Managed Jobs

JOB SCHEDULER

Technical Documentation
March 2009
Contact Information

Software- und Organisations-Service GmbH
Giesebrechtstr. 15
10629 Berlin
Germany

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

Last Updated: March 2009
Table of Contents

1 Introduction ................................................................. 5

2 Managed Jobs Versions .................................................. 6
  2.1 Release 1.3.1 Onwards .............................................. 6
  2.2 Release 1.3.5 Onwards .............................................. 7

3 The Managed Jobs Concept ............................................ 9
  3.1 Managed Job Starter ............................................... 11
  3.2 Configuration of a Single Job Scheduler ....................... 12
  3.3 Configuration of Multiple Job Schedulers ...................... 12

4 Database ........................................................................ 13
  4.1 Configuration .......................................................... 13
  4.2 Database Tables ....................................................... 13
  4.3 Database Table Content after Installation ..................... 14

5 Workflow Management .................................................. 16
  5.1 Example Job: Execution of a Shell Script ...................... 17
  5.2 Explorer ................................................................. 18
    5.2.1 Symbols Used in the Object Explorer ..................... 19
    5.2.2 Object Statuses and their Symbols ...................... 19
    5.2.3 Navigation Within the Explorer Tree ..................... 20
  5.3 The Context Menu Functions ..................................... 21
    5.3.1 Renaming Objects ............................................. 21
    5.3.2 Creating a New Object ....................................... 22
    5.3.3 Import Using the HTTP Upload Dialog .................... 22
    5.3.4 Copying Objects ............................................... 23
    5.3.5 Moving Objects ................................................. 24
    5.3.6 Exporting Objects Using the HTTP-Download Dialog .... 24
    5.3.7 Deleting Objects ............................................... 25
    5.3.8 Object Properties ............................................. 26
    5.3.9 Activation ....................................................... 26
    5.3.10 Deactivation ................................................... 26
    5.3.11 Additional Functions Available Within a Job Scheduler Hot Folder .. 27
  5.4 Permissions ............................................................ 27

5.5 Editor ....................................................................... 28
    5.5.1 Importing and Exporting Using the Clipboard (XML Editor View) .. 29
    5.5.2 Set Job & Order Types ....................................... 30
    5.5.3 Editing the Object Configuration ............................ 31
    5.5.4 Run Time Editor ................................................. 31
    5.5.5 Job and Order Parameter Management ...................... 33
    5.5.6 Scripts and Monitor Scripts in Jobs ....................... 34
    5.5.7 Commands Editor ............................................... 37
    5.5.8 Job Chain Editor ............................................... 37
    5.5.9 Buttons ............................................................ 38

5.6 Predefined Objects .................................................... 38
  5.7 Object Configuration ................................................ 39
    5.7.1 Locks ............................................................... 39
    5.7.2 Process Classes ................................................ 40
    5.7.3 Jobs ................................................................. 41
    5.7.4 Job Chains ....................................................... 42
    5.7.5 Orders ............................................................. 44
    5.7.6 Schedule .......................................................... 45

5.8 Job and Order Types ................................................ 46
  5.9 Database Connections ............................................ 48

6 Settings ........................................................................ 51
1 Introduction

The use of Managed Jobs is recommended in the following situations:

- when order controlled job chains are to be processed,
- when a number of Job Schedulers running on different servers are to be used.

The subject of this documentation is the Managed Jobs concept and the web interface for the administration of the database tables associated with Managed Jobs.

It is recommended that this documentation be fully read before starting the installation and configuration of Managed Jobs.

A knowledge of the Job Scheduler reference documentation is seen as a prerequisite for using Managed Jobs.

Managed Jobs are not configured using the Job Scheduler ./config/scheduler.xml XML configuration file but using a web interface. All relevant information is stored in a database and administered using the web interface whilst the Job Scheduler is running.

Managed Jobs - including the necessary Web Interface - are installed during a new installation of the Job Scheduler by selecting the Managed Jobs and Web Interface packages. Should it be necessary to add the Managed Jobs and Web Interface packages to an already existing Job Scheduler installation, then this can be done by repeating the Job Scheduler installation and selecting the Managed Jobs and Web Interface packages. Further details about installing the Job Scheduler can be found in the Job Scheduler Installation and Configuration documentation.

Note that the Managed Jobs web interface requires use of a web server that supports PHP in Version 4.3 or newer. However, all the features of the web interface will only be available when PHP 5.0 or higher is installed.

A database is required for Managed Jobs. The following database management systems are supported:

- MySQL 4.x, 5.x
- PostgreSQL 8.x
- Firebird 1.5
- Oracle 8.1.7, 9.2, 10g
- SQL Server 2000, 2005
- IBM DB2 8.x
- Sybase 15.x
2 Managed Jobs Versions

2.1 Release 1.3.1 Onwards

Managed Jobs were re-implemented with Release 1.3.1 of the Job Scheduler, acquiring a new interface, data model and Java classes. However, this chapter is also intended for people already using a version of Managed Jobs from an older Job Scheduler release - i.e. before Scheduler version 1.3.1. New users of Managed Jobs and those already having updated can jump to the next chapter.

Should you be using the Job Scheduler in a Release after 1.3.1 without having updated the Managed Jobs package, then you will still have the updated ./lib/sos_scheduler.jar library. In order to be able to use this library with Managed Jobs in the version before the Job Scheduler 1.3.1 release, then you need to change the ./config/scheduler_managed.xml as follows:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<spooler>
  <config>
    <script language="java"
      java_class="sos.scheduler.managed.JobSchedulerManagedStarter_1"/>
    <jobs>
      <job name="schedulerManagedStarter"
        title="start managed jobs">
        <description>
          <include file="jobs/JobSchedulerManagedStarter.xml"/>
        </description>
        <script language="java"
          process_class="single"
          java_class="sos.scheduler.managed.JobSchedulerManagedStarter_1"/>
      </job>
    </jobs>
  </config>
</spooler>
```

Please note that the class names now end with a "_1".

Should you be using the 1.3.1 Release of the Job Scheduler (or newer) with the actualised Managed Jobs, then because of the modified data model, your jobs, orders and job-chains will not function. However, you can convert 'old' Managed Jobs to the new data model using the ./bin/ManagedConverter.(sh|cmd) shell script. This script is called as follows:
Example: Calling the Managed Converter Script.(sh|cmd)
Parameters:
- **-target=** path to the factory.ini file, in which the database connection for the "new" data model is to be found.
- **-source=** path to the factory.ini file, in which the database connection for the "old" data model is to be found. (Note that this only need to be specified, when the value is different to that of -target)
- **-v =** Log-Level (optional) [0=info] [1=debug1]...[9=debug9]
- **-log =** log file (optional)
- **-folder=** Name of a folder which will be created below /Home in the Managed Jobs interface for the imported objects (default value: Import) (optional)

Unix-call: shell>bin/ManagedConverter.sh -target=config/factory.ini

Windows-call: shell>bin\ManagedConverter.cmd -target=config\factory.ini

After the script has run, the imported objects can be conveniently moved (page 24) in the interface.

2.2 Release 1.3.5 Onwards

With the Release 1.3.5 of the *Job Scheduler*, Managed Jobs come with an extended user interface, data model and new Java classes. Should you be using Managed Jobs from an older *Job Scheduler* release than 1.3.1, then we recommend that you read the previous section (page 6).

Should you be using a 1.3.5 or newer version of the *Job Scheduler* **without** having updated the *Managed Jobs* packet, then this new release will not affect you.

Should you be using version 1.3.5 of the *Job Scheduler* **and** have updated the *Managed Jobs* packet, then you will not be able to use the *Managed Jobs* interface without migrating the database tables. You will not be able to assign any interface objects to the *Job Scheduler*.

If you have installed the *Managed Jobs* packet as part of an update of the *Job Scheduler*, then you will need to add the APP_SCHEDULER_ID constant to the .//web/custom/custom.inc.php file as follows:

```php
<?php
...
// monitoring scheduler id
if(!defined('APP_SCHEDULER_ID')) {define('APP_SCHEDULER_ID','[scheduler id']);}
...
?>
```

The ID of your *Job Schedulers* is specified in the .//bin/jobscheduler.(cmd|sh) start script.

Migration is carried out as follows:

Call the Object-Explorer using the *Managed Jobs* interface. The Object-Explorer will show a /([your scheduler id]) directory, which is new with version 1.3.5. Objects can now only be assigned to a *Job Scheduler* when they lie in this new "Scheduler directory". Export your objects from your "old" /Home/[/user_name] directory, by calling the "Export" (page 24) entry in the "old" directory context menu (right-hand mouse-click). This opens a download dialog, which you then use to save an archive of the old directory. This archive will contain all objects from your /Home/[/user_name] directory that have not been suspended. Now select the new /([your scheduler id]/live directory in the Object Explorer and use the import function (page 22) in the context menu to import the archive exported from the /Home/[/user_name] directory. The archive objects will be saved in the /([your scheduler id]/live folder. You can now hand over the objects to the *Job Scheduler*, either individually or as a group, by calling the "Functions->Activate" (page 26) function from the object context menu.

However, it may be necessary to reconfigure objects before they can be handed over to the *Job Scheduler*. The following points should be observed:
• Orders must lie in the same directory as your job chains
• Objects, with attributes that reference other objects, can now be found using relative paths.
  Example: job chain node jobs are referenced in a job chain. Up till now, these were specified without a path.
  When the job lies in the same directory as the job chain, then nothing needs to be changed, otherwise the job
  must be named relative to the job chain.

Should you wish to reset the Managed Jobs interface to the version before 1.3.5, then start the scheduler_[os]
_update.1.3.4.(tar.gz|zip) update and select the "Web Interface" and "Web Interface for Managed Jobs" web
packets.
3 The Managed Jobs Concept

In its standard setup, the Job Scheduler uses an XML configuration file (.config/scheduler.xml). Alongside the basic settings for the Job Scheduler itself, this file contains job definition and configuration information about objects, such as jobs, job chains, orders, etc.. The information contained in this XML file is static and is not changed as the Job Scheduler runs. Orders for order controlled jobs are handed over to the Job Scheduler when it is running. Once an order has been successfully completed, it is then deleted from the system. Changes made to the .config/scheduler.xml file only take effect once the Job Scheduler is restarted.

Changes made to the configuration during runtime without restarting the Job Scheduler can, however, be carried out using Hot Folders. These are directories that are monitored by the Job Scheduler and in which information about objects is stored in individual files. Changes made to these files are then automatically taken on by the Job Scheduler. This has the following advantages:

• Changes made to an object are immediately effective even if the Job Scheduler is running. Further, they remain valid after the Job Scheduler has been restarted.
• Errors in the configuration are immediately visible - in run time - and not just after a restart of the Job Scheduler.
• Orders can be retained - i.e. made persistent.

The default Hot Folder is the .config/live directory. More detailed information about Hot Folders can be found in the Job Scheduler documentation.

The Managed Jobs interface serves as an editor for Hot Folder objects. In the interface Object Explorer, the object tree is represented as /[schedulerID]/live below the .config/live folder. The object definitions are not stored in the Job Scheduler scheduler.xml XML configuration file, but in a database. The objects are configured and submitted to a Job Scheduler using the web interface. The Managed Job Starter must be implemented (page 11) so that objects can be submitted to a Job Scheduler. The Managed Job Starter copies submitted objects from the database into the file system Hot Folder. Similarly, if changes are made to the file system, then the Managed Job Starter synchronises the Managed Jobs interface database tables.
Conflicts can occur when an object is modified both by the web interface and in the file system. It is possible, for example, to use the Job Editor (. /config/jobeditor.(sh|cmd)) in parallel with the web interface to configure objects. To avoid this problem, it is strongly recommended that only one of the two editors is used at any one time.

The Managed Jobs interface can be used to centrally configure objects for a number of Job Schedulers running on different computers. Each of the Job Schedulers is represented in the web interface Object Explorer by its own / [schedulerID] /live directory tree. The only pre-condition for this is that the Job Schedulers have unique SchedulerIDs. Central configuration can be used to set up one Job Scheduler as a so-called Supervisor, which then configures its Workload Schedulers. Workload Schedulers define their Supervisors with the following entry in their . /config/scheduler.xml XML configuration file:

```xml
...<config
tcp_port = "4455"
utp_port = "4455"
supervisor = "my_host:4444"
mail_xslt_stylesheet = "config/scheduler_mail.xsl">
...```

When operating with a Supervisor, the object tree of the Supervisor Job Scheduler in the Object Explorer is extended: a / [schedulerID] /remote directory being added parallel to Job Scheduler's own / [schedulerID] /live directory. This is reflected in the file system with a . /config/remote folder being added below the Supervisor's installation directory. Using information from the example above, the (new) remote directory would be named / [schedulerID] /remote/my_host#4455.

Further information about this subject can be found in the Job Scheduler reference documentation.
3.1 Managed Job Starter

The Managed Job Starter is defined in the ./config/scheduler Managed.xml XML configuration file. The following is added as a base element in the ./config/scheduler.xml XML configuration file if the Managed Jobs package is selected when setting up a Job Scheduler:

```xml
<!-- include job configurations -->
<base file = "scheduler Managed.xml"/>
```

The ./config/scheduler Managed.xml has the following content:

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<spooler>
<config>
 <scheduler_script name="scheduler Managed Starter">
  <script language = "java"
   java_class = "sos.scheduler.managed.JobSchedulerManagedStarter_3"/>
 </scheduler_script>
  
 <jobs>
   <job name = "scheduler Managed Starter"
    title = "start managed jobs"
    idle_timeout = "00:05"
    visible = "never"
    order = "yes"
    process_class= "single">
    <description>
    <include file = "jobs/JobSchedulerManagedStarter.xml"/>
    </description>
    
    <script language = "java"
     java_class = "sos.scheduler.managed.JobSchedulerManagedStarter_3"/>
    
    <!--
    <delay_order_after_setback setback_count="3"
     is_maximum="yes" delay="60"/>
    -->
    
   </job>
  </jobs>

  <job_chains>
   <job_chain name="scheduler Managed Starter" orders recoverable="yes"
    visible="never">
    <job_chain_node state="0" next_state="success" error_state="error"
     job="scheduler Managed Starter"/>
    <job_chain_node state="success"/>
    <job_chain_node state="error"/>
   </job_chain>
  </job_chains>

  <commands>
   <add_order id="managed Starter oder" job_chain="scheduler Managed Starter">
   </add_order>
  </commands>
</config>
</spooler>
```

Software- und Organisations-Service GmbH
March 2009
The scheduler_script element ensures that Job Scheduler Object configurations are submitted from the database when the Job Scheduler is starting.

The scheduler_managed_starter order job, synchronises the scheduler_managed_starter job chain with the database tables for every order being run.

The Run Time element in the add_order element at the end of the example ensures that the synchronisation starts every minute. In addition, the Managed Job Starter receives orders from the Managed Jobs interface, when objects there are updated.

The visible attribute in the above script is used to determine whether job chains and jobs are made visible in the Job Scheduler HTML interface: "yes" makes the objects visible.

3.2 Configuration of a Single Job Scheduler

When only a single Job Scheduler, which is not a Supervisor, is to be administered using the Managed Jobs interface, then the Job Scheduler's ID, host and port are set in the web interface configuration file (./web/custom/custom.inc.php). The Managed Jobs take over these parameters during their initial set-up. Note that when reinstalling or updating Managed Jobs the ID constants for the ID may be missing (see Migration (page 6)).

The following constants can be modified:

```php
if(!defined('APP_SCHEDULER_HOST')) {define('APP_SCHEDULER_HOST','my_host');}
if(!defined('APP_SCHEDULER_PORT')) {define('APP_SCHEDULER_PORT','my_port');}
if(!defined('APP_SCHEDULER_ID')) {define('APP_SCHEDULER_ID' ,'my_scheduler');}
```

The Managed Jobs interface uses the default http://my_host:my_port/ settings when calling the Job Scheduler HTML interface. If another address has to be used because, for example, of a proxy configuration, then the following constants can be modified:

```php
if(!defined('APP_SCHEDULER_ALIAS')) {define('APP_SCHEDULER_ALIAS','');}
```

3.3 Configuration of Multiple Job Schedulers

The web interface needs to be made aware of the ID, host and port of each Job Scheduler when more than one Job Scheduler is to be installed. If one of the Job Schedulers is to be a Supervisor then the web interface must also be made aware of the Workload Scheduler(s). This information is held in a database table (SETTINGS). If this table does not contain any relevant information, then the Job Schedulers specified in the web interface configuration file (./web/custom/custom.inc.php) will be supported as described further below (page 51).
4 Database

4.1 Configuration

The database connection must be made known to (all) the Job Scheduler(s) and to the Web Interface. This is normally done automatically during the Job Scheduler setup. Note that where multiple Job Schedulers are being set up to work together, the same database connection must be specified for them all.

The database connection can be set manually as follows:
• in the Web Interface ./web/custom/custom.inc.php file: the APP_CONNECTION_AUTH and APP_CONNECTION_CLASS constants should be modified to suit the database system being used.
• in the ./config/factory.ini configuration file of each Job Scheduler. Here, the db and db_class entries in the [scheduler] section should be modified.

More detailed information about this subject can be found in the Job Scheduler Installation and Configuration documentation.

4.2 Database Tables

The tables listed below are created when the Managed Jobs option is selected during the Job Scheduler installation. Further information about these tables - in particular for the situation where it is necessary to create these tables manually - can be found in the Job Scheduler Installation and Configuration documentation.

The database tables which are primarily used for Workflow Management are:
• SCHEDULER_MANAGED_OBJECTS
  Stores objects such as jobs, orders, job chains, process classes and locks
• SCHEDULER_MANAGED_ORDERS
  Stores orders
• SCHEDULER_MANAGED_TREE
  Stores the Object Explorer (tree) view and administers its own internal permissions
• SCHEDULER_MANAGED_JOB_TYPES
  Stores job and order types
• SCHEDULER_MANAGED_CONNECTIONS
  Stores database connections
• SCHEDULER_MANAGED_SUBMISSIONS
  Stores orders for synchronising interface and file system objects
• SETTINGS
  Stores the Job Scheduler network, settings and job and order type settings and administers counters

The Workflow Management also uses tables, with which, for example, the permissions to execute functions within the Workflow Management are managed. The administration of these tables is carried out by the User Management web interface.
• ACL
  Stores the permissions for a role or a user
- **USERS**
  Stores unit and user

- **USER_GROUPS**
  Stores roles which have been allocated to users

- **USER_VARIABLES**
  Stores unit and user variables

- **USER_ATTRIBUTES**
  Stores attributes of the above variables such as file type.

### 4.3 Database Table Content after Installation

All the database tables mentioned in the previous section are filled by the Job Scheduler installer when the Managed Jobs package is selected during installation. More detailed information about the content of these tables - in particular, about the manual addition of records - is to be found in the Job Scheduler Installation and Configuration documentation.

The following default records are to be found after installation:
- One unit with two users - the unit is sos and the users admin and demo.
- A role named admin, allocated to the admin user.
- A role named operator, allocated to the demo user.
- A role named developer, which is not allocated to any user.
- Seven permissions lists are available, in which the admin user and the admin role are granted all permissions. The Workflow Management permissions list is used to regulate access within the workflow management and the other six lists are used to regulate the user management.

When the Workflow management is opened, the object tree will have a structure, depending on the exact contents of the SCHEDULER_MANAGED_TREE and SCHEDULER_MANAGED_OBJECTS tables. This tree will appear something like:

```
+ /
  + Home
  + Samples
    + Launch Database Reports
      - launch_custom_report (job chain)
      - launch_custom_report,order (order)
      - launch_database_statement (job chain)
      - launch_database_statement,order (order)
      - schedulerManaged_custom_report (order job)
      - schedulerManaged_database_report (independent job)
      - schedulerManaged_database_statement (order job)
    + Launch Executable Files
      - launch_executable_file (job chain)
      - launch_executable_file,order (order)
      - schedulerManaged_executable_file (order job)
      - schedulerManaged_executable_php_file (independent job)
    + [your Job Scheduler ID]
      + live
```
In addition, eight job types are defined in the `SCHEDULER_MANAGED_JOB_TYPES` table: file, database, database_report, custom_report, ftp_send, ftp_receive, interpreter_php and standard. Sample entries for the administration of the database_report, custom_report, ftp_send, ftp_recieve and interpreter_php job types are to be found in the `SETTINGS` table. These sample entries simplify the Parameter Management (page 33) of jobs and orders in this type of job. In addition, the database connection specified during the `Job Scheduler` setup can be found in the `SCHEDULER_MANAGED_CONNECTIONS` table.
5 Workflow Management

The web interface opens with a login dialog, in which in unit, user and password are to be entered. The default unit and user after installation of the Job Scheduler are sos and admin respectively. The default password is nothing - i.e. it should be left blank.

If HTTP authentication is used, then this web interface login dialog will be omitted and the current HTTP authenticated user is automatically entered by the Job Scheduler in the Managed Jobs user management. This user is then given basic operator permissions (see Permission Management (page 27)). Should it be necessary to give a user administrator permissions during HTTP authentication when they are first registering, then the following user settings can be made in the ./web/custom/custom.inc.php file (note that the constants here may have to be set first):

```php
<?php
...
// admin permissions for user administration, if login via HTTP authentication
if(!defined('APP_HTTP_AUTH_ADMIN')) {define('APP_HTTP_AUTH_ADMIN','admin');}
...
?>
```

By default, this user is allocated to the sos unit. This allocation can be changed by opening the ./web/custom/custom.inc.php file and setting the value of the APP_HTTP_AUTH_UNIT constant (this constant must first be defined if it is not already listed). The following lines are used to configure the unit:

```php
<?php
...
// unit for user administration, if login via HTTP authentication
if(!defined('APP_HTTP_AUTH_UNIT')) {define('APP_HTTP_AUTH_UNIT','[your unit]');}
...
?>
```

Note that with only basic operator permissions, this user is not able to access, for example, the User Management.

In its 'as installed' state, the Managed Jobs package has only one user with full access to the User Management, including the permissions management. This user is admin/sos. Therefore it is important when using HTTP authentication that there is a user named admin. It is also important that the unit of the admin user, which is specified in the APP_HTTP_AUTH_UNIT constant, is not be changed until either a user from another unit has been created and given full administrator permissions or the APP_HTTP_AUTH_ADMIN constant, which specifies one user that is automatically given full user permissions, is set.

A frameset opens after a successful login, with a menu in the left frame. This documentation describes the part of the web interface which is reached using the Workflows and Settings items of this menu.

The Log-out menu item is used to end a session. However, if HTTP authentication is in use, in addition to logging out, then the browser window (and not just a browser tab) must be closed, in order to ensure that access to the Managed Jobs interface can only take place through renewed logging in.

The Workflows management link in the web interface is divided into a total of three sub-items:
- Job Chains / Jobs / Orders
- Job Types and Order Types
- Database Connections

The Workflows management link in the web interface is divided into a total of three sub-items:
- Job Chains/ Jobs/ Orders;
- Job / Order Types;
- Database Connections.
Note however that the functions and presentation shown in the workflow management depend on the rights a user has been given. The administration of user rights is done with the Workflow Management Access Control List, which is reached via the 'User Management → Rights' link (see also the User Administration documentation). In the Access Control List, the Group: root, Group: developer and Group: operator are allocated read, write and execute permissions. Roles are then allocated for each of these groups:

- the admin role is allocated the root group permissions;
- the developer role is allocated the developer group permissions and
- the operator role is allocated the operator group permissions.

Note also that it is necessary for a user who has logged into the system to have been given one of these group permissions (page 27) before they are able to see these links in the workflow management menu.

Selecting the "Job Chains / Jobs / Orders" item in the Workflow Management menu opens a web interface for the management of job chain, job and order objects. This web interface is called the (Job Scheduler) Explorer and is described in the Explorer (page 18) section below.

5.1 Example Job: Execution of a Shell Script

This section describes how a shell script can be submitted and started using the Job Scheduler Managed-Jobs Interface. Questions arising out of this example will be answered in subsequent sections. This section is intended as an introduction showing how easy it is to configure a job and to motivate the reader to continue with other chapters. The following script is to be found in ./jobs/test.sh or ./jobs/test.cmd directories:

```
Example: Content of test.sh
#!/bin/sh
echo "$1"
```

```
Example: Content of test.cmd
echo off
echo $~1
```

Note that on Unix systems the Job Scheduler user must be able to execute ./jobs/test.sh.

1. Log on to the Job Scheduler giving sos as unit, and admin as user, with no password.
2. Open the Workflow Management using the "Workflow Management → Job Chains/ Jobs/ Orders" entries in the left hand menu.
3. Navigate to the /\[your scheduler id]/live folder.
4. Now open the admin folder's context menu by clicking with the right hand mouse button in an empty space in the Editor window.
5. Select New → Order-Independent Job from the context menu.
6. Give this new job a name such as "Test". This is done by entering the desired name in the active "New Order-Independent Job" field which opens after the job has been created. Should this field not be visible, for example, because you clicked elsewhere in the browser window with the mouse, then open the job context menu using the right hand mouse button in the same way as described above for the new order and select the Rename item.
7. A single click on either the job icon or name text will open the Editor with which the job can be configured.
8. Select Executable File from the Job type selection box and confirm this selection by clicking the icon with the green arrow to the right of the Job Configuration area. This opens a list of predefined parameters for this job type in the bottom left hand part of the Editor window.
9. Click on the param text for the name="command" parameter in order to edit its values. This opens a text field in the area in the Editor window with the grey background. This field is used to enter the program call.
10. The shell script together with any parameters is entered in the name="command" text field. In this example, the code for either the ./jobs/test.sh "hello world" or the ./jobs/test.cmd "hello world") scripts as already listed should be entered. Should, for any reason, this script not be available, then either sh -c "echo hello world" or cmd /C "echo hello world" could be entered.
Congratulations, you have just executed your first job!

5.2 Explorer

The Explorer is a part of the Job Scheduler web interface and is used to manage job chain, job and order objects. It is called from the "Workflow" menu items Workflow->Job Chains/ Jobs/ Orders. The objects to be managed (folders, jobs, job chains, orders, process classes schedules, parameters and locks) are presented in a tree diagram, similar to that displayed, for example, in the Microsoft Windows Explorer "Folder" window. The Explorer is shown in the screen-shot below.

In the diagram, the object tree diagram is shown in the left hand part of the Explorer window and to the right, the content of the folder that has been selected. Should a folder contain further folders then these are listed first and in alphabetical order. Objects other than folders (jobs, orders, etc.) contained by selected folder are listed below any folders, again sorted alphabetically. The order in which the objects are listed can be changed by clicking on the title of the column according to which the objects are to be sorted. A small triangle in the in one of the column titles indicates the column according to which the objects are sorted. The orientation of the triangle (point upwards or downwards, indicates the direction in which the objects are then sorted. Note that when the column according to which the objects are sorted is not wide enough, then the triangle will be hidden by the next column. However, the widths of individual columns can be changed by moving the dividers between the column titles using the mouse.

Before each object shown in the right hand part of the Explorer is a checkbox, which can be used to select more than object. A function can then be used to change all the selected objects at once. The functions available for multiple operations are "copy", "cut/move", "delete", "activate", "deactivate" and "export".

2007-06-13 15:12:15.406  [info] stdout:
5.2.1 Symbols Used in the Object Explorer

The following symbols are used in the Object Explorer to represent different types of objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock</td>
<td></td>
<td>Locks represent the Job Scheduler configuration spooler/config/locks/lock elements.</td>
</tr>
<tr>
<td>Process class</td>
<td></td>
<td>Process classes represent the spooler/config/process_classes/process_class elements in the Job Scheduler configuration.</td>
</tr>
<tr>
<td>Order-independent job</td>
<td></td>
<td>Order-independent jobs represent the Job Scheduler configuration spooler/config/jobs/job elements, with the order= attribute set to &quot;no&quot;.</td>
</tr>
<tr>
<td>Order job</td>
<td></td>
<td>Order jobs represent the spooler/config/jobs/job elements in the Job Scheduler configuration, with the order= attribute set to &quot;yes&quot;.</td>
</tr>
<tr>
<td>Order</td>
<td></td>
<td>Orders represent the spooler/config/commands/add_order elements in the Job Scheduler configuration.</td>
</tr>
<tr>
<td>Job chain</td>
<td></td>
<td>Job chains represent the Job Scheduler configuration spooler/config/job_chains/job_chain elements.</td>
</tr>
<tr>
<td>Schedule</td>
<td></td>
<td>Schedules represent spooler/config/schedules/schedule elements in the Job Scheduler configuration, where the &quot;substitute&quot;, &quot;valid_from&quot; and &quot;valid_to&quot; attributes are not set.</td>
</tr>
<tr>
<td>Substituting schedule</td>
<td></td>
<td>Substituting schedules correspond with the spooler/config/schedules/schedule elements in the Job Scheduler configuration, with the &quot;substitute&quot;, &quot;valid_from&quot; and &quot;valid_to&quot; attributes set.</td>
</tr>
<tr>
<td>Parameter</td>
<td></td>
<td>Parameters represent Job Scheduler spooler/config/params configuration elements.</td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td>Documentation represents the file or live_file attribute in the .description/include element of a job or order in the Job Scheduler configuration</td>
</tr>
<tr>
<td>Other plain/text files</td>
<td></td>
<td>This object type is intended to be used to administer script files (shell, javascript, perl, vbs), which are referenced in the live_file attribute in the .script/include element of a job. Ini or csv files, which, for example, are included for a job implementation, can also be processed using this object.</td>
</tr>
</tbody>
</table>

Table 1. Object Symbols

Further information about these objects can be found in the Job Scheduler reference documentation and glossary.

5.2.2 Object Statuses and their Symbols

Further information about all the objects described in Table 1 can be represented through the use of the following markings:
### Table 2. Object Markings

<table>
<thead>
<tr>
<th>Marking</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚀</td>
<td>Job Scheduler</td>
<td>The Scheduler IDs of the supporting Job Schedulers form the names of these folders. Which Job Schedulers are supported is explained in Job Scheduler Network (page 51). These &quot;Hot Folders&quot; have &quot;live&quot; and eventually &quot;remote[/workload scheduler host][\workload scheduler port]&quot; sub-folders. Objects can only be forwarded to a Job Scheduler when they are added to the Job Scheduler's Hot Folder(s). Hot folders also provide a means for objects from a Job Scheduler file system to be made available to the Managed Jobs interface.</td>
</tr>
<tr>
<td>🌞</td>
<td>Empty object</td>
<td>After an object has been newly created, it is empty - i.e. it exists as an object in the explorer but does not contain any information relevant to the Job Scheduler. Element information - i.e. information relevant to the Job Scheduler - can be added to such an object using the Editor (page 28). An empty object cannot be handed over to the Job Scheduler.</td>
</tr>
<tr>
<td>🕹️</td>
<td>Inactive objects</td>
<td>Inactive objects are not empty but have not been handed over to the Job Scheduler. All non-empty objects, which are not stored in a Job Scheduler hot folder are automatically deactivated. The functions to activate and deactivate an object can be found in an object's context menu.</td>
</tr>
<tr>
<td>🛠️</td>
<td>Links</td>
<td>Objects can be copied as links. The path to the target object can be retrieved using the &quot;Properties&quot; entry in the object context menu. If the target object is a folder, then clicking on the originating object will open the linked directory. If the target object is not a folder, then clicking on the original element opens the Editor (page 28) containing the element information for the linked object in read-only mode. Note that linked objects possess the rights of their original objects.</td>
</tr>
<tr>
<td>🎯</td>
<td>Submitted objects</td>
<td>This symbol is used to indicate objects that have been submitted to a Job Scheduler but whose submission has not yet been confirmed. Submitted objects only become active in the Job Scheduler, once their submission has been confirmed. If an object is submitted to a Job Scheduler that is not running, then activation remains pending until the Job Scheduler starts. Activated objects have no additional symbol. The functions to activate and deactivate an object can be found in the object's context menu.</td>
</tr>
<tr>
<td>🕹️</td>
<td>Changed objects</td>
<td>This symbol indicates an object that has been activated and then modified. Such an object must be activated once more before the Job Scheduler takes account of the change. The functions to activate and deactivate an object can be found in the object's context menu.</td>
</tr>
<tr>
<td>🌾</td>
<td>Active Object</td>
<td>Active objects do not have a special symbol. These objects operate in the corresponding Job Scheduler and synchron with the configuration in der Managed Jobs interface. The function to deactivate active objects is to be found in the object's context menu.</td>
</tr>
</tbody>
</table>

#### 5.2.3 Navigation Within the Explorer Tree

Clicking on a folder symbol or on the text alongside the symbol in the explorer tree opens the directory. Clicking on a file symbol or on the text alongside it opens the (object) Editor (page 28), which allows form-based editing of the object. In addition, a context menu containing further object-dependent functions is available when the right-hand mouse button is clicked.
The separator between the tree in the left hand part of the object explorer and the contents list to the right can be moved as required. Furthermore, the columns of the contents list in the right hand window can be changed by moving the separators in the column titles.

The status bar below the explorer tree is used to indicate with *waiting for response* that a query has been sent to the web server. In the event of an error, the error message will be shown in this bar.

An address is shown above the tree. Clicking on the button to the right of this address causes the explorer to go to this address.

The buttons in the 2nd row have the following functions:

- Jump to the last address saved in the explorer history. The small triangle alongside opens the last 10 entries in the address history.
- Jump to the next address saved in the explorer history. The small triangle alongside opens up to 10 entries in the address history.
- Jump to the root directory (/)
- Jump a level higher in the directory
- Refresh the explorer tree and the content of the folder currently highlighted
- Open the search dialog for the current folder
- Return to the folder-view when the search dialog is open

Clicking on the "File", "Edit" and "?" menu items on the first row of the Explorer interface opens menus. Whilst the "?" menu opens links to the Explorer documentation, the "File" and "Edit" provide access to functions which are also available in the context menu. These functions will be described in the next chapter.

### 5.3 The Context Menu Functions

The context menu functions are accessed by clicking on an object in the Explorer object tree using the right hand mouse button.

The following context menu functions are available:

#### 5.3.1 Renaming Objects

The root directory and the *Job Scheduler Hot Folders* cannot be renamed. All other objects for which read and write permissions have been granted can be renamed as required. The following should be noted when renaming objects:

- Names are not case sensitive.
- A name must be unique amongst all the objects of a given type in a directory.
- A name may not contain one of the following special characters: /\?*<>|"
  Note that order names form an exception. A single comma is used here to separate the job chain name from the order name ([job chain name],[order name]).
- Empty spaces at either the start of a name or at its end will be removed.
- More than one successive empty space will be automatically reduced to one.
- A name may not contain more than 100 characters.
- An object name together with its full path may not contain more than 255 characters.
5.3.2 Creating a New Object

The function with which new objects are created can be found in the context menu of the (target) folder in which
the new object is to be created. This function opens a submenu alongside the "New" menu item, with which the
type of object to be created is selected. Note that this function is only active when the user possesses write
permissions for the target directory. After the type of object to be created has been specified, an entry field for
the name of the object opens in the right hand part of the Explorer. When selecting a name for an object, it is important
that the rules for object names as already specified in the Rename (page 21) section are observed. Newly created
objects (with the exception of directories) are empty at first and marked with a yellow background (see empty
objects (page 19)). New objects are allocated permissions according to the user group of the object owner as
follows:

- **Admin:** `rwxrwxr--` (774)
- **Developer:** `rwxrwxr--` (774)
- **Operator:** `rw-rwxr--` (674)

5.3.3 Import Using the HTTP Upload Dialog

The function with which objects can be imported is reached using the context menu of the folder into which the
object is to be imported (i.e. the target folder). Note that this function is only active for users with write permissions
for the target folder.

Selecting the type of object to be imported opens an upload dialog. The file name of the file to be uploaded is
entered in this dialog. The name and path of the file can either be entered directly or sought using the Browse ...
button.

The file to be imported can either be a single XML file or an archive containing XML files. Zip, tar, tgz and tar.gz
archive formats are supported.

The Close button is used to break off the import dialog without importing the object; the Confirm button loads the
file to be imported and saves the object or objects.

Imported objects are not automatically submitted to the Job Scheduler. This is done using the Activate (page 26)
link in the context menu of either individual objects or that of the imported directory.

Note that the XML files must have file names conforming with the requirements of the Hot Folders, as specified in
the Job Scheduler documentation: i.e. in the following object-dependent syntax:

- **Lock:** `[Name of the lock].lock.xml`
- **Process class:** `[Name of the process class].process_class.xml`
- **Job:** `[Name of the job].job.xml`
- **Job chain:** `[Name of the job chain].job_chain.xml`
- **Order:** `[Name of the job chain],[Name of the order].order.xml`
- **Schedule:** `[Name of the schedule].schedule.xml`
- **Parameter:** `[Name of the parameter].params.xml`

The XML files must be named according to the "Rename (page 21)" rules set out above. If the name is not unique,
then the dialog asks whether the target file should be overwritten. Empty directories in an archive will not be
imported. Note that the element information in XML files to be uploaded must conform with the requirements
specified in the Job Scheduler documentation: i.e. in the following object-dependent form::
When orders are imported, they will only be carried out by the *Job Scheduler* when they are saved in the same directory as the job chains allocated to them.

If the *Managed Jobs* interface is used together with PHP5 or higher, then the XML file can be tested against the appropriate schema. To do this, the `APP_SCHEDULER_SCHEMA_ENABLED` constant in the `.web/custom/custom.inc.php` file must be set to "true":

