returns `false` then the JobScheduler can be instructed to either:

- repeat the `spooler_open()` or `spooler_process()` calls as soon as the locks are available using `Task.call_me_again_when_locks_available()` or
- end `spooler_open()` or `spooler_process()` with `false`, without use of the above-mentioned call, (but with the expected effect),
- throw a `SCHEDULER-469` warning. This applies for `true`, which is interpreted as an error.

See also `<lock.use>`.

Returned value:
`boolean`

`true`, when the task retains the lock.

2.21.24 try_hold_lock_non_exclusive

Tries to acquire a non-exclusive lock

Syntax: `boolean` spooler_task. **`try_hold_lock_non_exclusive`** `( String lock_path )`

The same prerequisites apply as to `Task.try_hold_lock()`.

See <lock.use_exclusive="no">.

Returned value:
boolean

true, if the task successfully acquired the lock.

2.21.25 web_service

The Web Service which a task has been allocated to.

Syntax: Web_service spooler_task. **web_service** ()

This property causes an exception when a task has not been allocated to a Web Service.

See also Task.web_service_or_null.

Returned value:
Web_service

2.21.26 web_service_or_null

The Web Service to which a task has been allocated, or null.

Syntax: Web_service spooler_task. **web_service_or_null** ()

See also Task.web_service.

Returned value:
Web_service

2.22 Variable_set - A Variable_set may be used to pass parameters

Variable_set is used for the JobScheduler variables and task parameters. A new Variable_set is created using Spooler.create_variable_set().

Variable names are case independent.

The value of a variable is known as a variant in the COM interface (JavaScript, VBScript, Perl). Because variables are usually written in the JobScheduler database, only variant types which can be converted into strings should be used here.

The value of a variable in Java is a string. Therefore, a string value is returned when reading this variable, when it is set as a variant in the COM interface. Null and Empty are returned as null. An error is caused should the value of a variant not be convertible.

2.22.1 count

The number of variables

Syntax: `int` variable_set. **`count`** `()`

2.22.2 merge

Merges with values from another Variable_set

Syntax: void variable_set. **`merge`** `(` Variable_set `vs )`

Variables with the same name are overwritten.

2.22.3 names

The separation of variable names by semicolons

Syntax: `String` variable_set. **`names`** `()`

Example:

```
Variable_set variable_set = spooler.create_variable_set();
spooler_log.info( "\"" + variable_set.names() + "\"" );    // ==> ""

variable_set.set_var( "variable_1", "edno" );
variable_set.set_var( "variable_2", "dwa" );

spooler_log.info( "\"" + variable_set.names() + "\"" );    // ==>
"variable_1;variable_2"

java.util.StringTokenizer t = new java.util.StringTokenizer( variable_set.names(), ";"
);
while( t.hasMoreTokens() )
{
    String name = t.nextToken();
    spooler_log.info( name + "=" + variable_set.var( name ) );
}
```

Returned value:
`String`

All variable names should be separated by semicolons.

2.22.4 substitute

Replaces $-Variables in a String

Syntax: `String` variable_set. **`substitute`** `( String sustitution_string )`

Example: in javascript

```
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
```

In the example below, the `Subprocess.env` method is used.

References in the string in the form `$ name` and `${ name }` are replaced by variables.

Returned value:
`String`

The string containing the substituted $ variables.

2.22.5 value

A variable

Syntax: void variable_set.`set_value ( String name, String value )`

Syntax: `String` variable_set. `value ( String name )`

Parameters:

`name`	
`value`	`empty`, should a variable not exist.

Returned value:
`String`

`empty`, should a variable not exist.

2.22.6 var

A variable

Syntax: void variable_set.`set_var ( String name, String value )`

Syntax: `String` variable_set. `var ( String name )`

Use the `Variable_set.value`, which is available in all languages.

Parameters:

`name`	
`value`	`empty`, should a variable not exist.

Returned value:
`String`

`empty`, should a variable not exist.

2.22.7 xml

`Variable_set` as an XML document

Syntax: void variable_set.`set_xml ( String )`

Syntax: `String` variable_set. **`xml`** `()`

Example: in javascript

```
var variable_set = spooler.create_variable_set();
spooler_log.info( variable_set.xml );   // Liefert <?xml version='1.0'?><
sos.spooler.variable_set/>

variable_set.xml= "<?xml version='1.0'?>" +
                  "<params>" +
                      "<param name='surname' value='Meier'/>" +
                      "<param name='christian name' value='Hans'/>" +
                  "</params>";
spooler_log.info( variable_set.xml );
spooler_log.info( "nachname=" + variable_set.value( "surname" ) );
spooler_log.info( "vorname =" + variable_set.value( "christian name" ) );
```

See <sos.spooler.variable_set>, <params>.

Parameters:

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

Returned value:
`String`

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

2.23 Web_service

See also <web_service>

2.23.1 forward_xslt_stylesheet_path

Path to the forwarding XSLT stylesheets

Syntax: `String` web_service. **`forward_xslt_stylesheet_path`** `()`

See also <web_service_forward_xslt_stylesheet="">

2.23.2 name

The Name of the JobScheduler Web Service

Syntax: String web_service. **name** ()

See also <web_service_name="">

2.23.3 params

Freely definable parameters

Syntax: _Variable_set_ web_service. **params** ()

The Web Services parameters can be set using the <web_service> element.

Returned value:
Variable_set_

2.24 Web_service_operation

See also <web_service>

2.24.1 peer_hostname

Peer (Remote) Host Name

Syntax: String web_service_operation. **peer_hostname** ()

Returned value:
String

"", should it not be possible to determine the name.

2.24.2 peer_ip

Peer (Remote) IP Address

Syntax: String web_service_operation. **peer_ip** ()

2.24.3 request

Requests

Syntax: Web_service_request web_service_operation. **request** ()

Returned value:
Web_service_request

2.24.4 response

Answers

Syntax: Web_service_response web_service_operation. **response** ()

Returned value:
Web_service_response

2.24.5 web_service

Syntax: Web_service web_service_operation. **web_service** ()

Returned value:
Web_service

2.25 Web_service_request

See Web_service_operation.

2.25.1 binary_content

Payload as a Byte Array (Java only)

Syntax: byte[] web_service_request. **binary_content** ()

This property is only available under Java.

The ("Content-Type") header field is used to inform the client how binary content is to be interpreted (see HTTP/1.1 14.17 Content-Type) and Web_service_request. charset_name).

2.25.2 charset_name

Character Set

Syntax: `String` web_service_request. **`charset_name`** `()`

Example: in javascript

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Returns the `charset=` parameter from the `Content-Type:` header entry.

2.25.3 content_type

Content Type (without parameters)

Syntax: `String` web_service_request. **`content_type`** `()`

Returns the `Content-Type:` header entry, without parameters - e.g. `"text/plain"`.

2.25.4 header

Header Entries

Syntax: `String` web_service_request. **`header`** `( String name )`

Example: in javascript

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.request.header( "Content-Type" ) );
```

Parameters:

`name` Case is not relevant.

Returned value:
`String`

Returns `""` in event of an unrecognized entry.

2.25.5 string_content

Payload as Text

Syntax: `String` web_service_request. **`string_content`** `()`

The character set to be used is taken from the `charset` parameter in the `headers("Content-Type")` (see HTTP/1.1 14.17 Content-Type). ISO-8859-1 will be used as default, should this parameter not be specified.

The following character sets are recognized:

- ISO-8859-1
- UTF-8 (only on Windows systems and restricted to the ISO-8859-1 characters)

See also `Web_service_request.binary_content`.

2.25.6 url

Uniform Resource Locator

Syntax: `String` web_service_request. **`url`** `()`

`url = "http://" + header( "Host" ) +` *url_path*

2.26 Web_service_response

Note that the `binary_content` property is only available under Java.

See also `<web_service>`

2.26.1 charset_name

Character set

Syntax: `String` web_service_response. **`charset_name`** `()`

Example: in javascript

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Reads the `charset=` parameter from the `Content-Type:` header entry.

2.26.2 content_type

Content-Type (without parameters)

Syntax: `String` web_service_response. **`content_type`** `()`

Reads the `Content-Type:` header without any of the other associated parameters such as `charset=`.

2.26.3 header

Header Entries

Syntax: void web_service_response.`set_`**`header`** `( String value, String name )`

Syntax: `String` web_service_response. **`header`** `( String name )`

Example: in javascript

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.response.header( "Content-Type" ) );
```

Parameters:

`value`	`""` is used for unknown entries.
`name`	The case in which entries are written is not relevant here.

Returned value:
`String`

`""` is used for unknown entries.

2.26.4 send

Sends a Reply

Syntax: void web_service_response. **`send`** `()`

2.26.5 status_code

HTTP Status Code

Syntax: void web_service_response.`set_`**`status_code`** `( int )`

The default setting is 200 (OK).

2.26.6 string_content

Text payloads

Syntax: void web_service_response.set_**string_content** (String text)

Example: in javascript

```
var response = spooler_task.order.web_service_operation.response;
response.content_type = "text/plain";
response.charset_name = "iso-8859-1";
response.string_content = "This is the answer";
response.send();
```

The `header( "Content-Type" )` must first of all contain a `charset` parameter such as:

```
header( "Content-Type" ) = "text/plain; charset=iso-8859-1";
```

Text is coded as specified in the `charset` parameter. ISO-8859-1 will be used as the default value, should this parameter not be specified.

See `Web_service_request.string_content` for the character sets which are allowed.

See `Web_service_response.charset_name`.

2.27 Xslt_stylesheet

An XSLT style sheet contains the instructions for the transformation of an XML document.

The XSLT processor is implemented with libxslt .

2.27.1 apply_xml

Applies a style sheet to an XML document.

Syntax: String **x.** **apply_xml** (String xml)

2.27.2 close

Frees the style sheet resources

Syntax: void **x.** **close** ()

2.27.3 load_file

Loads the style sheet from an XML file

Syntax: void x. **load_file** (java.io.File path)

Syntax: void x. **load_file** (String path)

2.27.4 load_xml

Loads the style sheet from an XML document

Syntax: void x. **load_xml** (String xml)

3 Javascript API

The following classes are available for Javascript:

3.1 Error

3.1.1 code

The error code

Syntax: `string` error. **`code`**

3.1.2 is_error

`true`, should an error have occurred

Syntax: `boolean` error. **`is_error`**

3.1.3 text

The error text (with error code)

Syntax: `string` error. **`text`**

3.2 Job

A task can either be waiting in the order queue or be running.

3.2.1 clear_delay_after_error

Resets all delays which have previously been set using `delay_after_error`

Syntax: spooler_job. **`clear_delay_after_error`** `()`

3.2.2 clear_when_directory_changed

Resets directory notification for all directories which have previously been set using `start_when_directory_changed()`

Syntax: spooler_job. **clear_when_directory_changed** ()

3.2.3 configuration_directory

Directory for the job configuration file should dynamic configuration from hot folders be used

Syntax: string spooler_job. **configuration_directory**

"", when a job does not come from a configuration directory.

3.2.4 delay_after_error

Delays the restart of a job in case of an error

Syntax: spooler_job. **delay_after_error** (int error_steps) = double|int|string seconds_or_hhmm_ss

Example:

```
spooler_job.delay_after_error(  2 ) = 10;           // A 10 second delay after the 2nd
consecutive error
spooler_job.delay_after_error(  5 ) = "00:01";      // One minute delay after the 5th
consecutive error
spooler_job.delay_after_error( 10 ) = "24:00";      // A delay of one day after the
10th consecutive error
spooler_job.delay_after_error( 20 ) = "STOP";       // The Job is stopped after the
20th consecutive error
```

Should a (first) error occur whilst a job is being run, the JobScheduler will restart the job immediately.
However, after between two and four consecutive errors, the JobScheduler will wait 10 seconds before restarting the job;

After between five and nine consecutive errors, the job will be restarted after a delay of one minute; After between ten and nineteen errors, the delay is 24 hours.

The job is stopped after the twentieth consecutive error.

A delay can be specified, should a particular number of errors occur in series. In this case the job will be terminated and then restarted after the time specified.

This method call can be repeated for differing numbers of errors. A different delay can be specified for each new method call.

It is possible to set the value of the seconds_or_hhmm_ss parameter to "STOP" in order to restrict the number of (unsuccessful) repetitions of a job. The job then is stopped when the number of consecutive errors specified is reached.

A good position for this call is spooler_init().

See <delay_after_error>.

Parameters:

error_steps	The number of consecutive errors required to initiate the delay
seconds_or_hhmm_ss	The delay after which the job will be rerun

3.2.5 delay_order_after_setback

Delays after an order is setback

Syntax: spooler_job. **delay_order_after_setback** (int setback_count) = double|int|string seconds_or_hhmm_ss

Example:

```
spooler_job.delay_order_after_setback( 1 ) = 60;          // for the 1st and 2nd
consecutive setbacks of an order:
               // delay the order 60s.


spooler_job.delay_order_after_setback( 3 ) = "01:00";     // After the 3rd consecutive
setback of an order,
               // the order will be delayed an hour.


spooler_job.max_order_setbacks = 5;                  // The 5th setback sets the order
to the error state
```

A job can delay an order which is currently being carried out with Order.setback(). The order is then positioned at the rear of the order queue for that job and carried out after the specified time limit.

The number of consecutively occurring setbacks for an order is counted. The delay set after a setback can be changed using delay_order_after_setback in the event of consecutively occurring setbacks.

See
<delay_order_after_setback>,

Order.setback(),

Job.max_order_setbacks,

Job_chain.add_job(),

Job.delay_after_error().

Parameters:

setback_count	The number of consecutive errors and therefore setbacks for a job. The setback delay can be varied according to this parameter.
seconds_or_hhmm_ss	Time limit for the setback of the order. After expiry of the time limit, the order is reprocessed in the same job.

3.2.6 folder_path

The directory in which the job is to be found.

Syntax: `string` spooler_job. **`folder_path`**

`""`, when the job does come from the local (`<config configuration_directory="">`) configuration file.

Returns the job part relative to the live directory. The path is to start with a slash ("/") and all path components are to be separated by slashes.

Examples:

- `"/somewhere/excel"` will be returned for the `c:\scheduler\config\live\somewhere\excel\sample.job.xml` job;
- `"/"` returned for the `c:\scheduler\config\live\sample.xml` job and
- `""` (an empty string) returned for a job outside the live directory.

3.2.7 include_path

Value of the `-include-path=` option

Syntax: `string` spooler_job. **`include_path`**

See `-include-path`.

3.2.8 max_order_setbacks

Limits the number of setbacks for an order

Syntax: spooler_job. **`max_order_setbacks`** = `int`

An order state is set to "error" (see `Job_chain_node.error_state`) when it is set back more than the number of times specified here (see `Order.setback()`).

See `Job.delay_order_after_setback` and `<delay_order_after_setback is_maximum="yes">`.

3.2.9 name

The job path beginning without a backslash

Syntax: `string` spooler_job. **`name`**

See `<job name="">`.

3.2.10 order_queue

The job order queue

Syntax: _Order_queue_ spooler_job. **order_queue**

Example:

```
spooler_log.info( 'order=' + ( spooler_job.order_queue ? "yes" : "no" ) );
```

Every job order (<job order="yes">) has an order queue. This queue is filled by the job chain to which the job belongs.

See Job_chain.add_order() , and Job_chain.add_job() .

Returned value:
Order_queue

`null`, should the job have no queue (for <job order="no">).

3.2.11 process_class

The process class

Syntax: _Process_class_ spooler_job. **process_class**

See <job process_class="">.

Returned value:
Process_class

3.2.12 remove

Removes a job

Syntax: spooler_job. **remove** ()

The job is stopped - i.e. current tasks are terminated and no new ones are started. The job will be removed as soon as no more tasks are running.

Tasks queuing are ignored.

When no job task is running, the remove() function deletes the job immediately.

Job orders (<job order="yes">) cannot be removed.

See <modify_job cmd="remove">.

3.2.13 start

Creates a new task and places it in the task queue

Syntax: Task spooler_job. **start** (Variable_set variables (optional))

Example:

```
spooler.job( "job_a" ).start();

var parameters = spooler.create_variable_set();
parameters.value( "my_parameter" ) = "my_value";
parameters.value( "other_parameter" ) = "other_value";
spooler.job( "job_a" ).start( parameters );
```

The parameters are available to the Task.params task. Two parameters are particularly relevant here:

"spooler_task_name"	gives the task a name which then appears in the status display, e.g. in the web interface.
"spooler_start_after"	specifies a time in seconds (real number), after which the task is to start. The JobScheduler <run_time> is ignored in this case.

See Spooler.create_variable_set(), Spooler.job, Variable_set.value.

Returned value:
Task

3.2.14 start_when_directory_changed

Monitors a directory and starts a task should a notification of a change be received

Syntax: spooler_job. **start_when_directory_changed** (string directory_path, string filename_pattern (optional))

Example:

```
spooler_job.start_when_directory_changed( "c:/tmp" );

// only relevant for files whose names do not end in "~".
spooler_job.start_when_directory_changed( "c:/tmp", "^.*[^~]$" );
```

Should there not be a task belonging to this job running and a notification be received that a change in the directory being monitored has occurred (that a file has been added, changed or deleted), then this change can be used to prompt the JobScheduler to start a task if the current time falls within that allowed by the <run_time> parameter.

This method can be called a more than once in order to allow the monitoring of a number of directories. A repeat call can also be made to a directory in order to reactivate monitoring - if, for example, it has not been possible to access the directory.

This method call can be coded in the JobScheduler start script or in the spooler_init() method. In the latter case, the job must have been started at least once in order for the method call to be carried out. The <run_time once="yes"> setting should be used for this.

The job should be regularly <run_time repeat=""> restarted and <delay_after_error> set.

The same setting can be made in the XML configuration using the <start_when_directory_changed> element.

Parameters:

directory_path	the address of the directory being monitored
filename_pattern	restricts monitoring to files whose names correspond with the regular expression used.

3.2.15 state_text

Free text for the job state

Syntax: spooler_job. **state_text** = string

Example:

```
spooler_job.state_text = "Step C succeeded";
```

The text will be shown in the HTML interface.

3.2.16 title

The job title

Syntax: string spooler_job. **title**

Example:

```
spooler_log.info( "Job title=" + spooler_job.title );
```

See <job title="">.

3.2.17 wake

Causes a task to be started

Syntax: spooler_job. **wake** ()

Starts a task, should the job have the pending or stopped states.

See Job.start().

3.3 Job_chain - job chains for order processing

A job chain is a series of jobs (job chain nodes). Orders (Order) proceed along these chains.

Every position in a job chain is assigned a state and a job. When an order is added to the job chain, it is enqueued by the JobScheduler according to the state of the order. The job assigned to this position then carries out the order.

Additionally, each position in a job chain has a successor state and an error state. The JobScheduler changes the state of an order after each job in the job chain has been processed. Should the job step return `(spooler_process) true`, then the JobScheduler sets the succeeding state; otherwise it sets the error state. The order then moves to another position in the job chain as defined by the new state. However, this does not apply when the state is changed during execution with Order.state.

A job chain is created using Spooler.create_job_chain(); it is filled using Job_chain.add_job() and Job_chain.add_end_state() and finally made available with Spooler.add_job_chain().

Every node is allocated a unique state. Therefore either Job_chain.add_job() or Job_chain.add_end_state() must be called once for every state.

Example:

```
var my_job_chain = spooler.create_job_chain();
my_job_chain.name = "JobChain";

my_job_chain.add_job( "job_100", 100,  200, 999 );
my_job_chain.add_job( "job_200", 200, 1000, 999 );
my_job_chain.add_end_state(  999 );
my_job_chain.add_end_state( 1000 );
spooler.add_job_chain( my_job_chain );
```

3.3.1 add_end_state

Adds the end state to a job chain

Syntax: job_chain. **add_end_state** `( var state )`

This state is not assigned a job. An order that reaches the final state has completed the job chain and will be removed from the chain.

3.3.2 add_job

Adds a job to a job chain

Syntax: job_chain. **add_job** `( string job_name, var input_state, var output_state, var error_state )`

3.3.3 add_or_replace_order

Adds an order to a job chain and replaces any existing order having the same identifier

Syntax: job_chain. **add_or_replace_order** (Order order)

Should the job chain already contain an order with the same identifier, then this order will be replaced. More accurately: the original order will be deleted and the new one added to the job chain.

As long as an existing order having the same identifier as the new order is being carried out, both orders will be present. However, the original order will have already been deleted from the job chain and database; it is only available to the current task and will completely disappear after it has been completed.

In this case the JobScheduler will wait until the original order has been completed before starting the new one.

See Job_chain.add_order() and Order.remove_from_job_chain()

3.3.4 add_order

Adds an order to a job chain

Syntax: Order job_chain. **add_order** (Order | string order_or_payload)

Should an order already exist on another job chain, then the JobScheduler removes the order from this other chain.

An order is allocated to the job order queue corresponding to its state, and positioned according to its priority.

The job chain must be specified for the JobScheduler using <job_chain> or Spooler.add_job_chain().

Should an order with the same Order.id already exist in a job chain, then an exception with the error code SCHEDULER-186 is returned. However, see also Job_chain.add_or_replace_order().

Returned value:
Order

3.3.5 name

The name of a job chain

Syntax: job_chain. **name** = string

Syntax: string job_chain. **name**

Example:

```
var job_chain = spooler.create_job_chain();
job_chain.name = "JobChain";
```

3.3.6 node

The job chain nodes with a given state

Syntax: Job_chain_node job_chain. **node** (var state)

Returned value:
Job_chain_node

3.3.7 order_count

The number of orders in a job chain

Syntax: int job_chain. **order_count**

3.3.8 order_queue

```
= node( state).job().order_queue()
```

Syntax: Order_queue job_chain. **order_queue** (var state)

Returns the order queue which has a given state.

Returned value:
Order_queue

3.3.9 orders_recoverable

Syntax: job_chain. **orders_recoverable** = boolean

Syntax: boolean job_chain. **orders_recoverable**

See <job_chain orders_recoverable="">.

3.3.10 remove

Job chain deletion

Syntax: job_chain. **remove** ()

Should orders in a job chain still be being processed (in spooler_process()) when the chain is to be deleted, then the JobScheduler will wait until the last order has been processed before deleting the chain.

Orders remain in the database. Should a new job chain be added which has the same name as a deleted job chain (Spooler.add_job_chain()), then the JobScheduler will reload any orders from the original job chain which have remained in the database. Note however, that the states of the orders in the new job chain should be the same as those in the original chain at the time of its deletion.

3.3.11 title

Syntax: job_chain. **`title =`** `string`

Syntax: `string` job_chain. **`title`**

See `<job_chain title="">`.

3.4 Job_chain_node

A job chain node is assigned a position in a job chain (`Job_chain`). The following elements make up a job chain node: a state, a job, a successor state and an error state.

A job chain node is created either using `Job_chain.add_job()` or `Job_chain.add_end_state()`.

3.4.1 action

Stopping or missing out job chain nodes

Syntax: node. **`action =`** `string`

Syntax: `string` node. **`action`**

Example:

```
var job_chain_node = spooler.job_chain( "my_job_chain" ).node( 100 );
job_chain_node.action = "next_state";
```

This option is not possible with distributed job chains.

Possible settings are:

`action="process"`
: This is the default setting. Orders are carried out.

`action="stop"`
: Orders are not carried out, they collect in the order queue.

`action="next_state"`
: Orders are immediately handed over to the next node as specified with `next_state`.

See also `<job_chain_node.modify action="">`.

Character string constonants are defined in Java:

- `Job_chain_node.ACTION_PROCESS`
- `Job_chain_node.ACTION_STOP`
- `Job_chain_node.ACTION_NEXT_STATE`

3.4.2 error_node

The next node in a job chain in the event of an error

Syntax: Job_chain_node. node. **error_node**

Example:

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

     spooler_log.debug( "error state=" + job_chain_node.error_node.state );       //
"state=999"
```

Returned value:
Job_chain_node.

null, in the event of no error node being defined (the error state has not been specified)

3.4.3 error_state

State of a job chain in event of an error

Syntax: var node. **error_state**

Example:

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "error state=" + job_chain_node.error_node.state );       // "error
state=999"
```

3.4.4 job

The job allocated to a node

Syntax: Job. node. **job**

Example:

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "job=" + job_chain_node.job.name );                        //
"job=job_100"
```

Returned value:
Job.

3.4.5 next_node

Returns the next node or null if the current node is assigned the final state.

Syntax: Job_chain_node node. **next_node**

Returned value:
Job_chain_node

3.4.6 next_state

The order state in a job chain after successful completion of a job

Syntax: var node. **next_state**

Example:

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "next_state=" + job_chain_node.next_state );              //
"state=200"
```

3.4.7 state

The valid state for a job chain node

Syntax: var node. **state**

Example:

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.info( "state=" + job_chain_node.state );                         //
"state=100"
```

3.5 Job_impl - Super Class for a Job or the JobScheduler Script

Job methods are called in the following order:

```
spooler_init()
    spooler_open()
        spooler_process()
        spooler_process()
        ...
    spooler_close()
    spooler_on_success() or spooler_on_error()
```

spooler_exit()

None of these methods must be implemented. However, it is usual that at least the spooler_process() method is implemented.

An error during carrying out a job script whilst loading or during spooler_init() causes spooler_on_error(). to be called. The job is then stopped and spooler_exit() called (although spooler_init() has not been called!). The script is then unloaded.

Note that spooler_on_error() must also be able to handle errors which occur during loading or in spooler_init().

Note also that spooler_exit() is called even though spooler_init() has not been called.

3.5.1 spooler

The JobScheduler base object

Syntax: Spooler **spooler**

Example:

```
spooler_log.debug( "The working directory of the JobScheduler is " + spooler.directory
);
```

Returned value:
Spooler

3.5.2 spooler_close

Task end

Syntax: **spooler_close** ()

This method is called after a job has been completed. The opposite of this method is spooler_open().

3.5.3 spooler_exit

Destructor

Syntax: **spooler_exit** ()

Is called as the last method before the script is unloaded. This method can be used, for example, to close a database connection.

3.5.4 spooler_init

Initialization

Syntax: `boolean` **`spooler_init`** `()`

The JobScheduler calls these methods once before `spooler_open()`. This is analog to `spooler_exit()`. This method is suitable for initializing purposes (e.g. connecting to a database).

Returned value:
`boolean`

`false` ends a task. The JobScheduler continues using the `spooler_exit()` method. When the task is processing an order, then this return value makes the JobScheduler terminate the job with an error. That is, unless a repeated start interval has been set using `Job.delay_after_error`

3.5.5 spooler_job

The job object

Syntax: `Job` **`spooler_job`**

Example:

```
spooler_log.info( "The name of this job is " + spooler_job.name );
```

Returned value:
`Job`

3.5.6 spooler_log

Event logging object

Syntax: `Log` **`spooler_log`**

Example: in java

```
spooler_log.info( "Something has happened" );
```

Returned value:
`Log`

3.5.7 spooler_on_error

Unsuccessful completion of a job

Syntax: **`spooler_on_error`** `()`

Is called at the end of a job after an error has occurred (after spooler_close() but before spooler_exit()).

3.5.8 spooler_on_success

Successful completion of a job

Syntax: **spooler_on_success** ()

This method is called by the JobScheduler after spooler_close() and before spooler_exit() ; should no error have occurred.

3.5.9 spooler_open

The Start of a Task

Syntax: boolean **spooler_open** ()

This method is called immediately after spooler_init() . The opposite of this method is spooler_close() .

3.5.10 spooler_process

Job steps or the processing of an order

Syntax: boolean **spooler_process** ()

Processes a job step.

An order driven job stores the current order in Task. order .

The default implementation returns false. The implementation of an order driven job can set the successor state for an order by returning true.

Returned value:
boolean

In the event of standard jobs <job order="no"> : false the JobScheduler ends processing of this job; true> the JobScheduler continues calling the spooler_process() method.

In the event of order driven jobs <job order="yes"> : false the order acquires the error state (s. Job_chain_node and <job_chain_node>). true the order acquires the next state or is terminated if the next state is the final state. This, however, does not apply when the state is changed during execution using Order. state .

3.5.11 spooler_task

The task object

Syntax: Task **spooler_task**

Example:

```
spooler_log.info( "The task id is " + spooler_task.id );
```

Returned value:
Task

3.6 Lock

See also <lock name="">.

Example:

```
var locks = spooler.locks;
var lock = locks.create_lock();
lock.name = "my_lock";
locks.add_lock( lock );
```

3.6.1 max_non_exclusive

Limitation of non-exclusive allocation

Syntax: lock. **max_non_exclusive** = int

Syntax: int lock. **max_non_exclusive**

The default setting is unlimited (231-1), which means that with <lock.use_exclusive="no"> any number of non-exclusive tasks can be started (but only one exclusive task).
The number cannot be smaller than the number of non-exclusive allocations.

See also <lock max_non_exclusive="">.

3.6.2 name

The lock name

Syntax: lock. **name** = string

Syntax: string lock. **name**

The name can only be set once and cannot be changed.

See also <lock name="">.

3.6.3 remove

Removes a lock

Syntax: lock. **remove** ()

Example:

```
spooler.locks.lock( "my_lock" ).remove();
```

A lock can only be removed when it is not active - that is, it has not been allocated to a task and it is not being used by a job (<lock.use>).

See also <lock.remove>.

3.7 Locks

3.7.1 add_lock

Adds a lock to a JobScheduler

Syntax: locks. **add_lock** (Lock lck)

3.7.2 create_lock

Creates a new lock

Syntax: Lock locks. **create_lock** ()

Returns a new lock Lock. This lock can be added to the JobScheduler using Locks.add_lock().

Returned value:
Lock

3.7.3 lock

Returns a lock

Syntax: Lock locks. **lock** (string lock_name)

An exception will be returned if the lock is unknown.

Returned value:
Lock

3.7.4 lock_or_null

Returns a lock

Syntax: Lock locks. **lock_or_null** (string lock_name)

Returned value:
Lock

null, when the lock is unknown.

3.8 Log - Logging

The spooler_log method can be used in a job or in the JobScheduler start script with the methods described here.
Notification by e-mail

The JobScheduler can send a log file after a task has been completed per e-mail. The following properties define in which cases this should occur.

- Log.mail_on_error,
- Log.mail_on_warning,
- Log.mail_on_process,
- Log.mail_on_success and
- Log.mail_it

Only the end of a task - and not the end of an order - (i.e. spooler_process()) can initiate the sending of e-mails. However, see Task.end().

The Log.mail method makes the Mail object available, which in turn addresses the mails.

Example:

```
spooler_log.info( "Something for the Log" );

spooler_log.mail_on_warning = true;
spooler_log.mail.from    = "scheduler@company.com";
spooler_log.mail.to      = "admin@company.com";
spooler_log.mail.subject = "ended";
```

3.8.1 debug

Debug message (level -1)

Syntax: spooler_log. **debug** (string line)

3.8.2 debug1

Debug message (level -1)

Syntax: spooler_log. **debug1** (string line)

3.8.3 debug2

Debug message (level -2)

Syntax: spooler_log. **debug2** (string line)

3.8.4 debug3

Debug message (level -3)

Syntax: spooler_log. **debug3** (string line)

3.8.5 debug4

Debug message (level -4)

Syntax: spooler_log. **debug4** (string line)

3.8.6 debug5

Debug message (level -5)

Syntax: spooler_log. **debug5** (string line)

3.8.7 debug6

Debug message (level -6)

Syntax: spooler_log. **debug6** (string line)

3.8.8 debug7

Debug message (level -7)

Syntax: spooler_log. **debug7** (string line)

3.8.9 debug8

Debug message (level -8)

Syntax: spooler_log. **debug8** (string line)

3.8.10 debug9

Debug message (level -9)

Syntax: spooler_log. **debug9** (string line)

3.8.11 error

Error Message (Level 1)

Syntax: spooler_log. **error** (string line)

A job stops after a task has ended, should an error message have been written in the task log (spooler_log) and <job stop_on_error="no"> not have been set.

3.8.12 filename

Log file name

Syntax: string spooler_log. **filename**

3.8.13 info

Information message (Level 0)

Syntax: spooler_log. **info** (string line)

3.8.14 last

The last output with the level specified

Syntax: `string` spooler_log. **`last`** `( int| string level )`

3.8.15 last_error_line

The last output line with level 2 (error)

Syntax: `string` spooler_log. **`last_error_line`**

3.8.16 level

Limit protocol level

Syntax: spooler_log. **`level =`** `int`

Syntax: `int` spooler_log. **`level`**

Defines the level with which protocol entries should be written. Every protocol entry is given one of the following categories: `error`, `warn`, `info`, `debug1` to `debug9` (`debug1` is the same as `debug`).

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

The `-log-level` option has precedence over this parameter.

The `factory.ini` (section `[job]`, entry `log_level=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_level=...`) setting is overwritten by this parameter.

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

3.8.17 log

Writes in the log file with the specified level.

Syntax: spooler_log. **log** (int level, string line)

3.8.18 log_file

Adds the content of a file to the log file

Syntax: spooler_log. **log_file** (string path)

Log the content of a file with level 0 (info). An error occurring whilst accessing the file is logged as a warning.

Note that when executed on a remote computer with <process class remote scheduler=""> the file is read from the JobScheduler's file system and not that of the task.

3.8.19 mail

E-mail settings are made in the Mail Object

Syntax: spooler_log. **mail** = Mail

Syntax: Mail spooler_log. **mail**

Returned value:
Mail

3.8.20 mail_it

Force dispatch

Syntax: spooler_log. **mail_it** = boolean

If this property is set to true, then a log will be sent after a task has ended, independently of the following settings:

Log. mail on error, Log. mail on warning, Log. mail on success, Log. mail on process and Log. mail on error.

3.8.21 mail_on_error

Sends an e-mail should a job error occur. Errors are caused by the `Log.error()` method or by any exceptions that have not been caught by a job.

Syntax: spooler_log. **`mail_on_error`** = `boolean`

Syntax: `boolean` spooler_log. **`mail_on_error`**

Content of the e-mail is the error message. The log file is sent as an attachment.

The `factory.ini` (section `[job]`, entry `mail_on_error=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `mail_on_error=...`) setting is overwritten by this parameter.

Content of the e-mail is the error message. The log file is sent as an attachment.

3.8.22 mail_on_process

Sends an e-mail should a job have successfully processed the number of steps specified. Steps are caused by the `spooler_process()` methods:

Syntax: spooler_log. **`mail_on_process`** = `int`

Syntax: `int` spooler_log. **`mail_on_process`**

Causes the task log to be sent when a task has completed at least the specified number of steps - i.e. calls of `spooler_process()`. Because non-API tasks do not have steps, the JobScheduler counts each task as a single step.

Content of the e-mail is the success message. The log file is sent as an attachment.

The `factory.ini` (section `[job]`, entry `mail_on_process=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `mail_on_process=...`) setting is overwritten by this parameter.

Content of the e-mail is the success message. The log file is sent as an attachment.

3.8.23 mail_on_success

Sends an e-mail should a job terminate successfully.

Syntax: spooler_log. **`mail_on_success`** = `boolean`

Syntax: `boolean` spooler_log. **`mail_on_success`**

The success message forms the content of the e-mail. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_success=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_success=…) setting is overwritten by this parameter.

The success message forms the content of the e-mail. The log file is sent as an attachment.

3.8.24 mail_on_warning

Sends an e-mail should a job warning occur. Warnings are caused by the Log.warn() method.

Syntax: spooler_log. **mail_on_warning** = boolean

Syntax: boolean spooler_log. **mail_on_warning**

The warning forms the content of the e-mail. The log file is sent as an attachment.

The factory.ini (section [spooler], entry mail_on_warning=…) setting is overwritten by this parameter.

The warning forms the content of the e-mail. The log file is sent as an attachment.

3.8.25 new_filename

A new name for the log file

Syntax: spooler_log. **new_filename** = string

Syntax: string spooler_log. **new_filename**

Sets the name of the log file. The JobScheduler copies a log into this file after a log has been made. This file is then available to other applications.

3.8.26 start_new_file

Only for the main log file: closes the current log file and starts a new one

Syntax: spooler_log. **start_new_file** ()

3.8.27 warn

Warning (Level 2)

Syntax: spooler_log. **warn** (string line)

3.9 Mail - e-mail dispatch

See Log.mail.

3.9.1 add_file

Adds an attachment

Syntax: mail. **add_file** (string path, string filename_for_mail (optional) , string content_type (optional) , string encoding (optional))

Example:

```
spooler_log.mail.add_file( "c:/tmp/1.txt", "1.txt", "text/plain", "quoted-printable" );
```

Parameters:

path	path to the file to be appended
filename_for_mail	The file name to appear in the message
content_type	"text/plain" is the preset value.
encoding	e.g. "quoted printable"

3.9.2 add_header_field

Adds a field to the e-mail header

Syntax: mail. **add_header_field** (string field_name, string value)

3.9.3 bcc

Invisible recipient of a copy of a mail, (*blind carbon copy*)

Syntax: mail. **bcc** = string

Syntax: string mail. **bcc**

Example:

```
spooler_log.mail.bcc = "hans@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_bcc=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_bcc=...`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

3.9.4 body

Message content

Syntax: mail. **`body`** = `string`

Syntax: `string` mail. **`body`**

Example:

```
spooler_log.mail.body = "Job succeeded";
```

Line feed / carriage return is coded with `\n` (`chr(10)` in VBScript).