```php
<?php
...
// enable or disable validation against the schema (only for PHP5)
if(!defined('APP_SCHEDULER_SCHEMA_ENABLED'))
{define('APP_SCHEDULER_SCHEMA_ENABLED',true);}
...
?>
```

### 5.3.4 Copying Objects

One or more objects can be copied at once, with the exception of the root directory, which cannot be copied.

In order to copy one or more objects from a common directory, the directory containing the object(s) should first of all be selected in the Explorer object tree. This causes all the objects in the directory to be listed in the right hand part of the Explorer. The object (or objects) to be copied are then selected by marking the appropriate checkbox(es) in the Explorer. If all of the objects in the current directory are to be copied then they can be selected in a single operation using the *Highlight all* item in the context menu that opens when an open space in the right hand part of the Explorer interface is clicked with the right hand mouse button. After the required objects have been marked, the parent context menu (i.e. the context menu which opens when the folder containing the objects to be copied is clicked) is opened and the selected object(s) first of all copied into the clipboard by clicking *Copy*.

Note that if only one object is to be copied then it is not necessary to first of all select the object as just described - the *Copy* context menu item can be opened directly from the object to be copied. If a single folder is to be copied, then the corresponding context menu item can be reached from the object tree.

The target directory is now selected in the Explorer object tree. The object(s) to be copied from the clipboard are then pasted into the target object using the target object context menu and selecting either *Paste* or *Paste as link*.

If there are no suitable objects in the clipboard or the user is in possession of write permission for the target directory, then the *Paste* or *Paste as link* items in the context menu will not be active. Should an object in the clipboard have the same name as an object in the target directory, then the user will be asked in each case if the object in the target directory should be overwritten.

Objects which have been copied must not automatically submitted to the relevant *Job Scheduler*. If required, submission is done using Activate (page 26) function in the context menu from either the target directory or individual objects that have been copied. However, if the source object has already been submitted to the *Job Scheduler*, then the target object will be automatically submitted.
If an new object is added through copying, then this object will be allocated permissions as described in the Creating a New Object (page 22) section. If, however, an existing object is overwritten by copying, then both its owner and the permissions it has will be retained.

Links retain the permissions and owners of the original objects. Empty objects cannot be linked.

5.3.5 Moving Objects

One or more objects can be moved at the same time. However, a user must be in possession of read and write permissions in both source and target directories before being able to move objects. Note that neither the root directory nor the Job Scheduler Hot Folders can be moved and that root permissions are required before system directories can be moved (the owner is root).

Objects to be moved are first of all to be copied into the clipboard as described above in the copying (page 23) section with, however, the exception that the context menu Cut item is selected instead of Copy.

After copying the objects to be moved into the clipboard, the target directory is now selected and the object(s) in the clipboard written into the target directory by selecting Paste in the context menu. As with copy, the user will be asked whether an object should be overwritten when an object having the same name as an already existing object is to be moved into a directory.

Note that moved objects retain their permissions and owner.

If objects that are stored directly below the Job Scheduler Hot Folder are moved to a target direct directory that does not lie directly under the same Job Scheduler Hot Folder, then these files will be deleted in the Job Scheduler , which could have consequences (see Remove (page 25)). If required, the moved objects should now be Activated (page 26).

5.3.6 Exporting Objects Using the HTTP-Download Dialog

This function is available in the context menu of all objects. One or more objects can be exported at once. Only reading permissions are required to export an object.

Single objects are exported as XML files, multiple objects as archives. Zip,tar and tar.gz archives are supported. The files in exported archives are in XML format with paths relative to the location where the Export function was called. When exporting links, the linked object itself is exported, as long as the relative path does not contain . / elements, as such paths are not currently supported by the zip format.

Multiple objects are prepared for exporting by marking the object checkboxes alongside the object lists. The context menu for one of the marked objects is then opened and the "Export" item selected. It is not necessary to use the checkboxes when exporting a single object - the object context menu and "Export" item can be used directly. Selecting the "Export" item causes a download dialog window to open which is then used to specify the name and location for the exported file. The structure of the XML objects being exported depends on whether one or more objects are being exported and correspond with the structure already described in the Import (page 22) section.

Note that exported XML files are coded in UTF-8. Note also that the use of PHP5 or higher for the web interface leads to the exported XML being better structured than with PHP4.
5.3.7 Deleting Objects

One or more objects can be deleted at any one time. The root directory and Job Scheduler Hot Folder cannot be deleted. Root permissions are necessary to delete all system directories (root is the owner) - otherwise a user can delete every directory for which they have write permissions.

The directory containing the objects to be deleted is first of all selected in the Explorer object tree diagram as described in the copying (page 23) section.

When more than one object is to be deleted then the checkboxes in the right hand part of the Explorer interface can be used to select the relevant objects. After the objects have been selected, they are deleted by clicking on one of them with the right hand mouse key and selecting Delete from the context menu.

If all the objects in a directory are to be deleted then they can be selected in a single operation using the Highlight all item in the context menu which opens when an open space in the right hand part of the Explorer interface is clicked with the right hand mouse button.

Should only one object be deleted then it is not necessary to use the checkboxes - the context menu can be directly opened by clicking on the object to be deleted. When a folder is to be deleted then this can be directly carried out by opening the context menu in the object tree diagram in the left hand part of the Explorer and selecting Delete. Deleting an object also deletes all possible links to the object. Orders using a deleted job chain are automatically suspended.

Objects being deleted which have already been handed over to a Job Scheduler for processing will, with the exception of order jobs, also be deleted on the Job Scheduler. Note, however, that order jobs which have been handed over to the Job Scheduler will only be deleted after a restart.

It is important that dependencies between objects are taken into consideration when deleting objects which have already been handed over to a Job Scheduler. If such dependencies are not taken into consideration, this can lead to the Job Scheduler generating error messages. The dependencies which are to be considered are:

<table>
<thead>
<tr>
<th>Deleted Object</th>
<th>Error Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Class</td>
<td>Ensure that no job which has been handed over to the Job Scheduler uses the process class. If necessary, change the relevant element information and hand over the job once again.</td>
</tr>
<tr>
<td>Lock</td>
<td>Ensure than the lock which has been deleted is not used in any job which has been handed over. If necessary, change the relevant element information and hand over the job once again.</td>
</tr>
<tr>
<td>Order Job</td>
<td>Ensure that no job which has been handed over to the Job Scheduler uses the order job. If necessary, change the relevant element information and hand over the job once again.</td>
</tr>
<tr>
<td>Job Chain</td>
<td>Ensure that no order which has been handed over uses the deleted job chain. If possible, allocate another job chain to the order using the editor. Alternatively, the order itself can be suspended or deleted.</td>
</tr>
<tr>
<td>Schedule</td>
<td>Ensure that no submitted job or order uses the deleted schedule. If, however, this is the case, then change the relevant element information using the editor and resubmit the job or order.</td>
</tr>
</tbody>
</table>

Table 3. Consequences of Deleting Objects
5.3.8 Object Properties

Alongside general information about an object, the "Properties" item in the object context menu shows information about the path to linked objects. Users with root permissions and object owners can also set the permissions for the object. Further information about this subject can be found in the permissions (page 27) chapter.

<table>
<thead>
<tr>
<th>Property</th>
<th>Read</th>
<th>Write</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Developer</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Other</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permission mask</td>
<td>774</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.3.9 Activation

This function is available for all objects below a Job Scheduler Hot Folder in the Functions->Activate context menu. This function submits and updates the objects on the corresponding Job Scheduler. One or more objects can be activated at once. Empty directories cannot be submitted. Execute permissions are necessary before objects can be activated.

The following dependencies should be considered when activating individual objects:

<table>
<thead>
<tr>
<th>Object submitted</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>If a job is allocated a process class, schedule or lock in the Editor (page 28) then the process class, schedule or lock in question must also be submitted.</td>
</tr>
<tr>
<td>Job chains</td>
<td>All order jobs in a job chain should also be submitted.</td>
</tr>
<tr>
<td>Orders</td>
<td>Orders can only be carried out, when they are stored in the same directory as their job chain. If a schedule is allocated to an order in the Editor (page 28), then this Schedule must also be submitted.</td>
</tr>
</tbody>
</table>

Table 4. Dependencies in Object Submission

5.3.10 Deactivation

This function is available for all objects below a Job Scheduler Hot Folder in the Functions->Deactivate context menu. This function deletes objects from the corresponding Job Scheduler. However, deactivated objects remain in the Managed Jobs interface. One or more objects can be deactivated at once. Execute permissions are required to deactivate objects. Deactivation of an object has the same consequences as Deletion (page 25).
5.3.11 Additional Functions Available Within a Job Scheduler Hot Folder

The *Functions* item in the object context menu contains object-dependant functions in addition to *Activate* and *Deactivate*. Note that these functions are only available to users possessing execute permissions. Note also that calling a function using the context menu causes the function only to be applied to the object for which the context menu was opened. It is not possible to apply these functions to more than one object at once. These functions are also available as a button in the Editor (page 28).

The following table lists the functions available for each object and their purpose.

<table>
<thead>
<tr>
<th>Name</th>
<th>Object</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activate and</td>
<td>Order-independent</td>
<td>This function is similar to the Activate function but starts the job or</td>
</tr>
<tr>
<td>Start</td>
<td>jobs and orders</td>
<td>order immediately, regardless of any start times that have been set.</td>
</tr>
<tr>
<td>Test</td>
<td>Orders</td>
<td>This function is similar to the Activate function but the start interval is not handed over. This means that the order is not persistent - it will be started once and, once the end of the job chain has been reached, will be deleted.</td>
</tr>
<tr>
<td>Delete Orders</td>
<td>Job chains</td>
<td>This function corresponds to the Deactivation of all orders allocated to a job chain. The orders are retained in the web interface, so that they are available should it be necessary for them to be resubmitted to the Job Scheduler from the web interface.</td>
</tr>
<tr>
<td>History</td>
<td>Orders</td>
<td>Information about every order started is held in a history. This function shows the history for the order selected. The dialog which opens provides information about the length of processing and the status of the order. If a Job Scheduler running then a log file can be opened, in which each step of the order's progress can be examined.</td>
</tr>
</tbody>
</table>

Table 5. Hot folder functions

5.4 Permissions

There is a *Workflow Management* permissions list in the permissions management. In this list the *Group: root*, *Group: Developer* and *Group: Operator* are given, which, in turn, imply read, write and execute rights.

The appropriate write permissions are necessary before job type or database connections can be edited. These are provided by the *Group: root* and *Group: Developer* permissions set. The *Group: Operator* permissions set has only permission to read in this area.

Permissions for each object are given in the Workflow Management individually. It is necessary to be the owner of an object, in order to be able to change its permissions. The dialog with which permissions for an object are set is reached by way of the Properties (page 26) item in the object context menu. This dialog consists of a form with nine checkboxes and a related data entry field. Three numbers are shown in the entry field, which represent the object permissions.

- The 1st number specifies the permissions of the object owner.
- The 2nd number specifies the developer's permissions. Developers are the users who have been given the *Group: Developer* permissions set in the permissions management.
- The 3rd number specifies the permissions of the developers and operators, i.e. all the users who have been allocated to the *Group: Developer or Group: Operator* in the permissions management.

Each of these numbers can have a value from 0 to 7. The number is the sum of the permissions granted, where 4 denotes permission to read, 2 permission to write and 1 permission to execute. For example, 6 denotes that permission to read and write have been granted but not permission to execute.
The individual read, write and execute permissions which have been granted are also shown in the checkboxes in the dialog.

When a number in the entry field is changed, the values in the checkboxes will automatically be changed accordingly. However, it is recommended that the checkboxes are used for allocating permissions - the numbers in the entry field will then be changed accordingly. Once permissions have been set they are saved and the permissions dialog ended by clicking on the Save button. The dialog will also be entered - and entries saved - when the "Return" key is hit (as long as the cursor is focused on the properties dialog).

Users that have been allocated the Group: root permissions set in the permissions management, always have full permissions, regardless of the permissions set in object "Properties" menu.

Note that there are two protected directories - the root directory (/) and the Job Scheduler Hot Folder including its "live" sub-folder. The use of these protected directories is also restricted for root users, as these orders cannot be moved, deleted or renamed.

Should a user not possess read permissions for an object, then this object will be listed in the right hand Explorer contents area with an inactive checkbox. At the same time this object will not function for the user - i.e. it can neither be copied, moved, deleted nor renamed. Only the context menu "Properties" item will be available. However, when a user is also the owner of an object, the "Properties" dialog can be used to change permissions, if required. Note that the Editor (page 28) cannot be opened for folders. As folders cannot be edited, they are not shown in the Explorer object tree. When the read permissions are to be removed from several objects, then this can be done by removing the read permission from the folder containing the objects.

When an object is first created, then it is per default readable by all users.

Should a user not possess write permissions for a folder, then this means that the folder can neither be renamed nor deleted; that objects in the folder can neither be copied nor moved and that new objects cannot be created in the folder. Objects other than folders cannot be renamed or deleted by a user without write permissions. Furthermore, users without write permissions cannot change the configuration of objects - i.e. they can only use the Editor to read the object configuration but cannot change it.

Note that a newly created object can initially only be over-written by its owner and all developers.

Execute permissions are only relevant for the Job Scheduler Hot Folders. They are required for all functions available in the Functions context menu, which can be carried out on these folders. These include Activate (page 26) and Deactivate (page 26).

After setup, folders and other objects are available as specified in the Database Table Content after Installation table. The default permissions for these objects are rwxrwxr-- (774) and the internal Workflow Administration owner is root.

5.5 Editor

The object Editor is started either by clicking on an object (other than a folder) or by selecting the Edit Explorer context menu item.

The forms provided in the Editor are used to generate the relevant XML code and save this in the SCHEDULER_MANAGED_OBJECTS table in the database. When an already configured object is opened in the Editor, the XML is read out of the database and the corresponding form field in the Editor filled. Instead of the form fields, an XML view (page 29) can also be used to configure the object. The Editor layout is the same for all object types.
5.5.1 Importing and Exporting Using the Clipboard (XML Editor View)

The button in the top left part of the Editor window labelled XML can be used to open an XML view of the object configuration in a text field. This view allows the object configuration to be directly edited in XML format, without use of the fields presented in the Editor.
The `Submit` and `Cancel` buttons both close the XML view. Whilst the `Cancel` button causes any changes that may have been made to the object's XML configuration to be ignored, the `Submit` button causes changes to be saved and the Editor's object configuration form fields to be updated.

When the Managed Jobs interface is running on PHP5, the XML of an object is validated by the submission function against the schema specified in the `./web/config/scheduler.xsd` file, when the `./web/custom/custom.inc.php` is suitably configured (see the Import (page 22) section).

The XML view text field can also be used to import and export XML by way of the clipboard.

### 5.5.2 Set Job & Order Types

In order to simplify setting parameters, jobs and orders are grouped in the Workflow Management (page 46) together as predefined "types". A list of the job and order types available can be found in the Editor in the "Job Configuration" section. Note that this section of the Editor is only shown when a job or order object is being edited. Once a job or order type has been selected for an object, the parameters shown in the lower left hand area of the Editor will be modified accordingly and added to the job as a `<script>` element. Not only the parameters shown but also their values depend on the job or order type selected. Several standard job and order types are made available after installation of the Managed Jobs package (see Table 8). The parameters of these predefined job and order types are listed in Table 11.

Two further parameters, in addition to those listed in Table 11, can also be set for jobs and orders:
- The `database_connection` parameter is set by way of the `connection` selection box in the Job Configuration area of the Editor. The content of this selection box is determined by the Database Connections Management (page 48).
- These parameters are required for the Database Statement, Database Report and Customised Report job or order types.
- The `command` parameter is required for the Executable File, Executable PHP File, Database Statement, Database Report and Customised Report types. Note that whilst command line program calls are specified here for the Executable File and Executable PHP File types, the Database Statement, Database Report and Customised Report types.
Report expect SQL statements in the parameters. The values entered here are submitted to the job or order in hexadecimal code, which ensures that all characters are saved in a manner which is XML-conform. Changes made in the Job Type and database connection selection boxes are stored using the button to the right of the Job Configuration area.

5.5.3 Editing the Object Configuration

The object configuration itself is edited in a form in the lower part of the Editor interface. The individual elements of an object are shown in a tree diagram in the left hand part of this form. The top-most element in the tree diagram is the root element of the object to be configured. Each element in this tree has a context menu, which can be used to add child elements or to delete the element. The right hand part of the form, with the grey background, shows the attributes of the element selected. Should the selected element have only child elements and no attributes of its own, then an appropriate message will be shown. More detailed information about child elements and their attributes can be found in the Job Scheduler documentation.

Note also that an exclamation mark (!) to the right of a form field means that this attribute must contain a value. In addition, should a special data format be required, then this will be specified to the right of the form field and/or one of the following symbols shown:

- opens a calendar for the selection of dates
- opens a selection box containing a list of possible entries
- opens a dialog window with which the objects required in the form field can be added. Note that only objects and links are shown for which the necessary reading permissions are available. The dialog starts in the current directory.

The configuration of more complex objects - in particular jobs, job chains and orders - is simplified by special dialogues. The availability of such dialogues or assistants is symbolised by the icon alongside the following objects:

- ... Jobs for the params-, script-, commands- and run_time-Elements
- ... Aufträge for the params- und run_time-Elements
- ... Job-Ketten for the job_chain-Elements

Note that these assistants open a new window. Further information can be found in the following sections of this document: Parameter Management (page 33), Script Management (page 34) Run Time Editor (page 31) Commands Editor (page 37) and Job Chain Editor (page 37).

5.5.4 Run Time Editor

Because of the complexity involved in specifying start intervals (<run_time>-Element), a Run Time Editor is available as assistant to help in setting start intervals for jobs and orders. This editor can be opened by clicking on the symbol alongside the run_time text. The interface of the Run Time Editor is similar to that used for the Job Scheduler Job Editor (/bin/jobeditor.(sh|cmd)).
## Run Time

### Schedule:

- Everyday
- Weekdays
- Monthdays
- Ulimas
- Specific Weekdays
- Specific Days
- Holidays
- Specific Month

### Time Slot

- Lot Run:
  - Begin Time: `hh:mm:ss`
  - End Time: `hh:mm:ss`

- Run Once:
  - Begin Time: `hh:mm:ss`
  - End Time: `hh:mm:ss`

### Start Time

- Repeat Time: `hh:mm:ss or ss`
- Absolute Repeat: `hh:mm:ss or ss`
- Single Start: `12:00`

### When Holiday

- Suppress execution (default)
5.5.5 Job and Order Parameter Management

Order and job type settings made with a range are shown in this dialog. The uppermost *Local Settings* part of this dialog is used to show those job or order type parameters with .../mixed/... or .../local/... ranges. More detailed information about ranges and their schemes and sections can be found in the Chapter on Settings (page 51). Changes made here affect only the current jobs or order.

The third part of the dialog, shows the *Local Settings-Free Parameters*. The blank fields in this form are used to set the name and values of the parameters. These parameters are not specified in the job or order type settings and do not require the creation of ranges in the settings. They do, however, considerably simplify the parameter management.

Regardless of the job or order type, there are two *Local Settings-Free Parameters* which have a particular role. These are the *database_connection* and *command* parameters.

The *database_connection* parameter is applied from a selection box, whose content is drawn from the Database Connections (page 48) management.

The *command* parameter is set using a text field. Entries made in this field are coded hexadecimally, which ensures that all the characters (in particular line feeds) are correctly coded in XML, although the value of this parameter is an attribute of the XML configuration. Should a script, which evaluates a parameter named *command*, be implemented for a job, then it is important to ensure that the script first of all decodes the values of the parameters.

Note that within job chains, order parameters overwrite job parameters of the same name, in so far as the job is allocated one of the predefined job types (page 46). This also applies to the *database_connection* and *command* parameters.

The *Store* button saves all the parameters of the current job or order shown on the dialog and updates the global parameters of all jobs or orders of the same type. After the parameters have been saved, the dialog is closed and...
the user returned to the editing dialog from which the Parameter Management dialog was opened. Note that the Store button is only shown to users possessing write permissions.

Clicking the Cancel button returns the user to the editing dialog from which the Parameter Management dialog was opened without saving any changes made to parameters.

The More Inputs button adds fields for a further five Local Settings-Free Parameters. Values already entered will not be saved but are not changed.

5.5.6 Scripts and Monitor Scripts in Jobs

This dialog is intended to simplify the writing of scripts through the use of the large text field in the Code part of the dialog. Further, the text field can function as a shell in which already available methods from the Job Scheduler API can be easily "wrapped".

Note that this dialog is only available for the Standard Job job type for script elements directly below the job element. This is because the script element is automatically configured for all other job types. This dialog is, however, available for the script element underneath the monitor element.

![Script Dialog Example](Image)

Scripts provide functions that can be called during a task, if required. Further information about this subject can be found in the Job Scheduler reference documentation.

The functions available are available:

- `spooler_init`
- `spooler_open`
- `spooler_process`
spooler_init is called once after the script has been loaded. Return values of true, 1 or empty (no return value) are all interpreted as true and allow processing to continue. Return values of false, 0, nothing or null are interpreted as false and cause processing to be stopped. spooler_exit is then called and the script ended. Should an error occur in spooler_init then it will be discontinued and spooler_exit started. spooler_init is particularly useful for creating objects, database connections, etc.

spooler_open is called at the start of a task. The return value is interpreted in the same way as the return values for spooler_init. Should spooler_open return false or cause an error, then spooler_close will be closed, otherwise spooler_process will be called. spooler_open is particularly useful for creating a number of objects and/or opening a connection (database, FTP server, etc.).

spooler_process interprets return values in the same way as spooler_init. With an order independent job, false causes a task to continue with spooler_close and true with a repeat call of spooler_process. With an order-independent job, false in spooler_process sets the order to the error state, whereas true starts the next job in the job chain. spooler_process is particularly suited in order independent jobs for the incremental processing of a number of objects, which, for example, have been created using im spooler_open. The implementation of spooler_process offers the advantage that progress of the job can be followed in the Job Scheduler interface and that the task can be stopped at a particular step if required.

spooler_close ends a task should the spooler_open or spooler_process methods return false or an error occurs. Either spooler_on_success or, in the event of an error, spooler_on_error after spooler_close. spooler_close is particularly useful for closing possible connections (database, FTP server, etc.).

spooler_on_success is called after spooler_close as long as no error has occurred.

spooler_on_error is called after spooler_close in the event of an error. The job will then be stopped.

spooler_exit is called immediately after a script has been closed. spooler_exit is particularly useful for clearing up any objects which may have been created.

A monitor script implements functions that are executed before and after a task is carried out. The monitoring scripts for order processing may implement the functions spooler_process_before and spooler_process_after, which in turn are carried out before and after the spooler_process function. The monitoring script can start a task or stop the spooler_process from being executed. Further information can be found in the Job Scheduler reference handbook.

The following predefined monitoring script functions are available:

- spooler_process_before
- spooler_process_after
- spooler_task_before
- spooler_task_after

The spooler_process_before function is called by the Job Scheduler before each spooler_process function and returns true/false. Should a return not have been implemented - for example, when it has been forgotten - then
this function returns true. false stops the spooler_process function from being carried out. In this case, the Job Scheduler proceeds as though spooler_process had returned the false value.

The Job Scheduler executes the spooler_process_after function after every spooler_process function and cannot return a true/false value. The return value from the spooler_process_after function replaces the return value from the spooler_process. The return value from the spooler_process can be handed over to the function as a parameter.

The spooler_task_before is loaded by the Job Scheduler before the relevant task. Should a return value have been forgotten, then this function returns true. The false return value stops the task from being started and the spooler_task_after function will not be executed.

The spooler_task_after function is executed by the Job Scheduler after the relevant task has been ended.

If the Java script language is specified, then the Java class with which the above methods have been implemented must also be specified.
For the other supported script languages (JavaScript, Perl, VBScript) the "Add predefined functions" selection box is provided in the top right of the interface, which is used to write an empty function into the Code text field. Alternatively, instead of entering a script in the code text field, it could also be saved in a separate file, and specified in the "Included file" field.

The arrows to the left of the text fields can be used to enlarge or reduce the size of the text fields as required. Note that clicking within text fields which are smaller than 14 lines automatically enlarges such fields.

The Store button returns the user to the Object Editor, which then evaluated the entries made. Note that changes made using the Script Editor are only saved when the object configuration is saved using the Object Editor. Note also that the Store button is only shown users possessing write permissions.

The Cancel button returns the user to the Object editor, without retaining entries made in the Script Editor.

It is recommended that a text editor is used to formulate functions and that these are then copied into the code text field by way of the clipboard before being saved.
5.5.7 Commands Editor

For every job exit code, it is possible to have either a follow-on job (start_job) or an order (add_order) start. These commands can be administered using the Commands Editor, which is called using the (editor interface (page 31)).

Click on the commands link in the top left, in order to add either new commands elements or to delete existing ones. New commands elements added are created with an exit-code in the following order: error, success, 1, etc. Exit codes can be edited individually when they are selected (to the left) which opens a edit form in the right hand part of the window. Should no exit code have been specified, then this is shown as ... Further information about commands elements can be found in the Job Scheduler reference documentation.

5.5.8 Job Chain Editor

The Job Chain Editor is used to administer job chains and is called using (editor interface (page 31)). This editor is intended to simplify administration, in particular, changing the order of job chain nodes. Further information about job chains can be found in the Job Scheduler reference documentation.
5.5.9 Buttons

The button ...

- **Cancel** closes the Editor dialog without saving any changes that have been made in the parameter values shown.
- **Store** saves changes made to the parameter values shown in the Editor forms in the database (table SCHEDULER_MANAGED_OBJECTS) after several validity tests. Should an error in the parameters be found then an appropriate message will be given out. If no errors are found, the Editor window will be closed after saving the values.
- **... and Activate**, in addition to save changes function, opens the dialog for submitting an object to the Job Scheduler (see Operations -> Submit (page 26) in the context menu).
- **... and Start** is offered for order-independent jobs and orders, in so far as the object has been saved below a Job Scheduler Hot Folder in addition to saving the object this function submits the object to the Job Scheduler and starts it. (see Functions -> Activate and Start (page 27) in the context menu)

5.6 Predefined Objects

Sample jobs and job chains can be found in the /Samples folder after the Job Scheduler has been set up. These jobs and job chains are intended to be used as examples on which users can base development of their own jobs.
and job chains. Each job chain can contain several orders of a particular type for carrying out similar tasks. Orders can have a unique run_time configuration, which allows them to be processed independently of one another.

The predefined example job and job chain objects cover the following key tasks:

- **Start an Executable File on the Server**
  /Samples/Launch Executable Files/...
  ...launch_executable_file (job chain)
  ...scheduler_managed_executable_file (job chain order job)
  ...launch_executable_file, order (job chain order)

- **Start a PHP File on the Server**
  /Samples/Launch Executable Files/...
  ...scheduler_managed_executable_php_file (order-independent job)

- **Execute SQL Statements or SQL Procedures in a Database**
  /Samples/Launch Database Reports/...
  ...launch_database_statement (job chain)
  ...scheduler_managed_database_statement (job chain order job)
  ...launch_database_statement, order (job chain order)

- **Read Data from a Database Using SQL, Incorporate the Data into a Report and Send the Report by E-mail**
  /Samples/Launch Database Reports/...
  ...scheduler_managed_database_report (order-independent job)

The Java implementation in the script elements of all the above jobs can be used both as order jobs in a job chain and as order-independent jobs.

### 5.7 Object Configuration

There are several possible ways in which each object can be configured, regardless of the object type.

1. **Importing an XML file using HTTP upload**
2. **Using an existing object as a prototype**
3. **Importing an XML file from the clipboard in the Object Editor after it has been created in the Object Explorer**
4. **By hand, without the help of an example, by first of all creating the object using the Object Explorer and then editing it using the Object Editor**

An XML file can be created or copied from the clipboard using, for example, the Job Editor (./bin/jobeditor.(sh|cmd)).

With both the HTTP import (page 22) and using an existing object as a prototype (page 23) provide an already configured but possibly inactive object. The other two methods require that an object is created in the Explorer (page 22). With all four methods, however, it is necessary to use the Editor (page 28): on the one hand, simply to activate the object, and on the other, either to use the XML View of the Clipboard (page 29) for importing or to use the forms in the Editor.

#### 5.7.1 Locks

Locks have no child elements and only one attribute and the Object Editor lock edit form is therefore correspondingly simple.
The minimum requirement for the imported XML is:
<lock />

5.7.2 Process Classes

Process classes have no child elements and only two attributes and the Object Editor process class edit form is also relatively simple.

The minimum requirements for an imported XML file are:
<process_class name="[process class name]" />

The name of the process class must be unique among all the process classes being managed and may not contain any special characters such as /:*?"<>].
5.7.3 Jobs

Jobs have the largest number of child elements, and for this reason their configuration is supported by several dialogues: (Parameter Management (page 33), Script Management (page 34) and the Run Time Editor (page 31)).

The minimum requirements an imported XML job file must meet are:

```
<job>
  <script ...>...</script>
</job>
```

or

```
<job>
  <process .../>
</job>
```

Every job requires either a configured script or process element.

`script` and `process` elements are mutually exclusive - that is, once a decision for a `script` element has been reached, then a `process` element can only be submitted after the `script` element has been deleted - and vice-versa. The `job` element context menu in the Object Editor is now opened in order to add the `script` or `process` element (e.g. new child -> script) and then click in the newly added child element in the left hand object tree. This will now show the attributes of the child element in the right hand part of the Editor interface.

For a `script` child element, a script language must be specified using the "language" selection box and a script written in the language specified provided. This script can also be saved in a separate file and its file name specified in the "file name" field.
The script element is configured by clicking on the symbol alongside the script element. This opens the Script Configuration Dialog (page 34) in a new window.

For a script written in Java, only the Java class to be used need be specified. However, note that the Job Scheduler to which the job will be submitted, must have the corresponding Java archive in its class path. The class path is configured in the Job Scheduler's ./config/factory.ini file, in the classpath entry in the [java] section. Should it be necessary to add this information to the Job Scheduler factory.ini file, then it will be necessary to restart the Job Scheduler before the job can be carried out.

For scripts written in JavaScript, Perl or VBScript, the script can either be directly pasted or written into the large text field in the "Code" part of the Script Editor, or it can be saved in a separate file. If the script is saved in a separate file, then the path to this file must be entered in the large text field in the "Included files" part of the Script Editor. The path to this file must be relative to the Job Scheduler to which the file is to be submitted, i.e. when using ...

- ... file attributes in the include element:
  the path must either be absolute or relative to the Job Scheduler working directory;
- ... live_file attributes in the include element:
  the script file must also be saved in the Hot Folder and the path selected relative to the job.

Note also that it is important on Unix machines that sufficient permissions are given.

A process element must be specified in the the file attribute of the executing object. The path to this file must either be absolute or relative to the Job Scheduler to which the file is to be submitted. Note also that on Unix machines appropriate permissions must be allocated to this file.

At this point it is recommended that instead of using the process element, the Executable File job type is selected. This causes the script element to be automatically replaced by a Java class known to the Job Scheduler. Further, parameters will be offered which can be easily set using the Parameter Management (page 33). This is particularly relevant in the case of the timeout parameter, which stops the process after a specified time.

Should either the process_class attribute or the lock.use element be set, then it is important that the corresponding process class or lock have been submitted to the Job Scheduler, otherwise it will not be possible to initialise the job.

Note that when parameters are added to a job, it is important that no special characters (\/:*?"<>|) are used in the name attribute of the param element.

### 5.7.4 Job Chains

Job chains have only 5 child elements but nevertheless there are a number of points to observe. These are taken account of in the support provided by the Job Chain Editor (page 37) for this object type.

The minimum requirements an imported XML job chain file must meet are:

```
<job_chain>
  <[child_element] />
</job_chain>
```
where [child_element] is one of the 5 child elements.

The name of the job chain must be unique among all the job chains being managed and may not contain any special characters (V:,*?"<>|). Note that although the minimum requirement specified above is sufficient to allow the import, such a job chain would cause the Job Scheduler to return an error as it does not contain any job chain nodes.

The five job chain child element types cannot all be included in a single job chain - which can either contain the file_order_source, job_chain_node, job_chain_node.end and file_order_sink elements, or the job_chain_node.job_chain and job_chain_node.end elements.

In the first case, the jobs in the job chain are specified in the job attributes of the job_chain_node elements. The order in which the jobs are carried out are specified by the state, next_state and error_state attributes. For each job chain node in which either a next_state or error_state is specified, a node must be provided which accepts the corresponding state. In order to terminate a job chain, end nodes must be specified. These are either the job_chain_node.end or file_order_sink nodes, in which neither the next_state nor the error_state are specified.

The second case mentioned above (where the job chain contains job_chain_node.job_chain and job_chain_node.end elements) is a job chain consisting of job chains. Here job chains are specified in the (parent) chain by adding the job_chain attribute to the job_chain_node.job_chain element. The order in which the job chains are executed is, once again, specified using the state, next_state and error_state attributes. Furthermore, for each node in the (parent) job chain, in which a next_state or error_state is specified, a corresponding node must exist for the job chain.

Example: Example of a Simple Job Chain

```
<job_chain>
  <job_chain_node state="0"
    job="scheduler_managed_executable_file"
    next_state="success"
    error_state="error" />
  <job_chain_node_end state="success" />
  <job_chain_node_end state="error" />
</job_chain>
```

Note that the jobs in a job chain must be order jobs. It is also important to ensure that all the jobs specified in a job chain have already been submitted to the Job Scheduler. If this is not the case, then it will not be possible to initialise the job chain.

A job chain is started by an order. Such an order can also take the form of a file order, which is specified using the file_order_source element in the job chain. The file_order_sink element is used to delete or move the file order triggering file. The file_order_sink element also represents a job chain end node.

The second case mentioned above (where the job chain contains job_chain_node.job_chain and job_chain_node.end elements) is a job chain consisting of job chains. Here job chains are specified in the (parent) chain by adding the job_chain attribute to the job_chain_node.job_chain element. The order in which the job chains are executed is, once again, specified using the state, next_state and error_state attributes. Furthermore, for each node in the (parent) job chain, in which a next_state or error_state is specified, a corresponding node must exist for the job chain.
the *state* specified. In order to end the job chain, end nodes must be defined using the *job_chain_node.end* element.

Here too, it is important that all child (job chain) elements of the parent job chain have been submitted to the *Job Scheduler* before the parent chain, otherwise the parent job chain will not be initialised.

### 5.7.5 Orders

Orders only have two child elements - *params* and *run_time*.

The minimum requirements for an imported XML order are:

```
<order/>
```

If a *run_time* element is set for an order, then the order will become persistent - otherwise it will be deleted after the end of the job chain has been reached. Orders are particularly suited for starting a job in a job chain with parameters. However, such a job must be able to process the parameters handed over from the order.
The idea of using an order job in a job chain means that the order implementation can remain unchanged. Instead, the orders are run with differing parameters that cause them to achieve the desired results. This allows standard jobs to be developed, such as those delivered with the Job Scheduler Managed Jobs packet (see Predefined Objects (page 38)).

Note that it is important to ensure that the job chain specified in an order has already been submitted to the Job Scheduler, otherwise it will not be possible to initialise the order.

When parameters are specified for an order, it is important to ensure that no special characters (\/:*?"<>|) are included in the name attribute of the param element.

5.7.6 Schedule

Schedules are submitted to order-independent jobs and orders as Run Time, and their configuration is supported by the Run Time Editor (page 31).

The minimum requirements for an imported XML schedule are:

```xml
<job>
<script language="javascript" order="yes"> <![CDATA[
function spooler_process() {
    var order_params = spooler_task.order.params;
    var param_names = order_params.names.split("; ");
    for( var i in param_names ) {
        spooler_log.info( param_names[i] + "=" + order_params( param_names[i] ) );
    }
    return true;
}
]]> </script>
</job>
```

Example: Example of an Order Job, which Processes Order Parameters