3.9.5 cc

Recipient of a copy of a mail, (*carbon copy*)

Syntax: mail. **`cc`** = `string`

Syntax: `string` mail. **`cc`**

Example:

```
spooler_log.mail.cc = "hans@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_cc=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_cc=...`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

3.9.6 dequeue

Repeated attempts can be made to send messages from the `queue_dir` directory

Syntax: `int` mail. **`dequeue`** `()`

See `Mail.dequeue_log`, `factory.ini` (section `[spooler]`, entry `mail_queue_dir=…`).

Returned value:
`int`

The number of messages sent

3.9.7 dequeue_log

The `dequeue()` log

Syntax: `string` mail. **`dequeue_log`**

Example:

```
var count = spooler_log.mail.dequeue();
spooler_log.info( count + " messages from mail queue sent" );
spooler_log.info( spooler_log.mail.dequeue_log );
```

See `Mail.dequeue()`.

3.9.8 from

Sender

Syntax: mail. **`from`** = `string`

Syntax: `string` mail. **`from`**

Example:

```
spooler_log.mail.from = "scheduler@company.com";
```

The `factory.ini` (section `[job]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

3.9.9 queue_dir

The directory used for returned e-mails

Syntax: mail. **queue_dir** = string path

Syntax: string mail. **queue_dir**

E-mails which cannot be sent (because, for example, the SMTP server cannot be contacted) are stored in this directory.

In order to send these e-mails later it is necessary to write a job which calls up the Mail. dequeue( ) method.

This setting is generally made in sos. ini (section [mail], entry queue_dir=…).

Environment variables (e.g. $HOME) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The factory. ini (section [job], entry mail_queue_dir=…) setting is overwritten by this parameter.

The factory. ini (section [spooler], entry mail_queue_dir=…) setting is overwritten by this parameter.

The sos. ini (section [mail], entry queue_dir=…) setting is overwritten by this parameter.

3.9.10 smtp

The name of the SMTP server

Syntax: mail. **smtp** = string

Syntax: string mail. **smtp**

Example:

```
spooler_log. mail. smtp = "mail. company. com";
```

These settings are generally made using sos. ini (section [mail], entry smtp=…).

smtp=-queue stops e-mails being sent. Instead mails are written into the file specified in queue_dir. See also sos. ini (section [mail], entry queue_only=…).

The factory. ini (section [job], entry smtp=…) setting is overwritten by this parameter.

The factory. ini (section [spooler], entry smtp=…) setting is overwritten by this parameter.

The sos. ini (section [mail], entry smtp=…) setting is overwritten by this parameter.

3.9.11 subject

Subject, *re*

Syntax: mail. **subject** = string

Syntax: string mail. **subject**

Example:

```
spooler_log.mail.subject = "Job succeeded";
```

The factory.ini (section [job], entry log_mail_subject=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_subject=…) setting is overwritten by this parameter.

3.9.12 to

Recipient

Syntax: mail. **to** = string

Syntax: string mail. **to**

Example:

```
spooler_log.mail.to = "admin@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The factory.ini (section [job], entry log_mail_to=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_to=…) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

3.9.13 xslt_stylesheet

The XSLT style sheet for e-mail processing. Before sending an e-mail the JobScheduler creates an XML document containing the e-mail headers, subject and body. The content of these elements can be adjusted or overwritten by an individual XSLT style sheet. This can be used e.g. to create translations of e-mail content. Having processed the XSLT style sheet the JobScheduler sends the resulting content of the XML elements as e-mail.

Syntax: Xslt_stylesheet_ mail. **xslt_stylesheet**

Returned value:
Xslt_stylesheet_

The XSLT style sheet as a string

3.9.14 xslt_stylesheet_path

The path and file name of the XSL style sheet for e-mail processing.

Syntax: mail. **xslt_stylesheet_path =** string path

Example:

```
spooler_log.mail.xslt_stylesheet_path = "c:/stylesheets/mail.xslt";
```

The path to the XSLT style sheet. XSLT style sheets are used by the JobScheduler for the preparation of e-mails. At the time of writing (April 2006) this subject is not documented.

<config mail_xslt_stylesheet="…">

Parameters:

path The path of the file containing the XSLT style sheet

3.10 Monitor_impl - Using Super Classes for Start Scripts or Jobs

A job can be given a monitor using <monitor>.

A monitor can provide the following methods:

Monitor_impl.spooler_task_before()
Before starting a task - can prevent a task from being started.

Monitor_impl.spooler_task_after()
After a task has been completed.

Monitor_impl.spooler_process_before()
Before spooler_process() - this method can stop spooler_process() from being called.

Monitor_impl.spooler_process_after()

After spooler_process() - can be used to change its return value.

3.10.1 spooler

The JobScheduler Object

Syntax: _Spooler_ **spooler**

Example:

```
spooler_log.debug( "The working directory of the JobScheduler is " + spooler.directory
);
```

Is the same object as spooler in the Job_impl class.

Returned value:
Spooler_

3.10.2 spooler_job

The Job Object

Syntax: _Job_ **spooler_job**

Example:

```
spooler_log.info( "The name of this job is " + spooler_job.name );
```

Is the same object as spooler_job in the Job_impl class.

Returned value:
Job_

3.10.3 spooler_log

Writing Log Files

Syntax: _Log_ **spooler_log**

Example: in java

```
spooler_log.info( "Something has happened" );
```

Is the same object as spooler_log in the Job_impl class.

Returned value:
Log_

3.10.4 spooler_process_after

After `spooler_process()`

Syntax: `boolean` **`spooler_process_after`** `( boolean spooler_process_result )`

Example: in java

```
public boolean spooler_task_after( boolean spooler_process_result )  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    spooler_log.info( "spooler_process() didn't throw an exception and delivered " +
spooler_process_result );
    return spooler_process_result;   // Unchanged result
}
```

The JobScheduler calls this method after `spooler_process()` has been carried out.

Parameters:

`spooler_process_result`	The return value from the `spooler_process()` is set to `false`, should `spooler_process()` have ended with an exception.

Returned value:
`boolean`

Replaces the return value from the `spooler_process()` method or `false`, should `spooler_process()` have ended with an error.

3.10.5 spooler_process_before

Before `spooler_process()`

Syntax: `boolean` **`spooler_process_before`** `()`

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    spooler_log.info( "SPOOLER_PROCESS_BEFORE()" );
    return true;   // spooler_process() will be executed
}
```

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    boolean continue_with_spooler_process = true;

    if( !are_needed_ressources_available() )
    {
        spooler_task.order().setback();
        continue_with_spooler_process = false;
    }

    return continue_with_spooler_process;
}
```

This method is called by the JobScheduler before each call of spooler_process() .

Returned value:
boolean

false prevents further calls to spooler_process() . The JobScheduler continues as though false had been returned by spooler_process() false.

3.10.6 spooler_task

The Task Object

Syntax: Task **spooler_task**

Example:

```
spooler_log.info( "The task id is " + spooler_task.id );
```

Is the same object as spooler_task in the Job_impl class.

Returned value:
Task

3.10.7 spooler_task_after

After Completing a Task

Syntax: **spooler_task_after** ()

Example: in java

```
public void spooler_task_after()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_AFTER()" );
}
```

This method is called by the JobScheduler after a task has been completed.

3.10.8 spooler_task_before

Before Starting a Task

Syntax: `boolean spooler_task_before ()`

Example: in java

```
public boolean spooler_task_before()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    return true;    // Task will be started
  //return false;   // Task will not be started
}
```

This method is called by the JobScheduler before a task is loaded.

Returned value:
`boolean`

`false` does not allow a task to start and `Monitor_impl.spooler_task_after()` will not be called.

3.11 Order - Order

See JobScheduler Documentation, `Spooler.create_order()`, `Job_chain.add_order()`, `Task.order`.
File order

A file order is an order with for which the `scheduler_file_path` parameter has been set: `Order.params`. `Variable_set.value()`.

See JobScheduler Documentation.

Example: An Order with a simple Payload

```
// Create order:
{
    var order = spooler.create_order();
    order.id         = 1234;
    order.title      = "This is my order";
    order.state_text = "This is my state text";
    order.payload    = "This is my payload";
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    spooler_log.info( "order.payload=" + order.payload );
    return true;
}
```

Example: Creating an Order with a Variable_set as a Payload

```
// Create order:
{
    var variable_set = spooler.create_variable_set();
    variable_set.value( "param_one" ) = "11111";
    variable_set.value( "param_two" ) = "22222";

    var order = spooler.create_order();
    order.id      = 1234;
    order.payload = variable_set;
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    var variable_set = order.payload;
    spooler_log.info( "param_one=" + variable_set.value( "param_one" ) );
    spooler_log.info( "param_two=" + variable_set.value( "param_two" ) );
    return true;
}
```

3.11.1 at

The order start time

Syntax: order. **at =** `string|DATE`

Example:

```
order.at = "now+65";
spooler.job_chain( "my_job_chain" ).add_order( order );
```

Used to set the start time before an order is added to an order queue. The following can be specified as a string:

- `"now"`
- `"yyyy-mm-dd HH:MM[:SS]"`
- `"now + HH:MM[:SS]"`
- `"now + seconds"`

This setting changes start times set by `Order.run_time` or `Order.setback()`.

See `<add_order at="">`.

3.11.2 end_state

The state that should be reached when an order has been successfully completed

Syntax: order. **`end_state =`** `var`

Syntax: `var` order. **`end_state`**

When an order has its own `end_state` other than `""` then it is considered to be completed after the job allocated to this end state has been completed and before the order otherwise leaves this state (see `<job_chain_node>` for example to continue to another job which usually comprises a part of the job chain).

The state specified has to reference a valid state of a job node in the job chain.

3.11.3 id

Order Identification

Syntax: order. **`id =`** `var`

Syntax: `var` order. **`id`**

Every order has an identifier. This identifier must be unique within a job chain or job order queue. It should also correspond to the data being processed. Normally database record keys are used.

When an `id` is not set, then the JobScheduler automatically allocates one using `Job_chain.add_order()`.

3.11.4 job_chain

The job chain containing an order

Syntax: `Job_chain` order. **`job_chain`**

Returned value:
`Job_chain`

3.11.5 job_chain_node

The job chain nodes which correspond with the order state

Syntax: _Job_chain_node_ order. **job_chain_node**

Returned value:
Job_chain_node_

3.11.6 log

Order log

Syntax: _Log_ order. **log**

Example:

```
spooler_task.order.log.info( "Only for order log, not for task log" );
spooler_log.info( "For both order log and task log" );
```

Returned value:
Log_

3.11.7 params

The order parameters

Syntax: order. **params** = Variable_set

Syntax: _Variable_set_ order. **params**

`params` is held in `Order.payload`, the latter cannot, therefore, be used together with `params`.

See <add_order>.

Returned value:
Variable_set_

3.11.8 payload

Load - an order parameter.

Syntax: order. **payload** = Variable_set_|string|int|... payload

Syntax: _Variable_set_|string|int|... order. **payload**

Instead of this property, the use of Order.params is recommended, which corresponds to (Variable_set) order.payload.

In addition to Order.id which identifies an order, this field can be used for other information.

See Order.params and Order.xml_payload.

Parameters:

payload May be a string or a Variable_set.

Returned value:
Variable_set|string|int|…

May be a string or a Variable_set.

3.11.9 payload_is_type

Checks the payload COM-Type

Syntax: boolean order. **payload_is_type** (string type_name)

Parameters:

type_name "Spooler.Variable_set","Hostware.Dyn_obj" or "Hostware.Record".

3.11.10 priority

Orders with a higher priority are processed first

Syntax: order. **priority** = int

Syntax: int order. **priority**

3.11.11 remove_from_job_chain

Syntax: order. **remove_from_job_chain** ()

Note that when an order has just been started by a task, then the Order.job_chain property will still return the job chain from which the order has just been removed, using this call, even when "remove_from_job_chain" has been carried out. It is only when the execution has been ended that this method returns null. (other than when the order has just been added to a job chain). This ensures that the job_chain property remains stable whilst a task is being executed.

3.11.12 run_time

<run_time> is used to periodically repeat an order

Syntax: Run_time order. **run_time**

Example:

```
order.run_time.xml = "<run_time><at at='2006-05-23 11:43:00'/></run_time>";
```

See <run_time>.

The <modify_order at="now"> command causes an order which is waiting because of run_time to start immediately.

Returned value:
Run_time

3.11.13 setback

Delays an order back for a period of time

Syntax: order. **setback** ()

An order will be delayed and repeated after the period of time specified in either <delay_order_after_setback> or Job.delay_order_after_setback. When the job is repeated, only the spooler_process() job function is repeated. If the order.setback() function is called from spooler_process(), then the retrun value from spooler_process() will have no effect. .

An order counts the number of times this method is called in sequence. This count is then used by <delay_order_after_setback>. It is set to 0, when spooler_process() is completed without <delay_order_after_setback> being called. All counters are set to 0 when the JobScheduler is started.

The <modify_order at="now"> command causes a blocked order to start immediately.

3.11.14 setback_count

How many times the order is setting back?

Syntax: int order. **setback_count**

see also <delay_order_after_setback>.

3.11.15 state

The order state

Syntax: order. **state** = var

Syntax: `var` order. **`state`**

When an order is in a job chain, then its state must correspond with one of the states of the job chain.

Whilst an order is being processed by a job the following state, as defined in the job chain (`<job_chain_node next_state="">`) has no effect. Similarly, the return values from `spooler_process()` and `Monitor_impl.spooler_process_after()` are meaningless. This means that with `Order.state` the following state for a job can be set as required.

An order is added to the job order queue which is corresponding to its state. See `<job_chain_node>`. The execution by this job will be delayed until the job currently carrying out the order has been completed.

3.11.16 state_text

Free text for the order state

Syntax: order. **`state_text`** = `string`

Syntax: `string` order. **`state_text`**

This text is shown on the HTML interface.

For non-API jobs the JobScheduler fills this field with the first line from stdout, up to a maximum of 100 characters.

3.11.17 string_next_start_time

The next start time of an order when `<run_time>` is being used

Syntax: `string` order. **`string_next_start_time`**

Returned value:
`string`

`"yyyy-mm-dd HH:MM:SS.MMM"` or `"now"` or `"never"`.

3.11.18 suspended

Suspended order

Syntax: order. **`suspended`** = `boolean`

Syntax: `boolean` order. **`suspended`**

A suspended order will not be executed.

When an order is being carried out by a task when it is suspended, then the `spooler_process()` step will be completed and the order allocated the successor state before being suspended.

This means that an order can be set to an end state, which stops it from being removed. The JobScheduler can remove such an order only when it is not suspended - i.e. `order.suspended=false`).

A suspended order with the end state can be allocated a different state corresponding to a job node in the job chain. This is effected by using `Order.state`. In this case the order remains suspended.

3.11.19 title

Optionally a title can be allocated to an order that will show up in the HTML interface and in the logs.

Syntax: order. **`title =`** `string`

Syntax: `string` order. **`title`**

3.11.20 web_service

The web service to which an order has been allocated

Syntax: `Web_service` order. **`web_service`**

When an order has not been allocated to a web service, then this call returns the `SCHEDULER-240` error.

See also `Order.web_service_or_null`.

Returned value:
`Web_service`

3.11.21 web_service_operation

The web service operation to which an order has been allocated

Syntax: `Web_service_operation` order. **`web_service_operation`**

Example: in java

```
public boolean spooler_process()  throws Exception
{
    Order                 order                 = spooler_task.order();
    Web_service_operation web_service_operation = order.web_service_operation();
    Web_service_request   request               = web_service_operation.request();

    // Decode request data
    String request_string = new String( request.binary_content(),
request.charset_name() );


    process request_string ...;


    String                response_string = "This is my response";
    String                charset_name    = "UTF-8";
    ByteArrayOutputStream byos            = new ByteArrayOutputStream();

    // Encode response data
    Writer writer = new OutputStreamWriter( byos, charset_name );
    writer.write( response_string );
    writer.close();

    // Respond
    Web_service_response response = web_service_operation.response();

    response.set_content_type( "text/plain" );
    response.set_charset_name( charset_name );
    response.set_binary_content( byos.toByteArray() );
    response.send();

    // Web service operation has finished

    return true;
}
```

See <web_service>, Web_service_operation and Order.web_service_operation_or_null,

Returned value:
Web_service_operation

3.11.22 web_service_operation_or_null

The web service operation to which an order has been allocated, or `null`

Syntax: Web_service_operation order. **web_service_operation_or_null**

See Order.web_service_operation, Web_service_operation and <web_service>.

Returned value:
Web_service_operation

3.11.23 web_service_or_null

The web service to which an order has been allocated, or `null`.

Syntax: Web_service_ order. **web_service_or_null**

See also Order.web_service_.

Returned value:
Web_service_

3.11.24 xml

Order in XML: <order>...</order>

Syntax: `string` order. **xml**

Returned value:
`string`

See <order>

3.11.25 xml_payload

XML payload - an order parameter.

Syntax: order. **xml_payload** = `string xml`

Syntax: `string` order. **xml_payload**

This property can include an XML document (in addition to the Order.params_property).

<xml_payload>contains the XML document root element (instead of it being in `#PCDATA` coded form).

3.12 Order_queue - The order queue for an order controlled job

An order controlled job (<job order="yes">has an order queue, which is filled by the orders to be processed by a job. The orders are sorted according to their priority and the time at which they enter the queue.

Processing means that the JobScheduler calls the spooler_process()_method for a task. This method can access the order using the Task.order_property. Should the `spooler_process()` end without an error (i.e. without any exceptions), then the JobScheduler removes the order from the order queue. If the order is in a job chain then it is moved to the next position in the chain.

3.12.1 length

The number of orders in the order queue

Syntax: `int` q. **`length`**

3.13 Process_class

See also `<process_class name="">`.

Example:

```
var process_classs = spooler.process_classs;
var process_class = process_classs.create_process_class();
process_class.name = "my_process_class";
process_classs.add_process_class( process_class );
```

3.13.1 max_processes

The maximum number of processes that are executed in parallel

Syntax: process_class. **`max_processes =`** `int`

Syntax: `int` process_class. **`max_processes`**

Should more tasks have to be started than allowed by this setting, then these tasks starts would be delayed until processes become freed. The default setting is 10.

See also `<process_class max_processes="">`.

3.13.2 name

The process class name

Syntax: process_class. **`name =`** `string`

Syntax: `string` process_class. **`name`**

The name can only be set once and may not be changed.

See also `<process_class name="">`.

3.13.3 remote_scheduler

The address of the remote JobScheduler, which is to execute a process

Syntax: process_class. **remote_scheduler** = string

Syntax: string process_class. **remote_scheduler**

Example:

```
spooler.process_classes.process_class( "my_process_class" ).remote_scheduler =
"host:4444";
```

See also <process_class remote_scheduler="">.

Parameters:

The address is specified in the form: " *host* : *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber* : *portnumber*"

Returned value:
string

The address is specified in the form: " *host* : *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber* : *portnumber*"

3.13.4 remove

Removal of the process class

Syntax: process_class. **remove** ()

Example:

```
spooler.process_classs.process_class( "my_process_class" ).remove();
```

The JobScheduler delays deletion of the process class as long as tasks are still running. No new tasks will be started before the class is deleted.

See also <process_class.remove>.

3.14 Process_classes

3.14.1 add_process_class

Adds a process class to the JobScheduler

Syntax: process_classs. **add_process_class** (Process_class pc)

3.14.2 create_process_class

Creates a new process class

Syntax: Process_class process_classs. **create_process_class** ()

Returnds a new Process_class. This class can be made added to the JobScheduler using Process_classes.add_process_class().

Returned value:
Process_class

3.14.3 process_class

Returns a process class

Syntax: Process_class process_classs. **process_class** (string process_class_name)

An exception will occur if the process class is not known.

Returned value:
Process_class

3.14.4 process_class_or_null

Returns a process class

Syntax: Process_class process_classs. **process_class_or_null** (string process_class_name)

Returned value:
Process_class

null, when the process class is not known.

3.15 Run_time - Managing Time Slots and Starting Times

See <run_time>, Order.Schedule.

Example:

```
var order = spooler_task.order;

// Repeat order daily at 15:00
order.run_time.xml = "<run_time><period single_start='15:00'/></run_time>";
```

3.15.1 schedule

`<schedule>`

Syntax: _Schedule_ run_time. **schedule**

Returned value:
Schedule_

3.15.2 xml

`<run_time>`

Syntax: run_time. **xml** = string

Discards the current setting and resets `Run_time`.

Parameters:

XML document as a string

3.16 Schedule - Runtime

See <schedule>_, <run_time>_, Spooler.schedule_, Run_time_.

Example:

```
spooler.schedule( "my_schedule" ).xml = "<schedule><period single_start='15:00'/><
/schedule>";
```

3.16.1 xml

`<schedule>`

Syntax: schedule. **xml** = string

Syntax: string schedule. **xml**

Deletes the previous setting and resets `Schedule`.

Parameters:

XML document as a string

Returned value:
string

XML document as a string

3.17 Spooler

There is only one class for this object: spooler_.

3.17.1 abort_immediately

Aborts the JobScheduler immediately

Syntax: spooler. **abort_immediately** ()

Stops the JobScheduler immediately. Jobs do not have the possibility of reacting.

The JobScheduler kills all tasks and the processes that were started using the Task. create_subprocess() method. The JobScheduler also kills processes for which a process ID has been stored using the Task. add_pid() method.

See <modify_spooler cmd="abort_immediately"> and JobScheduler Documentation.

3.17.2 abort_immediately_and_restart

Aborts the JobScheduler immediately and then restarts it.

Syntax: spooler. **abort_immediately_and_restart** ()

Similar to the Spooler. abort_immediately() method, only that the JobScheduler restarts itself after aborting. It reuses the command line parameters to do this.

See <modify_spooler cmd="abort_immediately_and_restart"> and JobScheduler Documentation.

3.17.3 add_job_chain

Syntax: spooler. **add_job_chain** (Job_chain chain)

Job_chain. orders_recoverable_=true causes the JobScheduler to load the orders for a job chain from the database.

See Spooler. create_job_chain() . and <job_chain>.

3.17.4 configuration_directory

Path of the Configuration Directory with hot folders

Syntax: `string` spooler. **`configuration_directory`**

```
<config configuration_directory="…">
```

3.17.5 create_job_chain

Syntax: `Job_chain` spooler. **`create_job_chain`** `()`

Returns a new `Job_chain` object. This job chain can be added to the JobScheduler using `Spooler.add_job_chain()` after it has been filled with jobs.

See `<job_chain>`.

Returned value:
`Job_chain`

3.17.6 create_order

Syntax: `Order` spooler. **`create_order`** `()`

Creates a new order. This order can be assigned to a job chain using the `Job_chain.add_order()` method.

Returned value:
`Order`

3.17.7 create_variable_set

Syntax: `Variable_set` spooler. **`create_variable_set`** `()`

Returned value:
`Variable_set`

3.17.8 create_xslt_stylesheet

Syntax: `Xslt_stylesheet` spooler. **`create_xslt_stylesheet`** `( string xml (optional) )`

Parameters:

`xml` Creates an XSLT style sheet as an XML string.

Returned value:
Xslt_stylesheet_

3.17.9 db_history_table_name

The name of the database table used for the job history

Syntax: string spooler. **db_history_table_name**

See also Spooler. db_history_table_name()

The factory. ini (section [spooler], entry db_history_table=…) setting is overwritten by this parameter.

3.17.10 db_name

The database path

Syntax: string spooler. **db_name**

The database connection string for the history. Should no value be specified here, then the files will be saved in .csv format. See factory. ini (section [spooler], entry history_file=…).

A simple file name ending in . mdb (e.g. scheduler. mdb) can also be specified here when the JobScheduler is running on Windows. The JobScheduler then uses a Microsoft MS Access database of this name, which is located in the protocol directory (see the option -log-dir_). Should such a database not exist, then the JobScheduler will create this database.

The JobScheduler automatically creates the tables necessary for this database.

The factory. ini (section [spooler], entry db=…) setting is overwritten by this parameter.

3.17.11 db_order_history_table_name

The name of the order history database table

Syntax: string spooler. **db_order_history_table_name**

See also Spooler. db_order_history_table_name()

The factory. ini (section [spooler], entry db_order_history_table=…) setting is overwritten by this parameter.

3.17.12 db_orders_table_name

The name of the database table used for orders

Syntax: `string` spooler. **`db_orders_table_name`**

See also `Spooler.db_orders_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_orders_table=…`) setting is overwritten by this parameter.

3.17.13 db_tasks_table_name

The name of the task database table

Syntax: `string` spooler. **`db_tasks_table_name`**

See also `Spooler.db_tasks_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_tasks_table=…`) setting is overwritten by this parameter.

3.17.14 db_variables_table_name

The name of the database table used by the JobScheduler for internal variables

Syntax: `string` spooler. **`db_variables_table_name`**

The JobScheduler records internal counters, for example, the ID of the next free task, in this database table.

See also `Spooler.db_variables_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_variables_table=…`) setting is overwritten by this parameter.

3.17.15 directory

The working directory of the JobScheduler on starting

Syntax: `string` spooler. **`directory`**

Changes the Working Directory.

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

The -cd option has precedence over this parameter.

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

Returned value:
`string`

The directory ends on Unix with "/" and on Windows with "\".

3.17.16 execute_xml

Carries out XML commands

Syntax: `string` spooler. **`execute_xml`** `( string xml )`

Example:

```
spooler_log.info( spooler.execute_xml( "<show_state/>" ) );
```

Errors are returned as XML <ERROR> replies.

Parameters:

`xml` See JobScheduler Documentation.

Returned value:
`string`

Returns the answer to a command in XML format.

3.17.17 hostname

The name of the computer on which the JobScheduler is running.

Syntax: `string` spooler. **`hostname`**

3.17.18 id

The value of the command line `-id=` setting

Syntax: `string` spooler. **`id`**

The JobScheduler only selects elements in the XML configuration whose `spooler_id` attributes are either empty or set to the value given here.

When the JobScheduler ID is not specified here, then the JobScheduler ignores the `spooler_id=` XML attribute and selects all the elements in the XML configuration.

See, for example, `<config>`.

The `-id` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `id=…`) setting is overwritten by this parameter.

3.17.19 include_path

Returns the command line setting `-include-path=`.

Syntax: `string` spooler. **`include_path`**

The directory of the files which are to be included by the `<include>` element.

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

Environment variables (e.g. `$HOME`) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The `-include-path` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `include_path=…`) setting is overwritten by this parameter.

`<config include_path="…">`

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

3.17.20 ini_path

The value of the `-ini=` option (the name of the `factory.ini` file)

Syntax: `string` spooler. **`ini_path`**

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

See `-ini`, JobScheduler Documentation

3.17.21 is_service

Syntax: `boolean` spooler. **`is_service`**

Returned value:
`boolean`

is true, when the JobScheduler is running as a service (on Windows) or as a daemon (on Unix).

3.17.22 job

Returns a job

Syntax: `Job_` spooler. **`job`** `( string job_name )`

An exception is returned should the job name not be known.

Returned value:
`Job_`

3.17.23 job_chain

Returns a job chain

Syntax: `Job_chain_` spooler. **`job_chain`** `( string name )`

Should the name of the job chain not be known, then the JobScheduler returns an exception.

Returned value:
`Job_chain_`

3.17.24 job_chain_exists

Syntax: `boolean` spooler. **`job_chain_exists`** `( string name )`

3.17.25 let_run_terminate_and_restart

Syntax: spooler. **`let_run_terminate_and_restart`** `()`

The JobScheduler ends all tasks (by calling the `Job_impl_` method) as soon as all orders have been completed and then stops itself. It will then be restarted under the same command line parameters.

See <modify_spooler cmd="let_run_terminate_and_restart"> and JobScheduler Documentation.

3.17.26 locks

Returns the locks

Syntax: Locks spooler. **locks**

Returned value:
Locks

3.17.27 log

The main log

Syntax: Log spooler. **log**

spooler_log() is usually used for this property.

Returned value:
Log

3.17.28 log_dir

Protocol directory

Syntax: string spooler. **log_dir**

The directory in which the JobScheduler writes log files.

log_dir= *stderr allows the JobScheduler to write log files to the standard output (stderr, normally the screen) .

A task executed on a remote JobScheduler (<process class remote_scheduler="">) returns the value for the remote Scheduler.

The -log-dir option has precedence over this parameter.

The factory.ini (section [spooler], entry log_dir=…) setting is overwritten by this parameter.

A task executed on a remote JobScheduler (<process class remote_scheduler="">) returns the value for the remote Scheduler.

3.17.29 param

The command line option `-param=`

Syntax: `string` spooler. **`param`**

Free text. This parameter can be read using `spooler.param`.

The `-param` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `param=...`) setting is overwritten by this parameter.

3.17.30 process_classes

Returns the process classes

Syntax: `Process_classes` spooler. **`process_classes`**

Returned value:
`Process_classes`

3.17.31 schedule

Returns the `Schedule` with the name specified or `null`

Syntax: `Schedule` spooler. **`schedule`** `( string path )`

Returned value:
`Schedule`

3.17.32 supervisor_client

Returns the Supervisor_client or `null`

Syntax: `Supervisor_client` spooler. **`supervisor_client`**

Returned value:
`Supervisor_client`

3.17.33 tcp_port

Port for HTTP and TCP commands for the JobScheduler

Syntax: `int` spooler. **`tcp_port`**

The JobScheduler can accept commands via a TCP port whilst it is running. The number of this port is set here - depending on the operating system - with a number between 2048 and 65535. The default value is 4444.

The JobScheduler operates a HTTP/HTML server on the same port, enabling it to be reached using a web browser - e.g. via http://localhost:4444.

The JobScheduler does not respond to the `tcp_port=0` default setting either with TCP or HTTP protocols. This setting can therefore be used to block a JobScheduler from being accessed - for example via TCP.

The `-tcp-port` option has precedence over this parameter.

`<config tcp_port="…">`

Returned value:
`int`

0, when no port is open.

3.17.34 terminate

The proper ending of the JobScheduler and all related tasks

Syntax: spooler. **`terminate`** `( int timeout (optional) , boolean restart (optional) , boolean all_schedulers (optional) , boolean continue_exclusive_operation (optional) )`

Ends all tasks (by calling the `spooler_close()`) method and terminates the JobScheduler.

Should a time limit be specified, then the JobScheduler ends all processes still running after this limit has expired. (Typical processes are tasks which have remained too long in a method call such as `spooler_process()`.)

See `<modify_spooler cmd="terminate">` and JobScheduler Documentation.

Parameters:

`timeout`	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.
`restart`	`restart=true` allows the JobScheduler to restart after ending.
`all_schedulers`	`all_schedulers=true` ends all the JobSchedulers belonging to a cluster (see `-exclusive`). This may take a minute.
`continue_exclusive_operation`	`continue_exclusive_operation=true` causes another JobScheduler in the Cluster to take become active (see `-exclusive`).

3.17.35 terminate_and_restart

Correctly terminates the JobScheduler and all tasks before restarting

Syntax: spooler. **terminate_and_restart** (int timeout (optional))

Similar to the Spooler.terminate() method, but the JobScheduler restarts itself.

See <modify_spooler cmd="terminate_and_restart"> and JobScheduler Documentation.

Parameters:

timeout	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.

3.17.36 udp_port

Port for UDP commands for the JobScheduler

Syntax: int spooler. **udp_port**

The JobScheduler can also accept UDP commands addressed to the port specified in this setting. Note that a UDP command must fit in a message and that the JobScheduler does not answer UDP commands.

The default value of udp_port=0 does not allow the JobScheduler to open a UDP port.

The -udp-port option has precedence over this parameter.

<config udp_port="…">

Returned value:
int

0, when no port is open.

3.17.37 variables

The JobScheduler variables as a Variable_set

Syntax: Variable_set spooler. **variables**

The variables can be set in the configuration file using <config>.

Returned value:
Variable_set

3.18 Spooler_program - Debugging Jobs in Java

Starts the JobScheduler using Java, so that jobs written in Java can be debugged (e.g. using Eclipse). See Javadoc for information about the methods.

The JobScheduler is started as a Windows application and not as a console program. Output to `stderr` is lost - standard output is shown in Eclipse. `-log-dir` shows no output.