```xml
<job>
<script language="javascript" order="yes"> <![CDATA[
function spooler_process() {
    var order_params = spooler_task.order.params;
    var param_names = order_params.names.split("; ");
    for( var i in param_names ) {
        spooler_log.info( param_names[i] + "=" + order_params( param_names[i] ) );
    }
    return true;
}
]]> </script>
</job>
```
The substitute, valid_to and valid_from attributes are mandatory for replacement schedules, which contain a validity period and the schedule that is to be substituted for this period.

5.8 Job and Order Types

Jobs and orders are grouped together as types in the Workflow Management in order to simplify the assignment of parameters. The global and local parameters for each type of job and order are specified in the settings dialog. This information is then used in the Editor (page 28) to generate different parameters according to the type of job or order. Further information can be found in the Settings chapter (page 51). Note that all jobs of a particular type will be configured with the same script element.

Management of job and order types is carried out using the Job & Order Type Editor (not the Editor (page 28)). The Job & Order Type Editor is reached using the Workflow > Job Types and Order Types link in the main menu.

When first opened, the Job & Order Type Editor has two parts:

- An (edit order types / job types) form for the addition of new job and order types as well as for editing and deleting existing jobs and orders.
- A list of the order and job types that have already been saved. This list is used to select the job and order type to be edited by clicking on the Type entry for a job or order type. This transfers settings for the respective job or order type into the edit order types / job types form in the upper part of the editor.

Should global parameters have been created for a job type, then these will be shown in a (Global Parameters ...) table positioned between the form and list described above. This table can be seen in the screen shot below. Entries made here will be used as default parameters in newly created jobs. When the Web Interface is being used in conjunction with PHP 5 or higher, then, in addition, these parameters will be set for all existing jobs of this type but can be overwritten in individual jobs using the Editor.

The job and order types shown here can be selected in the Editor (page 28) and individually modified as required.
edit Order Type or Job Type Parameters:

The *type* and *title* fields at the top of the order and job type editor determine the internal names and title used for the job or order type in the edit job or edit order form selection boxes, both in this editor and in the (job and order) Editor (page 28). Values entered in both these fields must be unique amongst all (job and order) types. The *type* field may contain no special characters other than the underscore ("_").

Entries made in the *script* field specify the Java class, JavaScript, etc. which is to be executed by a job and must be in XML format. Sample content for the *script* element of the XML configuration can be found in the Job Scheduler documentation. Note that the *script* field is not offered for *standard* order or job types. The relevant information for these jobs or orders must be entered in the job or order editor.

Accept, Remove & set Standard Script Buttons:

The *accept* button is used to write entries in the `SCHEDULER_MANAGED_JOB_TYPES` database table. Should a new job or order *type* have been selected, then a new job or order type will be created. Should, however, the *type* already exist, then it will be updated. Note that the *accept* button is only shown to users with write permissions.
The `remove` button deletes the selected job or order type after a confirm removal dialog. Note also that the `database`, `database_report`, `custom_report`, `file` and `standard` types cannot generally be removed. Further, the `remove` button is only shown to users possessing write permissions.

The `set standard script` button is made available for the `database`, `database_report`, `custom_report`, `file`, `ftp_receive`, `ftp_send` and `interpreter_*` job and order types. Note that `interpreter_*` is used here to describe all types whose name begins with `interpreter_`.

The `set standard script` resets the content of the `script` field back to the following values:

<table>
<thead>
<tr>
<th>type</th>
<th>script</th>
</tr>
</thead>
</table>
| `file`          | `<script language="java"
java_class="sos.scheduler.managed.JobSchedulerManagedExecutableJob"/>` |
| `interpreter_*` | `<script language="java"
java_class="sos.scheduler.managed.JobSchedulerManagedExecutableJob"/>` |
| `database`      | `<script language="java"
java_class="sos.scheduler.managed.JobSchedulerManagedDatabaseJob"/>` |
| `database_report` | `<script language="java"
java_class="sos.scheduler.managed.JobSchedulerManagedDBReportJob"/>` |
| `custom_report` | `<script language="java"
java_class="sos.scheduler.managed.JobSchedulerManagedCustomReportJob"/>` |
| `ftp_receive`   | `<script language="java"
java_class="sos.scheduler.ftp.JobSchedulerFTPReceive"/>` |
| `ftp_send`      | `<script language="java"
java_class="sos.scheduler.ftp.JobSchedulerFTPSend"/>` |

Table 8. Standard Java Classes for Selected Job Types

The `set standard script` button does not save information contained in the form and is therefore available to all users.

### 5.9 Database Connections

The database management dialog is reached using the `Workflow > database connection` link in the web interface main menu.

The dialog which opens is similar to the job and order type dialogs described in the previous sections. The top part of this interface is a form which is used for adding a new database connection or for changing or deleting an existing connection. The bottom part of the interface is a list of the database connections which have already been created. As with the job and order type dialogs, the `name` entries in the leftmost column of this database connections list are links. When these links are clicked, the settings for the relevant connection are copied into the form above, where they can be modified or the connection deleted if required.

The database connections shown in this dialog are all available in the `connection` field of the Editor (page 28).
The name and title fields specify the internal name used for the connection and the title used in the job and order Editor. Values entered in both fields here must be unique amongst all connections. The name may not contain any special characters apart from the underscore ("_”). Selection of a database using the dbms selection box sets the JDBC driver and the connection url as follows:

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Driver</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>com.mysql.jdbc.Driver</td>
<td>jdbc:mysql://[host]:3306/[database]</td>
</tr>
<tr>
<td>Oracle</td>
<td>oracle.jdbc.driver.OracleDriver</td>
<td>jdbc:oracle:thin:@[host]:1521:[database]</td>
</tr>
<tr>
<td>MSSQL 2000</td>
<td>com.microsoft.jdbc.sqlserver.SQLServerDriver</td>
<td>jdbc:microsoft:sqlserver://[host]:1433;databaseName=[database]</td>
</tr>
<tr>
<td>MSSQL 2005</td>
<td>com.microsoft.sqlserver.jdbc.SQLServerDriver</td>
<td>jdbc:sqlserver://[host]:1433;databaseName=[datenbank]</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>org.postgresql.Driver</td>
<td>jdbc:postgresql://[host]:5432/[database]</td>
</tr>
<tr>
<td>DB2</td>
<td>com.ibm.db2.jcc.DB2Driver</td>
<td>jdbc:db2://[host]:50000/[database];driverType=2;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>retrieveMessagesFromServerOnGetMessage=true;</td>
</tr>
<tr>
<td>Sybase</td>
<td>com.sybase.jdbc3.jdbc.SybDriver</td>
<td>jdbc:sybase:Tds:[host]:5000/[database]</td>
</tr>
</tbody>
</table>

Table 9. Default Database Settings

Note that the value for the host field is entered in [host] and the value of the database field in [database]. Should no value be entered for Host, then localhost will be used. Similarly, test_db is the default value for the database field.

The name, title, dbms, driver and url fields are mandatory. The URL can be a standard URL which is generated by host, port and database or in any other syntax depending on the active radiobutton.

The accept button writes the entries held in the form into the SCHEDULER_MANAGED_CONNECTIONS database table. Should the name selected be new, then a new connection will be created. Should the name, however, already exist, then the values stored under this name will be overwritten. Note that the accept button is only available to users having write permissions.
The *remove* button deletes a connection after a confirm deletion dialog. Note that no job or order are being used the database connection. In this case the configuration of the relevant jobs and orders needs to be changed to another connection. Note that the *remove* button is only available to users having delete permissions.
6 Settings

The settings management dialog is reached using the *Settings* link in the left hand frame of the web interface.

After the settings dialog has been opened, the topmost element in the dialog is the *Ranges* item. (The *All Settings* link positioned at the top of this - and all other parts of the settings dialog - is a navigation aid used to return directly to this topmost level of the settings dialog.)

6.1 Job Scheduler Networks

Click on the "Job Scheduler Network" application in the settings management dialog. A new *Job Scheduler* is added to the network using the "(New Section)" link.

The basic attributes that are then to be set for the new Job Scheduler / section are:
- "Name": the Name entered must be unique amongst all the *Job Schedulers* in the network (e.g. [my_scheduler_id]@[my_host]),
- "Title": it is recommended that the title entered is identical to the "Name".

The Save as button is used to save the information entered and to return to the sections list.

When a particular section is to be modified or deleted, then the arrow next to the section title should be clicked. This opens the a dialog with which the Job Scheduler / section can be modified and saved.
Once the section has been added, click on the section title in the sections list, in order to open the section settings dialog. This dialog contains entry fields for the Alias, Host, Port, SchedulerID and Supervisor settings.

A proxy configuration may mean that the Job Scheduler address may differ from http://host:port/. In this case, enter the changed URL in the Alias field.

Host, Port and SchedulerID must be entered. The SchedulerID must be unique amongst all IDs. This must be the ID that is specified in the Job Scheduler start script (. /bin/jobscheduler.(sh|cmd)).

If the Job Scheduler is to be a Workload Scheduler then the host and port of the Supervisor (seperated by a colon ":") must be entered.

Close the dialog and save the settings using the Store button.

Once a change has been made to a network, the ...check link below "Registered Schedulers" should be clicked. This actualises the interface session.

6.2 Parameters

Settings can be created for each job and order type. The settings management dialog is reached using the "Settings" link in the web interface main menu. Settings are divided into so-called ranges and schemas. Parameter names are entered in the schemas and used in the Editor (page 28) where they are shown along with their parameters. Parameters can be either global or local:

- global parameters apply to all orders or jobs of a particular type
- local parameters apply only to a specific job or order.

At the time of writing only local parameters are supported in orders and global parameters are only used for jobs to set default values when creating a job.

6.2.1 Creation of Parameter Management Ranges

The (all settings) link positioned at the top of all the settings (sub) dialogs can be used to return to this topmost dialog level. The first link in the list of ranges opens a dialog for the creation of a new range. The dialog for editing existing ranges is opened by clicking on the arrow to the left of the title of a range.

In the dialog for the creation of a new range the Name, Title and Input format fields are mandatory and the Description field optional. The Input format should always be set to Schema table. There are no restrictions on the content of the Title field, which is used within the settings and Parameter Management (page 33) dialogs as title of the range. The Name must be unique amongst all ranges and conform with the following:

<table>
<thead>
<tr>
<th>Range Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_type/global/[typ]</td>
<td>Range for the global order parameters for the [type] order type (is currently not supported)</td>
</tr>
<tr>
<td>order_type/local/[typ]</td>
<td>Range for the local order parameters for the [type] order type</td>
</tr>
<tr>
<td>order_type/mixed/[typ]</td>
<td>Range for the mixed order parameters for the [type] order type (is currently not supported)</td>
</tr>
<tr>
<td>job_type/local/[typ]</td>
<td>Range for the local job parameters for the [type] job type</td>
</tr>
<tr>
<td>job_type/mixed/[typ]</td>
<td>Range for the mixed job parameters for the [type] job type</td>
</tr>
</tbody>
</table>

Table 10. Possible Range Names

Only local parameters are currently shown in the Editor, whereas global parameters are shown in the Job Types (page 46).
6.2.2 Creation of Parameter Management Schemes and Sections

Clicking on the title of a range in the list of ranges shown in the topmost level of the settings dialog opens the list of sections of the selected range (if any). The (Schema in the range) link, which opens the list of schemas belonging to the selected range, is also to be found here. The first item in this list of schemas is (new entry) which is used to open the dialog used to create a new schema. Under the (new entry) link is a list of entries which have already been made in this schema, and which can be edited using an edit entry dialog. This dialog is opened by clicking on the title of the entry.

Here the Name, Input (format, number of characters, mandatory) and Display (format, number of pixels) fields are mandatory and the Default and Description fields optional.

The text entered in the Name field must be unique amongst all similar entries in this schema and with the exception of the underscore (“_”) may not contain any special characters. Schema names are used in the Parameter Management for the names of parameters.

The Title of a schema is used in the settings and in the Parameter Management (page 33). Should a default entry be required to appear here, then this should be entered in the Default box.

Should either "List box" or "Radio" Display formats be selected, then the content of the list box or the texts and values of the radio buttons are entered in the Default field as a list separated by semicolons.

Should the Input "format" be set to "boolean", then radio buttons will automatically be given values of 0 and 1 and labelled "Yes" and "No". In this case default values will be overwritten.

The content of the Description field, when entered, is available as a help text in the Parameter Management (page 33).

The script allocated to a job or order type determines the parameters which are used for that job or order type. The table below contains the parameters which are used for the file, interpreter_*, database, database_report, ftp_receive and ftp_send job and order types. Note that more information about this topic can be found in the chapter on Job & Order Types (page 46) above, in so far as the standard Java classes mentioned above are used in the relevant job or order type.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter Name / Input Format / Number of Characters / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>ignore_error/Boolean//Ignore error on execution?</td>
</tr>
<tr>
<td></td>
<td>ignore_signal/Boolean//Ignore signals on execution ?</td>
</tr>
<tr>
<td></td>
<td>log_file/String/250/Log file for the executed command</td>
</tr>
<tr>
<td></td>
<td>timeout/Number/10/Timeout for the execution of commands</td>
</tr>
<tr>
<td>interpreter_*</td>
<td>as per file plus</td>
</tr>
<tr>
<td></td>
<td>interpreter/string/250/Interpreter</td>
</tr>
<tr>
<td>database_report</td>
<td>scheduler_order_report_asbody/Boolean//Send report as e-mail content?</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_body/String Text Area/1000/Layout of report e-mail</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_filename/String/250/Name of report file</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_mailbcc/String/250/E-mail addresses of blind carbon copy recipients</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_mailcc/String/250/E-mail addresses of carbon copy recipients</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_mailto/String/250/E-mail addresses of report recipients</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_path/String/250/Report path</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_stylesheet/String/250/Transformation style sheet</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_subject/String/250/Subject of report e-mail</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_send_if_no_result/Boolean//Send report if no result exists?</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_send_if_result/Boolean//Send report if a result exists?</td>
</tr>
</tbody>
</table>
Table 11. Parameters for use with standard Java classes

This table shows how schema entries are allocated in type-dependent ranges.

Sections can be created after the relevant schema has been created. Note that not every range requires that a section is created. This also means that not all sections are interpreted in every range.

Only section names which correspond with unit names stored in the User Administration are interpreted for the order_type/global/... and order_type/mixed/... ranges. Note that a section will only be shown in the Parameter Management when an order has been assigned to a unit in the User Administration. Note also that only the defaults section is allowed for the job_type/mixed/... range.

The (New section) link should be used to create a new section.

When creating a new section, the Name and Title fields are mandatory and the Description field optional. There are no restrictions on the Title field, which is used in the settings interface. Entries made in the Name field must follow the format described at the start of this section, regardless of the range in which the section is to be used.

When new scripts are being written that are allocated a job type and are to use parameters that are entered here, then it is important that the database_connection and command parameter names are reserved and not used here. Both these parameters give the (database) connection and SQL statement or command file fields from the edit orders and edit jobs dialogs.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter Name / Input Format / Number of Characters / Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>custom_report</td>
<td>sas per database_report plus</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_language/String Listbox/JavaScript;PerlScript;VBScript/250/Script language for reports</td>
</tr>
<tr>
<td></td>
<td>scheduler_order_report_template/String/250/Template (HTML, Text etc.)</td>
</tr>
<tr>
<td>ftp_receive</td>
<td>ftp_host/String/250/FTP server host</td>
</tr>
<tr>
<td></td>
<td>ftp_port/Number/5/FTP server port</td>
</tr>
<tr>
<td></td>
<td>ftp_user/String/250/FTP user</td>
</tr>
<tr>
<td></td>
<td>ftp_password/String/250/FTP password</td>
</tr>
<tr>
<td></td>
<td>ftp_account/String/250/FTP account</td>
</tr>
<tr>
<td></td>
<td>ftp_transfer_mode/String/250/transfer mode (binary or ASCII)</td>
</tr>
<tr>
<td></td>
<td>ftp_passive_mode/Boolean//Use passive mode?</td>
</tr>
<tr>
<td></td>
<td>ftp_remote_dir/String/250/FTP server directory</td>
</tr>
<tr>
<td></td>
<td>ftp_local_dir/String/250/Local directory</td>
</tr>
<tr>
<td>ftp_send</td>
<td>as per ftp_receive</td>
</tr>
</tbody>
</table>
7 Job Types

Managed Jobs have four predefined job types (database statement, database report, custom and executable file) for the Managed Database, the Managed Database Report, the Managed Custom Report Job and the Managed Executable Jobs respectively. All other jobs are allocated the "other jobs" job type.

7.1 The Managed Database Job Type

The Managed Database Job is a special Managed Job. This job is implemented by the JobSchedulerManagedDatabaseJob class, and can execute statements in a database. These can be database procedures or SQL statements. The Managed Database Job takes on orders which have been defined in the SCHEDULER_MANAGED_OBJECTS table. The job uses the database connection defined in the database_connection order parameter to carry out the instruction stored in the command parameter.

It is possible to define more than one instruction in the command parameter. Such instructions are then carried out in the order in which they are written and must be separated by a semicolon and a subsequent new line. In this way it is possible, for example, to realize time-controlled procedures for MySQL 4.x databases.

Note that the job does not return a COMMIT after carrying out the command. Should a COMMIT be required, then this must be stated in the database command.

Parameters can be set for database commands This is done by the addition of a variable in the form \${param} at any given point in a command. This variable is then given the value of the instruction parameter with the name param before execution.

Example: SQL Statement with a Parameter

UPDATE EMP SET SAL=SAL*1.2 WHERE JOB='\${jobname}';

Should the order pass a parameter named jobname and with value CLERK, then the instruction will be replaced by:

UPDATE EMP SET SAL=SAL*1.2 WHERE JOB='CLERK';

The following standard parameters are always present:

- \${scheduler_id} The Job Scheduler ID
- \${scheduler_order_id} Order ID
- \${scheduler_order_job_name} Job Name
- \${scheduler_order_job_id} Job ID

7.2 The Managed Database Report Job

This job is an extension of the Managed Database Job and executes database queries in exactly the same manner. A report is generated as the result of the last select statement and then either sent per e-mail or saved in a directory.

The report is first created in an XML format, which contains every line of the database query result set. The job can send the report in this format - it can also transform the report with the help of a style sheet into HTML or another XML format. The default.xslt style sheet, which is delivered in the Job Scheduler config directory is intended for use in such transformations. This style sheet transforms the report into a HTML table.
A Managed Database Report job is first of all configured in the same manner as a Managed Database Job. The following parameters are then specified in the order (or in the job) to configure the report.

- **scheduler_order_report_stylesheet**: The path to the transformation style sheet. Should this parameter not be set, then the report will not be transformed and it will be sent in the original XML format.
- **scheduler_order_report_mailto**: The e-mail recipient of the report
- **scheduler_order_report_mailcc**: The cc e-mail recipient of the report
- **scheduler_order_report_mailbcc**: The bcc e-mail recipient of the report
- **scheduler_order_report_subject**: The subject of the report e-mail
- **scheduler_order_report_body**: The body of the report e-mail
- **scheduler_order_report_asbody**: If this parameter is set to true, then the (HTML) report will be sent as the body of the e-mail. The **scheduler_order_report_body** parameter will then be ignored.
- **scheduler_order_report_filename**: The file name specified here is used for the mail attachment and for the saved version of the report, should a report path have be specified.
- **scheduler_order_report_path**: If a path is specified here, then a copy of the report will not only be sent by e-mail, but will also be saved in this directory.

The following tags can be used in specifying the **scheduler_order_report_path**, the **scheduler_order_report_filename**, the **scheduler_order_report_body** and the **scheduler_order_report_subject** parameters:

- **[date]**: The current date
- **[datetime]**: The current date and time
- **[orderid]**: The Order ID
- **[jobname]**: The Job Name
- **[taskid]**: The Task ID
- **[sql]**: The SQL Query used to create the report
- **[xml]**: The non-transformed XML result of the database query

### 7.3 Managed Custom Report Job

This job is an extension of the Managed Database Job. As with the Database Report Job, database statements are used to generate a report which in turn is sent by e-mail or may be saved in a directory.

There are, however, two basic differences between the Managed Custom Report Job and the Database Report Jobs:

- In the Managed Custom Report Job, not only the last statement is evaluated - results from all SELECT statements can be used;
- In the Managed Custom Report Job, the content of the (HTML) report is set with a template containing HTML and script code blocks and not using an XSLT style sheet. The script code for this template may be written in JavaScript, Perl, or VBScript. Although this approach allows more flexibility in the generation of reports, in some cases it can be more involved than using a simple style sheet.

The following example should illustrate the use of a Managed Custom Report:

Consider the situation where two reports are required which show all the jobs which have been run on a Job Scheduler in a table and where each report is compiled for a different time span - one report lists the jobs which have been run on the previous day and the other the jobs run over the last 7 days. Both reports are to be sent by e-mail and the reports should be created by two separate orders.

This task can be solved in a number of different ways:

- The SQL statements and the necessary text (template) for the reports can be written into the Custom Report Jobs job configuration. The different time spans for each report can be specified in the orders using variables and start times. This approach has the advantage of being modular, so that the SQL statements and the template need only be specified once. The disadvantage of this approach is that a job is used to carry out a very specialized task and thereby blocks the job chain for other purposes.
The SQL statements and the necessary template texts are written into each order along with the order start times. No further information is specified in the Custom Report Jobs job. This approach has the advantage that the job and job chain are freed up for other tasks and the disadvantage that the same SQL statements and templates must be specified more than once.

In this example the second solution will be followed as the advantage of the freeing up of the job chain outweighs the relatively small amount of additional work involved in specifying the statements and templates in each order.

The standard implementation of a Custom Report Job and the corresponding job chain can be found as samples in the /Samples/Launch Database Reports/scheduler_managed_custom_report and /Samples/Launch Database Reports/launch_custom_report directories respectively. The (predefined) Launch Custom Report job with Customized Report as job type will be used here. In this example it is not necessary to specify any SQL statements in this job. Only the times when the job is allowed to run are entered under "run time". The job is to be started by orders, which is the standard setting for the Launch Custom Report job and which can be used unchanged.

The following two orders are created for this example: "Job Summary - 1 day" and "Job Summary - 1 week". The Customised Report order type is selected.

As SQL is to be used, a database connection is required - MySQL will be used in this example. Note that the connection must be made using a database account with sufficient permissions - in this case reading, creating and deleting tables. These permissions are necessary as temporary tables will be created and then deleted after the report has been completed.

The SQL required is entered in the command parameter field. It is recommended that the code is first of all written with an editor, tested and then copied into this field. The start period is entered in the run time field. The following order should be run once a day at 00:01 (single_start = "00:01"):

In order to use the results of a database query in a HTML template, an SQL comment containing the keyword "variable" is written after the SELECT Statement. This keyword is then followed by the name of a variable - which can then be used in the template.

The job SQL queries contain the /* variable variable_name */ variables, which in turn contain the results of the query in the form of a two-dimensional JavaScript array. The first dimension of this array contains the query results records; the second dimension the field name (written in lower case).

Example: Assignment of a SELECT Result

```
SELECT * FROM emp;
/* variable emptest */
```

This makes a two dimensional array named "emptest" available in the template, containing all the rows and columns of the query results. The first element of, for example, a "JOB" column in this array would be accessed in JavaScript using `emptest[0]["job"]`. Note that the column name is written here in lower case.
Example: The Example SQL Statements

```
SELECT DATE_SUB(CURRENT_DATE(), INTERVAL ${retrospect} DAY) AS "ENTRY";
/* variable since */

SELECT "SPOOLER_ID" AS "JOB_SCHEDULER", "JOB_NAME", "ERROR", COUNT(*) AS "JOB_COUNT",
"END_TIME"-"START_TIME" AS "ELAPSED" FROM SCHEDULER_HISTORY
WHERE "START_TIME" >= DATE_SUB(CURRENT_DATE(), INTERVAL ${retrospect} DAY)
AND IFNULL("END_TIME", DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY)) <= CURRENT_DATE()
GROUP BY 1, 2, 3, 5;
/* variable overview */

SELECT "SPOOLER_ID" AS "JOB_SCHEDULER", "JOB_NAME", "ERROR", "START_TIME", "END_TIME",
"END_TIME"-"START_TIME" AS "ELAPSED" FROM SCHEDULER_HISTORY
WHERE "START_TIME" >= DATE_SUB(CURRENT_DATE(), INTERVAL ${retrospect} DAY)
AND IFNULL("END_TIME", DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY)) <= CURRENT_DATE()
GROUP BY 1, 2, 3, 4, 5, 6;
/* variable details */
```

This code creates three two-dimensional arrays named respectively "since", "overview" and "details" in the template.

The order uses the same parameters as the Database Report Job, plus the following:

- **scheduler_order_report_language**: - the script language used in the template. Valid values are JavaScript, PerlScript and VBScript.
- **scheduler_order_report_template**: - the template file - i.e. HTML and script - for the report.

The `${retrospect}` parameter is used in the SQL statement for this example, its value being manually set alongside the order type parameters.

For the daily report, retrospect is set to 1 and for the weekly report to 7. Order parameters are used in SQL scripts in the form ${parameter}. The order parameters must be valid SQL, and are automatically substituted when the job is being executed.

When an order is created for the Custom Report job, then the path to a template file must be specified in the order parameters. This file - which should be in HTML format - is used to layout the results of the SQL statement.

The HTML report template file is saved in the file system, and the file name and address specified in the scheduler_order_report_template parameter. The file address can be specified relative to the Job Scheduler installation directory. The report template can be edited using a HTML editor.

As described above, the template text can contain variables which can then be substituted during job execution or processed by a script language:

- variables are specified in HTML code in the form $(variable)
- script elements are bracketed using `<%` and `%>`.

In JavaScript the rows in the first dimension of the array are addressed numerically - in the example above `emptest[2]` would refer to the third row of results. The individual elements in a row are then addressed by the row name - e.g. `emptest[2]["job"]` would address the job element in the third row of the array.

The rows are also addressed numerically in VBScript - i.e. `emptest(2)`. Every line in VBScript contains a Scripting.Dictionary object, so that individual elements can be accessed by their names - e.g. `emptest(2).Item("job")`.

In Perl the rows are addressed numerically as well - e.g. `$emptest[2]` Elements are accessed in Perl using a HashMap - e.g. `$emptest[2]->{"job"}`.

The code block in a template is enclosed in `<%` and `%>` characters. A variable can also be output in HTML (i.e. outside a code block) using a $ or $(variable).
Note:
It is recommended that the "Try Out" function from the order context menu is used to test a template during its development. However, before this is done, both the order job and job chain mentioned above must be submitted to the Job Scheduler.
In addition, it is recommended that the log file is opened, so that the job progress can be followed. This is done using "Job Scheduler" -> "Monitoring" in the Web Interface main menu and then clicking on "Show log".

Example: Template for Outputting a Field (in JavaScript)

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<title>test</title>
</head>
<body>
<p>JavaScript test</p>
<% var line = emptest[2]; // third line
    var jobname = line["job"]; %>
<p>$jobname</p>
</body>
</html>
```

Should a query return a number of sets of results, these can be written into a table by iterating through the variables and writing each set of results into a table row.

Example: Outputting Results in a Table (in JavaScript)

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
<title>test</title>
</head>
<body>
<p>JavaScript test</p>
<table>
<% for(i=0; i<emptest.length; i++ ) {%>
<tr>
    <td>${emptest[i]["job"]}</td>
    <td>${emptest[i]["ename"]}</td>
</tr>
<% } %>
</table>
</body>
</html>
```
Example: Excerpt from Example Template

...  
```html
<p><strong>Job Runtimes since $(since)</strong></p>

<table width="100%" border="1" class="report">
<tr>
<th class="report" width="17%">JOB_SCHEDULER</th>
<th class="report" width="28%">JOB_NAME</th>
<th class="report" width="7%">ERROR</th>
<th class="report">JOB_COUNT</th>
<th class="report" width="10%">ELAPSED</th>
</tr>
<tr>
<td>$\{\text{overview[i]["job_scheduler"]}\}</td>
<td>$\{\text{overview[i]["job_name"]}\}</td>
<td>$\{\text{overview[i]["error"]}\}</td>
<td>$\{\text{overview[i]["job_count"]}\}</td>
<td>$\{\text{overview[i]["elapsed"]}\}</td>
</tr>
</table>
```

Job Order Summary

Job Runtimes since 2006-03-30

**Job Overview:**

<table>
<thead>
<tr>
<th>JOB_SCHEDULER</th>
<th>JOB_NAME</th>
<th>ERROR</th>
<th>JOB_COUNT</th>
<th>ELAPSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler.fs (Spoon)</td>
<td>(Spoon)</td>
<td>0</td>
<td>1</td>
<td>NULL</td>
</tr>
<tr>
<td>scheduler.fs (Spoon)</td>
<td>(Spoon)</td>
<td>0</td>
<td>1</td>
<td>2243</td>
</tr>
<tr>
<td>scheduler.fs</td>
<td>scheduler_managed_custom_report</td>
<td>0</td>
<td>1</td>
<td>NULL</td>
</tr>
<tr>
<td>scheduler.fs</td>
<td>scheduler_managed_custom_report</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>scheduler.fs</td>
<td>scheduler_managed_starter</td>
<td>0</td>
<td>1</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Job Details**

<table>
<thead>
<tr>
<th>JOB_SCHEDULER</th>
<th>JOB_NAME</th>
<th>ERROR</th>
<th>START_TIME</th>
<th>END_TIME</th>
<th>ELAPSED</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduler.fs (Spoon)</td>
<td>(Spoon)</td>
<td>0</td>
<td>2006-03-30 11:33:09.0</td>
<td>2006-03-30 11:55:52.0</td>
<td>2243</td>
</tr>
<tr>
<td>scheduler.fs (Spoon)</td>
<td>(Spoon)</td>
<td>0</td>
<td>2006-03-31 17:30:31.0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>scheduler.fs</td>
<td>scheduler_managed_custom_report</td>
<td>0</td>
<td>2006-03-31 19:26:42.0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>scheduler.fs</td>
<td>scheduler_managed_starter</td>
<td>0</td>
<td>2006-03-31 19:28:15.0</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
7.4 Managed Executable Jobs

This job starts executable files on the same computer as the Job Scheduler. To do this, the job executes the instruction in the command job parameters as a shell call in the COMMAND line. This means that programs which do not use the Job Scheduler API can be easily run as jobs, similar to cron jobs.

The output from an executable job which the job sends to stdout is included in the job protocol. Information about the course of a job in the form of the job log file can be obtained by e-mail. This can be obtained by activating the mail_on_success parameter in the Job Scheduler .ini file.

Should the executable file return an error message, then this is seen as failure of the job. This also applies to Posix Signals (SIGKILL, etc.). In such cases an error mail will be sent (in so far as mail_on_error is active) and the job ended. There are, however, three optional order parameters which can control the Managed Executable Jobs error handling more precisely:

- **ignore_error**: if true, then errors from executables are ignored
- **ignore_signal**: return signals from executables are ignored if true
- **timeout**: the maximum run time allowed for an executable file. The job will be terminated after this time has elapsed.

7.5 Should a Job be Run as an Order or as a Standard Job?

It is possible to run Managed Database Jobs and Managed Executable Jobs as either order-independent or as order controlled jobs. With order controlled jobs, a job chain is automatically created for each order controlled job. This job chain only contains a Managed Database Job or a Managed Executable Job. With an order controlled job, the executed command, the run time and the database connection (if present) are defined by the order. For use as a standard job, a new Managed Database job or Managed Executable job must be created with a new name and fixed instructions and run times.

What differences are there between the two methods? Both methods offer similar possibilities: database statements can be executed at predefined times with the option of notification about success or failure. In many cases, however, the predefined job chain offers advantages for order controlled jobs:

- An order is easier to set up than a job.
- The Job Scheduler can better manage the load should a number of orders be waiting.
- Only one job is required for a number of orders. This makes monitoring using the web interface easier, as only one job is shown. The progress of the individual orders by the job can still be followed.

However, there are situations where it can be advantageous to specify new Managed Database (or Executable) jobs. For example, should it be necessary to carry out database and executable jobs (or other custom jobs) in a particular order. In this case a job chain with jobs in the required order is made and the appropriate instructions are then allocated to these jobs. It is then necessary to create an order to use the job chain. This order is not used to convey instructions or parameters, but as a token to link these jobs. See the examples in the Appendices to this document for further information.
Appendix A: Example Showing the Creation of a Job Chain Using Jobs Realized in PHP

Steps 1 and 2 were carried out during the Job Scheduler setup. Step 3 was partly carried out during the setup. Only the path to the PHP interpreter needs to be specified here.

**Step 1: Create Ranges in the Settings**
The range is for the job type is given the name `job_type/mixed/interpreter_php`. "Executable PHP File Job Defaults" is selected as the title.
The range is for the order type is given the name `order_type/mixed/interpreter_php`. "Executable PHP File Orders" is selected as the title.

**Step 2: Create a Schema for the Range**
The range schema is completed as shown in Table 11 (page 53).

**Step 3: Create a Job Type**
The `interpreter_php` job type is created in the Job Type Management dialog. Here the job type is set to `interpreter_php` and the title to, for example, "executable php file". The sections which also have to be created are first shown after accept. The `interpreter` parameter must be completed. The complete (absolute) path to the local PHP command line interface should be entered here. If necessary, additional parameters required for the interpreter (e.g. "/usr/src/php-4.3.3/sapi/cli/php -c config" or "c:\php\cli\php.exe -c config") can be also added. In this example `php.ini` is required in the config directory for the Job Scheduler. "set script" should then be clicked and the standard Java class allocated to the job type as shown in Table 8 (page 46). This ensures that parameters added in the settings can be processed by a job of this type. The accept button should be clicked to store the job type.

**Step 4: Prepare the PHP Script**
The PHP script, which is to be executed in this example, has the following content:

```php
<?php
$pathinfo = pathinfo($argv[0]);