See JobScheduler Documentation.

Example:

```
C:\>java -Djava.library.path=… -classpath …\sos.spooler.jar sos.spooler.Spooler_program
 configuration.scheduler -log-dir=c:\tmp\scheduler
```

Should the location of the `scheduler.dll` not be specified in `%PATH%` then it may be set using `-Djava.library.path=…`.

3.19 Subprocess

A subprocess is a process which can be started using either `Task.create_subprocess()` or `Subprocess.start()`.

Example: system() - the Simple Execution of a Command

```
exit_code = my_system( "backup /" );

function system( cmd, timeout )
{
    var subprocess = spooler_task.create_subprocess();

    try
    {
        if( timeout )  subprocess.timeout = timeout;
        subprocess.start( cmd );
        subprocess.wait_for_termination();
        return subprocess.exit_code;
    }
    finally
    {
        subprocess.close();
    }
}
```

Example:

```
var subprocess = spooler_task.create_subprocess();

subprocess.environment( "test1" ) = "one";
subprocess.environment( "test2" ) = "two";
subprocess.ignore_error = true;

subprocess.start( "sleep 20" );

spooler_log.info( "pid=" + subprocess.pid );
subprocess.timeout = 10;

spooler_log.info( "wait_for_termination ..." );
var ok = subprocess.wait_for_termination( 10 );
spooler_log.info( "wait_for_termination ok=" + ok );

if( subprocess.terminated )
{
    spooler_log.info( "exit code=" + subprocess.exit_code );
    spooler_log.info( "termination signal=" + subprocess.termination_signal );
}
```

3.19.1 close

Frees system resources

Syntax: subprocess. **`close`** `()`

This method should only be called in language with a garbage collector (Java, JavaScript). In all other cases the task ends immediately.

Should this method have been called in a language with a garbage collector, then the `Subprocess` is no longer usable.

3.19.2 env

Environment Variables as Variable_sets

Syntax: `Variable_set` subprocess. **`env`**

Example:

```
var subprocess = spooler_task.create_subprocess();
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
subprocess.wait_for_termination();
```

Returns a `Variable_set` for the environment variables.

Initially the environment is filled by the environment variables from the calling process. Environment variables can be removed in that they are set to `""`. Calling `Subprocess.start()` hands over environment variables to the subprocess.

Note that the names of environment variables are case sensitive on UNIX systems.

Changes made to environment variables after the start of a subprocess have no effect. This is also true for environment variables changed by the process.

This object cannot be handed over to other objects - it is a part of the task process, whereas the majority of other objects are part of the JobScheduler process.

Returned value:
Variable_set_

3.19.3 environment

Environment variables

Syntax: subprocess. **environment** (string name) = string value

Example:

```
// The following two statements have the same effect
subprocess.environment( "my_variable" ) = "my_value"
subprocess.env.value( "my_variable" ) = "my_value"
```

Variables set here are handed over to a new subprocess together with any other environment variables belonging to the process.

Note that the names of environment variables are case sensitive on UNIX systems.

See also Subprocess.env_.

3.19.4 exit_code

Syntax: int subprocess. **exit_code**

Is only called after Subprocess.terminated_== true.

3.19.5 ignore_error

Prevents that a job is stopped, should exit_code != 0.

Syntax: subprocess. **ignore_error** = boolean

Syntax: boolean subprocess. **ignore_error**

Prevents a job from being stopped, when at the end of a task the subprocess ends with Subprocess.exit_code_!= 0.

Should a task not wait for the end of a subprocess with the Subprocess.wait_for_termination_method, then the JobScheduler waits at the end of the task for the end of any subprocesses. In this case the job is stopped with an error when a subprocess ends with Subprocess.exit_code_!= 0.

This may be avoided using `ignore_error`.

3.19.6 ignore_signal

Prevents a job from being stopped when the task is stopped with a UNIX signal.

Syntax: subprocess. **`ignore_signal =`** `int`

Syntax: `int` subprocess. **`ignore_signal`**

This property does not work on Windows systems, as this system does not support signals.

3.19.7 kill

Stops a subprocess

Syntax: subprocess. **`kill`** `( int signal (optional) )`

Parameters:

`signal` Only on UNIX systems: The `kill()` signal. 0 is interpreted here as 9 (`SIGKILL`, immediate ending).

3.19.8 own_process_group

Subprocesses as a Process Group

Syntax: subprocess. **`own_process_group =`** `boolean`

Syntax: `boolean` subprocess. **`own_process_group`**

Only available for UNIX systems.

The default setting can be made using `factory.ini` (section `[spooler]`, entry `subprocess.own_process_group=…`).

`own_process_group` allows a subprocess to run in its own process group, by executing the `setpgid(0,0)` system call. When the JobScheduler then stops the subprocess, then it stops the complete process group.

3.19.9 pid

Process identification

Syntax: `int` subprocess. **`pid`**

3.19.10 priority

Process Priority

Syntax: subprocess. **priority** = int

Syntax: int subprocess. **priority**

Example:

```
spooler_task.priority = +5;     // UNIX: reduce the priority a little
```

UNIX: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 "idle", 6 "below_normal", 8 "normal", 10 "above_normal" and 13 "high" (other values are rounded down). See also Task.priority_class.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Subprocess.priority_class. See also Task.priority.

3.19.11 priority_class

Priority Class

Syntax: subprocess. **priority_class** = string

Syntax: string subprocess. **priority_class**

Example:

```
subprocess.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and UNIX Systems:

Priority Class	Windows	UNIX
"idle"	4	16
"below_normal"	6	6
"normal"	8	0
"above_normal"	10	-6
"high"	13	-16

Note that when it is not possible to set a priority for a task - for example, because of inappropriate permissions - then this must not cause an error. On the other hand, an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Subprocess.priority, Task.priority_class and Microsoft® Windows® Scheduling Priorities.

3.19.12 start

Starts the process

Syntax: subprocess. **start** (string| string[] command_line)

Windows immediately detects whether the program cannot be executed. In this case the method returns an error.

On UNIX systems the Subprocess.exit_code property is set to 99. Before this is done, the end of the process must be waited on with Subprocess.wait_for_termination().

Shell operators such as | , && and > are not interpreted. The /bin/sh or c:\windows\system32\cmd.exe programs must be used to do this. (Note that the actual paths will depend on the installation.)

This process is started on UNIX systems using execvp() and with CreateProcess() on Windows systems.

3.19.13 terminated

Syntax: boolean subprocess. **terminated**

Verifies that a process has ended. Should the process in question have ended, then the Subprocess.exit_code and Subprocess.termination_signal classes may be called.

3.19.14 termination_signal

Signal with which a process (only on UNIX systems) ends

Syntax: int subprocess. **termination_signal**

Is only called, after Subprocess.terminated == true.

3.19.15 timeout

Time limit for a subprocess

Syntax: subprocess. **timeout** = double seconds

After the time allowed, the JobScheduler stops the subprocess (UNIX: with SIGKILL).

This time limit does not apply to processes running on remote computers with <process_class remote_scheduler="">.

3.19.16 wait_for_termination

Syntax: subprocess. **wait_for_termination** ()

Syntax: boolean subprocess. **wait_for_termination** (double seconds)

Parameters:

seconds	Waiting time. Should this parameter not be specified, then the call will take place after the subprocess has ended.

Returned value:
boolean

true, after a subprocess has ended.
false, should the subprocess continue beyond the waiting time.

3.20 Supervisor_client

This object is returned by Spooler. supervisor_client_.

Example:

```
var supervisor_hostname = spooler. supervisor_client. hostname;
```

3.20.1 hostname

The name or IPnumber of the host computer on which the suupervising JobScheduler is running

Syntax: string supervisor_client. **hostname**

See also <config supervisor=""> .

3.20.2 tcp_port

the TCP port of the supervisor

Syntax: int supervisor_client. **tcp_port**

See also <config supervisor=""> .

3.21 Task

A task is an instance of a job which is currently running.

A task can either be waiting in a job queue or being carried out.

3.21.1 add_pid

Makes an independent, temporary process known to the JobScheduler

Syntax: spooler_task. **add_pid** (int pid, string| double| int timeout (optional))

This call is used to restrict the time allowed for processes that have been launched by a task. The JobScheduler ends all independent processes still running at the end of a task.

A log entry is made each time the JobScheduler stops a process. This does not affect the state of a task.

The <kill_task> method stops all processes for which the add_pid() method has been called.

A process group ID can be handed over on Unix systems as a negative pid. kill then stops the complete process group.

This time limit does not apply for processes being run on remote computers with <process_class remote_scheduler="">.

3.21.2 call_me_again_when_locks_available

Repeats spooler_open() or spooler_process() as soon as locks become available

Syntax: spooler_task. **call_me_again_when_locks_available** ()

Causes the JobScheduler to repeat a call of spooler_open() or spooler_process(), after an unsuccessful Task.try_hold_lock() or Task.try_hold_lock_non_exclusive() as soon as the locks required are available. The JobScheduler then repeats the call once it holds the locks, so that the first call (i.e. spooler_open()) will be successful.

After this call, true/false values returned by spooler_open() or spooler_process() has no effect. The JobScheduler leaves the state of the Task.order unchanged.

3.21.3 changed_directories

The directory in which the change which started a task occurred

Syntax: string spooler_task. **changed_directories**

See Job.start_when_directory_changed(), Task.trigger_files.

Returned value:
string

Directory names are to be separated using a semicolon.

"", should no change have occurred in a directory.

3.21.4 create_subprocess

Starts a monitored subprocess

Syntax: _Subprocess_ spooler_task. **create_subprocess** (string| string[] filename_and_arguments (optional))

Returned value:
Subprocess_

3.21.5 delay_spooler_process

Delays the next call of spooler_process()

Syntax: spooler_task. **delay_spooler_process** = string|double|int seconds_or_hhmm_ss

Only functions in spooler_process() .

3.21.6 end

Ends a task

Syntax: spooler_task. **end** ()

The JobScheduler no longer calls the spooler_process() method. Instead the spooler_close() method is called.

This method call can be used at the end of a task to trigger sending a task log. See Log_.

3.21.7 error

Sets an error and stops the current job

Syntax: spooler_task. **error** = string

Syntax: _Error_ spooler_task. **error**

This method call returns the last error which has occurred with the current task. Should no error have occurred, an Error_object is returned, with the is_error property set to false.

An error message can also be written in the task log file using Log. error()

Returned value:
string Error_

3.21.8 exit_code

Exit-Code

Syntax: spooler_task. **exit_code** = int

Syntax: int spooler_task. **exit_code**

Example:

```
spooler_log.error( "This call of spooler_log.error() sets the exit code to 1" );
spooler_task.exit_code = 0;   // Reset the exit code
```

The initial exit-code value is 0 - this is changed to 1 should an error occur. Note that an error is defined here as occurring when the JobScheduler writes a line in the task log containing "[ERROR] ":

- calling the Log.error() method;
- setting the Task.error property;
- the script returns an exception.

The job can then set the Task.exit_code property - e.g. in the spooler_on_error() method.

The exit code resulting from an operating system process executing a task is not relevant here and, in contrast to jobs with <process> or <script language="shell">, is not automatically handed over to this property.

The exit code determines the commands to be subsequently carried out. See <job> <commands on_exit_code="" > for more information.

The exit codes have no influence for API jobs on whether or not a job is stopped (a task error message causes jobs to be stopped).

3.21.9 history_field

A field in the task history

Syntax: spooler_task. **history_field** (string name) = var value

Example:

```
spooler_task.history_field( "extra" ) = 4711;
```

The database table (see factory.ini (section [spooler], entry db_history_table=…)) must have a column with this name and have been declared in the factory.ini (section [job], entry history_columns=…) file.

3.21.10 id

The task identifier

Syntax: int spooler_task. **id**

The unique numerical identifier of every task run by a JobScheduler.

3.21.11 job

The job which a task belongs to

Syntax: Job spooler_task. **job**

Returned value:
Job

3.21.12 order

The current order

Syntax: Order spooler_task. **order**

Example:

```
var order = spooler_task.order;

spooler_log.info( "order.id=" + order.id + ", order.title=" + order.title );
```

Returned value:
Order

`null`, should no order exist.

3.21.13 params

The task parameters

Syntax: Variable_set spooler_task. **params**

Example:

```
var value = spooler_task.params.value( "parameter3" );
```

Example:

```
var parameters = spooler_task.params;
if( parameters.count > 0 )  spooler_log.info( "Parameters given" );

var value1 = parameters.value( "parameter1" );
var value2 = parameters.value( "parameter2" );
```

A task can have parameters. These parameters can be set using:

- <params> in the <job> element in the configuration file;
- Job.start() and

- <start_job>.

Returned value:
Variable_set

!= null

3.21.14 priority

Priority of the Current Task

Syntax: spooler_task. **priority** = int

Syntax: int spooler_task. **priority**

Example:

```
spooler_task.priority = +5;    // Unix: reduce the priority a little
```

Unix: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 "idle", 6 "below_normal", 8 "normal", 10 "above_normal" and 13 "high" (other values are rounded down). See also Task.priority_class.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Task.priority_class.

3.21.15 priority_class

Priority Class of the Current Class

Syntax: spooler_task. **priority_class** = string

Syntax: string spooler_task. **priority_class**

Example:

```
spooler_task.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and Unix Systems:

Priority Class	Windows	Unix
"idle"	4	16
"below_normal"	6	6
"normal"	8	0

"above_normal"	10	-6
"high"	13	-16

Note that an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Task. priority_, Subprocess. priority_class_ and Microsoft® Windows® Scheduling Priorities.

3.21.16 remove_pid

The opposite to add_pid()

Syntax: spooler_task. **remove_pid** (int pid)

An error does not occur when the pid has not been added using Task_.

See Task. add_pid() _.

3.21.17 repeat

Restarts a task after the specified time

Syntax: spooler_task. **repeat** = double

(This method actually belongs to the Job_class and has nothing to do with the task currently being processed.)

Should there be no task belonging to the current job running after the time specified has expired, then the JobScheduler starts a new task. Note that the <run_time>_element is considered here, and that the <period repeat="">_attribute may be temporarily ignored.

Job. delay_after_error_has priority, should a task return an error.

3.21.18 stderr_path

The path to the file in which stderr task output is captured

Syntax: string spooler_task. **stderr_path**

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
string

"", should a task not run in a separate <process_classes>_process.

3.21.19 stderr_text

Text written to `stderr` up to this point by the process that was started by the task.

Syntax: `string` spooler_task. **`stderr_text`**

Text in `stderr` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`string`

`""`, should the task not have been started in a separate process <process_classes>.

3.21.20 stdout_path

The path of the file in which `stdout` task output is captured

Syntax: `string` spooler_task. **`stdout_path`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`string`

`""`, should a task not run in a separate <process_classes> process.

3.21.21 stdout_text

Text written to `stdout` up to this point by the process that was started by the task.

Syntax: `string` spooler_task. **`stdout_text`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`string`

`""`, should a task not run in a separate <process_classes> process.

3.21.22 trigger_files

File paths in folders monitored with regex

Syntax: `string` spooler_task. **`trigger_files`**

Returns the file paths from monitored directories (Job.start_when_directory_changed() or <start_when_directory_changed>) at the time a task is started. Only applies to directories for which a regular expression has been defined (`regex`).

The paths are taken from the addresses defined in Job.start_when_directory_changed() or <start_when_directory_changed> and combined with the file names.

The non-API <process> and <script language="shell"> jobs make the content of Task.trigger_files available to the SCHEDULER_TASK_TRIGGER_FILES environment variable.

See Job.start_when_directory_changed() and Task.changed_directories().

Returned value:
string

The file paths are separated by semicolons.

"" otherwise

3.21.23 try_hold_lock

Try to hold a lock

Syntax: boolean spooler_task. **try_hold_lock** (string lock_path)

Example:

```
function spooler_process()
{
    var result = false;

    if( spooler_task.try_hold_lock( "Georgien" )  &&
        spooler_task.try_hold_lock_non_exlusive( "Venezuela" ) )
    {
        // Task is holding the two locks. Insert processing code here.
        result = ...
    }
    else
    {
        spooler_task.call_me_again_when_locks_available();
    }

    return result;
}
```

try_lock_hold() attempts to retain the lock specified (Lock), and can be called in:

- spooler_open(): the lock is held for the task being carried out and will be freed after the task has been completed,
- spooler_process(): the lock is only held for the job step currently being carried out and will be given up after the step has been completed - i.e. after leaving spooler_process().

When the lock is not available and calling this method returns false then the JobScheduler can be instructed to either:

- repeat the spooler_open() or spooler_process() calls as soon as the locks are available using Task.call_me_again_when_locks_available() or
- end spooler_open() or spooler_process() with false, without use of the above-mentioned call, (but with the expected effect),
- throw a SCHEDULER-469 warning. This applies for true, which is interpreted as an error.

See also <lock.use>.

Returned value:
boolean

true, when the task retains the lock.

3.21.24 try_hold_lock_non_exclusive

Tries to acquire a non-exclusive lock

Syntax: boolean spooler_task. **try_hold_lock_non_exclusive** (string lock_path)

The same prerequisites apply as to Task.try_hold_lock().

See <lock.use exclusive="no">.

Returned value:
boolean

true, if the task successfully acquired the lock.

3.21.25 web_service

The Web Service which a task has been allocated to.

Syntax: Web_service spooler_task. **web_service**

This property causes an exception when a task has not been allocated to a Web Service.

See also Task.web_service_or_null.

Returned value:
Web_service

3.21.26 web_service_or_null

The Web Service to which a task has been allocated, or null.

Syntax: Web_service spooler_task. **web_service_or_null**

See also Task.web_service.

Returned value:
Web_service

3.22 Variable_set - A Variable_set may be used to pass parameters

Variable_set is used for the JobScheduler variables and task parameters. A new Variable_set is created using `Spooler.create_variable_set()`.

Variable names are case independent.

The value of a variable is known as a variant in the COM interface (JavaScript, VBScript, Perl). Because variables are usually written in the JobScheduler database, only variant types which can be converted into strings should be used here.

The value of a variable in Java is a string. Therefore, a string value is returned when reading this variable, when it is set as a variant in the COM interface. `Null` and `Empty` are returned as `null`. An error is caused should the value of a variant not be convertible.

3.22.1 count

The number of variables

Syntax: `int` variable_set. **`count`**

3.22.2 merge

Merges with values from another Variable_set

Syntax: variable_set. **`merge`** `(` `Variable_set` `vs` `)`

Variables with the same name are overwritten.

3.22.3 names

The separation of variable names by semicolons

Syntax: `string` variable_set. **`names`**

Example:

```
var variable_set = spooler.create_variable_set();
spooler_log.info( '"' + variable_set.names + '"' );        // ==> ""

variable_set( "variable_1" ) = "edno";
variable_set( "variable_2" ) = "dwa";

spooler_log.info( '"' + variable_set.names + '"' );        // ==>
"variable_1;variable_2"

var names = variable_set.names.split( ";" );
for( var i in names )  spooler_log.info( names[i] + "=" + variable_set( names[i] ) );
```

Returned value:
`string`

All variable names should be separated by semicolons.

3.22.4 set_var

Sets a variable

Syntax: variable_set. **`set_var`** `( string name, var value )`

3.22.5 substitute

Replaces $-Variables in a String

Syntax: `string` variable_set. **`substitute`** `( string sustitution_string )`

Example:

```
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
```

In the example below, the `Subprocess.env` method is used.

References in the string in the form `$` *name* and `${` *name* `}` are replaced by variables.

Returned value:
`string`

The string containing the substituted $ variables.

3.22.6 value

A variable

Syntax: variable_set. **`value`** `( string name )` **=** `var value`

Syntax: `var` variable_set. **`value`** `( string name )`

Parameters:

`name`

`value` `empty`, should a variable not exist.

Returned value:
`var`

`empty`, should a variable not exist.

3.22.7 xml

`Variable_set` as an XML document

Syntax: variable_set. **`xml`** = `string`

Syntax: `string` variable_set. **`xml`**

Example:

```
var variable_set = spooler.create_variable_set();
spooler_log.info( variable_set.xml );   // Liefert <?xml version='1.0'?><
sos.spooler.variable_set/>

variable_set.xml= "<?xml version='1.0'?>" +
                  "<params>" +
                      "<param name='surname' value='Meier'/>" +
                      "<param name='christian name' value='Hans'/>" +
                  "</params>";
spooler_log.info( variable_set.xml );
spooler_log.info( "nachname=" + variable_set.value( "surname" ) );
spooler_log.info( "vorname =" + variable_set.value( "christian name" ) );
```

See <sos.spooler.variable_set>, <params>.

Parameters:

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

Returned value:
`string`

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

3.23 Web_service

See also <web_service>

3.23.1 forward_xslt_stylesheet_path

Path to the forwarding XSLT stylesheets

Syntax: `string` web_service. **`forward_xslt_stylesheet_path`**

See also <web_service forward_xslt_stylesheet="">

3.23.2 name

The Name of the JobScheduler Web Service

Syntax: string web_service. **name**

See also <web_service name="">

3.23.3 params

Freely definable parameters

Syntax: Variable_set web_service. **params**

The Web Services parameters can be set using the <web_service> element.

Returned value:
Variable_set

3.24 Web_service_operation

See also <web_service>

3.24.1 peer_hostname

Peer (Remote) Host Name

Syntax: string web_service_operation. **peer_hostname**

Returned value:
string

"", should it not be possible to determine the name.

3.24.2 peer_ip

Peer (Remote) IP Address

Syntax: string web_service_operation. **peer_ip**

3.24.3 request

Requests

Syntax: Web_service_request web_service_operation. **request**

Returned value:
Web_service_request

3.24.4 response

Answers

Syntax: Web_service_response web_service_operation. **response**

Returned value:
Web_service_response

3.24.5 web_service

Syntax: Web_service web_service_operation. **web_service**

Returned value:
Web_service

3.25 Web_service_request

See Web_service_operation.

3.25.1 binary_content

Payload as a Byte Array (Java only)

Syntax: web_service_request. **binary_content**

This property is only available under Java.

The `( "Content-Type" )header field` is used to inform the client how binary content is to be interpreted (see HTTP/1.1 14.17 Content-Type) and Web_service_request.charset_name).

3.25.2 charset_name

Character Set

Syntax: `string` web_service_request. **`charset_name`**

Example:

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Returns the `charset=` parameter from the `Content-Type:` header entry.

3.25.3 content_type

Content Type (without parameters)

Syntax: `string` web_service_request. **`content_type`**

Returns the `Content-Type:` header entry, without parameters - e.g. `"text/plain"`.

3.25.4 header

Header Entries

Syntax: `string` web_service_request. **`header`** `( string name )`

Example:

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.request.header( "Content-Type" ) );
```

Parameters:

`name` Case is not relevant.

Returned value:
`string`

Returns `""` in event of an unrecognized entry.

3.25.5 string_content

Payload as Text

Syntax: `string` web_service_request. **`string_content`**

The character set to be used is taken from the `charset` parameter in the `headers("Content-Type")` (see HTTP/1.1 14.17 Content-Type). ISO-8859-1 will be used as default, should this parameter not be specified.

The following character sets are recognized:

- ISO-8859-1
- UTF-8 (only on Windows systems and restricted to the ISO-8859-1 characters)

See also `Web_service_request.binary_content`.

3.25.6 url

Uniform Resource Locator

Syntax: `string` web_service_request. **`url`**

`url = "http://" + header( "Host" ) +` *url_path*

3.26 Web_service_response

Note that the `binary_content` property is only available under Java.

See also `<web_service>`

3.26.1 charset_name

Character set

Syntax: `string` web_service_response. **`charset_name`**

Example:

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Reads the `charset=` parameter from the `Content-Type:` header entry.

3.26.2 content_type

Content-Type (without parameters)

Syntax: `string` web_service_response. **`content_type`**

Reads the `Content-Type:` header without any of the other associated parameters such as `charset=`.

3.26.3 header

Header Entries

Syntax: web_service_response. **`header`** `( string name )` **`=`** `string value`

Syntax: `string` web_service_response. **`header`** `( string name )`

Example:

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.response.header( "Content-Type" ) );
```

Parameters:

`value`	`""` is used for unknown entries.
`name`	The case in which entries are written is not relevant here.

Returned value:
`string`

`""` is used for unknown entries.

3.26.4 send

Sends a Reply

Syntax: web_service_response. **`send`** `()`

3.26.5 status_code

HTTP Status Code

Syntax: web_service_response. **`status_code`** **`=`** `int`

The default setting is 200 (OK).

3.26.6 string_content

Text payloads

Syntax: web_service_response. **string_content** = string text

Example:

```
var response = spooler_task.order.web_service_operation.response;
response.content_type = "text/plain";
response.charset_name = "iso-8859-1";
response.string_content = "This is the answer";
response.send();
```

The `header( "Content-Type" )` must first of all contain a `charset` parameter such as:

```
header( "Content-Type" ) = "text/plain; charset=iso-8859-1";
```

Text is coded as specified in the `charset` parameter. ISO-8859-1 will be used as the default value, should this parameter not be specified.

See `Web_service_request.string_content` for the character sets which are allowed.

See `Web_service_response.charset_name`.

3.27 Xslt_stylesheet

An XSLT style sheet contains the instructions for the transformation of an XML document.

The XSLT processor is implemented with libxslt .

3.27.1 apply_xml

Applies a style sheet to an XML document.

Syntax: `string x.` **`apply_xml`** `( string xml )`

3.27.2 close

Frees the style sheet resources

Syntax: `x.` **`close`** `()`

3.27.3 load_file

Loads the style sheet from an XML file

Syntax: x. **load_file** (string path)

3.27.4 load_xml

Loads the style sheet from an XML document

Syntax: x. **load_xml** (string xml)

4 Perl API

The following classes are available for Perl:

4.1 Error

4.1.1 code

The error code

Syntax: `BSTR` $error-> **`code`**

4.1.2 is_error

`true`, should an error have occurred

Syntax: `Boolean` $error-> **`is_error`**

4.1.3 text

The error text (with error code)

Syntax: `BSTR` $error-> **`text`**

4.2 Job

A task can either be waiting in the order queue or be running.

4.2.1 clear_delay_after_error

Resets all delays which have previously been set using `delay_after_error`

Syntax: $spooler_job-> **`clear_delay_after_error`**(

4.2.2 clear_when_directory_changed

Resets directory notification for all directories which have previously been set using `start_when_directory_changed()`

Syntax: $spooler_job-> **clear_when_directory_changed**(

4.2.3 configuration_directory

Directory for the job configuration file should dynamic configuration from hot folders be used

Syntax: `BSTR` $spooler_job-> **configuration_directory**

"", when a job does not come from a configuration directory.

4.2.4 delay_after_error

Delays the restart of a job in case of an error

Syntax: $spooler_job->`LetProperty( 'delay_after_error', int error_steps, double| int| BSTR seconds_or_hhmm_ss )`

Example:

```
$spooler_job->LetProperty( 'delay_after_error',  2,  10 );          # A 10 second delay
 after the 2nd consecutive error
$spooler_job->LetProperty( 'delay_after_error',  5, '00:01' );      # One minute delay
after the 5th consecutive error
$spooler_job->LetProperty( 'delay_after_error', 10, '24:00' );      # A delay of one
day after the 10th consecutive error
$spooler_job->LetProperty( 'delay_after_error', 20, 'STOP' );       # The Job is
stopped after the 20th consecutive error
```

Should a (first) error occur whilst a job is being run, the JobScheduler will restart the job immediately.
However, after between two and four consecutive errors, the JobScheduler will wait 10 seconds before restarting the job;

After between five and nine consecutive errors, the job will be restarted after a delay of one minute; After between ten and nineteen errors, the delay is 24 hours.

The job is stopped after the twentieth consecutive error.

A delay can be specified, should a particular number of errors occur in series. In this case the job will be terminated and then restarted after the time specified.

This method call can be repeated for differing numbers of errors. A different delay can be specified for each new method call.

It is possible to set the value of the `seconds_or_hhmm_ss` parameter to `"STOP"` in order to restrict the number of (unsuccessful) repetitions of a job. The job then is stopped when the number of consecutive errors specified is reached.

A good position for this call is `spooler_init()`.

See `<delay_after_error>`.

Parameters:

error_steps	The number of consecutive errors required to initiate the delay
seconds_or_hhmm_ss	The delay after which the job will be rerun

4.2.5 delay_order_after_setback

Delays after an order is setback

Syntax: $spooler_job->LetProperty('**delay_order_after_setback**', int setback_count, double| int| BSTR seconds_or_hhmm_ss)

Example:

```
    $spooler_job->LetProperty( 'delay_order_after_setback', 1, 60 );     # for the 1st
and 2nd consecutive setbacks of an order:
                    # delay the order 60s.
    $spooler_job->LetProperty( 'delay_order_after_setback', 3, '01:00' );  # After the
3rd consecutive setback of an order,
                    # the order will be delayed an hour.
    $spooler_job->LetProperty( 'max_order_setbacks', 5 );              # The 5th setback
 sets the order to the error state
```

A job can delay an order which is currently being carried out with Order.setback(). The order is then positioned at the rear of the order queue for that job and carried out after the specified time limit.

The number of consecutively occurring setbacks for an order is counted. The delay set after a setback can be changed using delay_order_after_setback in the event of consecutively occurring setbacks.

See
<delay_order_after_setback>,

Order.setback(),

Job.max_order_setbacks,

Job_chain.add_job(),

Job.delay_after_error().

Parameters:

setback_count	The number of consecutive errors and therefore setbacks for a job. The setback delay can be varied according to this parameter.
seconds_or_hhmm_ss	Time limit for the setback of the order. After expiry of the time limit, the order is reprocessed in the same job.

4.2.6 folder_path

The directory in which the job is to be found.

Syntax: BSTR $spooler_job-> **folder_path**

"", when the job does come from the local (<config configuration_directory="">) configuration file.

Returns the job part relative to the live directory. The path is to start with a slash ("/") and all path components are to be separated by slashes.

Examples:

- "/somewhere/excel" will be returned for the c:\scheduler\config\live\somewhere\excel\sample.job.xml job;
- "/" returned for the c:\scheduler\config\live\sample.xml job and
- "" (an empty string) returned for a job outside the live directory.

4.2.7 include_path

Value of the -include-path= option

Syntax: BSTR $spooler_job-> **include_path**

See -include-path.

4.2.8 max_order_setbacks

Limits the number of setbacks for an order

Syntax: $spooler_job->LetProperty('**max_order_setbacks**', int)

An order state is set to "error" (see Job_chain_node.error_state) when it is set back more than the number of times specified here (see Order.setback()).

See Job.delay_order_after_setback and <delay_order_after_setback is_maximum="yes">.

4.2.9 name

The job path beginning without a backslash

Syntax: BSTR $spooler_job-> **name**

See <job name="">.

4.2.10 order_queue

The job order queue

Syntax: Order_queue $spooler_job-> **order_queue**

Example:

```
$spooler_log->info( 'order=' . ( defined $spooler_job->order_queue ? "yes" : "no" ) );
```

Every job order (<job order="yes">) has an order queue. This queue is filled by the job chain to which the job belongs.

See Job_chain.add_order() , and Job_chain.add_job() .

Returned value:
Order_queue

null, should the job have no queue (for <job order="no">).

4.2.11 process_class

The process class

Syntax: Process_class $spooler_job-> **process_class**

See <job process_class="">.

Returned value:
Process_class

4.2.12 remove

Removes a job

Syntax: $spooler_job-> **remove**(

The job is stopped - i.e. current tasks are terminated and no new ones are started. The job will be removed as soon as no more tasks are running.

Tasks queuing are ignored.

When no job task is running, the remove() function deletes the job immediately.

Job orders (<job order="yes">) cannot be removed.

See <modify_job cmd="remove">.

4.2.13 start

Creates a new task and places it in the task queue

Syntax: Task $spooler_job-> **start**(Variable_set variables (optional))

Example:

```
$spooler->job( 'job_a' )->start();

my $parameters = $spooler->create_variable_set();
$parameters->LetProperty( 'var', 'my_parameter', 'my_value' );
$parameters->LetProperty( 'var', 'other_parameter', 'other_value' );
$spooler->job( 'job_a' )->start( $parameters );
```

The parameters are available to the Task.params task. Two parameters are particularly relevant here:

"spooler_task_name"	gives the task a name which then appears in the status display, e.g. in the web interface.
"spooler_start_after"	specifies a time in seconds (real number), after which the task is to start. The JobScheduler <run_time> is ignored in this case.

See Spooler.create_variable_set(), Spooler.job, Variable_set.value.

Returned value:
Task

4.2.14 start_when_directory_changed

Monitors a directory and starts a task should a notification of a change be received

Syntax: $spooler_job-> **start_when_directory_changed**(BSTR directory_path, BSTR filename_pattern (optional))

Example:

```
$spooler_job->start_when_directory_changed( 'c:/tmp' );

# only relevant for files whose names do not end in "~".
$spooler_job->start_when_directory_changed( 'c:/tmp', '^.*[^~]$' );
```

Should there not be a task belonging to this job running and a notification be received that a change in the directory being monitored has occurred (that a file has been added, changed or deleted), then this change can be used to prompt the JobScheduler to start a task if the current time falls within that allowed by the <run_time> parameter.

This method can be called a more than once in order to allow the monitoring of a number of directories. A repeat call can also be made to a directory in order to reactivate monitoring - if, for example, it has not been possible to access the directory.

This method call can be coded in the JobScheduler start script or in the spooler_init() method. In the latter case, the job must have been started at least once in order for the method call to be carried out. The <run_time once="yes"> setting should be used for this.

The job should be regularly <run_time repeat=""> restarted and <delay_after_error> set.

The same setting can be made in the XML configuration using the <start_when_directory_changed> element.

Parameters:

directory_path the address of the directory being monitored

`filename_pattern`	restricts monitoring to files whose names correspond with the regular expression used.

4.2.15 state_text

Free text for the job state

Syntax: $spooler_job->`LetProperty( 'state_text', BSTR )`

Example:

```
$spooler_job->LetProperty( 'state_text', 'Step C succeeded' );
```

The text will be shown in the HTML interface.

4.2.16 title

The job title

Syntax: `BSTR` $spooler_job-> **`title`**

Example:

```
$spooler_log->info( 'Job title=' . $spooler_job->title );
```

See `<job title="">`.

4.2.17 wake

Causes a task to be started

Syntax: $spooler_job-> **`wake`**`(`

Starts a task, should the job have the `pending` or `stopped` states.

See `Job.start()`.

4.3 Job_chain - job chains for order processing

A job chain is a series of jobs (job chain nodes). Orders (`Order`) proceed along these chains.

Every position in a job chain is assigned a state and a job. When an order is added to the job chain, it is enqueued by the JobScheduler according to the state of the order. The job assigned to this position then carries out the order.

Additionally, each position in a job chain has a successor state and an error state. The JobScheduler changes the state of an order after each job in the job chain has been processed. Should the job step return `(spooler_process) true`, then the JobScheduler sets the succeeding state; otherwise it sets the error state. The order then moves to another position in the job chain as defined by the new state. However, this does not apply when the state is changed during execution with `Order.state`.

A job chain is created using `Spooler.create_job_chain()`; it is filled using `Job_chain.add_job()` and `Job_chain.add_end_state()` and finally made available with `Spooler.add_job_chain()`.

Every node is allocated a unique state. Therefore either `Job_chain.add_job()` or `Job_chain.add_end_state()` must be called once for every state.

Example:

```
my $my_job_chain = $spooler->create_job_chain();
$my_job_chain->LetProperty( 'name', 'JobChain' );
$my_job_chain->add_job( 'job_100', 100,  200, 999 );
$my_job_chain->add_job( 'job_200', 200, 1000, 999 );
$my_job_chain->add_end_state( 999 );
$my_job_chain->add_end_state( 1000 );
$spooler->add_job_chain( $my_job_chain );
```

4.3.1 add_end_state

Adds the end state to a job chain

Syntax: $job_chain-> **add_end_state**(Variant state)

This state is not assigned a job. An order that reaches the final state has completed the job chain and will be removed from the chain.

4.3.2 add_job

Adds a job to a job chain

Syntax: $job_chain-> **add_job**(BSTR job_name, Variant input_state, Variant output_state, Variant error_state)

4.3.3 add_or_replace_order

Adds an order to a job chain and replaces any existing order having the same identifier

Syntax: $job_chain-> **add_or_replace_order**(Order order)

Should the job chain already contain an order with the same identifier, then this order will be replaced. More accurately: the original order will be deleted and the new one added to the job chain.

As long as an existing order having the same identifier as the new order is being carried out, both orders will be present. However, the original order will have already been deleted from the job chain and database; it is only available to the current task and will completely disappear after it has been completed.

In this case the JobScheduler will wait until the original order has been completed before starting the new one.

See Job_chain.add_order() and Order.remove_from_job_chain()

4.3.4 add_order

Adds an order to a job chain

Syntax: Order $job_chain-> **add_order**(Order | BSTR order_or_payload)

Should an order already exist on another job chain, then the JobScheduler removes the order from this other chain.

An order is allocated to the job order queue corresponding to its state, and positioned according to its priority.

The job chain must be specified for the JobScheduler using <job_chain> or Spooler.add_job_chain() .

Should an order with the same Order.id already exist in a job chain, then an exception with the error code SCHEDULER-186 is returned. However, see also Job_chain.add_or_replace_order() .

Returned value:
Order

4.3.5 name

The name of a job chain

Syntax: $job_chain->LetProperty('**name**', BSTR)

Syntax: BSTR $job_chain-> **name**

Example:

```
my $job_chain = $spooler->create_job_chain();
$job_chain->LetProperty( 'name', 'JobChain' );
```

4.3.6 node

The job chain nodes with a given state

Syntax: Job_chain_node $job_chain-> **node**(Variant state)

Returned value:
Job_chain_node

4.3.7 order_count

The number of orders in a job chain

Syntax: `int` $job_chain-> **`order_count`**

4.3.8 order_queue

```
= node(state).job().order_queue()
```

Syntax: `Order_queue` $job_chain-> **`order_queue`**`( Variant state )`

Returns the order queue which has a given state.

Returned value:
`Order_queue`

4.3.9 orders_recoverable

Syntax: $job_chain->`LetProperty( 'orders_recoverable', Boolean )`

Syntax: `Boolean` $job_chain-> **`orders_recoverable`**

See `<job_chain orders_recoverable="">`.

4.3.10 remove

Job chain deletion

Syntax: $job_chain-> **`remove`**`(`

Should orders in a job chain still be being processed (in `spooler_process()`) when the chain is to be deleted, then the JobScheduler will wait until the last order has been processed before deleting the chain.

Orders remain in the database. Should a new job chain be added which has the same name as a deleted job chain (`Spooler.add_job_chain()`), then the JobScheduler will reload any orders from the original job chain which have remained in the database. Note however, that the states of the orders in the new job chain should be the same as those in the original chain at the time of its deletion.

4.3.11 title

Syntax: $job_chain->`LetProperty( 'title', BSTR )`

Syntax: `BSTR` $job_chain-> **`title`**

See <job_chain title="">.

4.4 Job_chain_node

A job chain node is assigned a position in a job chain (Job_chain). The following elements make up a job chain node: a state, a job, a successor state and an error state.

A job chain node is created either using Job_chain.add_job() or Job_chain.add_end_state().

4.4.1 action

Stopping or missing out job chain nodes

Syntax: $node->LetProperty(**'action'**, BSTR)

Syntax: BSTR $node-> **action**

Example:

```
my $job_chain_node = $spooler->job_chain( 'my_job_chain' )->node( 100 );
$job_chain_node->LetProperty( 'action', 'next_state' );
```

This option is not possible with distributed job chains.

Possible settings are:

action="process"
: This is the default setting. Orders are carried out.

action="stop"
: Orders are not carried out, they collect in the order queue.

action="next_state"
: Orders are immediately handed over to the next node as specified with next_state.

See also <job_chain_node.modify action="">.

Character string constonants are defined in Java:

- Job_chain_node.ACTION_PROCESS
- Job_chain_node.ACTION_STOP
- Job_chain_node.ACTION_NEXT_STATE

4.4.2 error_node

The next node in a job chain in the event of an error

Syntax: Job_chain_node $node-> **error_node**

Example:

```
my $job_chain_node = $spooler->job_chain( 'Jobchain' )->node( 100 );
     $spooler_log->debug( 'error state=' . $job_chain_node->error_node->state );
// "state=999"
```

Returned value:
Job_chain_node_

`null`, in the event of no error node being defined (the error state has not been specified)

4.4.3 error_state

State of a job chain in event of an error

Syntax: `Variant` $node-> **`error_state`**

Example:

```
my $job_chain_node = $spooler->job_chain( 'Jobchain' )->node( 100 );
$spooler_log->debug( 'error state=' . $job_chain_node->error_node->state );       #
"error state=999"
```

4.4.4 job

The job allocated to a node

Syntax: _Job_ $node-> **`job`**

Example:

```
my $job_chain_node = $spooler->job_chain( 'Jobchain' )->node( 100 );
$spooler_log->debug( 'job=' . $job_chain_node->job->name );                       #
"job=job_100"
```

Returned value:
Job_

4.4.5 next_node

Returns the next node or null if the current node is assigned the final state.

Syntax: _Job_chain_node_ $node-> **`next_node`**

Returned value:
Job_chain_node_

4.4.6 next_state

The order state in a job chain after successful completion of a job

Syntax: `Variant` $node-> **`next_state`**

Example:

```
my $job_chain_node = $spooler->job_chain( 'Jobchain' )->node( 100 );
$spooler_log->debug( 'next_state=' . $job_chain_node->next_state );              #
"state=200"
```

4.4.7 state

The valid state for a job chain node

Syntax: `Variant` $node-> **`state`**

Example:

```
my $job_chain_node = $spooler->job_chain( 'Jobchain' )->node( 100 );
$spooler_log->info( 'state=' . $job_chain_node->state );                        #
"state=100"
```

4.5 Job_impl - Super Class for a Job or the JobScheduler Script

Job methods are called in the following order:

```
spooler_init()
    spooler_open()
        spooler_process()
        spooler_process()
        ...
    spooler_close()
    spooler_on_success() or spooler_on_error()

spooler_exit()
```

None of these methods must be implemented. However, it is usual that at least the `spooler_process()` method is implemented.

An error during carrying out a job script whilst loading or during `spooler_init()` causes `spooler_on_error()`. to be called. The job is then stopped and `spooler_exit()` called (although `spooler_init()` has not been called!). The script is then unloaded.

Note that spooler_on_error() must also be able to handle errors which occur during loading or in spooler_init().

Note also that spooler_exit() is called even though spooler_init() has not been called.

4.5.1 spooler

The JobScheduler base object

Syntax: _Spooler_ **spooler**

Example:

```
$spooler_log->debug( 'The working directory of the JobScheduler is ' . $spooler->
directory );
```

Returned value:
Spooler_

4.5.2 spooler_close

Task end

Syntax: **spooler_close**(

This method is called after a job has been completed. The opposite of this method is spooler_open().

4.5.3 spooler_exit

Destructor

Syntax: **spooler_exit**(

Is called as the last method before the script is unloaded. This method can be used, for example, to close a database connection.

4.5.4 spooler_init

Initialization

Syntax: Boolean **spooler_init**(

The JobScheduler calls these methods once before spooler_open(). This is analog to spooler_exit(). This method is suitable for initializing purposes (e.g. connecting to a database).

Returned value:

`Boolean`

`false` ends a task. The JobScheduler continues using the `spooler_exit()` method. When the task is processing an order, then this return value makes the JobScheduler terminate the job with an error. That is, unless a repeated start interval has been set using `Job.delay_after_error`

4.5.5 spooler_job

The job object

Syntax: `Job` **`spooler_job`**

Example:

```
$spooler_log->info( 'The name of this job is ' . $spooler_job->name );
```

Returned value:
`Job`

4.5.6 spooler_log

Event logging object

Syntax: `Log` **`spooler_log`**

Example:

```
$spooler_log->info( 'Something has happened' );
```

Returned value:
`Log`

4.5.7 spooler_on_error

Unsuccessful completion of a job

Syntax: **`spooler_on_error`**(

Is called at the end of a job after an error has occurred (after `spooler_close()` but before `spooler_exit()`).

4.5.8 spooler_on_success

Successful completion of a job

Syntax: **spooler_on_success**(

This method is called by the JobScheduler after spooler_close() and before spooler_exit(); should no error have occurred.

4.5.9 spooler_open

The Start of a Task

Syntax: Boolean **spooler_open**(

This method is called immediately after spooler_init(). The opposite of this method is spooler_close().

4.5.10 spooler_process

Job steps or the processing of an order

Syntax: Boolean **spooler_process**(

Processes a job step.

An order driven job stores the current order in Task.order.

The default implementation returns false. The implementation of an order driven job can set the successor state for an order by returning true.

Returned value:
Boolean

In the event of standard jobs <job order="no">: false the JobScheduler ends processing of this job; true> the JobScheduler continues calling the spooler_process() method.

In the event of order driven jobs <job order="yes">: false the order acquires the error state (s. Job_chain_node and <job_chain_node>). true the order acquires the next state or is terminated if the next state is the final state. This, however, does not apply when the state is changed during execution using Order.state.

4.5.11 spooler_task

The task object

Syntax: Task **spooler_task**

Example:

```
$spooler_log->info( 'The task id is ' . $spooler_task->id );
```

Returned value:
Task

4.6 Lock

See also <lock name="">.

Example:

```
my $locks = $spooler->locks;
my $lock = $locks->create_lock();
$lock->LetProperty( 'name', 'my_lock' );
$locks->add_lock( $lock );
```

4.6.1 max_non_exclusive

Limitation of non-exclusive allocation

Syntax: $lock->`LetProperty( 'max_non_exclusive', int )`

Syntax: `int` $lock-> **`max_non_exclusive`**

The default setting is unlimited (231-1), which means that with <lock.use_exclusive="no"> any number of non-exclusive tasks can be started (but only one exclusive task).
The number cannot be smaller than the number of non-exclusive allocations.

See also <lock max_non_exclusive="">.

4.6.2 name

The lock name

Syntax: $lock->`LetProperty( 'name', BSTR )`

Syntax: `BSTR` $lock-> **`name`**

The name can only be set once and cannot be changed.

See also <lock name="">.

4.6.3 remove

Removes a lock

Syntax: $lock-> **`remove`**(

Example:

```
$spooler->locks->lock( 'my_lock' )->remove();
```

A lock can only be removed when it is not active - that is, it has not been allocated to a task and it is not being used by a job (<lock.use>_).

See also <lock.remove>_.

4.7 Locks

4.7.1 add_lock

Adds a lock to a JobScheduler

Syntax: $locks-> **add_lock**(Lock_ lck)

4.7.2 create_lock

Creates a new lock

Syntax: _Lock_ $locks-> **create_lock**(

Returns a new lock Lock_. This lock can be added to the JobScheduler using Locks.add_lock()_.

Returned value:
Lock_

4.7.3 lock

Returns a lock

Syntax: _Lock_ $locks-> **lock**(BSTR lock_name)

An exception will be returned if the lock is unknown.

Returned value:
Lock_

4.7.4 lock_or_null

Returns a lock

Syntax: _Lock_ $locks-> **lock_or_null**(BSTR lock_name)

Returned value:
Lock

null, when the lock is unknown.

4.8 Log - Logging

The spooler_log method can be used in a job or in the JobScheduler start script with the methods described here.
Notification by e-mail

The JobScheduler can send a log file after a task has been completed per e-mail. The following properties define in which cases this should occur.

- Log.mail_on_error,
- Log.mail_on_warning,
- Log.mail_on_process,
- Log.mail_on_success and
- Log.mail_it

Only the end of a task - and not the end of an order - (i.e. spooler_process()) can initiate the sending of e-mails. However, see Task.end() .

The Log.mail method makes the Mail object available, which in turn addresses the mails.

Example:

```
$spooler_log->info( "Something for the Log" );

$spooler_log->LetProperty( 'mail_on_warning', 1 );
$spooler_log->mail->LetProperty( 'from', 'scheduler@company.com' );
$spooler_log->mail->LetProperty( 'to', 'admin@company.com' );
$spooler_log->mail->LetProperty( 'subject', 'ended' );
```

4.8.1 debug

Debug message (level -1)

Syntax: $spooler_log-> **debug**(BSTR line)

4.8.2 debug1

Debug message (level -1)

Syntax: $spooler_log-> **debug1**(BSTR line)

4.8.3 debug2

Debug message (level -2)

Syntax: $spooler_log-> **debug2**(BSTR line)

4.8.4 debug3

Debug message (level -3)

Syntax: $spooler_log-> **debug3**(BSTR line)

4.8.5 debug4

Debug message (level -4)

Syntax: $spooler_log-> **debug4**(BSTR line)

4.8.6 debug5

Debug message (level -5)

Syntax: $spooler_log-> **debug5**(BSTR line)

4.8.7 debug6

Debug message (level -6)

Syntax: $spooler_log-> **debug6**(BSTR line)

4.8.8 debug7

Debug message (level -7)

Syntax: $spooler_log-> **debug7**(BSTR line)

4.8.9 debug8

Debug message (level -8)

Syntax: $spooler_log-> **debug8**(BSTR line)

4.8.10 debug9

Debug message (level -9)

Syntax: $spooler_log-> **debug9**(BSTR line)

4.8.11 error

Error Message (Level 1)

Syntax: $spooler_log-> **error**(BSTR line)

A job stops after a task has ended, should an error message have been written in the task log (spooler_log) and <job stop_on_error="no"> not have been set.

4.8.12 filename

Log file name

Syntax: BSTR $spooler_log-> **filename**

4.8.13 info

Information message (Level 0)

Syntax: $spooler_log-> **info**(BSTR line)

4.8.14 last

The last output with the level specified

Syntax: BSTR $spooler_log-> **last**(int| BSTR level)

4.8.15 last_error_line

The last output line with level 2 (error)

Syntax: `BSTR` $spooler_log-> **`last_error_line`**

4.8.16 level

Limit protocol level

Syntax: $spooler_log->`LetProperty( 'level', int )`

Syntax: `int` $spooler_log-> **`level`**

Defines the level with which protocol entries should be written. Every protocol entry is given one of the following categories: `error`, `warn`, `info`, `debug1` to `debug9` (`debug1` is the same as `debug`).

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

The `-log-level` option has precedence over this parameter.

The `factory.ini` (section `[job]`, entry `log_level=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_level=…`) setting is overwritten by this parameter.

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

4.8.17 log

Writes in the log file with the specified level.

Syntax: $spooler_log-> **`log`**`( int level, BSTR line )`

4.8.18 log_file

Adds the content of a file to the log file

Syntax: $spooler_log-> **log_file**(BSTR path)

Log the content of a file with level 0 (info). An error occurring whilst accessing the file is logged as a warning.

Note that when executed on a remote computer with <process class remote scheduler=""> the file is read from the JobScheduler's file system and not that of the task.

4.8.19 mail

E-mail settings are made in the Mail Object

Syntax: $spooler_log->LetProperty('**mail**', Mail)

Syntax: Mail $spooler_log-> **mail**

Returned value:
Mail

4.8.20 mail_it

Force dispatch

Syntax: $spooler_log->LetProperty('**mail_it**', Boolean)

If this property is set to true, then a log will be sent after a task has ended, independently of the following settings:

Log.mail_on_error, Log.mail_on_warning, Log.mail_on_success, Log.mail_on_process and Log.mail_on_error.

4.8.21 mail_on_error

Sends an e-mail should a job error occur. Errors are caused by the Log.error() method or by any exceptions that have not been caught by a job.

Syntax: $spooler_log->LetProperty('**mail_on_error**', Boolean)

Syntax: Boolean $spooler_log-> **mail_on_error**

Content of the e-mail is the error message. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_error=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_error=…) setting is overwritten by this parameter.

Content of the e-mail is the error message. The log file is sent as an attachment.

4.8.22 mail_on_process

Sends an e-mail should a job have successfully processed the number of steps specified. Steps are caused by the spooler_process() methods:

Syntax: $spooler_log->LetProperty(**'mail_on_process'**, int)

Syntax: int $spooler_log-> **mail_on_process**

Causes the task log to be sent when a task has completed at least the specified number of steps - i.e. calls of spooler_process(). Because non-API tasks do not have steps, the JobScheduler counts each task as a single step.

Content of the e-mail is the success message. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_process=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_process=…) setting is overwritten by this parameter.

Content of the e-mail is the success message. The log file is sent as an attachment.

4.8.23 mail_on_success

Sends an e-mail should a job terminate successfully.

Syntax: $spooler_log->LetProperty(**'mail_on_success'**, Boolean)

Syntax: Boolean $spooler_log-> **mail_on_success**

The success message forms the content of the e-mail. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_success=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_success=…) setting is overwritten by this parameter.

The success message forms the content of the e-mail. The log file is sent as an attachment.

4.8.24 mail_on_warning

Sends an e-mail should a job warning occur. Warnings are caused by the Log. warn( ) method.

Syntax: $spooler_log->LetProperty(**'mail_on_warning'**, Boolean)

Syntax: Boolean $spooler_log-> **mail_on_warning**

The warning forms the content of the e-mail. The log file is sent as an attachment.

The factory. ini (section [spooler], entry mail_on_warning=…) setting is overwritten by this parameter.

The warning forms the content of the e-mail. The log file is sent as an attachment.

4.8.25 new_filename

A new name for the log file

Syntax: $spooler_log->LetProperty(**'new_filename'**, BSTR)

Syntax: BSTR $spooler_log-> **new_filename**

Sets the name of the log file. The JobScheduler copies a log into this file after a log has been made. This file is then available to other applications.

4.8.26 start_new_file

Only for the main log file: closes the current log file and starts a new one

Syntax: $spooler_log-> **start_new_file**(

4.8.27 warn

Warning (Level 2)

Syntax: $spooler_log-> **warn**(BSTR line)

4.9 Mail - e-mail dispatch

See Log. mail .

4.9.1 add_file

Adds an attachment

Syntax: $mail-> **add_file**(BSTR path, BSTR filename_for_mail (optional) , BSTR content_type (optional) , BSTR encoding (optional))

Example:

```
$spooler_log->mail->add_file( 'c:/tmp/1.txt', '1.txt', 'text/plain', 'quoted-printable' );
```

Parameters:

path	path to the file to be appended
filename_for_mail	The file name to appear in the message
content_type	"text/plain" is the preset value.
encoding	e.g. "quoted printable"

4.9.2 add_header_field

Adds a field to the e-mail header

Syntax: $mail-> **add_header_field**(BSTR field_name, BSTR value)

4.9.3 bcc

Invisible recipient of a copy of a mail, (*blind carbon copy*)

Syntax: $mail->LetProperty(**'bcc'**, BSTR)

Syntax: BSTR $mail-> **bcc**

Example:

```
$spooler_log->mail->LetProperty( 'bcc', 'hans@company.com' );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The factory.ini (section [job], entry log_mail_bcc=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_bcc=…) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

4.9.4 body

Message content

Syntax: $mail->`LetProperty( 'body', BSTR )`

Syntax: `BSTR` $mail-> **`body`**

Example:

```
$spooler_log->mail->LetProperty( 'body', 'Job succeeded' );
```

Line feed / carriage return is coded with `\n` (`chr(10)` in VBScript).

4.9.5 cc

Recipient of a copy of a mail, (*carbon copy*)

Syntax: $mail->`LetProperty( 'cc', BSTR )`

Syntax: `BSTR` $mail-> **`cc`**

Example:

```
$spooler_log->mail->LetProperty( 'cc', 'hans@company.com' );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_cc=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_cc=…`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

4.9.6 dequeue

Repeated attempts can be made to send messages from the `queue_dir` directory

Syntax: `int` $mail-> **`dequeue`**(

See `Mail.dequeue_log`, `factory.ini` (section `[spooler]`, entry `mail_queue_dir=…`).

Returned value:
`int`

The number of messages sent

4.9.7 dequeue_log

The `dequeue()` log

Syntax: `BSTR` $mail-> **`dequeue_log`**

Example:

```
my $count = $spooler_log->mail->dequeue();
$spooler_log->info( $count . ' messages from mail queue sent' );
$spooler_log->info( $spooler_log->mail->dequeue_log );
```

See `Mail.dequeue()`.

4.9.8 from

Sender

Syntax: $mail->`LetProperty( 'from', BSTR )`

Syntax: `BSTR` $mail-> **`from`**

Example:

```
$spooler_log->mail->LetProperty( 'from', 'scheduler@company.com' );
```

The `factory.ini` (section `[job]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

4.9.9 queue_dir

The directory used for returned e-mails

Syntax: $mail->LetProperty('**queue_dir**', BSTR path)

Syntax: BSTR $mail-> **queue_dir**

E-mails which cannot be sent (because, for example, the SMTP server cannot be contacted) are stored in this directory.

In order to send these e-mails later it is necessary to write a job which calls up the Mail.dequeue() method.

This setting is generally made in sos.ini (section [mail], entry queue_dir=...).

Environment variables (e.g. $HOME) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The factory.ini (section [job], entry mail_queue_dir=...) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_queue_dir=...) setting is overwritten by this parameter.

The sos.ini (section [mail], entry queue_dir=...) setting is overwritten by this parameter.

4.9.10 smtp

The name of the SMTP server

Syntax: $mail->LetProperty('**smtp**', BSTR)

Syntax: BSTR $mail-> **smtp**

Example:

```
$spooler_log->mail->LetProperty( 'smtp', 'mail.company.com' );
```

These settings are generally made using sos.ini (section [mail], entry smtp=...).

smtp=-queue stops e-mails being sent. Instead mails are written into the file specified in queue_dir. See also sos.ini (section [mail], entry queue_only=...).

The factory.ini (section [job], entry smtp=...) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry smtp=...) setting is overwritten by this parameter.

The sos.ini (section [mail], entry smtp=...) setting is overwritten by this parameter.

4.9.11 subject

Subject, *re*

Syntax: $mail->`LetProperty( 'subject', BSTR )`

Syntax: `BSTR` $mail-> **`subject`**

Example:

```
$spooler_log->mail->LetProperty( 'subject', 'Job succeeded' );
```

The `factory.ini` (section `[job]`, entry `log_mail_subject=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_subject=...`) setting is overwritten by this parameter.

4.9.12 to

Recipient

Syntax: $mail->`LetProperty( 'to', BSTR )`

Syntax: `BSTR` $mail-> **`to`**

Example:

```
$spooler_log->mail->LetProperty( 'to', 'admin@company.com' );
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_to=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_to=...`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

4.9.13 xslt_stylesheet

The XSLT style sheet for e-mail processing. Before sending an e-mail the JobScheduler creates an XML document containing the e-mail headers, subject and body. The content of these elements can be adjusted or overwritten by

an individual XSLT style sheet. This can be used e.g. to create translations of e-mail content. Having processed the XSLT style sheet the JobScheduler sends the resulting content of the XML elements as e-mail.

Syntax: Xslt_stylesheet $mail-> **xslt_stylesheet**

Returned value:
Xslt_stylesheet

The XSLT style sheet as a string

4.9.14 xslt_stylesheet_path

The path and file name of the XSL style sheet for e-mail processing.

Syntax: $mail->LetProperty('**xslt_stylesheet_path**', BSTR path)

Example:

```
$spooler_log->mail->LetProperty( 'xslt_stylesheet_path', 'c:/stylesheets/mail.xslt' );
```

The path to the XSLT style sheet. XSLT style sheets are used by the JobScheduler for the preparation of e-mails. At the time of writing (April 2006) this subject is not documented.

<config mail_xslt_stylesheet="…">

Parameters:

path	The path of the file containing the XSLT style sheet

4.10 Monitor_impl - Using Super Classes for Start Scripts or Jobs

A job can be given a monitor using <monitor>.

A monitor can provide the following methods:

Monitor_impl.spooler_task_before()
: Before starting a task - can prevent a task from being started.

Monitor_impl.spooler_task_after()
: After a task has been completed.

Monitor_impl.spooler_process_before()
: Before spooler_process() - this method can stop spooler_process() from being called.

Monitor_impl.spooler_process_after()
: After spooler_process() - can be used to change its return value.

4.10.1 spooler

The JobScheduler Object

Syntax: _Spooler_ **spooler**

Example:

```
$spooler_log->debug( 'The working directory of the JobScheduler is ' . $spooler->
directory );
```

Is the same object as spooler in the Job_impl class.

Returned value:
Spooler_

4.10.2 spooler_job

The Job Object

Syntax: _Job_ **spooler_job**

Example:

```
$spooler_log->info( 'The name of this job is ' . $spooler_job->name );
```

Is the same object as spooler_job in the Job_impl class.

Returned value:
Job_

4.10.3 spooler_log

Writing Log Files

Syntax: _Log_ **spooler_log**

Example:

```
$spooler_log->info( 'Something has happened' );
```

Is the same object as spooler_log in the Job_impl class.

Returned value:
Log_

4.10.4 spooler_process_after

After `spooler_process()`

Syntax: `Boolean` **`spooler_process_after`**`( Boolean spooler_process_result )`

Example: in java

```
public boolean spooler_task_after( boolean spooler_process_result )  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    spooler_log.info( "spooler_process() didn't throw an exception and delivered " +
spooler_process_result );
    return spooler_process_result;   // Unchanged result
}
```

The JobScheduler calls this method after `spooler_process()` has been carried out.

Parameters:

`spooler_process_result`	The return value from the `spooler_process()` is set to `false`, should `spooler_process()` have ended with an exception.

Returned value:
`Boolean`

Replaces the return value from the `spooler_process()` method or `false`, should `spooler_process()` have ended with an error.

4.10.5 spooler_process_before

Before `spooler_process()`

Syntax: `Boolean` **`spooler_process_before`**`(`

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    spooler_log.info( "SPOOLER_PROCESS_BEFORE()" );
    return true;   // spooler_process() will be executed
}
```

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    boolean continue_with_spooler_process = true;

    if( !are_needed_ressources_available() )
    {
        spooler_task.order().setback();
        continue_with_spooler_process = false;
    }

    return continue_with_spooler_process;
}
```

This method is called by the JobScheduler before each call of spooler_process() .

Returned value:
Boolean

false prevents further calls to spooler_process() . The JobScheduler continues as though false had been returned by spooler_process() false.

4.10.6 spooler_task

The Task Object

Syntax: Task **spooler_task**

Example:

```
$spooler_log->info( 'The task id is ' . $spooler_task->id );
```

Is the same object as spooler_task in the Job_impl class.

Returned value:
Task

4.10.7 spooler_task_after

After Completing a Task

Syntax: **spooler_task_after**(

Example: in java

```
public void spooler_task_after()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_AFTER()" );
}
```

This method is called by the JobScheduler after a task has been completed.

4.10.8 spooler_task_before

Before Starting a Task

Syntax: `Boolean` **`spooler_task_before`**(

Example: in java

```
public boolean spooler_task_before()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    return true;    // Task will be started
  //return false;   // Task will not be started
}
```

This method is called by the JobScheduler before a task is loaded.

Returned value:
`Boolean`

`false` does not allow a task to start and `Monitor_impl.spooler_task_after()` will not be called.

4.11 Order - Order

See JobScheduler Documentation, `Spooler.create_order()`, `Job_chain.add_order()`, `Task.order`.
File order

A file order is an order with for which the `scheduler_file_path` parameter has been set: `Order.params`. `Variable_set.value()`.

See JobScheduler Documentation.

Example: An Order with a simple Payload

```
# Create order:
{
    my $order = $spooler->create_order();
    $order->LetProperty( 'id'         , 1234 );
    $order->LetProperty( 'title'      , 'This is my order' );
    $order->LetProperty( 'state_text', 'This is my state text' );
    $order->LetProperty( 'payload'    , 'This is my payload' );
    $spooler->job_chain( 'my_job_chain' )->add_order( $order );
}
  ...

# Process order:
sub spooler_process()
{
    my $order = $spooler_task->order;
    $spooler_log->info( 'order.payload=' + $order->payload );
    return 1;
}
```

Example: Creating an Order with a Variable_set as a Payload

```
# Create order:
{
    my $variable_set = $spooler->create_variable_set();
    $variable_set->set_var( 'param_one', 111 );
    $variable_set->set_var( 'param_two', 222 );

    my $order = $spooler->create_order();
    $order->LetProperty( 'id'      , 1234 );
    $order->LetProperty( 'payload', $variable_set );
    $spooler->job_chain( 'my_job_chain' )->add_order( $order );
}
...