$default_arguments = array( 'bad' => 0 );
$arguments = array();
if( $argc > 1 ) { parse_str( $argv[1], $arguments ); } $arguments = array_merge( $default_arguments, $arguments );
if( $arguments['bad'] ) { fwrite(STDERR, 'Error at '.$pathinfo['basename']."\n"); exit(200); }
else {
    fwrite(STDOUT, $pathinfo['basename']." starts with parameters:
".
    foreach( $arguments as $key => $val ) { fwrite(STDOUT, $key.'='.$val."\n"); }
    exit(0);
}
?>
```

A file containing this script and named `php_sample.php` should be copied into the `/jobs` directory in a Job Scheduler installation, supported by the Managed Jobs interface. Assume that the ID of this Job Scheduler is `my_scheduler`. This file will be called later with the parameters "bad=0" and "bad=1". In the first - successful - case, the script will send output to StdOut and return 0 as exit code. In the second - unsuccessful - case, the script will send output to StdErr and return 200 as exit code.

**Step 5: Create the Project**
A new folder should be created in the user's "Home" directory. This is done by opening the Object Explorer (page 18) interface using Workflow management "Job Chains/ Jobs/ Orders" menu item and then navigating to /my_scheduler/live in the Object Explorer interface. The context menu is then opened and a new folder created using "New -> Folder" (page 22). This folder is then given a name such as "PHP Test". The purpose of this folder is simply to help keep this and other jobs with very different purposes well sorted. The new folder should now be opened and the following added to it:

- a new order job (context menu "New-> Order Job") under the name of "PHP Sample Job".
- a new job chain(context menu "New-> Job Chain") under the name of "Launch PHP Sample".
- two new orders (context menu "New-> Order") named "PHP Successful Order" and "PHP Error Order" respectively.

**Step 6: Configure and Submit the Job**

The order job, which is still empty, can be edited by opening it with the Editor (page 28). This is done by clicking on the order job object in the Explorer (page 18) or by selecting "Edit" from the context menu. The Executable PHP File job type, which was created in step 3 of this example, can now be selected as Job Type and this change saved by clicking on the button to the right of the "Job Type" selection box. The job now contains the script and parameters from the job type which were set in steps 1 - 3 above. The parameters can be checked and / or modified by clicking on the relevant "param" text in the Editor. In particular, it is recommended that the "interpreter" field is checked to ensure that it has been properly filled.

Where more than one Job Scheduler is in operation, the Scheduler ID parameter should be set to the ID of the Job Scheduler in whose jobs directory the PHP file from step 4 above was saved. Note that the button alongside the Scheduler ID field shows a list of the possible IDs from which the appropriate one can be selected. The dialog step is completed by clicking the ... and activate button.

**Step 7: Configure and Submit the Job Chain**

The Explorer (page 28), which is used to edit the still empty job chain, is opened either by clicking on the job chain object in the Explorer (page 18) dialog or selecting "Edit" from the Explorer context menu. First, the Scheduler ID for the job chain should be set as described in Step 6 above. Next, in the Editor dialog, opening the "New Child -> job_chain_node" or "New Child -> job_chain_node.end" context menu items for the job chain object shown in the lower left part of the Editor allows a node to be inserted into the job chain.

For this example a total of five nodes should be added to this job chain and for each node the "State", "Job", etc. attributes in the grey part of the Editor should be set as listed in the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Job</th>
<th>State</th>
<th>Next State</th>
<th>Error State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Node</td>
<td>job_chain_node</td>
<td>PHP Sample Job</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2nd Node</td>
<td>job_chain_node</td>
<td>PHP Sample Job</td>
<td>1</td>
<td>success</td>
</tr>
<tr>
<td>3rd Node</td>
<td>job_chain_node</td>
<td>PHP Sample Job</td>
<td>2</td>
<td>success</td>
</tr>
<tr>
<td>4th Node</td>
<td>job_chain_node.end</td>
<td></td>
<td>success</td>
<td></td>
</tr>
<tr>
<td>5th Node</td>
<td>job_chain_node.end</td>
<td></td>
<td>error</td>
<td></td>
</tr>
</tbody>
</table>

In the job chain created here, the job at the second node is started after the job at the first node has been completed, as long as the job at the first node runs without errors. Should, however, the job at the first node however return an error, then the third job will be started, after the first. The 4th and 5th nodes represent end statuses which the jobs at the 2nd and 3rd nodes reach in the event of success or error of the first job. As no jobs are specified for the 4th and 5th nodes, the processing within this job chain then ends when one of these nodes is reached. The dialog is closed by clicking on the ... and activate button.

The newly created job chain can now be found in the main Job Scheduler interface when the Job chains tab is selected. At the moment this job chain has no orders ....

**Step 8: Configure Orders**
Two orders will be created in this example: The first order should end the job created in the previous steps without errors. The second order should end the same job with an error, in order to demonstrate the different progressions within the job chain.

The Editor is opened with each order by either clicking on the order in the Explorer dialog or using the context menu "Edit" item. For each order, the procedure with which the order type is set and which was described in step 6 above should be followed. The same parameters as described for the job in step 6 will be offered. Note that parameters set in an order overwrite parameters with the same name in a job.

The `command` parameter form should now be opened. This form is used to set program call - i.e. to specify the arguments with which the PHP interpreter is started. For the "Successful" order the value `-f jobs/php_sample.php "bad=0"` should be inserted here and for the "Error" order the value `-f jobs/php_sample.php "bad=1"`.

When orders are to be persistent - i.e. they are to be saved so that they can be rerun - then they should be given a `run_time` such as Single Start = 12:00. The Editor should now be closed by clicking the Store button.

**Step 9: Test the Orders**

Submit the orders to the Job Scheduler using the "Functions -> Test" context menu item. The "Test" function does not make an order persistent when submitting it but starts the order immediately. Note that before an order can be submitted to the Job Scheduler, all the job chains listed in the order must have been submitted.

After the orders have completed their test runs, it is possible to examine the log files for each order test in the Job Scheduler web interface. This is done by selecting the "Launch PHP Sample" job chain from the Job chains tab and activating "Show order history" in the right hand area. The change in statuses in the orders can be seen in the log files as well as the output written to StdOut and StdErr. When `mail_on_warning` is set in the `config/factory.ini` file, then users will be sent an error message per e-mail using the "Error" order.

**Step 10: Activating the Orders**

When no PHP-errors occur when testing, then orders can be submitted using "Functions->Activate". Should a start interval have been specified (`run_time`), then the orders will be made persistent in the job chain.
Appendix B: Examples Showing the Use of Managed Jobs

The following examples show how Managed Jobs can be used:

B.1 A Simple Database Order

This example shows how a new order can be created for a predefined job chain. It is assumed that a table with data about a group of people exists in a database and that a connection for this database has been configured (see Database connections (page 48)). A column in the table contains the person’s age in years. The people are to age one year every day.

The Job Scheduler supported by the Managed Job interface has the ID my_scheduler. Use the context menu to create a /my_scheduler/live sub-folder and select "New->folder" (page 22). Enter a name, such as "Launch Database Statements". Copy (page 23) the launch_database_statement, launch_database_statement, order and scheduler_managed_database_statement objects from /Samples/Launch Database Reports to /my_scheduler/live/Launch Database Statements. Configure the order by clicking on the order to open the Editor, or by selecting "Edit" in the order context menu. "Database Statement" should be selected as order type. The database connection described above should be selected and the following SQL statement should be entered in the "command" parameter:

```
UPDATE persondata SET age=age+1;
```

The order should be carried out once a day. The following run time value causes the statement to be executed at 03:00:

```
<run_time let_run = "yes">
<period single_start = "3:00"/>
</run_time>
```

The order can now be saved and submitted to the Job Scheduler.

B.2 An Individual Job Chain with Standard Jobs

In this example a MySQL database containing weather information should be updated on a daily basis with new data from a csv file. The current csv file is not directly available to the database server and first of all needs to be copied over. The WEATHER_DATA table is created using the following SQL statement:

```
CREATE TABLE "WEATHER_DATA" ("ID" INT(10) UNSIGNED NOT NULL AUTO_INCREMENT, "CITY" VARCHAR(45) NOT NULL, "DAY" TIMESTAMP NOT NULL, "TEMPERATURE" SMALLINT NOT NULL, PRIMARY KEY("ID") ) TYPE = InnoDB;
```

The csv file with CITY und TEMPERATURE could look like this (Careful! Tabs are used here to separate fields instead of commas.):
Two jobs are necessary, which should be executed one after the other. The first job copies the csv file and the second imports it into the database. The use of a **Managed Executable Job** followed by a **Managed Database Job** would be appropriate here.

First of all a new job chain needs to be created using the workflow management. Both the job chain and the jobs which will be later added to it should be allocated to a **Job Scheduler** which has access to the csv file directory by creating the new job chain below the **Job Scheduler's** "live" directory. Two jobs are then added to this job chain. The first job is allocated the "Executable File" job type. A copy command (under the "command" parameter) is given as program call. For Unix, for example, this could look like:

```
cp /home/test/scheduler.sos/weather/weather.csv /home/mysql/weather.csv
```

An empty run_time element is entered as start interval (<run_time/>), as the actual start time should be set later by orders. The entry status must be 0. Exit and error statuses can, for example, be set to 100 and 1100.

The second job is given the "Database Statement" job type and the corresponding database connection specified. The following SQL statement (with suitable path descriptions) is then entered under the "command" parameter:

```
LOAD DATA INFILE '/home/mysql/weather.csv' INTO TABLE WEATHER_DATA (CITY, TEMPERATURE);
```

In order that the job can work, the MySQL user specified must have the appropriate FILE authorization. The entry status for this job is set to the (successful) exit status of the first job (that is 100), thereby linking the jobs to a chain. Exit and error statuses can, for example, be set to 200 and 1200.

The "other jobs" type order can now be defined for the new job chain by selecting "Standard Job" from the "order type" select box. This order requires neither parameters nor database connection, as these have already been defined in the job. Should this order be started, for example, by way of a specified start time, then it has an initial status of 0. Should everything run correctly, then the executable file job will be carried out and the order given the status 100. This, in turn, causes the second database job to be carried out, after which the order status is set to 200.

### B.3 An Individual Job Chain with Individual Jobs

This example builds on the previous "Individual Job Chain with Standard Jobs" example, with the weather data no longer being taken from a local file, but downloaded from the internet. In this example, the weather data will be taken from http://weather.unisys.com/xml/observation.cgi where weather data from different cities is available in XML format. The second job from the example above - which writes the data in the database - remains unchanged. However, additional jobs will be required for the following operations: downloading the weather data from the internet; reading the data; converting the temperatures from Fahrenheit and writing the date in a csv file.

The first job is to download the data. Here the **JobSchedulerManagedJob** class is used to write the new job in Java. The new **DownloadWeatherJob** class is derived from this class. In the example below, all parameters are hard coded. If required, however, the job could be parameterized.

```
DownloadWeatherJob.java
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.URL;
import sos.scheduler.managed.JobSchedulerManagedJob;

public class DownloadWeatherJob extends JobSchedulerManagedJob {

    private String weatherURL="http://weather.unisys.com/xml/observation.cgi";
    private String fileName="/home/test/scheduler.sos/weather/weather.xml";

    public boolean spooler_process() throws Exception {
        getLogger().info("Downloading Weatherdata from: "+weatherURL);

        File outFile = new File(fileName);
        FileOutputStream fos = new FileOutputStream(outFile);

        URL url = new URL(weatherURL);
        BufferedReader in = new BufferedReader(
                new InputStreamReader(url.openStream()));

        int input;

        while ((input = in.read()) != -1)
            fos.write(input);

        fos.flush();
        fos.close();
        in.close();
        return true;
    }
}

Only the spooler_process() method is overwritten here. This is called at every process step in the job. The getLogger() function returns a SOSLogger, which has already been initialized by one of the parent classes. For details of the development of this job, see the Job Scheduler tutorial.

The next job reads the downloaded XML file and writes a file in the same format as in the previous example. The downloaded file contains weather from many weather stations and in the example only data for Berlin, New York, Paris und Miami is required. The data from a weather station is represented in the XML format by an <observation> element. This element has an ID attribute, which identifies the station; for example, Berlin has the ID "EDDB". The temperature is contained in the temp.f attribute. The following code carries out the necessary task:

WeatherXML2CSVJob.java

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.HashMap;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

WeatherXML2CSVJob.java

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.HashMap;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import sos.scheduler.managed.JobSchedulerManagedJob;

public class WeatherXML2CVSJob extends JobSchedulerManagedJob {

    private String xmlFileName = "/home/test/scheduler.sos/weather/weather.xml";
    private String csvFileName = "/home/test/scheduler.sos/weather.csv";
    private HashMap cities;

    public boolean spooler_init() {
        boolean rc = super.spooler_init();
        cities = new HashMap();
        cities.put("EDDB","Berlin");
        cities.put("KJFK","New York");
        cities.put("LFPO","Paris");
        cities.put("K Mia","Miami");
        return rc;
    }

    public boolean spooler_process() throws Exception {
        try{
            getLogger().info("Reading xml Document");
            DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
            Document weatherDocument = docBuilder.parse(new FileInputStream(xmlFileName));
            NodeList observations = weatherDocument.getElementsByTagName("observation");
            getLogger().debug3("found "+observations.getLength()+" observations.");
            for(int i=0; i<observations.getLength(); i++){
                if (observation!=null){
                    String id = observation.getAttribute("id");
                    String tempF = observation.getAttribute("temp.F");
                    if (cities.keySet().contains(id)){
                        int iTempF = Integer.parseInt(tempF);
                        int fTempC = Math.round((iTempF-32)*(5f/9));
                        String csvLine = cities.get(id)+"\t"+fTempC+"\n";
                        getLogger().debug3("Found city "+cities.get(id)+" with temperature "+fTempC);
                        fos.write(csvLine.getBytes());
                    }
                }
            }
        } catch (Exception e){
            getLogger().error("Failed to read xml Document");
            return false;
        }
    }
}

To use this job, it is necessary that it is copied in a directory which can be reached by the Job Scheduler, such as scheduler/lib. This means that the directory or JAR file(s) containing the compiled classes has to be included in the class_path entry in the Job Scheduler factory.ini file and that the Job Scheduler must be restarted.

The new jobs now require a job chain. This can be easily made by copying the chain from the previous example under a new name. The first job of this chain should then be given a different name, and the job type changed to "standard job". The script element should also be changed in that "DownloadWeatherJob" is entered in the java_class attribute. The start, exit, and error statuses for the second job also need to be changed - for example to 200, 300 and 1300 respectively, as this job will now become the third job in the chain.

The XML conversion has now to be added as a new job. This can be done by making a copy of the first job with a new name and entering "WeatherXML2CVSJob" as the Java class. The start, exit, and error statuses for this job need to be selected so that it passes between the other two jobs - e.g. 100, 200 and 1200. As with the previous example, an empty order leads to the job chain being carried out.

Note that it can occur that when this job is run, not all the data for the four cities is in the table. This is because the XML source file does not always have complete data for every weather station.
Appendix C: Examples of run_time Elements

Examples are given in this Appendix of how <run_time> elements are used to determine the run times of jobs or orders. Note that for order controlled jobs, the run_time is not the time at which a job should start but rather the time period in which the job may run.

The order itself determines the start time of an order controlled job. Should the start time specified in an order lie outside the time period in which the job is allowed to be executed, then the order must wait until the job is allowed to run.

Repeat an order at fixed intervals (180s):
<pre class="xml">
<run_time let_run = "yes">
  <period repeat = "180"/>
</run_time>
</pre>

Start an order at fixed times Mondays to Wednesdays:
<pre class="xml">
<run_time let_run = "yes">
  <weekdays>
    <day day = "1 2 3">
      <period single_start = "10:00"/>
      <period single_start = "14:00"/>
    </day>
  </weekdays>
</run_time>
</pre>

Repeat an order at fixed intervals Mondays to Wednesdays:
<pre class="xml">
<run_time let_run = "yes">
  <weekdays>
    <day day = "1 2 3">
      <period repeat = "180"/>
    </day>
  </weekdays>
</run_time>
</pre>

Start an order three days before the last day of the month:
<pre class="xml">
<run_time let_run = "yes">
  <ultimos>
    <day day="3">
      <period single_start = "23:30"/>
    </day>
  </ultimos>
</run_time>
</pre>
Start an order on the 1st., 15th and 25th of the month except on holidays (Christmas):

```xml
<run_time let_run = "yes">
  <monthdays>
    <day day = "1 15 25"><period single_start = "10:00"/></day>
  </monthdays>
  <holidays>
    <date date = "2006-12-25"/>
  </holidays>
</run_time>
```

Only allow a job to run every day between 9:00 and 18:00:

```xml
<run_time let_run = "yes" begin="09:00" end="18:00"/>
```
Glossary

Job

Programs and scripts which are to be executed by the Job Scheduler must be embedded in jobs. Jobs can either start executable files or contain job scripts which use the Job Scheduler programming interface. Jobs can be run in more than one process or task, when more throughput is required.

Job Chain

A consecutive series of jobs which process orders. The Job Scheduler starts the jobs in a job chain automatically after an order is received. Depending on the load, the jobs in a job chain can be run in parallel in order to complete a larger number of orders.

Locks

Locks in the Job Scheduler stop jobs from being executed, so long as a particular lock is occupied by one or more other Job(s). Locks are can be used in order jobs in job chains and in order independent jobs. Should a job be waiting for a lock to be released, then it will be automatically started as soon as the lock is opened.

Managed Jobs

Jobs which are administered using a database and which are automatically allocated to one or more Job Schedulers. A database is a prerequisite for the use of Managed Jobs.

Order

An order activates the processing of a job chain and contains parameters for one or more jobs in the job chain. A Persistent Order contains a start time or a start interval for repeated starts. Should an order not be persistent, then it is deleted after it has been processed in the job chain.

An order proceeds along the jobs in a job chain one after the other. Should a processing error occur, then the order is stopped and removed from the job chain. Every job in a job chain has access to the parameters of an order.

Process Classes

Process classes are used to allow job tasks to run in their own processes. Otherwise job tasks run in the Job Scheduler process.