# Process order:
sub spooler_process()
{
    my $order = $spooler_task->order;
    my $variable_set = $order->payload;
    $spooler_log->info( 'param_one=' . $variable_set->value( 'param_one' ) );
    $spooler_log->info( 'param_two=' . $variable_set->value( 'param_two' ) );
    return 1;
}
```

4.11.1 at

The order start time

Syntax: $order->`LetProperty( 'at', BSTR| DATE )`

Example:

```
$order->LetProperty( 'at', "now+60" );
$spooler->job_chain( "my_job_chain" )->add_order( $order );
```

Used to set the start time before an order is added to an order queue. The following can be specified as a string:

- "now"
- "*yyyy-mm-dd HH:MM[:SS]*"
- "now + *HH:MM[:SS]*"
- "now + *seconds*"

This setting changes start times set by Order.run_time or Order.setback().

See <add_order at="">.

4.11.2 end_state

The state that should be reached when an order has been successfully completed

Syntax: $order->LetProperty(**'end_state'**, Variant)

Syntax: Variant $order-> **end_state**

When an order has its own end_state other than "" then it is considered to be completed after the job allocated to this end state has been completed and before the order otherwise leaves this state (see <job_chain_node> for example to continue to another job which usually comprises a part of the job chain).

The state specified has to reference a valid state of a job node in the job chain.

4.11.3 id

Order Identification

Syntax: $order->LetProperty(**'id'**, Variant)

Syntax: Variant $order-> **id**

Every order has an identifier. This identifier must be unique within a job chain or job order queue. It should also correspond to the data being processed. Normally database record keys are used.

When an id is not set, then the JobScheduler automatically allocates one using Job_chain.add_order().

4.11.4 job_chain

The job chain containing an order

Syntax: Job_chain $order-> **job_chain**

Returned value:
Job_chain

4.11.5 job_chain_node

The job chain nodes which correspond with the order state

Syntax: _Job_chain_node_ $order-> **job_chain_node**

Returned value:
Job_chain_node_

4.11.6 log

Order log

Syntax: _Log_ $order-> **log**

Example:

```
spooler_task.order.log.info( "Only for order log, not for task log" );
spooler_log.info( "For both order log and task log" );
```

Example:

```
$spooler_task->order->log->info( 'Only for order log, not for task log' );
                $spooler_log->info( 'For both order log and task log' );
```

Returned value:
Log_

4.11.7 params

The order parameters

Syntax: $order->LetProperty('**params**', Variable_set)

Syntax: _Variable_set_ $order-> **params**

`params` is held in Order.payload_, the latter cannot, therefore, be used together with `params`.

See <add_order>_.

Returned value:
Variable_set_

4.11.8 payload

Load - an order parameter.

Syntax: $order->LetProperty(**'payload'**, Variable_set | BSTR| int| … payload)

Syntax: Variable_set|BSTR|int|… $order-> **payload**

Instead of this property, the use of Order.params is recommended, which corresponds to (Variable_set)order.payload.

In addition to Order.id which identifies an order, this field can be used for other information.

See Order.params and Order.xml_payload.

Parameters:

payload May be a string or a Variable_set.

Returned value:
Variable_set|BSTR|int|…

May be a string or a Variable_set.

4.11.9 payload_is_type

Checks the payload COM-Type

Syntax: Boolean $order-> **payload_is_type**(BSTR type_name)

Parameters:

type_name "Spooler.Variable_set","Hostware.Dyn_obj" or "Hostware.Record".

4.11.10 priority

Orders with a higher priority are processed first

Syntax: $order->LetProperty(**'priority'**, int)

Syntax: int $order-> **priority**

4.11.11 remove_from_job_chain

Syntax: $order-> **remove_from_job_chain**(

Note that when an order has just been started by a task, then the Order.job_chain property will still return the job chain from which the order has just been removed, using this call, even when "remove_from_job_chain" has been carried out. It is only when the execution has been ended that this method returns null. (other than when the order has just been added to a job chain). This ensures that the job_chain property remains stable whilst a task is being executed.

4.11.12 run_time

<run_time> is used to periodically repeat an order

Syntax: Run_time $order-> **run_time**

Example:

```
$order->run_time->LetProperty( "xml", "<run_time><at at='2006-05-23 11:43:00'/><
/run_time>" );
```

See <run_time>.

The <modify_order at="now"> command causes an order which is waiting because of run_time to start immediately.

Returned value:
Run_time

4.11.13 setback

Delays an order back for a period of time

Syntax: $order-> **setback**(

An order will be delayed and repeated after the period of time specified in either <delay_order_after_setback> or Job.delay_order_after_setback. When the job is repeated, only the spooler_process() job function is repeated. If the order.setback() function is called from spooler_process(), then the retrun value from spooler_process() will have no effect. .

An order counts the number of times this method is called in sequence. This count is then used by <delay_order_after_setback>. It is set to 0, when spooler_process() is completed without <delay_order_after_setback> being called. All counters are set to 0 when the JobScheduler is started.

The <modify_order at="now"> command causes a blocked order to start immediately.

4.11.14 setback_count

How many times the order is setting back?

Syntax: int $order-> **setback_count**

see also <delay_order_after_setback>.

4.11.15 state

The order state

Syntax: $order->LetProperty('state', Variant)

Syntax: Variant $order-> state

When an order is in a job chain, then its state must correspond with one of the states of the job chain.

Whilst an order is being processed by a job the following state, as defined in the job chain (<job_chain_node next_state="">) has no effect. Similarly, the return values from spooler_process() and Monitor_impl.spooler_process_after() are meaningless. This means that with Order.state the following state for a job can be set as required.

An order is added to the job order queue which is corresponding to its state. See <job_chain_node>. The execution by this job will be delayed until the job currently carrying out the order has been completed.

4.11.16 state_text

Free text for the order state

Syntax: $order->LetProperty('state_text', BSTR)

Syntax: BSTR $order-> state_text

This text is shown on the HTML interface.

For non-API jobs the JobScheduler fills this field with the first line from stdout, up to a maximum of 100 characters.

4.11.17 string_next_start_time

The next start time of an order when <run_time> is being used

Syntax: BSTR $order-> string_next_start_time

Returned value:
BSTR

"yyyy-mm-dd HH:MM:SS.MMM" or "now" or "never".

4.11.18 suspended

Suspended order

Syntax: $order->LetProperty('suspended', Boolean)

Syntax: Boolean $order-> suspended

A suspended order will not be executed.

When an order is being carried out by a task when it is suspended, then the spooler_process() step will be completed and the order allocated the successor state before being suspended.

This means that an order can be set to an end state, which stops it from being removed. The JobScheduler can remove such an order only when it is not suspended - i.e. `order.suspended=false`).

A suspended order with the end state can be allocated a different state corresponding to a job node in the job chain. This is effected by using `Order.state`. In this case the order remains suspended.

4.11.19 title

Optionally a title can be allocated to an order that will show up in the HTML interface and in the logs.

Syntax: $order->`LetProperty( 'title', BSTR )`

Syntax: `BSTR` $order-> **`title`**

4.11.20 web_service

The web service to which an order has been allocated

Syntax: `Web_service` $order-> **`web_service`**

When an order has not been allocated to a web service, then this call returns the `SCHEDULER-240` error.

See also `Order.web_service_or_null`.

Returned value:
`Web_service`

4.11.21 web_service_operation

The web service operation to which an order has been allocated

Syntax: `Web_service_operation` $order-> **`web_service_operation`**

Example: in java

```
public boolean spooler_process()  throws Exception
{
    Order                 order                 = spooler_task.order();
    Web_service_operation web_service_operation = order.web_service_operation();
    Web_service_request   request               = web_service_operation.request();

    // Decode request data
    String request_string = new String( request.binary_content(),
request.charset_name() );


    process request_string ...;


    String                response_string = "This is my response";
    String                charset_name    = "UTF-8";
    ByteArrayOutputStream byos            = new ByteArrayOutputStream();

    // Encode response data
    Writer writer = new OutputStreamWriter( byos, charset_name );
    writer.write( response_string );
    writer.close();

    // Respond
    Web_service_response response = web_service_operation.response();

    response.set_content_type( "text/plain" );
    response.set_charset_name( charset_name );
    response.set_binary_content( byos.toByteArray() );
    response.send();

    // Web service operation has finished

    return true;
}
```

See <web_service>, Web_service_operation and Order.web_service_operation_or_null,

Returned value:
Web_service_operation

4.11.22 web_service_operation_or_null

The web service operation to which an order has been allocated, or `null`

Syntax: Web_service_operation $order-> **web_service_operation_or_null**

See Order.web_service_operation, Web_service_operation and <web_service>.

Returned value:
Web_service_operation

4.11.23 web_service_or_null

The web service to which an order has been allocated, or `null`.

Syntax: Web_service_ $order-> **web_service_or_null**

See also Order.web_service_.

Returned value:
Web_service_

4.11.24 xml

Order in XML: <order>...</order>

Syntax: `BSTR` $order-> **xml**

Returned value:
`BSTR`

See <order>

4.11.25 xml_payload

XML payload - an order parameter.

Syntax: $order->`LetProperty( 'xml_payload', BSTR xml )`

Syntax: `BSTR` $order-> **xml_payload**

This property can include an XML document (in addition to the Order.params_property).

<xml_payload>_contains the XML document root element (instead of it being in `#PCDATA` coded form).

4.12 Order_queue - The order queue for an order controlled job

An order controlled job (<job order="yes">_has an order queue, which is filled by the orders to be processed by a job. The orders are sorted according to their priority and the time at which they enter the queue.

Processing means that the JobScheduler calls the spooler_process()_method for a task. This method can access the order using the Task.order_property. Should the `spooler_process()` end without an error (i.e. without any exceptions), then the JobScheduler removes the order from the order queue. If the order is in a job chain then it is moved to the next position in the chain.

4.12.1 length

The number of orders in the order queue

Syntax: `int` $q-> **`length`**

4.13 Process_class

See also `<process_class name="">`.

Example:

```
my $process_classs = $spooler->process_classes;
my $process_class = $process_classs->create_process_class();
$process_class->LetProperty( 'name', 'my_process_class' );
$process_classs->add_process_class( $process_class );
```

4.13.1 max_processes

The maximum number of processes that are executed in parallel

Syntax: $process_class->`LetProperty( 'max_processes', int )`

Syntax: `int` $process_class-> **`max_processes`**

Should more tasks have to be started than allowed by this setting, then these tasks starts would be delayed until processes become freed. The default setting is 10.

See also `<process_class max_processes="">`.

4.13.2 name

The process class name

Syntax: $process_class->`LetProperty( 'name', BSTR )`

Syntax: `BSTR` $process_class-> **`name`**

The name can only be set once and may not be changed.

See also `<process_class name="">`.

4.13.3 remote_scheduler

The address of the remote JobScheduler, which is to execute a process

Syntax: $process_class->LetProperty('**remote_scheduler**', BSTR)

Syntax: BSTR $process_class-> **remote_scheduler**

Example:

```
$spooler->process_classes->process_class( 'my_process_class' )->remote_scheduler(
'host:4444' );
```

See also <process_class remote_scheduler="">.

Parameters:

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

Returned value:
BSTR

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

4.13.4 remove

Removal of the process class

Syntax: $process_class-> **remove**(

Example:

```
$spooler->process_classs->process_class( 'my_process_class' )->remove();
```

The JobScheduler delays deletion of the process class as long as tasks are still running. No new tasks will be started before the class is deleted.

See also <process_class.remove>.

4.14 Process_classes

4.14.1 add_process_class

Adds a process class to the JobScheduler

Syntax: $process_classs-> **add_process_class**(Process_class pc)

4.14.2 create_process_class

Creates a new process class

Syntax: Process_class $process_classs-> **create_process_class**(

Returnds a new Process_class. This class can be made added to the JobScheduler using Process_classes.add_process_class().

Returned value:
Process_class

4.14.3 process_class

Returns a process class

Syntax: Process_class $process_classs-> **process_class**(BSTR process_class_name)

An exception will occur if the process class is not known.

Returned value:
Process_class

4.14.4 process_class_or_null

Returns a process class

Syntax: Process_class $process_classs-> **process_class_or_null**(BSTR process_class_name)

Returned value:
Process_class

null, when the process class is not known.

4.15 Run_time - Managing Time Slots and Starting Times

See <run_time>, Order.Schedule.

Example:

```
my $order = $spooler_task->order;

# Repeat order daily at 15:00
$order->run_time->LetProperty( "xml", "<run_time><period single_start='15:00'/><
/run_time>" );
```

4.15.1 schedule

`<schedule>`

Syntax: _Schedule_ $run_time-> **schedule**

Returned value:
Schedule_

4.15.2 xml

`<run_time>`

Syntax: $run_time->`LetProperty( '`**xml**`', BSTR )`

Discards the current setting and resets `Run_time`.

Parameters:

XML document as a string

4.16 Schedule - Runtime

See <schedule>_, <run_time>_, Spooler.schedule_, Run_time_.

Example:

```
$spooler->schedule( 'my_schedule' )->LetProperty( "xml", "
                                " );
```

4.16.1 xml

`<schedule>`

Syntax: $schedule->`LetProperty( '`**xml**`', BSTR )`

Syntax: `BSTR` $schedule-> **xml**

Deletes the previous setting and resets `Schedule`.

Parameters:

XML document as a string

Returned value:
`BSTR`

XML document as a string

4.17 Spooler

There is only one class for this object: `spooler_`.

4.17.1 abort_immediately

Aborts the JobScheduler immediately

Syntax: $spooler-> **`abort_immediately`**(

Stops the JobScheduler immediately. Jobs do not have the possibility of reacting.

The JobScheduler kills all tasks and the processes that were started using the `Task.create_subprocess()` method. The JobScheduler also kills processes for which a process ID has been stored using the `Task.add_pid()` method.

See `<modify_spooler cmd="abort_immediately">` and JobScheduler Documentation.

4.17.2 abort_immediately_and_restart

Aborts the JobScheduler immediately and then restarts it.

Syntax: $spooler-> **`abort_immediately_and_restart`**(

Similar to the `Spooler.abort_immediately()` method, only that the JobScheduler restarts itself after aborting. It reuses the command line parameters to do this.

See `<modify_spooler cmd="abort_immediately_and_restart">` and JobScheduler Documentation.

4.17.3 add_job_chain

Syntax: $spooler-> **add_job_chain**(Job_chain chain)

Job_chain.orders_recoverable=true causes the JobScheduler to load the orders for a job chain from the database.

See Spooler.create_job_chain() . and <job_chain> .

4.17.4 configuration_directory

Path of the Configuration Directory with hot folders

Syntax: BSTR $spooler-> **configuration_directory**

<config configuration_directory="…">

4.17.5 create_job_chain

Syntax: Job_chain $spooler-> **create_job_chain**(

Returns a new Job_chain object. This job chain can be added to the JobScheduler using Spooler.add_job_chain() after it has been filled with jobs.

See <job_chain> .

Returned value:
Job_chain

4.17.6 create_order

Syntax: Order $spooler-> **create_order**(

Creates a new order. This order can be assigned to a job chain using the Job_chain.add_order() method.

Returned value:
Order

4.17.7 create_variable_set

Syntax: Variable_set $spooler-> **create_variable_set**(

Returned value:

Variable_set

4.17.8 create_xslt_stylesheet

Syntax: Xslt_stylesheet $spooler-> **create_xslt_stylesheet**(BSTR xml (optional))

Parameters:

xml Creates an XSLT style sheet as an XML string.

Returned value:
Xslt_stylesheet

4.17.9 db_history_table_name

The name of the database table used for the job history

Syntax: BSTR $spooler-> **db_history_table_name**

See also Spooler.db_history_table_name()

The factory.ini (section [spooler], entry db_history_table=...) setting is overwritten by this parameter.

4.17.10 db_name

The database path

Syntax: BSTR $spooler-> **db_name**

The database connection string for the history. Should no value be specified here, then the files will be saved in .csv format. See factory.ini (section [spooler], entry history_file=...).

A simple file name ending in .mdb (e.g. scheduler.mdb) can also be specified here when the JobScheduler is running on Windows. The JobScheduler then uses a Microsoft MS Access database of this name, which is located in the protocol directory (see the option -log-dir). Should such a database not exist, then the JobScheduler will create this database.

The JobScheduler automatically creates the tables necessary for this database.

The factory.ini (section [spooler], entry db=...) setting is overwritten by this parameter.

4.17.11 db_order_history_table_name

The name of the order history database table

Syntax: `BSTR` $spooler-> **`db_order_history_table_name`**

See also `Spooler.db_order_history_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_order_history_table=…`) setting is overwritten by this parameter.

4.17.12 db_orders_table_name

The name of the database table used for orders

Syntax: `BSTR` $spooler-> **`db_orders_table_name`**

See also `Spooler.db_orders_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_orders_table=…`) setting is overwritten by this parameter.

4.17.13 db_tasks_table_name

The name of the task database table

Syntax: `BSTR` $spooler-> **`db_tasks_table_name`**

See also `Spooler.db_tasks_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_tasks_table=…`) setting is overwritten by this parameter.

4.17.14 db_variables_table_name

The name of the database table used by the JobScheduler for internal variables

Syntax: `BSTR` $spooler-> **`db_variables_table_name`**

The JobScheduler records internal counters, for example, the ID of the next free task, in this database table.

See also Spooler.db_variables_table_name()

The factory.ini (section [spooler], entry db_variables_table=…) setting is overwritten by this parameter.

4.17.15 directory

The working directory of the JobScheduler on starting

Syntax: BSTR $spooler-> **directory**

Changes the Working Directory.

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

The -cd option has precedence over this parameter.

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

Returned value:
BSTR

The directory ends on Unix with "/" and on Windows with "\".

4.17.16 execute_xml

Carries out XML commands

Syntax: BSTR $spooler-> **execute_xml**(BSTR xml)

Example:

```
$spooler_log->info( $spooler->execute_xml( '<show_state/>' ) );
```

Errors are returned as XML <ERROR> replies.

Parameters:

xml See JobScheduler Documentation.

Returned value:
BSTR

Returns the answer to a command in XML format.

4.17.17 hostname

The name of the computer on which the JobScheduler is running.

Syntax: `BSTR` $spooler-> **`hostname`**

4.17.18 id

The value of the command line `-id=` setting

Syntax: `BSTR` $spooler-> **`id`**

The JobScheduler only selects elements in the XML configuration whose `spooler_id` attributes are either empty or set to the value given here.

When the JobScheduler ID is not specified here, then the JobScheduler ignores the `spooler_id=` XML attribute and selects all the elements in the XML configuration.

See, for example, `<config>`.

The `-id` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `id=…`) setting is overwritten by this parameter.

4.17.19 include_path

Returns the command line setting `-include-path=`.

Syntax: `BSTR` $spooler-> **`include_path`**

The directory of the files which are to be included by the `<include>` element.

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

Environment variables (e.g. `$HOME`) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The `-include-path` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `include_path=…`) setting is overwritten by this parameter.

`<config include_path="…">`

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

4.17.20 ini_path

The value of the -ini= option (the name of the factory.ini file)

Syntax: BSTR $spooler-> **ini_path**

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

See -ini , JobScheduler Documentation

4.17.21 is_service

Syntax: Boolean $spooler-> **is_service**

Returned value:
Boolean

is true, when the JobScheduler is running as a service (on Windows) or as a daemon (on Unix).

4.17.22 job

Returns a job

Syntax: Job $spooler-> **job**(BSTR job_name)

An exception is returned should the job name not be known.

Returned value:
Job

4.17.23 job_chain

Returns a job chain

Syntax: Job_chain $spooler-> **job_chain**(BSTR name)

Should the name of the job chain not be known, then the JobScheduler returns an exception.

Returned value:
Job_chain

4.17.24 job_chain_exists

Syntax: `Boolean` $spooler-> **`job_chain_exists`**`( BSTR name )`

4.17.25 let_run_terminate_and_restart

Syntax: $spooler-> **`let_run_terminate_and_restart`**`(`

The JobScheduler ends all tasks (by calling the `Job_impl` method) as soon as all orders have been completed and then stops itself. It will then be restarted under the same command line parameters.

See `<modify_spooler cmd="let_run_terminate_and_restart">` and JobScheduler Documentation.

4.17.26 locks

Returns the locks

Syntax: `_Locks_` $spooler-> **`locks`**

Returned value:
`Locks_`

4.17.27 log

The main log

Syntax: `_Log_` $spooler-> **`log`**

`spooler_log()` is usually used for this property.

Returned value:
`Log_`

4.17.28 log_dir

Protocol directory

Syntax: `BSTR` $spooler-> **`log_dir`**

The directory in which the JobScheduler writes log files.

`log_dir= *stderr` allows the JobScheduler to write log files to the standard output (`stderr`, normally the screen) .

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

The -log-dir option has precedence over this parameter.

The factory.ini (section [spooler], entry log_dir=…) setting is overwritten by this parameter.

A task executed on a remote JobScheduler (<process_class remote_scheduler="">) returns the value for the remote Scheduler.

4.17.29 param

The command line option -param=

Syntax: BSTR $spooler-> **param**

Free text. This parameter can be read using spooler.param.

The -param option has precedence over this parameter.

The factory.ini (section [spooler], entry param=…) setting is overwritten by this parameter.

4.17.30 process_classes

Returns the process classes

Syntax: Process_classes $spooler-> **process_classes**

Returned value:
Process_classes

4.17.31 schedule

Returns the Schedule with the name specified or null

Syntax: Schedule $spooler-> **schedule**(BSTR path)

Returned value:
Schedule

4.17.32 supervisor_client

Returns the Supervisor_client or null

Syntax: Supervisor_client $spooler-> **supervisor_client**

Returned value:
Supervisor_client

4.17.33 tcp_port

Port for HTTP and TCP commands for the JobScheduler

Syntax: `int` $spooler-> **tcp_port**

The JobScheduler can accept commands via a TCP port whilst it is running. The number of this port is set here - depending on the operating system - with a number between 2048 and 65535. The default value is 4444.

The JobScheduler operates a HTTP/HTML server on the same port, enabling it to be reached using a web browser - e.g. via http://localhost:4444.

The JobScheduler does not respond to the `tcp_port=0` default setting either with TCP or HTTP protocols. This setting can therefore be used to block a JobScheduler from being accessed - for example via TCP.

The -tcp-port option has precedence over this parameter.

<config tcp_port="…">

Returned value:
`int`

0, when no port is open.

4.17.34 terminate

The proper ending of the JobScheduler and all related tasks

Syntax: $spooler-> **terminate**`( int timeout (optional) , Boolean restart (optional) , boolean all_schedulers (optional) , boolean continue_exclusive_operation (optional) )`

Ends all tasks (by calling the spooler_close()) method and terminates the JobScheduler.

Should a time limit be specified, then the JobScheduler ends all processes still running after this limit has expired. (Typical processes are tasks which have remained too long in a method call such as spooler_process() .)

See <modify_spooler cmd="terminate"> and JobScheduler Documentation.

Parameters:

`timeout`	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.
`restart`	`restart=true` allows the JobScheduler to restart after ending.

all_schedulers	all_schedulers=true ends all the JobSchedulers belonging to a cluster (see -exclusive). This may take a minute.
continue_exclusive_operation	continue_exclusive_operation=true causes another JobScheduler in the Cluster to take become active (see -exclusive).

4.17.35 terminate_and_restart

Correctly terminates the JobScheduler and all tasks before restarting

Syntax: $spooler-> **terminate_and_restart**(int timeout (optional))

Similar to the Spooler.terminate() method, but the JobScheduler restarts itself.

See <modify_spooler cmd="terminate_and_restart"> and JobScheduler Documentation.

Parameters:

timeout	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.

4.17.36 udp_port

Port for UDP commands for the JobScheduler

Syntax: int $spooler-> **udp_port**

The JobScheduler can also accept UDP commands addressed to the port specified in this setting. Note that a UDP command must fit in a message and that the JobScheduler does not answer UDP commands.

The default value of udp_port=0 does not allow the JobScheduler to open a UDP port.

The -udp-port option has precedence over this parameter.

<config udp_port="…">

Returned value:
int

0, when no port is open.

4.17.37 var

Allows access to variables defined in the JobScheduler start script

Syntax: $spooler->`LetProperty( 'var', BSTR name, Variant )`

Syntax: `Variant` $spooler-> **`var`**`( BSTR name )`

The variables are used by all JobScheduler job implementations.

4.17.38 variables

The JobScheduler variables as a `Variable_set`

Syntax: `Variable_set` $spooler-> **`variables`**

The variables can be set in the configuration file using `<config>`.

Returned value:
`Variable_set`

4.18 Spooler_program - Debugging Jobs in Java

Starts the JobScheduler using Java, so that jobs written in Java can be debugged (e.g. using Eclipse). See Javadoc for information about the methods.

The JobScheduler is started as a Windows application and not as a console program. Output to `stderr` is lost - standard output is shown in Eclipse. `-log-dir` shows no output.

See JobScheduler Documentation.

Example:

```
C:\>java -Djava.library.path=… -classpath …\sos.spooler.jar sos.spooler.Spooler_program
 configuration.scheduler -log-dir=c:\tmp\scheduler
```

Should the location of the `scheduler.dll` not be specified in `%PATH%` then it may be set using `-Djava.library.path=…`.

4.19 Subprocess

A subprocess is a process which can be started using either `Task.create_subprocess()` or `Subprocess.start()`.

Example: my_system() - the Simple Execution of a Command

```
sub my_system
{
    # Executes the command without processing the shell characters

    my $cmd     = shift;
    my $timeout = shift;

    my $subprocess = $spooler_task->create_subprocess();

    $subprocess->LetProperty( "timeout", $timeout )  if defined $timeout;
    $subprocess->start( $cmd );
    $subprocess->wait_for_termination();
    return $subprocess->exit_code;
}


sub shell
{
    # Executes the command under the shell ( UNIX only)

    my $cmd     = shift;
    my $timeout = shift;
    my $subprocess = $spooler_task->create_subprocess();

    $subprocess->LetProperty( "timeout", $timeout )  if defined $timeout;
    $subprocess->start( [ "/bin/sh", "-c", $cmd ] );
    $subprocess->wait_for_termination();
    return $subprocess->exit_code;
}
```

Example:

```
my $subprocess = $spooler_task->create_subprocess();

$subprocess->environment->LetProperty( "test1", "one" );
$subprocess->environment->LetProperty( "test2", "two" );
$subprocess->environment->LetProperty( "ignore_error"; true );

$subprocess->start( "sleep 20" );

$spooler_log->info( "pid=" . $subprocess->pid );
$subprocess->timeout( 10 );

$spooler_log->info( "wait_for_termination ..." );
my #ok = $subprocess->wait_for_termination( 10 );
$spooler_log->info( "wait_for_termination ok=" . ok );

if( $subprocess->terminated )
{
    $spooler_log->info( "exit code=" . $subprocess->exit_code );
    $spooler_log->info( "termination signal=" . $subprocess->termination_signal );
}
```

4.19.1 close

Frees system resources

Syntax: $subprocess-> **close**(

This method should only be called in language with a garbage collector (Java, JavaScript). In all other cases the task ends immediately.

Should this method have been called in a language with a garbage collector, then the `Subprocess` is no longer usable.

4.19.2 env

Environment Variables as Variable_sets

Syntax: _Variable_set_ $subprocess-> **env**

Example:

```
my $subprocess = $spooler_task->create_subprocess();
$subprocess->LetProperty( 'start', $subprocess->env->substitute(
'${MY_HOME}/my_program' ) );
$subprocess->wait_for_termination();
```

Returns a Variable_set for the environment variables.

Initially the environment is filled by the environment variables from the calling process. Environment variables can be removed in that they are set to "". Calling Subprocess.start() hands over environment variables to the subprocess.

Note that the names of environment variables are case sensitive on UNIX systems.

Changes made to environment variables after the start of a subprocess have no effect. This is also true for environment variables changed by the process.

This object cannot be handed over to other objects - it is a part of the task process, whereas the majority of other objects are part of the JobScheduler process.

Returned value:
Variable_set_

4.19.3 environment

Environment variables

Syntax: $subprocess->LetProperty('**environment**', BSTR name, BSTR value)

Example:

```
// The following two statements have the same effect
$subprocess->LetProperty( 'environment', 'my_variable', 'my_value' )
$subprocess->env->LetProperty( 'value', 'my_variable', 'my_value' )
```

Variables set here are handed over to a new subprocess together with any other environment variables belonging to the process.

Note that the names of environment variables are case sensitive on UNIX systems.

See also `Subprocess.env`.

4.19.4 exit_code

Syntax: `int` $subprocess-> **`exit_code`**

Is only called after `Subprocess.terminated` `== true`.

4.19.5 ignore_error

Prevents that a job is stopped, should `exit_code != 0`.

Syntax: $subprocess->`LetProperty( 'ignore_error', Boolean )`

Syntax: `Boolean` $subprocess-> **`ignore_error`**

Prevents a job from being stopped, when at the end of a task the subprocess ends with `Subprocess.exit_code` `!= 0`.

Should a task not wait for the end of a subprocess with the `Subprocess.wait_for_termination` method, then the JobScheduler waits at the end of the task for the end of any subprocesses. In this case the job is stopped with an error when a subprocess ends with `Subprocess.exit_code` `!= 0`.

This may be avoided using `ignore_error`.

4.19.6 ignore_signal

Prevents a job from being stopped when the task is stopped with a UNIX signal.

Syntax: $subprocess->`LetProperty( 'ignore_signal', int )`

Syntax: `int` $subprocess-> **`ignore_signal`**

This property does not work on Windows systems, as this system does not support signals.

4.19.7 kill

Stops a subprocess

Syntax: $subprocess-> **`kill`**`( int signal (optional) )`

Parameters:

`signal` Only on UNIX systems: The `kill()` signal. 0 is interpreted here as 9 (`SIGKILL`, immediate ending).

4.19.8 own_process_group

Subprocesses as a Process Group

Syntax: $subprocess->`LetProperty( '`**`own_process_group`**`', Boolean )`

Syntax: `Boolean` $subprocess-> **`own_process_group`**

Only available for UNIX systems.

The default setting can be made using `factory.ini` (section `[spooler]`, entry `subprocess.own_process_group=…`).

`own_process_group` allows a subprocess to run in its own process group, by executing the `setpgid(0,0)` system call. When the JobScheduler then stops the subprocess, then it stops the complete process group.

4.19.9 pid

Process identification

Syntax: `int` $subprocess-> **`pid`**

4.19.10 priority

Process Priority

Syntax: $subprocess->`LetProperty( '`**`priority`**`', int )`

Syntax: `int` $subprocess-> **`priority`**

Example:

```
$spooler_task->LetProperty( 'priority', +5 );   // UNIX: reduce the priority a little
```

UNIX: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 `"idle"`, 6 `"below_normal"`, 8 `"normal"`, 10 `"above_normal"` and 13 `"high"` (other values are rounded down). See also `Task.priority_class`.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Subprocess.priority_class. See also Task.priority.

4.19.11 priority_class

Priority Class

Syntax: $subprocess->LetProperty('priority_class', BSTR)

Syntax: BSTR $subprocess-> priority_class

Example:

```
$subprocess->LetProperty( 'priority_class', 'below_normal' );
```

The following priority classes can be used to set priorities on Windows and UNIX Systems:

Priority Class	Windows	UNIX
"idle"	4	16
"below_normal"	6	6
"normal"	8	0
"above_normal"	10	-6
"high"	13	-16

Note that when it is not possible to set a priority for a task - for example, because of inappropriate permissions - then this must not cause an error. On the other hand, an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Subprocess.priority, Task.priority_class and Microsoft® Windows® Scheduling Priorities.

4.19.12 start

Starts the process

Syntax: $subprocess-> **start**(BSTR| BSTR command_line)

Windows immediately detects whether the program cannot be executed. In this case the method returns an error.

On UNIX systems the Subprocess.exit_code property is set to 99. Before this is done, the end of the process must be waited on with Subprocess.wait_for_termination().

Shell operators such as | , && and > are not interpreted. The /bin/sh or c:\windows\system32\cmd.exe programs must be used to do this. (Note that the actual paths will depend on the installation.)

This process is started on UNIX systems using execvp() and with CreateProcess() on Windows systems.

4.19.13 terminated

Syntax: `Boolean` $subprocess-> **`terminated`**

Verifies that a process has ended. Should the process in question have ended, then the `Subprocess.exit_code` and `Subprocess.termination_signal` classes may be called.

4.19.14 termination_signal

Signal with which a process (only on UNIX systems) ends

Syntax: `int` $subprocess-> **`termination_signal`**

Is only called, after `Subprocess.terminated == true`.

4.19.15 timeout

Time limit for a subprocess

Syntax: $subprocess->`LetProperty( 'timeout', double seconds )`

After the time allowed, the JobScheduler stops the subprocess (UNIX: with `SIGKILL`).

This time limit does not apply to processes running on remote computers with `<process_class remote_scheduler="">`.

4.19.16 wait_for_termination

Syntax: $subprocess-> **`wait_for_termination`**`(`

Syntax: `Boolean` $subprocess-> **`wait_for_termination`**`( double seconds )`

Parameters:

`seconds`	Waiting time. Should this parameter not be specified, then the call will take place after the subprocess has ended.

Returned value:
`Boolean`

`true`, after a subprocess has ended.
`false`, should the subprocess continue beyond the waiting time.

4.20 Supervisor_client

This object is returned by Spooler.supervisor_client_.

Example:

```
my $supervisor_hostname = $spooler->supervisor_client->hostname;
```

4.20.1 hostname

The name or IPnumber of the host computer on which the suupervising JobScheduler is running

Syntax: BSTR $supervisor_client-> **hostname**

See also <config supervisor="">_.

4.20.2 tcp_port

the TCP port of the supervisor

Syntax: int $supervisor_client-> **tcp_port**

See also <config supervisor="">_.

4.21 Task

A task is an instance of a job which is currently running.

A task can either be waiting in a job queue or being carried out.

4.21.1 add_pid

Makes an independent, temporary process known to the JobScheduler

Syntax: $spooler_task-> **add_pid**(int pid, BSTR| double| int timeout (optional))

This call is used to restrict the time allowed for processes that have been launched by a task. The JobScheduler ends all independent processes still running at the end of a task.

A log entry is made each time the JobScheduler stops a process. This does not affect the state of a task.

The <kill_task>_method stops all processes for which the add_pid() method has been called.

A process group ID can be handed over on Unix systems as a negative pid. kill then stops the complete process group.

This time limit does not apply for processes being run on remote computers with <process_class remote_scheduler="">.

4.21.2 call_me_again_when_locks_available

Repeats spooler_open() or spooler_process() as soon as locks become available

Syntax: $spooler_task-> **call_me_again_when_locks_available**(

Causes the JobScheduler to repeat a call of spooler_open() or spooler_process(), after an unsuccessful Task.try_hold_lock() or Task.try_hold_lock_non_exclusive() as soon as the locks required are available. The JobScheduler then repeats the call once it holds the locks, so that the first call (i.e. spooler_open()) will be successful.

After this call, true/false values returned by spooler_open() or spooler_process() has no effect. The JobScheduler leaves the state of the Task.order unchanged.

4.21.3 changed_directories

The directory in which the change which started a task occurred

Syntax: BSTR $spooler_task-> **changed_directories**

See Job.start_when_directory_changed(), Task.trigger_files.

Returned value:
BSTR

Directory names are to be separated using a semicolon.

"", should no change have occurred in a directory.

4.21.4 create_subprocess

Starts a monitored subprocess

Syntax: __Subprocess__ $spooler_task-> **create_subprocess**(BSTR| BSTR filename_and_arguments (optional))

Returned value:
Subprocess

4.21.5 delay_spooler_process

Delays the next call of spooler_process()

Syntax: $spooler_task->LetProperty(**'delay_spooler_process'**, BSTR| double| int seconds_or_hhmm_ss)

Only functions in spooler_process() .

4.21.6 end

Ends a task

Syntax: $spooler_task-> **end**(

The JobScheduler no longer calls the spooler_process() method. Instead the spooler_close() method is called.

This method call can be used at the end of a task to trigger sending a task log. See Log .

4.21.7 error

Sets an error and stops the current job

Syntax: $spooler_task->LetProperty(**'error'**, BSTR)

Syntax: Error $spooler_task-> **error**

This method call returns the last error which has occurred with the current task. Should no error have occurred, an Error object is returned, with the is_error property set to false.

An error message can also be written in the task log file using Log. error()

Returned value:
BSTR Error

4.21.8 exit_code

Exit-Code

Syntax: $spooler_task->LetProperty(**'exit_code'**, int)

Syntax: int $spooler_task-> **exit_code**

Example:

```
$spooler_log->error( 'This call of spooler_log.error() sets the exit code to 1' );
    $spooler_task->LetProperty( 'exit_code', 0 );   # Reset the exit code
```

The initial exit-code value is 0 - this is changed to 1 should an error occur. Note that an error is defined here as occurring when the JobScheduler writes a line in the task log containing "[ERROR] ":

- calling the Log. error() method;

- setting the Task.error property;
- the script returns an exception.

The job can then set the Task.exit_code property - e.g. in the spooler_on_error() method.

The exit code resulting from an operating system process executing a task is not relevant here and, in contrast to jobs with <process>or <script language="shell">, is not automatically handed over to this property.

The exit code determines the commands to be subsequently carried out. See <job> <commands on_exit_code="">for more information.

The exit codes have no influence for API jobs on whether or not a job is stopped (a task error message causes jobs to be stopped).

4.21.9 history_field

A field in the task history

Syntax: $spooler_task->LetProperty(**'history_field'**, BSTR name, Variant value)

Example:

```
$spooler_task->LetProperty( 'history_field', 'extra', 4711 );
```

The database table (see factory.ini (section [spooler], entry db_history_table=…)) must have a column with this name and have been declared in the factory.ini (section [job], entry history_columns=…) file.

4.21.10 id

The task identifier

Syntax: int $spooler_task-> **id**

The unique numerical identifier of every task run by a JobScheduler.

4.21.11 job

The job which a task belongs to

Syntax: Job $spooler_task-> **job**

Returned value:
Job

4.21.12 order

The current order

Syntax: Order $spooler_task-> **order**

Example:

```
my $order = $spooler_task->order();

    $spooler_log->info( 'order.id=' . $order->id . ', order.title=' . $order->title );
```

Returned value:
Order

null, should no order exist.

4.21.13 params

The task parameters

Syntax: Variable_set $spooler_task-> **params**

Example:

```
my $value = $spooler_task->params->var( "parameter3" );

my $parameters = $spooler_task->params;
my $value1 = $parameters->var( "parameter1" );
my $value2 = $parameters->var( "parameter2" );
```

A task can have parameters. These parameters can be set using:

- <params> in the <job> element in the configuration file;
- Job.start() and
- <start_job>.

Returned value:
Variable_set

!= null

4.21.14 priority

Priority of the Current Task

Syntax: $spooler_task->LetProperty('**priority**', int)

Syntax: int $spooler_task-> **priority**

Example:

```
$spooler_task->LetProperty('priority', +5 );    // Unix: reduce the priority a little
```

Unix: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 `"idle"`, 6 `"below_normal"`, 8 `"normal"`, 10 `"above_normal"` and 13 `"high"` (other values are rounded down). See also `Task. priority_class`.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with `Task. priority_class`.

4.21.15 priority_class

Priority Class of the Current Class

Syntax: $spooler_task->`LetProperty( 'priority_class', BSTR )`

Syntax: `BSTR` $spooler_task-> **`priority_class`**

Example:

```
$spooler_task->LetProperty('priority_class', 'below_normal' );
```

The following priority classes can be used to set priorities on Windows and Unix Systems:

Priority Class	Windows	Unix
`"idle"`	4	16
`"below_normal"`	6	6
`"normal"`	8	0
`"above_normal"`	10	-6
`"high"`	13	-16

Note that an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also `Task. priority`, `Subprocess. priority_class` and Microsoft® Windows® Scheduling Priorities.

4.21.16 remove_pid

The opposite to `add_pid()`

Syntax: $spooler_task-> **`remove_pid`**`( int pid )`

An error does not occur when the pid has not been added using Task.

See Task.add_pid().

4.21.17 repeat

Restarts a task after the specified time

Syntax: $spooler_task->LetProperty('repeat', double)

(This method actually belongs to the Job class and has nothing to do with the task currently being processed.)

Should there be no task belonging to the current job running after the time specified has expired, then the JobScheduler starts a new task. Note that the <run_time> element is considered here, and that the <period repeat=""> attribute may be temporarily ignored.

Job.delay_after_error has priority, should a task return an error.

4.21.18 stderr_path

The path to the file in which stderr task output is captured

Syntax: BSTR $spooler_task-> **stderr_path**

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
BSTR

"", should a task not run in a separate <process_classes> process.

4.21.19 stderr_text

Text written to stderr up to this point by the process that was started by the task.

Syntax: BSTR $spooler_task-> **stderr_text**

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
BSTR

"", should the task not have been started in a separate process <process_classes>.

4.21.20 stdout_path

The path of the file in which `stdout` task output is captured

Syntax: `BSTR` $spooler_task-> **`stdout_path`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`BSTR`

`""`, should a task not run in a separate `<process_classes>` process.

4.21.21 stdout_text

Text written to `stdout` up to this point by the process that was started by the task.

Syntax: `BSTR` $spooler_task-> **`stdout_text`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`BSTR`

`""`, should a task not run in a separate `<process_classes>` process.

4.21.22 trigger_files

File paths in folders monitored with regex

Syntax: `BSTR` $spooler_task-> **`trigger_files`**

Returns the file paths from monitored directories (`Job.start_when_directory_changed()` or `<start_when_directory_changed>`) at the time a task is started. Only applies to directories for which a regular expression has been defined (`regex`).

The paths are taken from the addresses defined in `Job.start_when_directory_changed()` or `<start_when_directory_changed>` and combined with the file names.

The non-API `<process>` and `<script language="shell">` jobs make the content of `Task.trigger_files` available to the `SCHEDULER_TASK_TRIGGER_FILES` environment variable.

See `Job.start_when_directory_changed()` and `Task.changed_directories()`.

Returned value:
`BSTR`

The file paths are separated by semicolons.

`""` otherwise

4.21.23 try_hold_lock

Try to hold a lock

Syntax: `boolean` $spooler_task-> **`try_hold_lock`**`( BSTR lock_path )`

Example: in javascript

```
function spooler_process()
{
    var result = false;

    if( spooler_task.try_hold_lock( "Georgien" )  &&
        spooler_task.try_hold_lock_non_exlusive( "Venezuela" ) )
    {
        // Task is holding the two locks. Insert processing code here.
        result = ...
    }
    else
    {
        spooler_task.call_me_again_when_locks_available();
    }

    return result;
}
```

`try_lock_hold()` attempts to retain the lock specified (`Lock`), and can be called in:

- `spooler_open()`: the lock is held for the task being carried out and will be freed after the task has been completed,
- `spooler_process()`: the lock is only held for the job step currently being carried out and will be given up after the step has been completed - i.e. after leaving `spooler_process()`.

When the lock is not available and calling this method returns `false` then the JobScheduler can be instructed to either:

- repeat the `spooler_open()` or `spooler_process()` calls as soon as the locks are available using `Task.call_me_again_when_locks_available()` or
- end `spooler_open()` or `spooler_process()` with `false`, without use of the above-mentioned call, (but with the expected effect),
- throw a `SCHEDULER-469` warning. This applies for `true`, which is interpreted as an error.

See also `<lock.use>`.

Returned value:
`boolean`

`true`, when the task retains the lock.

4.21.24 try_hold_lock_non_exclusive

Tries to acquire a non-exclusive lock

Syntax: `boolean` $spooler_task-> **`try_hold_lock_non_exclusive`**`( BSTR lock_path )`

The same prerequisites apply as to `Task.try_hold_lock()`.

See <lock.use_exclusive="no">.

Returned value:
boolean

true, if the task successfully acquired the lock.

4.21.25 web_service

The Web Service which a task has been allocated to.

Syntax: Web_service $spooler_task-> **web_service**

This property causes an exception when a task has not been allocated to a Web Service.

See also Task.web_service_or_null.

Returned value:
Web_service

4.21.26 web_service_or_null

The Web Service to which a task has been allocated, or null.

Syntax: Web_service $spooler_task-> **web_service_or_null**

See also Task.web_service.

Returned value:
Web_service

4.22 Variable_set - A Variable_set may be used to pass parameters

Variable_set is used for the JobScheduler variables and task parameters. A new Variable_set is created using Spooler.create_variable_set().

Variable names are case independent.

The value of a variable is known as a variant in the COM interface (JavaScript, VBScript, Perl). Because variables are usually written in the JobScheduler database, only variant types which can be converted into strings should be used here.

The value of a variable in Java is a string. Therefore, a string value is returned when reading this variable, when it is set as a variant in the COM interface. Null and Empty are returned as null. An error is caused should the value of a variant not be convertible.

4.22.1 count

The number of variables

Syntax: `int` $variable_set-> **`count`**

4.22.2 merge

Merges with values from another Variable_set

Syntax: $variable_set-> **`merge`**`( Variable_set vs )`

Variables with the same name are overwritten.

4.22.3 names

The separation of variable names by semicolons

Syntax: `BSTR` $variable_set-> **`names`**

Example:

```
my $variable_set = $spooler->create_variable_set();
   $spooler_log->info( '"' . $variable_set->names . '"' );        # ==> ""

   $variable_set->( "variable_1", "edno");
   $variable_set->( "variable_2", "dwa";

   $spooler_log->info( '"' . $variable_set->names . '"' );        # ==>
"variable_1;variable_2"

   my @names = $variable_set->names->split( ";" );
    foreach my $name(@names){$spooler_log->info( $name . " = " . $variable_set( $name )
 );
```

Returned value:
`BSTR`

All variable names should be separated by semicolons.

4.22.4 set_var

Sets a variable

Syntax: $variable_set-> **`set_var`**`( BSTR name, Variant value )`

4.22.5 substitute

Replaces $-Variables in a String

Syntax: `BSTR` $variable_set-> **`substitute`**`( BSTR sustitution_string )`

Example: in javascript

```
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
```

In the example below, the `Subprocess.env` method is used.

References in the string in the form `$` *name* and `${` *name* `}` are replaced by variables.

Returned value:
`BSTR`

The string containing the substituted $ variables.

4.22.6 value

A variable

Syntax: $variable_set->`LetProperty( '`**`value`**`', BSTR name, Variant value )`

Syntax: `Variant` $variable_set-> **`value`**`( BSTR name )`

Parameters:

`name`

`value` `empty`, should a variable not exist.

Returned value:
`Variant`

`empty`, should a variable not exist.

4.22.7 var

A variable

Syntax: $variable_set->`LetProperty( '`**`var`**`', BSTR name, Variant value )`

Syntax: `Variant` $variable_set-> **`var`**`( BSTR name )`

Use the `Variable_set.value`, which is available in all languages.

Parameters:

`name`

`value` `empty`, should a variable not exist.

Returned value:
`Variant`

`empty`, should a variable not exist.

4.22.8 xml

`Variable_set` as an XML document

Syntax: $variable_set->`LetProperty( 'xml', BSTR )`

Syntax: `BSTR` $variable_set-> **`xml`**

Example:

```
my $variable_set = $spooler->create_variable_set();
$spooler_log->info( $variable_set->xml );   // Liefert <?xml version='1.0'?><
sos.spooler.variable_set/>

my $variable_set->LetProperty( 'xml', '<?xml version='1.0'?>' .
                  '<params>' .
                      '<param name='surname' value='Meier'/>' .
                      '<param name='christian name' value='Hans'/>' .
                  '</params>';
$spooler_log->info( $variable_set->xml );
$spooler_log->info( 'nachname=' . $variable_set->value( 'surname' ) );
$spooler_log->info( 'vorname =' . $variable_set->value( 'christian name' ) );
```

See <sos.spooler.variable_set>, <params>.

Parameters:

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

Returned value:
`BSTR`

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

4.23 Web_service

See also <web_service>

4.23.1 forward_xslt_stylesheet_path

Path to the forwarding XSLT stylesheets

Syntax: BSTR $web_service-> **forward_xslt_stylesheet_path**

See also <web_service forward_xslt_stylesheet="">

4.23.2 name

The Name of the JobScheduler Web Service

Syntax: BSTR $web_service-> **name**

See also <web_service name="">

4.23.3 params

Freely definable parameters

Syntax: _Variable_set_ $web_service-> **params**

The Web Services parameters can be set using the <web_service> element.

Returned value:
Variable_set_

4.24 Web_service_operation

See also <web_service>

4.24.1 peer_hostname

Peer (Remote) Host Name

Syntax: BSTR $web_service_operation-> **peer_hostname**

Returned value:
BSTR

"", should it not be possible to determine the name.

4.24.2 peer_ip

Peer (Remote) IP Address

Syntax: BSTR $web_service_operation-> **peer_ip**

4.24.3 request

Requests

Syntax: _Web_service_request_ $web_service_operation-> **request**

Returned value:
Web_service_request_

4.24.4 response

Answers

Syntax: _Web_service_response_ $web_service_operation-> **response**

Returned value:
Web_service_response_

4.24.5 web_service

Syntax: _Web_service_ $web_service_operation-> **web_service**

Returned value:
Web_service_

4.25 Web_service_request

See Web_service_operation_.

4.25.1 binary_content

Payload as a Byte Array (Java only)

Syntax: $web_service_request-> **binary_content**

This property is only available under Java.

The `( "Content-Type" )header field` is used to inform the client how binary content is to be interpreted (see HTTP/1.1 14.17 Content-Type) and `Web_service_request.charset_name`).

4.25.2 charset_name

Character Set

Syntax: `BSTR` $web_service_request-> **charset_name**

Example:

```
my $request = $spooler_task->order->web_service_operation->request;

$spooler_log->info( $request->header( 'Content-Type' ) );    # ==> text/xml;
charset=utf-8
$spooler_log->info( $request->content_type );                # ==> text/xml
$spooler_log->info( $request->charset_name );                # ==> utf-8
```

Returns the `charset=` parameter from the `Content-Type:` header entry.

4.25.3 content_type

Content Type (without parameters)

Syntax: `BSTR` $web_service_request-> **content_type**

Returns the `Content-Type:` header entry, without parameters - e.g. `"text/plain"`.

4.25.4 header

Header Entries

Syntax: `BSTR` $web_service_request-> **header**`( BSTR name )`

Example:

```
$spooler_log->info( 'Content-Type: ' . $spooler_task->order->web_service_operation->
request->header( 'Content-Type' ) );
```

Parameters:

`name` Case is not relevant.

Returned value:

BSTR

Returns "" in event of an unrecognized entry.

4.25.5 string_content

Payload as Text

Syntax: BSTR $web_service_request-> **string_content**

The character set to be used is taken from the charset parameter in the headers("Content-Type") (see HTTP/1.1 14.17 Content-Type). ISO-8859-1 will be used as default, should this parameter not be specified.

The following character sets are recognized:

- ISO-8859-1
- UTF-8 (only on Windows systems and restricted to the ISO-8859-1 characters)

See also Web_service_request.binary_content.

4.25.6 url

Uniform Resource Locator

Syntax: BSTR $web_service_request-> **url**

```
url = "http://" + header( "Host" ) + url_path
```

4.26 Web_service_response

Note that the binary_content property is only available under Java.

See also <web_service>

4.26.1 charset_name

Character set

Syntax: BSTR $web_service_response-> **charset_name**

Example:

```
my $request = $spooler_task->order->web_service_operation->request;

$spooler_log->info( $request->header( 'Content-Type' ) );   // ==> text/xml;
charset=utf-8
$spooler_log->info( $request->content_type );               // ==> text/xml
$spooler_log->info( $request->charset_name );               // ==> utf-8
```

Reads the `charset=` parameter from the `Content-Type:` header entry.

4.26.2 content_type

Content-Type (without parameters)

Syntax: `BSTR` $web_service_response-> **`content_type`**

Reads the `Content-Type:` header without any of the other associated parameters such as `charset=`.

4.26.3 header

Header Entries

Syntax: $web_service_response->`LetProperty( '`**`header`**`', BSTR value, BSTR name )`

Syntax: `BSTR` $web_service_response-> **`header`**`( BSTR name )`

Example:

```
$spooler_log->info( 'Content-Type: ' . $spooler_task->order->web_service_operation->
response->header( 'Content-Type' ) );
```

Parameters:

`value`	`""` is used for unknown entries.
`name`	The case in which entries are written is not relevant here.

Returned value:
`BSTR`

`""` is used for unknown entries.

4.26.4 send

Sends a Reply

Syntax: $web_service_response-> **`send`**`(`

4.26.5 status_code

HTTP Status Code

Syntax: $web_service_response->`LetProperty( 'status_code', int )`

The default setting is 200 (OK).

4.26.6 string_content

Text payloads

Syntax: $web_service_response->`LetProperty( 'string_content', BSTR text )`

Example:

```
my $response = $spooler_task->order->web_service_operation->response;
    $response->LetProperty( 'content_type', 'text/plain' );
    $response->LetProperty( 'charset_name', 'iso-8859-1' );
    $response->LetProperty( 'string_content', 'This is the answer' );
    $response->send();
```

The `header( "Content-Type" )` must first of all contain a `charset` parameter such as:

```
header( "Content-Type" ) = "text/plain; charset=iso-8859-1";
```

Text is coded as specified in the `charset` parameter. ISO-8859-1 will be used as the default value, should this parameter not be specified.

See `Web_service_request.string_content` for the character sets which are allowed.

See `Web_service_response.charset_name`.

4.27 Xslt_stylesheet

An XSLT style sheet contains the instructions for the transformation of an XML document.

The XSLT processor is implemented with libxslt .

4.27.1 apply_xml

Applies a style sheet to an XML document.

Syntax: `BSTR` $x-> **`apply_xml`**`( BSTR xml )`

4.27.2 close

Frees the style sheet resources

Syntax: $x-> **close**(

4.27.3 load_file

Loads the style sheet from an XML file

Syntax: $x-> **load_file**(BSTR path)

4.27.4 load_xml

Loads the style sheet from an XML document

Syntax: $x-> **load_xml**(BSTR xml)

5 VBScript API

The following classes are available for VBScript:

5.1 Error

5.1.1 code

The error code

Syntax: `String` error. **`code`**

5.1.2 is_error

`true`, should an error have occurred

Syntax: `Boolean` error. **`is_error`**

5.1.3 text

The error text (with error code)

Syntax: `String` error. **`text`**

5.2 Job

A task can either be waiting in the order queue or be running.

5.2.1 clear_delay_after_error

Resets all delays which have previously been set using `delay_after_error`

Syntax: spooler_job. **`clear_delay_after_error`** `()`

5.2.2 clear_when_directory_changed

Resets directory notification for all directories which have previously been set using `start_when_directory_changed()`

Syntax: spooler_job. **clear_when_directory_changed** ()

5.2.3 configuration_directory

Directory for the job configuration file should dynamic configuration from hot folders be used

Syntax: String spooler_job. **configuration_directory**

"", when a job does not come from a configuration directory.

5.2.4 delay_after_error

Delays the restart of a job in case of an error

Syntax: spooler_job. **delay_after_error** (Integer error_steps) = Double|Integer|String seconds_or_hhmm_ss

Example: in javascript

```
spooler_job.delay_after_error(  2 ) = 10;          // A 10 second delay after the 2nd
consecutive error
spooler_job.delay_after_error(  5 ) = "00:01";     // One minute delay after the 5th
consecutive error
spooler_job.delay_after_error( 10 ) = "24:00";     // A delay of one day after the
10th consecutive error
spooler_job.delay_after_error( 20 ) = "STOP";      // The Job is stopped after the
20th consecutive error
```

Should a (first) error occur whilst a job is being run, the JobScheduler will restart the job immediately.
However, after between two and four consecutive errors, the JobScheduler will wait 10 seconds before restarting the job;

After between five and nine consecutive errors, the job will be restarted after a delay of one minute; After between ten and nineteen errors, the delay is 24 hours.

The job is stopped after the twentieth consecutive error.

A delay can be specified, should a particular number of errors occur in series. In this case the job will be terminated and then restarted after the time specified.

This method call can be repeated for differing numbers of errors. A different delay can be specified for each new method call.

It is possible to set the value of the seconds_or_hhmm_ss parameter to "STOP" in order to restrict the number of (unsuccessful) repetitions of a job. The job then is stopped when the number of consecutive errors specified is reached.

A good position for this call is spooler_init().

See <delay_after_error>.

Parameters:

error_steps	The number of consecutive errors required to initiate the delay
seconds_or_hhmm_ss	The delay after which the job will be rerun

5.2.5 delay_order_after_setback

Delays after an order is setback

Syntax: spooler_job. **delay_order_after_setback** (Integer setback_count) = Double|Integer|String seconds_or_hhmm_ss

Example: in javascript

```
spooler_job.delay_order_after_setback( 1 ) = 60;          // for the 1st and 2nd
consecutive setbacks of an order:
               // delay the order 60s.


spooler_job.delay_order_after_setback( 3 ) = "01:00";     // After the 3rd consecutive
setback of an order,
               // the order will be delayed an hour.


spooler_job.max_order_setbacks = 5;                  // The 5th setback sets the order
to the error state
```

A job can delay an order which is currently being carried out with Order.setback(). The order is then positioned at the rear of the order queue for that job and carried out after the specified time limit.

The number of consecutively occurring setbacks for an order is counted. The delay set after a setback can be changed using delay_order_after_setback in the event of consecutively occurring setbacks.

See
<delay_order_after_setback>,

Order.setback(),

Job.max_order_setbacks,

Job_chain.add_job(),

Job.delay_after_error().

Parameters:

setback_count	The number of consecutive errors and therefore setbacks for a job. The setback delay can be varied according to this parameter.
seconds_or_hhmm_ss	Time limit for the setback of the order. After expiry of the time limit, the order is reprocessed in the same job.

5.2.6 folder_path

The directory in which the job is to be found.

Syntax: `String` spooler_job. **`folder_path`**

`""`, when the job does come from the local (`<config configuration_directory="">`) configuration file.

Returns the job part relative to the live directory. The path is to start with a slash ("/") and all path components are to be separated by slashes.

Examples:

- `"/somewhere/excel"` will be returned for the `c:\scheduler\config\live\somewhere\excel\sample.job.xml` job;
- `"/"` returned for the `c:\scheduler\config\live\sample.xml` job and
- `""` (an empty string) returned for a job outside the live directory.

5.2.7 include_path

Value of the `-include-path=` option

Syntax: `String` spooler_job. **`include_path`**

See `-include-path`.

5.2.8 max_order_setbacks

Limits the number of setbacks for an order

Syntax: spooler_job. **`max_order_setbacks`** = `Integer`

An order state is set to "error" (see `Job_chain_node.error_state`) when it is set back more than the number of times specified here (see `Order.setback()`).

See `Job.delay_order_after_setback` and `<delay_order_after_setback is_maximum="yes">`.

5.2.9 name

The job path beginning without a backslash

Syntax: `String` spooler_job. **`name`**

See `<job name="">`.

5.2.10 order_queue

The job order queue

Syntax: Order_queue spooler_job. **order_queue**

Example: in javascript

```
spooler_log.info( 'order=' + ( spooler_job.order_queue ? "yes" : "no" ) );
```

Every job order (<job order="yes">) has an order queue. This queue is filled by the job chain to which the job belongs.

See Job_chain.add_order() , and Job_chain.add_job() .

Returned value:
Order_queue

`null`, should the job have no queue (for <job order="no">).

5.2.11 process_class

The process class

Syntax: Process_class spooler_job. **process_class**

See <job process_class=""> .

Returned value:
Process_class

5.2.12 remove

Removes a job

Syntax: spooler_job. **remove** ()

The job is stopped - i.e. current tasks are terminated and no new ones are started. The job will be removed as soon as no more tasks are running.

Tasks queuing are ignored.

When no job task is running, the remove() function deletes the job immediately.

Job orders (<job order="yes">) cannot be removed.

See <modify_job cmd="remove"> .

5.2.13 start

Creates a new task and places it in the task queue

Syntax: Task spooler_job. **start** (Variable_set variables (optional))

Example:

```
spooler.job( "job_a" ).start

Dim parameters
Set parameters = spooler.create_variable_set()
parameters.var( "my_parameter" ) = "my_value"
parameters.var( "other_parameter" ) = "other_value"
spooler.job( "job_a" ).start( parameters )
```

The parameters are available to the Task.params task. Two parameters are particularly relevant here:

"spooler_task_name"	gives the task a name which then appears in the status display, e.g. in the web interface.
"spooler_start_after"	specifies a time in seconds (real number), after which the task is to start. The JobScheduler <run_time> is ignored in this case.

See Spooler.create_variable_set(), Spooler.job, Variable_set.value.

Returned value:
Task

5.2.14 start_when_directory_changed

Monitors a directory and starts a task should a notification of a change be received

Syntax: spooler_job. **start_when_directory_changed** (String directory_path, String filename_pattern (optional))

Example: in javascript

```
spooler_job.start_when_directory_changed( "c:/tmp" );

// only relevant for files whose names do not end in "~".
spooler_job.start_when_directory_changed( "c:/tmp", "^.*[^~]$" );
```

Should there not be a task belonging to this job running and a notification be received that a change in the directory being monitored has occurred (that a file has been added, changed or deleted), then this change can be used to prompt the JobScheduler to start a task if the current time falls within that allowed by the <run_time> parameter.

This method can be called a more than once in order to allow the monitoring of a number of directories. A repeat call can also be made to a directory in order to reactivate monitoring - if, for example, it has not been possible to access the directory.

This method call can be coded in the JobScheduler start script or in the spooler_init() method. In the latter case, the job must have been started at least once in order for the method call to be carried out. The <run_time once="yes"> setting should be used for this.

The job should be regularly <run_time repeat=""> restarted and <delay_after_error> set.

The same setting can be made in the XML configuration using the <start_when_directory_changed> element.

Parameters:

directory_path	the address of the directory being monitored
filename_pattern	restricts monitoring to files whose names correspond with the regular expression used.

5.2.15 state_text

Free text for the job state

Syntax: spooler_job. **state_text** = String

Example: in javascript

```
spooler_job.state_text = "Step C succeeded";
```

The text will be shown in the HTML interface.

5.2.16 title

The job title

Syntax: String spooler_job. **title**

Example: in javascript

```
spooler_log.info( "Job title=" + spooler_job.title );
```

See <job title="">.

5.2.17 wake

Causes a task to be started

Syntax: spooler_job. **wake** ()

Starts a task, should the job have the pending or stopped states.

See Job.start().

5.3 Job_chain - job chains for order processing

A job chain is a series of jobs (job chain nodes). Orders (Order) proceed along these chains.

Every position in a job chain is assigned a state and a job. When an order is added to the job chain, it is enqueued by the JobScheduler according to the state of the order. The job assigned to this position then carries out the order.

Additionally, each position in a job chain has a successor state and an error state. The JobScheduler changes the state of an order after each job in the job chain has been processed. Should the job step return `(spooler_process) true`, then the JobScheduler sets the succeeding state; otherwise it sets the error state. The order then moves to another position in the job chain as defined by the new state. However, this does not apply when the state is changed during execution with Order.state.

A job chain is created using Spooler.create_job_chain(); it is filled using Job_chain.add_job() and Job_chain.add_end_state() and finally made available with Spooler.add_job_chain().

Every node is allocated a unique state. Therefore either Job_chain.add_job() or Job_chain.add_end_state() must be called once for every state.

Example: in javascript

```
var my_job_chain = spooler.create_job_chain();
my_job_chain.name = "JobChain";

my_job_chain.add_job( "job_100", 100,  200, 999 );
my_job_chain.add_job( "job_200", 200, 1000, 999 );
my_job_chain.add_end_state(  999 );
my_job_chain.add_end_state( 1000 );
spooler.add_job_chain( my_job_chain );
```

5.3.1 add_end_state

Adds the end state to a job chain

Syntax: job_chain. **add_end_state** `( Variant state )`

This state is not assigned a job. An order that reaches the final state has completed the job chain and will be removed from the chain.

5.3.2 add_job

Adds a job to a job chain

Syntax: job_chain. **add_job** `( String job_name, Variant input_state, Variant output_state, Variant error_state )`

5.3.3 add_or_replace_order

Adds an order to a job chain and replaces any existing order having the same identifier

Syntax: job_chain. **add_or_replace_order** (Order order)

Should the job chain already contain an order with the same identifier, then this order will be replaced. More accurately: the original order will be deleted and the new one added to the job chain.

As long as an existing order having the same identifier as the new order is being carried out, both orders will be present. However, the original order will have already been deleted from the job chain and database; it is only available to the current task and will completely disappear after it has been completed.

In this case the JobScheduler will wait until the original order has been completed before starting the new one.

See Job_chain.add_order() and Order.remove_from_job_chain()

5.3.4 add_order

Adds an order to a job chain

Syntax: Order job_chain. **add_order** (Order | String order_or_payload)

Should an order already exist on another job chain, then the JobScheduler removes the order from this other chain.

An order is allocated to the job order queue corresponding to its state, and positioned according to its priority.

The job chain must be specified for the JobScheduler using <job_chain> or Spooler.add_job_chain().

Should an order with the same Order.id already exist in a job chain, then an exception with the error code SCHEDULER-186 is returned. However, see also Job_chain.add_or_replace_order().

Returned value:
Order

5.3.5 name

The name of a job chain

Syntax: job_chain. **name** = String

Syntax: String job_chain. **name**

Example: in javascript

```
var job_chain = spooler.create_job_chain();
job_chain.name = "JobChain";
```

5.3.6 node

The job chain nodes with a given state

Syntax: Job_chain_node job_chain. **node** (Variant state)

Returned value:
Job_chain_node

5.3.7 order_count

The number of orders in a job chain

Syntax: Integer job_chain. **order_count**

5.3.8 order_queue

```
= node( state).job().order_queue()
```

Syntax: Order_queue job_chain. **order_queue** (Variant state)

Returns the order queue which has a given state.

Returned value:
Order_queue

5.3.9 orders_recoverable

Syntax: job_chain. **orders_recoverable** = Boolean

Syntax: Boolean job_chain. **orders_recoverable**

See <job_chain orders_recoverable="">.

5.3.10 remove

Job chain deletion

Syntax: job_chain. **remove** ()

Should orders in a job chain still be being processed (in spooler_process()) when the chain is to be deleted, then the JobScheduler will wait until the last order has been processed before deleting the chain.

Orders remain in the database. Should a new job chain be added which has the same name as a deleted job chain (Spooler.add_job_chain()), then the JobScheduler will reload any orders from the original job chain which have remained in the database. Note however, that the states of the orders in the new job chain should be the same as those in the original chain at the time of its deletion.

5.3.11 title

Syntax: job_chain. **title** = String

Syntax: String job_chain. **title**

See <job_chain title="">.

5.4 Job_chain_node

A job chain node is assigned a position in a job chain (Job_chain). The following elements make up a job chain node: a state, a job, a successor state and an error state.

A job chain node is created either using Job_chain.add_job() or Job_chain.add_end_state().

5.4.1 action

Stopping or missing out job chain nodes

Syntax: node. **action** = String

Syntax: String node. **action**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "my_job_chain" ).node( 100 );
job_chain_node.action = "next_state";
```

This option is not possible with distributed job chains.

Possible settings are:

action="process"
: This is the default setting. Orders are carried out.

action="stop"
: Orders are not carried out, they collect in the order queue.

action="next_state"
: Orders are immediately handed over to the next node as specified with next_state.

See also <job_chain_node.modify action="">.

Character string constonants are defined in Java:

- Job_chain_node.ACTION_PROCESS
- Job_chain_node.ACTION_STOP
- Job_chain_node.ACTION_NEXT_STATE

5.4.2 error_node

The next node in a job chain in the event of an error

Syntax: Job_chain_node. node. **error_node**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

     spooler_log.debug( "error state=" + job_chain_node.error_node.state );       //
"state=999"
```

Returned value:
Job_chain_node.

null, in the event of no error node being defined (the error state has not been specified)

5.4.3 error_state

State of a job chain in event of an error

Syntax: Variant node. **error_state**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "error state=" + job_chain_node.error_node.state );       // "error
state=999"
```

5.4.4 job

The job allocated to a node

Syntax: Job. node. **job**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "job=" + job_chain_node.job.name );                        //
"job=job_100"
```

Returned value:
Job.

5.4.5 next_node

Returns the next node or null if the current node is assigned the final state.

Syntax: _Job_chain_node_ node. **next_node**

Returned value:
Job_chain_node_

5.4.6 next_state

The order state in a job chain after successful completion of a job

Syntax: Variant node. **next_state**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.debug( "next_state=" + job_chain_node.next_state );              //
"state=200"
```

5.4.7 state

The valid state for a job chain node

Syntax: Variant node. **state**

Example: in javascript

```
var job_chain_node = spooler.job_chain( "Jobchain" ).node( 100 );

spooler_log.info( "state=" + job_chain_node.state );                         //
"state=100"
```

5.5 Job_impl - Super Class for a Job or the JobScheduler Script

Job methods are called in the following order:

```
spooler_init()
    spooler_open()
        spooler_process()
        spooler_process()
        ...
    spooler_close()
    spooler_on_success() or spooler_on_error()
```

spooler_exit()

None of these methods must be implemented. However, it is usual that at least the spooler_process() method is implemented.

An error during carrying out a job script whilst loading or during spooler_init() causes spooler_on_error(). to be called. The job is then stopped and spooler_exit() called (although spooler_init() has not been called!). The script is then unloaded.

Note that spooler_on_error() must also be able to handle errors which occur during loading or in spooler_init().

Note also that spooler_exit() is called even though spooler_init() has not been called.

5.5.1 spooler

The JobScheduler base object

Syntax: Spooler **spooler**

Example: in javascript

```
spooler_log.debug( "The working directory of the JobScheduler is " + spooler.directory
);
```

Returned value:
Spooler

5.5.2 spooler_close

Task end

Syntax: **spooler_close** ()

This method is called after a job has been completed. The opposite of this method is spooler_open().

5.5.3 spooler_exit

Destructor

Syntax: **spooler_exit** ()

Is called as the last method before the script is unloaded. This method can be used, for example, to close a database connection.

5.5.4 spooler_init

Initialization

Syntax: `Boolean` **`spooler_init`** `()`

The JobScheduler calls these methods once before `spooler_open()`. This is analog to `spooler_exit()`. This method is suitable for initializing purposes (e.g. connecting to a database).

Returned value:
`Boolean`

`false` ends a task. The JobScheduler continues using the `spooler_exit()` method. When the task is processing an order, then this return value makes the JobScheduler terminate the job with an error. That is, unless a repeated start interval has been set using `Job.delay_after_error`

5.5.5 spooler_job

The job object

Syntax: `Job` **`spooler_job`**

Example: in javascript

```
spooler_log.info( "The name of this job is " + spooler_job.name );
```

Returned value:
`Job`

5.5.6 spooler_log

Event logging object

Syntax: `Log` **`spooler_log`**

Example: in java

```
spooler_log.info( "Something has happened" );
```

Returned value:
`Log`

5.5.7 spooler_on_error

Unsuccessful completion of a job

Syntax: **`spooler_on_error`** `()`

Is called at the end of a job after an error has occurred (after spooler_close() but before spooler_exit()).

5.5.8 spooler_on_success

Successful completion of a job

Syntax: **spooler_on_success** ()

This method is called by the JobScheduler after spooler_close() and before spooler_exit() ; should no error have occurred.

5.5.9 spooler_open

The Start of a Task

Syntax: Boolean **spooler_open** ()

This method is called immediately after spooler_init() . The opposite of this method is spooler_close() .

5.5.10 spooler_process

Job steps or the processing of an order

Syntax: Boolean **spooler_process** ()

Processes a job step.

An order driven job stores the current order in Task. order .

The default implementation returns false. The implementation of an order driven job can set the successor state for an order by returning true.

Returned value:
Boolean

In the event of standard jobs <job order="no"> : false the JobScheduler ends processing of this job; true> the JobScheduler continues calling the spooler_process() method.

In the event of order driven jobs <job order="yes"> : false the order acquires the error state (s. Job_chain_node and <job_chain_node>). true the order acquires the next state or is terminated if the next state is the final state. This, however, does not apply when the state is changed during execution using Order. state .

5.5.11 spooler_task

The task object

Syntax: Task **spooler_task**

Example: in javascript

```
spooler_log.info( "The task id is " + spooler_task.id );
```

Returned value:
Task

5.6 Lock

See also <lock name="">.

Example: in javascript

```
var locks = spooler.locks;
var lock = locks.create_lock();
lock.name = "my_lock";
locks.add_lock( lock );
```

5.6.1 max_non_exclusive

Limitation of non-exclusive allocation

Syntax: lock. **max_non_exclusive** = Integer

Syntax: Integer lock. **max_non_exclusive**

The default setting is unlimited (231-1), which means that with <lock.use_exclusive="no"> any number of non-exclusive tasks can be started (but only one exclusive task).
The number cannot be smaller than the number of non-exclusive allocations.

See also <lock max_non_exclusive="">.

5.6.2 name

The lock name

Syntax: lock. **name** = String

Syntax: String lock. **name**

The name can only be set once and cannot be changed.

See also <lock name="">.

5.6.3 remove

Removes a lock

Syntax: lock. **remove** ()

Example: in javascript

```
spooler.locks.lock( "my_lock" ).remove();
```

A lock can only be removed when it is not active - that is, it has not been allocated to a task and it is not being used by a job (<lock.use>).

See also <lock.remove>.

5.7 Locks

5.7.1 add_lock

Adds a lock to a JobScheduler

Syntax: locks. **add_lock** (Lock lck)

5.7.2 create_lock

Creates a new lock

Syntax: Lock locks. **create_lock** ()

Returns a new lock Lock. This lock can be added to the JobScheduler using Locks.add_lock().

Returned value:
Lock

5.7.3 lock

Returns a lock

Syntax: Lock locks. **lock** (String lock_name)

An exception will be returned if the lock is unknown.

Returned value:
Lock

5.7.4 lock_or_null

Returns a lock

Syntax: Lock_ locks. **lock_or_null** (String lock_name)

Returned value:
Lock_

null, when the lock is unknown.

5.8 Log - Logging

The spooler_log method can be used in a job or in the JobScheduler start script with the methods described here.
Notification by e-mail

The JobScheduler can send a log file after a task has been completed per e-mail. The following properties define in which cases this should occur.

- Log.mail_on_error,
- Log.mail_on_warning,
- Log.mail_on_process,
- Log.mail_on_success and
- Log.mail_it

Only the end of a task - and not the end of an order - (i.e. spooler_process()) can initiate the sending of e-mails. However, see Task.end().

The Log.mail method makes the Mail object available, which in turn addresses the mails.

Example: in javascript

```
spooler_log.info( "Something for the Log" );

spooler_log.mail_on_warning = true;
spooler_log.mail.from    = "scheduler@company.com";
spooler_log.mail.to      = "admin@company.com";
spooler_log.mail.subject = "ended";
```

5.8.1 debug

Debug message (level -1)

Syntax: spooler_log. **debug** (String line)

5.8.2 debug1

Debug message (level -1)

Syntax: spooler_log. **debug1** (String line)

5.8.3 debug2

Debug message (level -2)

Syntax: spooler_log. **debug2** (String line)

5.8.4 debug3

Debug message (level -3)

Syntax: spooler_log. **debug3** (String line)

5.8.5 debug4

Debug message (level -4)

Syntax: spooler_log. **debug4** (String line)

5.8.6 debug5

Debug message (level -5)

Syntax: spooler_log. **debug5** (String line)

5.8.7 debug6

Debug message (level -6)

Syntax: spooler_log. **debug6** (String line)

5.8.8 debug7

Debug message (level -7)

Syntax: spooler_log. **debug7** (String line)

5.8.9 debug8

Debug message (level -8)

Syntax: spooler_log. **debug8** (String line)

5.8.10 debug9

Debug message (level -9)

Syntax: spooler_log. **debug9** (String line)

5.8.11 error

Error Message (Level 1)

Syntax: spooler_log. **error** (String line)

A job stops after a task has ended, should an error message have been written in the task log (spooler_log) and <job stop_on_error="no"> not have been set.

5.8.12 filename

Log file name

Syntax: String spooler_log. **filename**

5.8.13 info

Information message (Level 0)

Syntax: spooler_log. **info** (String line)

5.8.14 last

The last output with the level specified

Syntax: `String` spooler_log. **`last`** `( Integer| String level )`

5.8.15 last_error_line

The last output line with level 2 (error)

Syntax: `String` spooler_log. **`last_error_line`**

5.8.16 level

Limit protocol level

Syntax: spooler_log. **`level`** = `Integer`

Syntax: `Integer` spooler_log. **`level`**

Defines the level with which protocol entries should be written. Every protocol entry is given one of the following categories: `error`, `warn`, `info`, `debug1` to `debug9` (`debug1` is the same as `debug`).

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

The `-log-level` option has precedence over this parameter.

The `factory.ini` (section `[job]`, entry `log_level=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_level=…`) setting is overwritten by this parameter.

Only messages above the level specified will be given out.

The meanings of the numerical values are:

-9 to -2:	`debug9` to `debug2`
-1:	`debug`
0:	`info`
1:	`warn`
2:	`error`

5.8.17 log

Writes in the log file with the specified level.

Syntax: spooler_log. **log** (Integer level, String line)

5.8.18 log_file

Adds the content of a file to the log file

Syntax: spooler_log. **log_file** (String path)

Log the content of a file with level 0 (info). An error occurring whilst accessing the file is logged as a warning.

Note that when executed on a remote computer with <process_class remote_scheduler=""> the file is read from the JobScheduler's file system and not that of the task.

5.8.19 mail

E-mail settings are made in the Mail Object

Syntax: spooler_log. **mail** = Mail

Syntax: Mail spooler_log. **mail**

Returned value:
Mail

5.8.20 mail_it

Force dispatch

Syntax: spooler_log. **mail_it** = Boolean

If this property is set to true, then a log will be sent after a task has ended, independently of the following settings:

Log.mail_on_error, Log.mail_on_warning, Log.mail_on_success, Log.mail_on_process and Log.mail_on_error.

5.8.21 mail_on_error

Sends an e-mail should a job error occur. Errors are caused by the `Log.error()` method or by any exceptions that have not been caught by a job.

Syntax: spooler_log. **`mail_on_error`** = `Boolean`

Syntax: `Boolean` spooler_log. **`mail_on_error`**

Content of the e-mail is the error message. The log file is sent as an attachment.

The `factory.ini` (section `[job]`, entry `mail_on_error=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `mail_on_error=...`) setting is overwritten by this parameter.

Content of the e-mail is the error message. The log file is sent as an attachment.

5.8.22 mail_on_process

Sends an e-mail should a job have successfully processed the number of steps specified. Steps are caused by the `spooler_process()` methods:

Syntax: spooler_log. **`mail_on_process`** = `Integer`

Syntax: `Integer` spooler_log. **`mail_on_process`**

Causes the task log to be sent when a task has completed at least the specified number of steps - i.e. calls of `spooler_process()`. Because non-API tasks do not have steps, the JobScheduler counts each task as a single step.

Content of the e-mail is the success message. The log file is sent as an attachment.

The `factory.ini` (section `[job]`, entry `mail_on_process=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `mail_on_process=...`) setting is overwritten by this parameter.

Content of the e-mail is the success message. The log file is sent as an attachment.

5.8.23 mail_on_success

Sends an e-mail should a job terminate successfully.

Syntax: spooler_log. **`mail_on_success`** = `Boolean`

Syntax: `Boolean` spooler_log. **`mail_on_success`**

The success message forms the content of the e-mail. The log file is sent as an attachment.

The factory.ini (section [job], entry mail_on_success=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry mail_on_success=…) setting is overwritten by this parameter.

The success message forms the content of the e-mail. The log file is sent as an attachment.

5.8.24 mail_on_warning

Sends an e-mail should a job warning occur. Warnings are caused by the Log.warn() method.

Syntax: spooler_log. **mail_on_warning** = Boolean

Syntax: Boolean spooler_log. **mail_on_warning**

The warning forms the content of the e-mail. The log file is sent as an attachment.

The factory.ini (section [spooler], entry mail_on_warning=…) setting is overwritten by this parameter.

The warning forms the content of the e-mail. The log file is sent as an attachment.

5.8.25 new_filename

A new name for the log file

Syntax: spooler_log. **new_filename** = String

Syntax: String spooler_log. **new_filename**

Sets the name of the log file. The JobScheduler copies a log into this file after a log has been made. This file is then available to other applications.

5.8.26 start_new_file

Only for the main log file: closes the current log file and starts a new one

Syntax: spooler_log. **start_new_file** ()

5.8.27 warn

Warning (Level 2)

Syntax: spooler_log. **warn** (String line)

5.9 Mail - e-mail dispatch

See Log.mail.

5.9.1 add_file

Adds an attachment

Syntax: mail. **add_file** (String path, String filename_for_mail (optional) , String content_type (optional) , String encoding (optional))

Example: in javascript

```
spooler_log.mail.add_file( "c:/tmp/1.txt", "1.txt", "text/plain", "quoted-printable" );
```

Parameters:

path	path to the file to be appended
filename_for_mail	The file name to appear in the message
content_type	"text/plain" is the preset value.
encoding	e.g. "quoted printable"

5.9.2 add_header_field

Adds a field to the e-mail header

Syntax: mail. **add_header_field** (String field_name, String value)

5.9.3 bcc

Invisible recipient of a copy of a mail, (*blind carbon copy*)

Syntax: mail. **bcc** = String

Syntax: String mail. **bcc**

Example: in javascript

```
spooler_log.mail.bcc = "hans@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_bcc=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_bcc=...`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

5.9.4 body

Message content

Syntax: mail. **`body`** = `String`

Syntax: `String` mail. **`body`**

Example: in javascript

```
spooler_log.mail.body = "Job succeeded";
```

Line feed / carriage return is coded with `\n` (`chr(10)` in VBScript).

5.9.5 cc

Recipient of a copy of a mail, (*carbon copy*)

Syntax: mail. **`cc`** = `String`

Syntax: `String` mail. **`cc`**

Example: in javascript

```
spooler_log.mail.cc = "hans@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The `factory.ini` (section `[job]`, entry `log_mail_cc=...`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_cc=...`) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

5.9.6 dequeue

Repeated attempts can be made to send messages from the `queue_dir` directory

Syntax: `Integer` mail. **`dequeue`** `()`

See `Mail.dequeue_log`, `factory.ini` (section `[spooler]`, entry `mail_queue_dir=…`).

Returned value:
`Integer`

The number of messages sent

5.9.7 dequeue_log

The `dequeue()` log

Syntax: `String` mail. **`dequeue_log`**

Example: in javascript

```
var count = spooler_log.mail.dequeue();
spooler_log.info( count + " messages from mail queue sent" );
spooler_log.info( spooler_log.mail.dequeue_log );
```

See `Mail.dequeue()`.

5.9.8 from

Sender

Syntax: mail. **`from`** = `String`

Syntax: `String` mail. **`from`**

Example: in javascript

```
spooler_log.mail.from = "scheduler@company.com";
```

The `factory.ini` (section `[job]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

The `factory.ini` (section `[spooler]`, entry `log_mail_from=…`) setting is overwritten by this parameter.

5.9.9 queue_dir

The directory used for returned e-mails

Syntax: mail. **queue_dir** = String path

Syntax: String mail. **queue_dir**

E-mails which cannot be sent (because, for example, the SMTP server cannot be contacted) are stored in this directory.

In order to send these e-mails later it is necessary to write a job which calls up the Mail. dequeue( ) method.

This setting is generally made in sos. ini (section [mail], entry queue_dir=…).

Environment variables (e.g. $HOME) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The factory. ini (section [job], entry mail_queue_dir=…) setting is overwritten by this parameter.

The factory. ini (section [spooler], entry mail_queue_dir=…) setting is overwritten by this parameter.

The sos. ini (section [mail], entry queue_dir=…) setting is overwritten by this parameter.

5.9.10 smtp

The name of the SMTP server

Syntax: mail. **smtp** = String

Syntax: String mail. **smtp**

Example: in javascript

```
spooler_log.mail.smtp = "mail.company.com";
```

These settings are generally made using sos. ini (section [mail], entry smtp=…).

smtp=-queue stops e-mails being sent. Instead mails are written into the file specified in queue_dir. See also sos. ini (section [mail], entry queue_only=…).

The factory. ini (section [job], entry smtp=…) setting is overwritten by this parameter.

The factory. ini (section [spooler], entry smtp=…) setting is overwritten by this parameter.

The sos. ini (section [mail], entry smtp=…) setting is overwritten by this parameter.

5.9.11 subject

Subject, *re*

Syntax: mail. **subject** = String

Syntax: String mail. **subject**

Example: in javascript

```
spooler_log.mail.subject = "Job succeeded";
```

The factory.ini (section [job], entry log_mail_subject=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_subject=…) setting is overwritten by this parameter.

5.9.12 to

Recipient

Syntax: mail. **to** = String

Syntax: String mail. **to**

Example: in javascript

```
spooler_log.mail.to = "admin@company.com";
```

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

The factory.ini (section [job], entry log_mail_to=…) setting is overwritten by this parameter.

The factory.ini (section [spooler], entry log_mail_to=…) setting is overwritten by this parameter.

Multiple addresses (separated by commas) can be specified when the hostware uses JavaMail to send e-mails.

See javax.mail.InternetAddress.parse(String).

5.9.13 xslt_stylesheet

The XSLT style sheet for e-mail processing. Before sending an e-mail the JobScheduler creates an XML document containing the e-mail headers, subject and body. The content of these elements can be adjusted or overwritten by an individual XSLT style sheet. This can be used e.g. to create translations of e-mail content. Having processed the XSLT style sheet the JobScheduler sends the resulting content of the XML elements as e-mail.

Syntax: `Xslt_stylesheet` mail. **`xslt_stylesheet`**

Returned value:
`Xslt_stylesheet`

The XSLT style sheet as a string

5.9.14 xslt_stylesheet_path

The path and file name of the XSL style sheet for e-mail processing.

Syntax: mail. **`xslt_stylesheet_path`** = `String path`

Example: in javascript

```
spooler_log.mail.xslt_stylesheet_path = "c:/stylesheets/mail.xslt";
```

The path to the XSLT style sheet. XSLT style sheets are used by the JobScheduler for the preparation of e-mails. At the time of writing (April 2006) this subject is not documented.

`<config mail_xslt_stylesheet="…">`

Parameters:

`path` The path of the file containing the XSLT style sheet

5.10 Monitor_impl - Using Super Classes for Start Scripts or Jobs

A job can be given a monitor using `<monitor>`.

A monitor can provide the following methods:

`Monitor_impl.spooler_task_before()`
Before starting a task - can prevent a task from being started.

`Monitor_impl.spooler_task_after()`
After a task has been completed.

`Monitor_impl.spooler_process_before()`
Before `spooler_process()` - this method can stop `spooler_process()` from being called.

Monitor_impl.spooler_process_after()

After spooler_process() - can be used to change its return value.

5.10.1 spooler

The JobScheduler Object

Syntax: _Spooler_ **spooler**

Example: in javascript

```
spooler_log.debug( "The working directory of the JobScheduler is " + spooler.directory
);
```

Is the same object as spooler in the Job_impl class.

Returned value:
Spooler_

5.10.2 spooler_job

The Job Object

Syntax: _Job_ **spooler_job**

Example: in javascript

```
spooler_log.info( "The name of this job is " + spooler_job.name );
```

Is the same object as spooler_job in the Job_impl class.

Returned value:
Job_

5.10.3 spooler_log

Writing Log Files

Syntax: _Log_ **spooler_log**

Example: in java

```
spooler_log.info( "Something has happened" );
```

Is the same object as spooler_log in the Job_impl class.

Returned value:
Log_

5.10.4 spooler_process_after

After `spooler_process()`

Syntax: `Boolean` **`spooler_process_after`** `( Boolean spooler_process_result )`

Example: in java

```
public boolean spooler_task_after( boolean spooler_process_result )  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    spooler_log.info( "spooler_process() didn't throw an exception and delivered " +
spooler_process_result );
    return spooler_process_result;   // Unchanged result
}
```

The JobScheduler calls this method after `spooler_process()` has been carried out.

Parameters:

`spooler_process_result`	The return value from the `spooler_process()` is set to `false`, should `spooler_process()` have ended with an exception.

Returned value:
`Boolean`

Replaces the return value from the `spooler_process()` method or `false`, should `spooler_process()` have ended with an error.

5.10.5 spooler_process_before

Before `spooler_process()`

Syntax: `Boolean` **`spooler_process_before`** `()`

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    spooler_log.info( "SPOOLER_PROCESS_BEFORE()" );
    return true;   // spooler_process() will be executed
}
```

Example: in java

```
public boolean spooler_process_before()  throws Exception
{
    boolean continue_with_spooler_process = true;

    if( !are_needed_ressources_available() )
    {
        spooler_task.order().setback();
        continue_with_spooler_process = false;
    }

    return continue_with_spooler_process;
}
```

This method is called by the JobScheduler before each call of spooler_process() .

Returned value:
Boolean

false prevents further calls to spooler_process() . The JobScheduler continues as though false had been returned by spooler_process() false.

5.10.6 spooler_task

The Task Object

Syntax: Task **spooler_task**

Example: in javascript

```
spooler_log.info( "The task id is " + spooler_task.id );
```

Is the same object as spooler_task in the Job_impl class.

Returned value:
Task

5.10.7 spooler_task_after

After Completing a Task

Syntax: **spooler_task_after** ()

Example: in java

```
public void spooler_task_after()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_AFTER()" );
}
```

This method is called by the JobScheduler after a task has been completed.

5.10.8 spooler_task_before

Before Starting a Task

Syntax: `Boolean` **`spooler_task_before`** `()`

Example: in java

```
public boolean spooler_task_before()  throws Exception
{
    spooler_log.info( "SPOOLER_TASK_BEFORE()" );
    return true;    // Task will be started
  //return false;   // Task will not be started
}
```

This method is called by the JobScheduler before a task is loaded.

Returned value:
`Boolean`

`false` does not allow a task to start and `Monitor_impl.spooler_task_after()` will not be called.

5.11 Order - Order

See JobScheduler Documentation, `Spooler.create_order()`, `Job_chain.add_order()`, `Task.order`.
File order

A file order is an order with for which the `scheduler_file_path` parameter has been set: `Order.params`. `Variable_set.value()`.

See JobScheduler Documentation.

Example: An Order with a simple Payload, in javascript

```
// Create order:
{
    var order = spooler.create_order();
    order.id         = 1234;
    order.title      = "This is my order";
    order.state_text = "This is my state text";
    order.payload    = "This is my payload";
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    spooler_log.info( "order.payload=" + order.payload );
    return true;
}
```

Example: Creating an Order with a Variable_set as a Payload, in javascript

```
// Create order:
{
    var variable_set = spooler.create_variable_set();
    variable_set.value( "param_one" ) = "11111";
    variable_set.value( "param_two" ) = "22222";

    var order = spooler.create_order();
    order.id      = 1234;
    order.payload = variable_set;
    spooler.job_chain( "my_job_chain" ).add_order( order );
}
...

// Process order:
function spooler_process()
{
    var order = spooler_task.order;
    var variable_set = order.payload;
    spooler_log.info( "param_one=" + variable_set.value( "param_one" ) );
    spooler_log.info( "param_two=" + variable_set.value( "param_two" ) );
    return true;
}
```

5.11.1 at

The order start time

Syntax: order. **at =** `String|DATE`

Example: in javascript

```
order.at = "now+65";
spooler.job_chain( "my_job_chain" ).add_order( order );
```

Used to set the start time before an order is added to an order queue. The following can be specified as a string:

- "now"
- "*yyyy-mm-dd HH:MM[:SS]*"
- "now + *HH:MM[:SS]*"
- "now + *seconds*"

This setting changes start times set by Order.run_time or Order.setback().

See <add_order at="">.

5.11.2 end_state

The state that should be reached when an order has been successfully completed

Syntax: order. **end_state** = Variant

Syntax: Variant order. **end_state**

When an order has its own end_state other than "" then it is considered to be completed after the job allocated to this end state has been completed and before the order otherwise leaves this state (see <job_chain_node> for example to continue to another job which usually comprises a part of the job chain).

The state specified has to reference a valid state of a job node in the job chain.

5.11.3 id

Order Identification

Syntax: order. **id** = Variant

Syntax: Variant order. **id**

Every order has an identifier. This identifier must be unique within a job chain or job order queue. It should also correspond to the data being processed. Normally database record keys are used.

When an id is not set, then the JobScheduler automatically allocates one using Job_chain.add_order().

5.11.4 job_chain

The job chain containing an order

Syntax: Job_chain order. **job_chain**

Returned value:
Job_chain

5.11.5 job_chain_node

The job chain nodes which correspond with the order state

Syntax: _Job_chain_node_ order. **job_chain_node**

Returned value:
Job_chain_node_

5.11.6 log

Order log

Syntax: _Log_ order. **log**

Example:

```
spooler_task.order.log.info( "Only for order log, not for task log" );
spooler_log.info( "For both order log and task log" );
```

Returned value:
Log_

5.11.7 params

The order parameters

Syntax: order. **params** = Variable_set

Syntax: _Variable_set_ order. **params**

`params` is held in `Order.payload_`, the latter cannot, therefore, be used together with `params`.

See <add_order>_.

Returned value:
Variable_set_

5.11.8 payload

Load - an order parameter.

Syntax: order. **payload** = Variable_set_|String|Integer|... payload

Syntax: _Variable_set_|String|Integer|... order. **payload**

Instead of this property, the use of Order.params is recommended, which corresponds to (Variable_set)order.payload.

In addition to Order.id which identifies an order, this field can be used for other information.

See Order.params and Order.xml_payload.

Parameters:

payload May be a string or a Variable_set.

Returned value:
Variable_set|String|Integer|…

May be a string or a Variable_set.

5.11.9 payload_is_type

Checks the payload COM-Type

Syntax: Boolean order.**payload_is_type**(String type_name)

Parameters:

type_name "Spooler.Variable_set","Hostware.Dyn_obj" or "Hostware.Record".

5.11.10 priority

Orders with a higher priority are processed first

Syntax: order.**priority** = Integer

Syntax: Integer order.**priority**

5.11.11 remove_from_job_chain

Syntax: order.**remove_from_job_chain**()

Note that when an order has just been started by a task, then the Order.job_chain property will still return the job chain from which the order has just been removed, using this call, even when "remove_from_job_chain" has been carried out. It is only when the execution has been ended that this method returns null. (other than when the order has just been added to a job chain). This ensures that the job_chain property remains stable whilst a task is being executed.

5.11.12 run_time

<run_time> is used to periodically repeat an order

Syntax: _Run_time_ order. **run_time**

Example: in javascript

```
order.run_time.xml = "<run_time><at at='2006-05-23 11:43:00'/></run_time>";
```

See <run_time>.

The <modify_order at="now"> command causes an order which is waiting because of run_time to start immediately.

Returned value:
Run_time

5.11.13 setback

Delays an order back for a period of time

Syntax: order. **setback** ()

An order will be delayed and repeated after the period of time specified in either <delay_order_after_setback> or Job.delay_order_after_setback. When the job is repeated, only the spooler_process() job function is repeated. If the order.setback() function is called from spooler_process(), then the retrun value from spooler_process() will have no effect. .

An order counts the number of times this method is called in sequence. This count is then used by <delay_order_after_setback>. It is set to 0, when spooler_process() is completed without <delay_order_after_setback> being called. All counters are set to 0 when the JobScheduler is started.

The <modify_order at="now"> command causes a blocked order to start immediately.

5.11.14 setback_count

How many times the order is setting back?

Syntax: Integer order. **setback_count**

see also <delay_order_after_setback>.

5.11.15 state

The order state

Syntax: order. **state** = Variant

Syntax: `Variant` order. **`state`**

When an order is in a job chain, then its state must correspond with one of the states of the job chain.

Whilst an order is being processed by a job the following state, as defined in the job chain (`<job_chain_node next_state="">`) has no effect. Similarly, the return values from `spooler_process()` and `Monitor_impl.spooler_process_after()` are meaningless. This means that with `Order.state` the following state for a job can be set as required.

An order is added to the job order queue which is corresponding to its state. See `<job_chain_node>`. The execution by this job will be delayed until the job currently carrying out the order has been completed.

5.11.16 state_text

Free text for the order state

Syntax: order. **`state_text`** = `String`

Syntax: `String` order. **`state_text`**

This text is shown on the HTML interface.

For non-API jobs the JobScheduler fills this field with the first line from stdout, up to a maximum of 100 characters.

5.11.17 string_next_start_time

The next start time of an order when `<run_time>` is being used

Syntax: `String` order. **`string_next_start_time`**

Returned value:
`String`

`"yyyy-mm-dd HH:MM:SS.MMM"` or `"now"` or `"never"`.

5.11.18 suspended

Suspended order

Syntax: order. **`suspended`** = `Boolean`

Syntax: `Boolean` order. **`suspended`**

A suspended order will not be executed.

When an order is being carried out by a task when it is suspended, then the `spooler_process()` step will be completed and the order allocated the successor state before being suspended.

This means that an order can be set to an end state, which stops it from being removed. The JobScheduler can remove such an order only when it is not suspended - i.e. `order.suspended=false`).

A suspended order with the end state can be allocated a different state corresponding to a job node in the job chain. This is effected by using `Order.state`. In this case the order remains suspended.

5.11.19 title

Optionally a title can be allocated to an order that will show up in the HTML interface and in the logs.

Syntax: order. **`title`** = `String`

Syntax: `String` order. **`title`**

5.11.20 web_service

The web service to which an order has been allocated

Syntax: `Web_service` order. **`web_service`**

When an order has not been allocated to a web service, then this call returns the `SCHEDULER-240` error.

See also `Order.web_service_or_null`.

Returned value:
`Web_service`

5.11.21 web_service_operation

The web service operation to which an order has been allocated

Syntax: `Web_service_operation` order. **`web_service_operation`**

Example: in java

```
public boolean spooler_process()  throws Exception
{
    Order                 order                 = spooler_task.order();
    Web_service_operation web_service_operation = order.web_service_operation();
    Web_service_request   request               = web_service_operation.request();

    // Decode request data
    String request_string = new String( request.binary_content(),
request.charset_name() );


    process request_string ...;


    String                response_string = "This is my response";
    String                charset_name    = "UTF-8";
    ByteArrayOutputStream byos            = new ByteArrayOutputStream();

    // Encode response data
    Writer writer = new OutputStreamWriter( byos, charset_name );
    writer.write( response_string );
    writer.close();

    // Respond
    Web_service_response response = web_service_operation.response();

    response.set_content_type( "text/plain" );
    response.set_charset_name( charset_name );
    response.set_binary_content( byos.toByteArray() );
    response.send();

    // Web service operation has finished

    return true;
}
```

See <web_service>, Web_service_operation and Order.web_service_operation_or_null,

Returned value:
Web_service_operation

5.11.22 web_service_operation_or_null

The web service operation to which an order has been allocated, or `null`

Syntax: Web_service_operation order. **web_service_operation_or_null**

See Order.web_service_operation, Web_service_operation and <web_service>.

Returned value:
Web_service_operation

5.11.23 web_service_or_null

The web service to which an order has been allocated, or `null`.

Syntax: Web_service order. **web_service_or_null**

See also Order.web_service.

Returned value:
Web_service

5.11.24 xml

Order in XML: <order>...</order>

Syntax: `String` order. **xml**

Returned value:
`String`

See <order>

5.11.25 xml_payload

XML payload - an order parameter.

Syntax: order. **xml_payload** = `String` xml

Syntax: `String` order. **xml_payload**

This property can include an XML document (in addition to the Order.params property).

<xml_payload> contains the XML document root element (instead of it being in `#PCDATA` coded form).

5.12 Order_queue - The order queue for an order controlled job

An order controlled job (<job order="yes"> has an order queue, which is filled by the orders to be processed by a job. The orders are sorted according to their priority and the time at which they enter the queue.

Processing means that the JobScheduler calls the spooler_process() method for a task. This method can access the order using the Task.order property. Should the `spooler_process()` end without an error (i.e. without any exceptions), then the JobScheduler removes the order from the order queue. If the order is in a job chain then it is moved to the next position in the chain.

5.12.1 length

The number of orders in the order queue

Syntax: `Integer` q. **`length`**

5.13 Process_class

See also <process_class name="">.

Example: in javascript

```
var process_classs = spooler.process_classs;
var process_class = process_classs.create_process_class();
process_class.name = "my_process_class";
process_classs.add_process_class( process_class );
```

5.13.1 max_processes

The maximum number of processes that are executed in parallel

Syntax: process_class. **`max_processes`** = `Integer`

Syntax: `Integer` process_class. **`max_processes`**

Should more tasks have to be started than allowed by this setting, then these tasks starts would be delayed until processes become freed. The default setting is 10.

See also <process_class max_processes="">.

5.13.2 name

The process class name

Syntax: process_class. **`name`** = `String`

Syntax: `String` process_class. **`name`**

The name can only be set once and may not be changed.

See also <process_class name="">.

5.13.3 remote_scheduler

The address of the remote JobScheduler, which is to execute a process

Syntax: process_class. **remote_scheduler** = String

Syntax: String process_class. **remote_scheduler**

Example: in javascript

```
spooler.process_classes.process_class( "my_process_class" ).remote_scheduler =
"host:4444";
```

See also <process_class remote_scheduler="">.

Parameters:

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

Returned value:
String

The address is specified in the form: " *host*: *portnumber*".

In addition, the IP address is returned on reading: " *hostname* / *ipnumber*: *portnumber*"

5.13.4 remove

Removal of the process class

Syntax: process_class. **remove** ()

Example: in javascript

```
spooler.process_classs.process_class( "my_process_class" ).remove();
```

The JobScheduler delays deletion of the process class as long as tasks are still running. No new tasks will be started before the class is deleted.

See also <process_class.remove>.

5.14 Process_classes

5.14.1 add_process_class

Adds a process class to the JobScheduler

Syntax: process_classs. **add_process_class** (Process_class pc)

5.14.2 create_process_class

Creates a new process class

Syntax: Process_class process_classs. **create_process_class** ()

Returnds a new Process_class. This class can be made added to the JobScheduler using Process_classes.add_process_class().

Returned value:
Process_class

5.14.3 process_class

Returns a process class

Syntax: Process_class process_classs. **process_class** (String process_class_name)

An exception will occur if the process class is not known.

Returned value:
Process_class

5.14.4 process_class_or_null

Returns a process class

Syntax: Process_class process_classs. **process_class_or_null** (String process_class_name)

Returned value:
Process_class

`null`, when the process class is not known.

5.15 Run_time - Managing Time Slots and Starting Times

See <run_time>, Order.Schedule.

Example: in javascript

```
var order = spooler_task.order;

// Repeat order daily at 15:00
order.run_time.xml = "<run_time><period single_start='15:00'/></run_time>";
```

5.15.1 schedule

`<schedule>`

Syntax: Schedule run_time. **schedule**

Returned value:
Schedule

5.15.2 xml

`<run_time>`

Syntax: run_time. **xml** = String

Discards the current setting and resets Run_time.

Parameters:

XML document as a string

5.16 Schedule - Runtime

See <schedule>, <run_time>, Spooler.schedule, Run_time.

Example: in javascript

```
spooler.schedule( "my_schedule" ).xml = "<schedule><period single_start='15:00'/><
/schedule>";
```

5.16.1 xml

`<schedule>`

Syntax: schedule. **xml** = String

Syntax: String schedule. **xml**

Deletes the previous setting and resets Schedule.

Parameters:

XML document as a string

Returned value:
String

XML document as a string

5.17 Spooler

There is only one class for this object: spooler_.

5.17.1 abort_immediately

Aborts the JobScheduler immediately

Syntax: spooler. **abort_immediately** ()

Stops the JobScheduler immediately. Jobs do not have the possibility of reacting.

The JobScheduler kills all tasks and the processes that were started using the Task. create_subprocess() method. The JobScheduler also kills processes for which a process ID has been stored using the Task. add_pid() method.

See <modify_spooler cmd="abort_immediately"> and JobScheduler Documentation.

5.17.2 abort_immediately_and_restart

Aborts the JobScheduler immediately and then restarts it.

Syntax: spooler. **abort_immediately_and_restart** ()

Similar to the Spooler. abort_immediately() method, only that the JobScheduler restarts itself after aborting. It reuses the command line parameters to do this.

See <modify_spooler cmd="abort_immediately_and_restart"> and JobScheduler Documentation.

5.17.3 add_job_chain

Syntax: spooler. **add_job_chain** (Job_chain chain)

Job_chain. orders_recoverable_=true causes the JobScheduler to load the orders for a job chain from the database.

See Spooler. create_job_chain() _. and <job_chain>_.

5.17.4 configuration_directory

Path of the Configuration Directory with hot folders

Syntax: `String` spooler. **`configuration_directory`**

```
<config configuration_directory="…">
```

5.17.5 create_job_chain

Syntax: `Job_chain` spooler. **`create_job_chain`** `()`

Returns a new `Job_chain` object. This job chain can be added to the JobScheduler using `Spooler.add_job_chain()` after it has been filled with jobs.

See `<job_chain>`.

Returned value:
`Job_chain`

5.17.6 create_order

Syntax: `Order` spooler. **`create_order`** `()`

Creates a new order. This order can be assigned to a job chain using the `Job_chain.add_order()` method.

Returned value:
`Order`

5.17.7 create_variable_set

Syntax: `Variable_set` spooler. **`create_variable_set`** `()`

Returned value:
`Variable_set`

5.17.8 create_xslt_stylesheet

Syntax: `Xslt_stylesheet` spooler. **`create_xslt_stylesheet`** `( String xml (optional) )`

Parameters:

`xml` Creates an XSLT style sheet as an XML string.

Returned value:
Xslt_stylesheet_

5.17.9 db_history_table_name

The name of the database table used for the job history

Syntax: String spooler. **db_history_table_name**

See also Spooler. db_history_table_name()

The factory. ini (section [spooler], entry db_history_table=…) setting is overwritten by this parameter.

5.17.10 db_name

The database path

Syntax: String spooler. **db_name**

The database connection string for the history. Should no value be specified here, then the files will be saved in .csv format. See factory. ini (section [spooler], entry history_file=…).

A simple file name ending in . mdb (e.g. scheduler. mdb) can also be specified here when the JobScheduler is running on Windows. The JobScheduler then uses a Microsoft MS Access database of this name, which is located in the protocol directory (see the option -log-dir_). Should such a database not exist, then the JobScheduler will create this database.

The JobScheduler automatically creates the tables necessary for this database.

The factory. ini (section [spooler], entry db=…) setting is overwritten by this parameter.

5.17.11 db_order_history_table_name

The name of the order history database table

Syntax: String spooler. **db_order_history_table_name**

See also Spooler. db_order_history_table_name()

The factory. ini (section [spooler], entry db_order_history_table=…) setting is overwritten by this parameter.

5.17.12 db_orders_table_name

The name of the database table used for orders

Syntax: `String` spooler. **`db_orders_table_name`**

See also `Spooler.db_orders_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_orders_table=…`) setting is overwritten by this parameter.

5.17.13 db_tasks_table_name

The name of the task database table

Syntax: `String` spooler. **`db_tasks_table_name`**

See also `Spooler.db_tasks_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_tasks_table=…`) setting is overwritten by this parameter.

5.17.14 db_variables_table_name

The name of the database table used by the JobScheduler for internal variables

Syntax: `String` spooler. **`db_variables_table_name`**

The JobScheduler records internal counters, for example, the ID of the next free task, in this database table.

See also `Spooler.db_variables_table_name()`

The `factory.ini` (section `[spooler]`, entry `db_variables_table=…`) setting is overwritten by this parameter.

5.17.15 directory

The working directory of the JobScheduler on starting

Syntax: `String` spooler. **`directory`**

Changes the Working Directory.

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

The -cd option has precedence over this parameter.

A task executed on a remote JobScheduler (<process class remote scheduler="">) returns the value for the remote Scheduler.

Returned value:
`String`

The directory ends on Unix with "/" and on Windows with "\".

5.17.16 execute_xml

Carries out XML commands

Syntax: `String` spooler. **`execute_xml`** `( String xml )`

Example: in javascript

```
spooler_log.info( spooler.execute_xml( "<show_state/>" ) );
```

Errors are returned as XML <ERROR> replies.

Parameters:

`xml` See JobScheduler Documentation.

Returned value:
`String`

Returns the answer to a command in XML format.

5.17.17 hostname

The name of the computer on which the JobScheduler is running.

Syntax: `String` spooler. **`hostname`**

5.17.18 id

The value of the command line `-id=` setting

Syntax: `String` spooler. **`id`**

The JobScheduler only selects elements in the XML configuration whose `spooler_id` attributes are either empty or set to the value given here.

When the JobScheduler ID is not specified here, then the JobScheduler ignores the `spooler_id=` XML attribute and selects all the elements in the XML configuration.

See, for example, `<config>`.

The `-id` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `id=…`) setting is overwritten by this parameter.

5.17.19 include_path

Returns the command line setting `-include-path=`.

Syntax: `String` spooler. **`include_path`**

The directory of the files which are to be included by the `<include>` element.

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

Environment variables (e.g. `$HOME`) are replaced by this attribute (see Settings which Allow Environment Variables to be Called).

The `-include-path` option has precedence over this parameter.

The `factory.ini` (section `[spooler]`, entry `include_path=…`) setting is overwritten by this parameter.

`<config include_path="…">`

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

5.17.20 ini_path

The value of the `-ini=` option (the name of the `factory.ini` file)

Syntax: `String` spooler. **`ini_path`**

A task executed on a remote JobScheduler (`<process_class remote_scheduler="">`) returns the value for the remote Scheduler.

See `-ini`, JobScheduler Documentation

5.17.21 is_service

Syntax: `Boolean` spooler. **`is_service`**

Returned value:
`Boolean`

is true, when the JobScheduler is running as a service (on Windows) or as a daemon (on Unix).

5.17.22 job

Returns a job

Syntax: `Job_` spooler. **`job`** `( String job_name )`

An exception is returned should the job name not be known.

Returned value:
`Job_`

5.17.23 job_chain

Returns a job chain

Syntax: `Job_chain_` spooler. **`job_chain`** `( String name )`

Should the name of the job chain not be known, then the JobScheduler returns an exception.

Returned value:
`Job_chain_`

5.17.24 job_chain_exists

Syntax: `Boolean` spooler. **`job_chain_exists`** `( String name )`

5.17.25 let_run_terminate_and_restart

Syntax: spooler. **`let_run_terminate_and_restart`** `()`

The JobScheduler ends all tasks (by calling the `Job_impl_` method) as soon as all orders have been completed and then stops itself. It will then be restarted under the same command line parameters.

See <modify_spooler cmd="let_run_terminate_and_restart"> and JobScheduler Documentation.

5.17.26 locks

Returns the locks

Syntax: Locks spooler. **locks**

Returned value:
Locks

5.17.27 log

The main log

Syntax: Log spooler. **log**

spooler_log() is usually used for this property.

Returned value:
Log

5.17.28 log_dir

Protocol directory

Syntax: String spooler. **log_dir**

The directory in which the JobScheduler writes log files.

log_dir= *stderr allows the JobScheduler to write log files to the standard output (stderr, normally the screen) .

A task executed on a remote JobScheduler (<process class remote_scheduler="">) returns the value for the remote Scheduler.

The -log-dir option has precedence over this parameter.

The factory.ini (section [spooler], entry log_dir=…) setting is overwritten by this parameter.

A task executed on a remote JobScheduler (<process class remote_scheduler="">) returns the value for the remote Scheduler.

5.17.29 param

The command line option `-param=`

Syntax: `String` spooler. **`param`**

Free text. This parameter can be read using `spooler. param`.

The `-param` option has precedence over this parameter.

The `factory. ini` (section `[ spooler]`, entry `param=…`) setting is overwritten by this parameter.

5.17.30 process_classes

Returns the process classes

Syntax: `Process_classes` spooler. **`process_classes`**

Returned value:
`Process_classes`

5.17.31 schedule

Returns the `Schedule` with the name specified or `null`

Syntax: `Schedule` spooler. **`schedule`** `( String path )`

Returned value:
`Schedule`

5.17.32 supervisor_client

Returns the Supervisor_client or `null`

Syntax: `Supervisor_client` spooler. **`supervisor_client`**

Returned value:
`Supervisor_client`

5.17.33 tcp_port

Port for HTTP and TCP commands for the JobScheduler

Syntax: `Integer` spooler. **`tcp_port`**

The JobScheduler can accept commands via a TCP port whilst it is running. The number of this port is set here - depending on the operating system - with a number between 2048 and 65535. The default value is 4444.

The JobScheduler operates a HTTP/HTML server on the same port, enabling it to be reached using a web browser - e.g. via http://localhost:4444.

The JobScheduler does not respond to the `tcp_port=0` default setting either with TCP or HTTP protocols. This setting can therefore be used to block a JobScheduler from being accessed - for example via TCP.

The `-tcp-port` option has precedence over this parameter.

`<config tcp_port="…">`

Returned value:
`Integer`

0, when no port is open.

5.17.34 terminate

The proper ending of the JobScheduler and all related tasks

Syntax: spooler. **`terminate`** `( Integer timeout (optional) , Boolean restart (optional) , boolean all_schedulers (optional) , boolean continue_exclusive_operation (optional) )`

Ends all tasks (by calling the `spooler_close()`) method and terminates the JobScheduler.

Should a time limit be specified, then the JobScheduler ends all processes still running after this limit has expired. (Typical processes are tasks which have remained too long in a method call such as `spooler_process()`.)

See `<modify_spooler cmd="terminate">` and JobScheduler Documentation.

Parameters:

`timeout`	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.
`restart`	`restart=true` allows the JobScheduler to restart after ending.
`all_schedulers`	`all_schedulers=true` ends all the JobSchedulers belonging to a cluster (see `-exclusive`). This may take a minute.
`continue_exclusive_operation`	`continue_exclusive_operation=true` causes another JobScheduler in the Cluster to take become active (see `-exclusive`).

5.17.35 terminate_and_restart

Correctly terminates the JobScheduler and all tasks before restarting

Syntax: spooler. **terminate_and_restart** (Integer timeout (optional))

Similar to the Spooler.terminate() method, but the JobScheduler restarts itself.

See <modify_spooler cmd="terminate_and_restart"> and JobScheduler Documentation.

Parameters:

timeout	The time in seconds which the JobScheduler allows for a task to end. After this time the JobScheduler stops all processes before stopping itself. If this parameter is not set then the JobScheduler will wait on tasks indefinitely.

5.17.36 udp_port

Port for UDP commands for the JobScheduler

Syntax: Integer spooler. **udp_port**

The JobScheduler can also accept UDP commands addressed to the port specified in this setting. Note that a UDP command must fit in a message and that the JobScheduler does not answer UDP commands.

The default value of udp_port=0 does not allow the JobScheduler to open a UDP port.

The -udp-port option has precedence over this parameter.

<config udp_port="…">

Returned value:
Integer

0, when no port is open.

5.17.37 var

Allows access to variables defined in the JobScheduler start script

Syntax: spooler. **var** (String name) = Variant

Syntax: Variant spooler. **var** (String name)

The variables are used by all JobScheduler job implementations.

5.17.38 variables

The JobScheduler variables as a Variable_set

Syntax: Variable_set spooler. **variables**

The variables can be set in the configuration file using <config>.

Returned value:
Variable_set

5.18 Spooler_program - Debugging Jobs in Java

Starts the JobScheduler using Java, so that jobs written in Java can be debugged (e.g. using Eclipse). See Javadoc for information about the methods.

The JobScheduler is started as a Windows application and not as a console program. Output to stderr is lost - standard output is shown in Eclipse. -log-dir shows no output.

See JobScheduler Documentation.

Example:

```
C:\>java -Djava.library.path=… -classpath …\sos.spooler.jar sos.spooler.Spooler_program
 configuration.scheduler -log-dir=c:\tmp\scheduler
```

Should the location of the scheduler.dll not be specified in %PATH% then it may be set using -Djava.library.path=….

5.19 Subprocess

A subprocess is a process which can be started using either Task.create_subprocess() or Subprocess.start()
.

Example: system() - the Simple Execution of a Command, in javascript

```
exit_code = my_system( "backup /" );

function system( cmd, timeout )
{
    var subprocess = spooler_task.create_subprocess();

    try
    {
        if( timeout )  subprocess.timeout = timeout;
        subprocess.start( cmd );
        subprocess.wait_for_termination();
        return subprocess.exit_code;
    }
    finally
    {
        subprocess.close();
    }
}
```

Example: in javascript

```
var subprocess = spooler_task.create_subprocess();

subprocess.environment( "test1" ) = "one";
subprocess.environment( "test2" ) = "two";
subprocess.ignore_error = true;

subprocess.start( "sleep 20" );

spooler_log.info( "pid=" + subprocess.pid );
subprocess.timeout = 10;

spooler_log.info( "wait_for_termination ..." );
var ok = subprocess.wait_for_termination( 10 );
spooler_log.info( "wait_for_termination ok=" + ok );

if( subprocess.terminated )
{
    spooler_log.info( "exit code=" + subprocess.exit_code );
    spooler_log.info( "termination signal=" + subprocess.termination_signal );
}
```

5.19.1 close

Frees system resources

Syntax: subprocess. **`close`** `()`

This method should only be called in language with a garbage collector (Java, JavaScript). In all other cases the task ends immediately.

Should this method have been called in a language with a garbage collector, then the `Subprocess` is no longer usable.

5.19.2 env

Environment Variables as Variable_sets

Syntax: `Variable_set` subprocess. **`env`**

Example: in javascript

```
var subprocess = spooler_task.create_subprocess();
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
subprocess.wait_for_termination();
```

Returns a `Variable_set` for the environment variables.

Initially the environment is filled by the environment variables from the calling process. Environment variables can be removed in that they are set to `""`. Calling `Subprocess.start()` hands over environment variables to the subprocess.

Note that the names of environment variables are case sensitive on UNIX systems.

Changes made to environment variables after the start of a subprocess have no effect. This is also true for environment variables changed by the process.

This object cannot be handed over to other objects - it is a part of the task process, whereas the majority of other objects are part of the JobScheduler process.

Returned value:
Variable_set_

5.19.3 environment

Environment variables

Syntax: subprocess. **environment** (String name) = String value

Example: in javascript

```
// The following two statements have the same effect
subprocess.environment( "my_variable" ) = "my_value"
subprocess.env.value( "my_variable" ) = "my_value"
```

Variables set here are handed over to a new subprocess together with any other environment variables belonging to the process.

Note that the names of environment variables are case sensitive on UNIX systems.

See also Subprocess.env_.

5.19.4 exit_code

Syntax: Integer subprocess. **exit_code**

Is only called after Subprocess.terminated_== true.

5.19.5 ignore_error

Prevents that a job is stopped, should exit_code != 0.

Syntax: subprocess. **ignore_error** = Boolean

Syntax: Boolean subprocess. **ignore_error**

Prevents a job from being stopped, when at the end of a task the subprocess ends with Subprocess.exit_code_!= 0.

Should a task not wait for the end of a subprocess with the Subprocess.wait_for_termination_method, then the JobScheduler waits at the end of the task for the end of any subprocesses. In this case the job is stopped with an error when a subprocess ends with Subprocess.exit_code_!= 0.

This may be avoided using `ignore_error`.

5.19.6 ignore_signal

Prevents a job from being stopped when the task is stopped with a UNIX signal.

Syntax: subprocess. **`ignore_signal =`** `Integer`

Syntax: `Integer` subprocess. **`ignore_signal`**

This property does not work on Windows systems, as this system does not support signals.

5.19.7 kill

Stops a subprocess

Syntax: subprocess. **`kill`** `( Integer signal (optional) )`

Parameters:

`signal` Only on UNIX systems: The `kill()` signal. 0 is interpreted here as 9 (`SIGKILL`, immediate ending).

5.19.8 own_process_group

Subprocesses as a Process Group

Syntax: subprocess. **`own_process_group =`** `Boolean`

Syntax: `Boolean` subprocess. **`own_process_group`**

Only available for UNIX systems.

The default setting can be made using `factory.ini` (section `[spooler]`, entry `subprocess.own_process_group=…`).

`own_process_group` allows a subprocess to run in its own process group, by executing the `setpgid(0,0)` system call. When the JobScheduler then stops the subprocess, then it stops the complete process group.

5.19.9 pid

Process identification

Syntax: `Integer` subprocess. **`pid`**

5.19.10 priority

Process Priority

Syntax: subprocess. **priority** = Integer

Syntax: Integer subprocess. **priority**

Example: in javascript

```
spooler_task.priority = +5;     // UNIX: reduce the priority a little
```

UNIX: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 "idle", 6 "below_normal", 8 "normal", 10 "above_normal" and 13 "high" (other values are rounded down). See also Task.priority_class.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Subprocess.priority_class. See also Task.priority.

5.19.11 priority_class

Priority Class

Syntax: subprocess. **priority_class** = String

Syntax: String subprocess. **priority_class**

Example: in javascript

```
subprocess.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and UNIX Systems:

Priority Class	Windows	UNIX
"idle"	4	16
"below_normal"	6	6
"normal"	8	0
"above_normal"	10	-6
"high"	13	-16

Note that when it is not possible to set a priority for a task - for example, because of inappropriate permissions - then this must not cause an error. On the other hand, an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Subprocess. priority_, Task. priority_class_and Microsoft® Windows® Scheduling Priorities.

5.19.12 start

Starts the process

Syntax: subprocess. **start** (String| String[] command_line)

Windows immediately detects whether the program cannot be executed. In this case the method returns an error.

On UNIX systems the Subprocess. exit_code_property is set to 99. Before this is done, the end of the process must be waited on with Subprocess. wait_for_termination() _.

Shell operators such as | , && and > are not interpreted. The /bin/sh or c: \windows\system32\cmd. exe programs must be used to do this. (Note that the actual paths will depend on the installation.)

This process is started on UNIX systems using execvp() and with CreateProcess() on Windows systems.

5.19.13 terminated

Syntax: Boolean subprocess. **terminated**

Verifies that a process has ended. Should the process in question have ended, then the Subprocess. exit_code and Subprocess. termination_signal_classes may be called.

5.19.14 termination_signal

Signal with which a process (only on UNIX systems) ends

Syntax: Integer subprocess. **termination_signal**

Is only called, after Subprocess. terminated_== true.

5.19.15 timeout

Time limit for a subprocess

Syntax: subprocess. **timeout =** Double seconds

After the time allowed, the JobScheduler stops the subprocess (UNIX: with SIGKILL).

This time limit does not apply to processes running on remote computers with <process_class remote_scheduler="">_.

5.19.16 wait_for_termination

Syntax: subprocess. **wait_for_termination** ()

Syntax: Boolean subprocess. **wait_for_termination** (Double seconds)

Parameters:

seconds	Waiting time. Should this parameter not be specified, then the call will take place after the subprocess has ended.

Returned value:
Boolean

true, after a subprocess has ended.
false, should the subprocess continue beyond the waiting time.

5.20 Supervisor_client

This object is returned by Spooler. supervisor_client_.

Example: in javascript

```
var supervisor_hostname = spooler. supervisor_client. hostname;
```

5.20.1 hostname

The name or IPnumber of the host computer on which the suupervising JobScheduler is running

Syntax: String supervisor_client. **hostname**

See also <config supervisor="">_.

5.20.2 tcp_port

the TCP port of the supervisor

Syntax: Integer supervisor_client. **tcp_port**

See also <config supervisor="">_.

5.21 Task

A task is an instance of a job which is currently running.

A task can either be waiting in a job queue or being carried out.

5.21.1 add_pid

Makes an independent, temporary process known to the JobScheduler

Syntax: spooler_task. **add_pid** (Integer pid, String| Double| Integer timeout (optional))

This call is used to restrict the time allowed for processes that have been launched by a task. The JobScheduler ends all independent processes still running at the end of a task.

A log entry is made each time the JobScheduler stops a process. This does not affect the state of a task.

The <kill_task> method stops all processes for which the add_pid() method has been called.

A process group ID can be handed over on Unix systems as a negative pid. kill then stops the complete process group.

This time limit does not apply for processes being run on remote computers with <process_class remote_scheduler="">.

5.21.2 call_me_again_when_locks_available

Repeats spooler_open() or spooler_process() as soon as locks become available

Syntax: spooler_task. **call_me_again_when_locks_available** ()

Causes the JobScheduler to repeat a call of spooler_open() or spooler_process(), after an unsuccessful Task.try_hold_lock() or Task.try_hold_lock_non_exclusive() as soon as the locks required are available. The JobScheduler then repeats the call once it holds the locks, so that the first call (i.e. spooler_open()) will be successful.

After this call, true/false values returned by spooler_open() or spooler_process() has no effect. The JobScheduler leaves the state of the Task.order unchanged.

5.21.3 changed_directories

The directory in which the change which started a task occurred

Syntax: String spooler_task. **changed_directories**

See Job.start_when_directory_changed(), Task.trigger_files.

Returned value:
String

Directory names are to be separated using a semicolon.

"", should no change have occurred in a directory.

5.21.4 create_subprocess

Starts a monitored subprocess

Syntax: _Subprocess_ spooler_task. **create_subprocess** (String| String[] filename_and_arguments (optional))

Returned value:
Subprocess_

5.21.5 delay_spooler_process

Delays the next call of spooler_process()

Syntax: spooler_task. **delay_spooler_process** = String|Double|Integer seconds_or_hhmm_ss

Only functions in spooler_process() .

5.21.6 end

Ends a task

Syntax: spooler_task. **end** ()

The JobScheduler no longer calls the spooler_process() method. Instead the spooler_close() method is called.

This method call can be used at the end of a task to trigger sending a task log. See Log_.

5.21.7 error

Sets an error and stops the current job

Syntax: spooler_task. **error** = String

Syntax: _Error_ spooler_task. **error**

This method call returns the last error which has occurred with the current task. Should no error have occurred, an Error_object is returned, with the is_error property set to false.

An error message can also be written in the task log file using Log. error()

Returned value:
String Error_

5.21.8 exit_code

Exit-Code

Syntax: spooler_task. **exit_code** = Integer

Syntax: Integer spooler_task. **exit_code**

Example: in javascript

```
spooler_log.error( "This call of spooler_log.error() sets the exit code to 1" );
spooler_task.exit_code = 0;   // Reset the exit code
```

The initial exit-code value is 0 - this is changed to 1 should an error occur. Note that an error is defined here as occurring when the JobScheduler writes a line in the task log containing "[ERROR] ":

- calling the Log. error( ) method;
- setting the Task. error property;
- the script returns an exception.

The job can then set the Task. exit_code property - e.g. in the spooler_on_error( ) method.

The exit code resulting from an operating system process executing a task is not relevant here and, in contrast to jobs with <process> or <script language="shell">, is not automatically handed over to this property.

The exit code determines the commands to be subsequently carried out. See <job> <commands on_exit_code="" > for more information.

The exit codes have no influence for API jobs on whether or not a job is stopped (a task error message causes jobs to be stopped).

5.21.9 history_field

A field in the task history

Syntax: spooler_task. **history_field** (String name) = Variant value

Example: in javascript

```
spooler_task.history_field( "extra" ) = 4711;
```

The database table (see factory. ini (section [spooler], entry db_history_table=…)) must have a column with this name and have been declared in the factory. ini (section [job], entry history_columns=…) file.

5.21.10 id

The task identifier

Syntax: Integer spooler_task. **id**

The unique numerical identifier of every task run by a JobScheduler.

5.21.11 job

The job which a task belongs to

Syntax: Job spooler_task. **job**

Returned value:
Job

5.21.12 order

The current order

Syntax: Order spooler_task. **order**

Example: in javascript

```
var order = spooler_task.order;

spooler_log.info( "order.id=" + order.id + ", order.title=" + order.title );
```

Returned value:
Order

`null`, should no order exist.

5.21.13 params

The task parameters

Syntax: Variable_set spooler_task. **params**

Example: in javascript

```
var value = spooler_task.params.value( "parameter3" );
```

Example: in javascript

```
var parameters = spooler_task.params;
if( parameters.count > 0 )  spooler_log.info( "Parameters given" );

var value1 = parameters.value( "parameter1" );
var value2 = parameters.value( "parameter2" );
```

A task can have parameters. These parameters can be set using:

- <params> in the <job> element in the configuration file;
- Job.start() and

- <start_job>.

Returned value:
Variable_set

!= null

5.21.14 priority

Priority of the Current Task

Syntax: spooler_task. **priority** = Integer

Syntax: Integer spooler_task. **priority**

Example: in javascript

```
spooler_task.priority = +5;    // Unix: reduce the priority a little
```

Unix: The highest priority is -20, the lowest 20. The priority of a task can generally only be reduced and not increased.

The following priority classes are available on Windows systems 4 "idle", 6 "below_normal", 8 "normal", 10 "above_normal" and 13 "high" (other values are rounded down). See also Task.priority_class.

Note that an error does not occur, should it not be possible to set the priority of a task.

Note also that a process with a higher priority can block a computer.

The priority of a task can be set independently of the operating system with Task.priority_class.

5.21.15 priority_class

Priority Class of the Current Class

Syntax: spooler_task. **priority_class** = String

Syntax: String spooler_task. **priority_class**

Example: in javascript

```
spooler_task.priority_class = "below_normal";
```

The following priority classes can be used to set priorities on Windows and Unix Systems:

Priority Class	Windows	Unix
"idle"	4	16
"below_normal"	6	6
"normal"	8	0

"above_normal"	10	-6
"high"	13	-16

Note that an error will occur should it be attempted to allocate a task a priority class not listed here.

Note also that a higher priority process can block a computer.

See also Task. priority, Subprocess. priority_class and Microsoft® Windows® Scheduling Priorities.

5.21.16 remove_pid

The opposite to add_pid()

Syntax: spooler_task. **remove_pid** (Integer pid)

An error does not occur when the pid has not been added using Task.

See Task. add_pid().

5.21.17 repeat

Restarts a task after the specified time

Syntax: spooler_task. **repeat** = Double

(This method actually belongs to the Job class and has nothing to do with the task currently being processed.)

Should there be no task belonging to the current job running after the time specified has expired, then the JobScheduler starts a new task. Note that the <run_time> element is considered here, and that the <period repeat=""> attribute may be temporarily ignored.

Job. delay_after_error has priority, should a task return an error.

5.21.18 stderr_path

The path to the file in which stderr task output is captured

Syntax: String spooler_task. **stderr_path**

Text in stderr is currently interpreted in the ISO-8859-1 character set.

Returned value:
String

"", should a task not run in a separate <process_classes> process.

5.21.19 stderr_text

Text written to `stderr` up to this point by the process that was started by the task.

Syntax: `String` spooler_task. **`stderr_text`**

Text in `stderr` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`String`

`""`, should the task not have been started in a separate process <process_classes>.

5.21.20 stdout_path

The path of the file in which `stdout` task output is captured

Syntax: `String` spooler_task. **`stdout_path`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`String`

`""`, should a task not run in a separate <process_classes> process.

5.21.21 stdout_text

Text written to `stdout` up to this point by the process that was started by the task.

Syntax: `String` spooler_task. **`stdout_text`**

Text in `stdout` is currently interpreted in the ISO-8859-1 character set.

Returned value:
`String`

`""`, should a task not run in a separate <process_classes> process.

5.21.22 trigger_files

File paths in folders monitored with regex

Syntax: `String` spooler_task. **`trigger_files`**

Returns the file paths from monitored directories (_Job.start_when_directory_changed() or <start_when_directory_changed>) at the time a task is started. Only applies to directories for which a regular expression has been defined (`regex`).

The paths are taken from the addresses defined in Job.start_when_directory_changed() or <start_when_directory_changed> and combined with the file names.

The non-API <process> and <script language="shell"> jobs make the content of Task.trigger_files available to the SCHEDULER_TASK_TRIGGER_FILES environment variable.

See Job.start_when_directory_changed() and Task.changed_directories().

Returned value:
String

The file paths are separated by semicolons.

"" otherwise

5.21.23 try_hold_lock

Try to hold a lock

Syntax: boolean spooler_task. **try_hold_lock** (String lock_path)

Example: in javascript

```
function spooler_process()
{
    var result = false;

    if( spooler_task.try_hold_lock( "Georgien" )  &&
        spooler_task.try_hold_lock_non_exlusive( "Venezuela" ) )
    {
        // Task is holding the two locks. Insert processing code here.
        result = ...
    }
    else
    {
        spooler_task.call_me_again_when_locks_available();
    }

    return result;
}
```

try_lock_hold() attempts to retain the lock specified (Lock), and can be called in:

- spooler_open(): the lock is held for the task being carried out and will be freed after the task has been completed,
- spooler_process(): the lock is only held for the job step currently being carried out and will be given up after the step has been completed - i.e. after leaving spooler_process().

When the lock is not available and calling this method returns false then the JobScheduler can be instructed to either:

- repeat the spooler_open() or spooler_process() calls as soon as the locks are available using Task.call_me_again_when_locks_available() or
- end spooler_open() or spooler_process() with false, without use of the above-mentioned call, (but with the expected effect),
- throw a SCHEDULER-469 warning. This applies for true, which is interpreted as an error.

See also <lock.use>.

Returned value:
boolean

true, when the task retains the lock.

5.21.24 try_hold_lock_non_exclusive

Tries to acquire a non-exclusive lock

Syntax: boolean spooler_task. **try_hold_lock_non_exclusive** (String lock_path)

The same prerequisites apply as to Task.try_hold_lock().

See <lock.use_exclusive="no">.

Returned value:
boolean

true, if the task successfully acquired the lock.

5.21.25 web_service

The Web Service which a task has been allocated to.

Syntax: Web_service spooler_task. **web_service**

This property causes an exception when a task has not been allocated to a Web Service.

See also Task.web_service_or_null.

Returned value:
Web_service

5.21.26 web_service_or_null

The Web Service to which a task has been allocated, or null.

Syntax: Web_service spooler_task. **web_service_or_null**

See also Task.web_service.

Returned value:
Web_service

5.22 Variable_set - A Variable_set may be used to pass parameters

Variable_set is used for the JobScheduler variables and task parameters. A new Variable_set is created using `Spooler.create_variable_set()`.

Variable names are case independent.

The value of a variable is known as a variant in the COM interface (JavaScript, VBScript, Perl). Because variables are usually written in the JobScheduler database, only variant types which can be converted into strings should be used here.

The value of a variable in Java is a string. Therefore, a string value is returned when reading this variable, when it is set as a variant in the COM interface. `Null` and `Empty` are returned as `null`. An error is caused should the value of a variant not be convertible.

5.22.1 count

The number of variables

Syntax: `Integer` variable_set. **`count`**

5.22.2 merge

Merges with values from another Variable_set

Syntax: variable_set. **`merge`** `(` `Variable_set` `vs` `)`

Variables with the same name are overwritten.

5.22.3 names

The separation of variable names by semicolons

Syntax: `String` variable_set. **`names`**

Example: in javascript

```
var variable_set = spooler.create_variable_set();
spooler_log.info( '"' + variable_set.names + '"' );        // ==> ""

variable_set( "variable_1" ) = "edno";
variable_set( "variable_2" ) = "dwa";

spooler_log.info( '"' + variable_set.names + '"' );        // ==>
"variable_1;variable_2"

var names = variable_set.names.split( ";" );
for( var i in names )  spooler_log.info( names[i] + "=" + variable_set( names[i] ) );
```

Returned value:
`String`

All variable names should be separated by semicolons.

5.22.4 set_var

Sets a variable

Syntax: variable_set. **`set_var`** `( String name, Variant value )`

5.22.5 substitute

Replaces $-Variables in a String

Syntax: `String` variable_set. **`substitute`** `( String sustitution_string )`

Example: in javascript

```
subprocess.start( subprocess.env.substitute( "${MY_HOME}/my_program" ) );
```

In the example below, the `Subprocess.env` method is used.

References in the string in the form `$` *name* and `${` *name* `}` are replaced by variables.

Returned value:
`String`

The string containing the substituted $ variables.

5.22.6 value

A variable

Syntax: variable_set. **`value`** `( String name )` **=** `Variant value`

Syntax: `Variant` variable_set. **`value`** `( String name )`

Parameters:

`name`	
`value`	`empty`, should a variable not exist.

Returned value:
`Variant`

`empty`, should a variable not exist.

5.22.7 var

A variable

Syntax: variable_set. **var** (String name) = Variant value

Syntax: Variant variable_set. **var** (String name)

Use the Variable_set.value_, which is available in all languages.

Parameters:

name

value — empty, should a variable not exist.

Returned value:
Variant

empty, should a variable not exist.

5.22.8 xml

Variable_set as an XML document

Syntax: variable_set. **xml** = String

Syntax: String variable_set. **xml**

Example: in javascript

```
var variable_set = spooler.create_variable_set();
spooler_log.info( variable_set.xml );   // Liefert <?xml version='1.0'?><
sos.spooler.variable_set/>

variable_set.xml= "<?xml version='1.0'?>" +
                  "<params>" +
                      "<param name='surname' value='Meier'/>" +
                      "<param name='christian name' value='Hans'/>" +
                  "</params>";
spooler_log.info( variable_set.xml );
spooler_log.info( "nachname=" + variable_set.value( "surname" ) );
spooler_log.info( "vorname =" + variable_set.value( "christian name" ) );
```

See <sos.spooler.variable_set>_, <params>_.

Parameters:

XML document as a string. Returns <sos.spooler.variable_set>_. When setting this property to an XML value, then the name of the root element is ignored; <params>_ or <sos.spooler.variable_set>_ may be returned.

Returned value:
`String`

XML document as a string. Returns <sos.spooler.variable_set>. When setting this property to an XML value, then the name of the root element is ignored; <params> or <sos.spooler.variable_set> may be returned.

5.23 Web_service

See also <web_service>

5.23.1 forward_xslt_stylesheet_path

Path to the forwarding XSLT stylesheets

Syntax: `String` web_service. **`forward_xslt_stylesheet_path`**

See also <web_service forward_xslt_stylesheet="">

5.23.2 name

The Name of the JobScheduler Web Service

Syntax: `String` web_service. **`name`**

See also <web_service name="">

5.23.3 params

Freely definable parameters

Syntax: _Variable_set_ web_service. **`params`**

The Web Services parameters can be set using the <web_service> element.

Returned value:
Variable_set_

5.24 Web_service_operation

See also <web_service>

5.24.1 peer_hostname

Peer (Remote) Host Name

Syntax: `String` web_service_operation. **`peer_hostname`**

Returned value:
`String`

"", should it not be possible to determine the name.

5.24.2 peer_ip

Peer (Remote) IP Address

Syntax: `String` web_service_operation. **`peer_ip`**

5.24.3 request

Requests

Syntax: `_Web_service_request_` web_service_operation. **`request`**

Returned value:
`Web_service_request_`

5.24.4 response

Answers

Syntax: `_Web_service_response_` web_service_operation. **`response`**

Returned value:
`Web_service_response_`

5.24.5 web_service

Syntax: `_Web_service_` web_service_operation. **`web_service`**

Returned value:

Web_service_

5.25 Web_service_request

See Web_service_operation_.

5.25.1 binary_content

Payload as a Byte Array (Java only)

Syntax: web_service_request. **binary_content**

This property is only available under Java.

The ("Content-Type") header field is used to inform the client how binary content is to be interpreted (see HTTP/1.1 14.17 Content-Type) and Web_service_request.charset_name_).

5.25.2 charset_name

Character Set

Syntax: String web_service_request. **charset_name**

Example: in javascript

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Returns the charset= parameter from the Content-Type: header entry.

5.25.3 content_type

Content Type (without parameters)

Syntax: String web_service_request. **content_type**

Returns the Content-Type: header entry, without parameters - e.g. "text/plain".

5.25.4 header

Header Entries

Syntax: `String` web_service_request. **`header`** `( String name )`

Example: in javascript

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.request.header( "Content-Type" ) );
```

Parameters:

`name` Case is not relevant.

Returned value:
`String`

Returns `""` in event of an unrecognized entry.

5.25.5 string_content

Payload as Text

Syntax: `String` web_service_request. **`string_content`**

The character set to be used is taken from the `charset` parameter in the `headers("Content-Type")` (see HTTP/1.1 14.17 Content-Type). ISO-8859-1 will be used as default, should this parameter not be specified.

The following character sets are recognized:

- ISO-8859-1
- UTF-8 (only on Windows systems and restricted to the ISO-8859-1 characters)

See also `Web_service_request.binary_content`.

5.25.6 url

Uniform Resource Locator

Syntax: `String` web_service_request. **`url`**

`url = "http://" + header( "Host" ) +` *url_path*

5.26 Web_service_response

Note that the `binary_content` property is only available under Java.

See also <web_service>

5.26.1 charset_name

Character set

Syntax: String web_service_response. **charset_name**

Example: in javascript

```
var request = spooler_task.order.web_service_operation.request;

spooler_log.info( request.header( "Content-Type" ) );    // ==> text/xml; charset=utf-8
spooler_log.info( request.content_type );                // ==> text/xml
spooler_log.info( request.charset_name );                // ==> utf-8
```

Reads the charset= parameter from the Content-Type: header entry.

5.26.2 content_type

Content-Type (without parameters)

Syntax: String web_service_response. **content_type**

Reads the Content-Type: header without any of the other associated parameters such as charset=.

5.26.3 header

Header Entries

Syntax: web_service_response. **header** (String name) = String value

Syntax: String web_service_response. **header** (String name)

Example: in javascript

```
spooler_log.info( "Content-Type: " +
spooler_task.order.web_service_operation.response.header( "Content-Type" ) );
```

Parameters:

value	"" is used for unknown entries.
name	The case in which entries are written is not relevant here.

Returned value:
String

"" is used for unknown entries.

5.26.4 send

Sends a Reply

Syntax: web_service_response. **send** ()

5.26.5 status_code

HTTP Status Code

Syntax: web_service_response. **status_code** = Integer

The default setting is 200 (OK).

5.26.6 string_content

Text payloads

Syntax: web_service_response. **string_content** = String text

Example: in javascript

```
var response = spooler_task.order.web_service_operation.response;
response.content_type = "text/plain";
response.charset_name = "iso-8859-1";
response.string_content = "This is the answer";
response.send();
```

The `header( "Content-Type" )` must first of all contain a `charset` parameter such as:

```
header( "Content-Type" ) = "text/plain; charset=iso-8859-1";
```

Text is coded as specified in the `charset` parameter. ISO-8859-1 will be used as the default value, should this parameter not be specified.

See `Web_service_request.string_content` for the character sets which are allowed.

See `Web_service_response.charset_name`.

5.27 Xslt_stylesheet

An XSLT style sheet contains the instructions for the transformation of an XML document.

The XSLT processor is implemented with libxslt .

5.27.1 apply_xml

Applies a style sheet to an XML document.

Syntax: `String x. apply_xml ( String xml )`

5.27.2 close

Frees the style sheet resources

Syntax: `x. close ()`

5.27.3 load_file

Loads the style sheet from an XML file

Syntax: `x. load_file ( String path )`

5.27.4 load_xml

Loads the style sheet from an XML document

Syntax: `x. load_xml ( String xml )`

Index