

independIT Integrative Technologies GmbH  
Bergstraße 6  
D-86529 Schrobenhausen



**schedulix!Web**

**User Guide  
Release 2.10**

Dieter Stubler

Ronald Jeninga

September 5, 2023

Copyright © 2023 independIT GmbH

**Legal notice**

This work is copyright protected

Copyright © 2023 independIT Integrative Technologies GmbH

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronically or mechanically, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner.

# Contents

<b>Contents</b>	<b>3</b>
<b>List of tables</b>	<b>3</b>
<b>List of illustrations</b>	<b>5</b>
<b>1 General</b>	<b>1</b>
1.1 Overview . . . . .	1
1.2 Login . . . . .	2
1.3 Main Desktop . . . . .	3
1.4 Layout of the object window . . . . .	5
1.4.1 Title bar . . . . .	5
1.4.2 Navigator . . . . .	7
1.4.2.1 Standard buttons . . . . .	7
1.4.3 Editor . . . . .	9
1.4.3.1 Standard buttons . . . . .	10
1.5 Selecting values . . . . .	16
1.5.1 Selecting values from drop-down lists . . . . .	16
1.5.2 Choosing values using the Select button . . . . .	16
1.6 Handling standard lists . . . . .	18
1.6.1 Adding a line . . . . .	18
1.6.2 Deleting a line . . . . .	19
1.6.3 Changing line values . . . . .	20
1.7 Copy & paste and the clipboard . . . . .	21
1.7.1 Moving objects in the hierarchy . . . . .	21
1.7.2 Linking objects . . . . .	25
1.8 Graphic Legend . . . . .	27
1.8.1 How schedulix objects are represented . . . . .	27
1.8.2 Representation of parent-child relationships in workflows . . . . .	27
1.8.3 Representation of dependencies between Scheduling Entities . . . . .	28
1.9 Comments . . . . .	29
1.10 Standard fields . . . . .	29
<b>2 Exit State Definitions</b>	<b>31</b>
2.1 View . . . . .	31
2.2 Concept . . . . .	31
2.2.1 In short . . . . .	31

2.2.2	Detailed description . . . . .	31
2.3	Editor . . . . .	31
<b>3</b>	<b>Exit State Mappings</b>	<b>33</b>
3.1	View . . . . .	33
3.2	Concept . . . . .	33
3.2.1	In short . . . . .	33
3.2.2	Detailed description . . . . .	33
3.3	Editor . . . . .	34
<b>4</b>	<b>Exit State Profiles</b>	<b>37</b>
4.1	View . . . . .	37
4.2	Concept . . . . .	37
4.2.1	In short . . . . .	37
4.2.2	Detailed description . . . . .	37
4.3	Editor . . . . .	38
<b>5</b>	<b>Exit State Translations</b>	<b>41</b>
5.1	View . . . . .	41
5.2	Concept . . . . .	41
5.2.1	In short . . . . .	41
5.2.2	Detailed description . . . . .	41
5.3	Editor . . . . .	42
<b>6</b>	<b>Resource State Definitions</b>	<b>43</b>
6.1	View . . . . .	43
6.2	Concept . . . . .	43
6.2.1	In short . . . . .	43
6.2.2	Detailed description . . . . .	43
6.3	Editor . . . . .	43
<b>7</b>	<b>Resource State Profiles</b>	<b>45</b>
7.1	View . . . . .	45
7.2	Concept . . . . .	45
7.2.1	In short . . . . .	45
7.2.2	Detailed description . . . . .	45
7.3	Editor . . . . .	46
7.3.1	Example . . . . .	46
<b>8</b>	<b>Resource State Mappings</b>	<b>49</b>
8.1	View . . . . .	49
8.2	Concept . . . . .	49
8.2.1	In short . . . . .	49
8.2.2	Detailed description . . . . .	49

8.3	Editor	50
<b>9</b>	<b>Named Resources</b>	<b>53</b>
9.1	View	53
9.2	Concept	53
9.2.1	In short	53
9.2.2	Detailed description	53
9.3	Editor	54
9.3.1	Properties tab for categories	54
9.3.2	Content tab	55
9.3.3	Properties tab for Named Resource Definitions	55
9.3.4	Parameters tab	57
9.3.4.1	Parameter Details tab	57
9.3.4.2	Standard parameters	58
9.3.5	Resources tab	58
9.3.6	Job Definitions tab	60
9.4	Named Resource selector	61
<b>10</b>	<b>Environments</b>	<b>63</b>
10.1	View	63
10.2	Concept	63
10.2.1	In short	63
10.2.2	Detailed description	63
10.3	Navigator	64
10.4	Editor	65
10.4.1	Properties tab	65
10.4.2	References tab	66
<b>11</b>	<b>Footprints</b>	<b>67</b>
11.1	View	67
11.2	Concept	67
11.2.1	In short	67
11.2.2	Detailed description	67
11.3	Editor	68
11.3.1	Properties tab	68
11.3.2	References tab	69
<b>12</b>	<b>Jobscribers and resources</b>	<b>71</b>
12.1	View	71
12.2	Concept	71
12.2.1	In short	71
12.2.2	Detailed description	71
12.3	Navigator	73

12.4	Editor	73
12.4.1	Properties tab	73
12.4.2	Resources tab	76
12.4.2.1	Resource Details tab	77
12.4.2.2	Parameters tab	79
12.4.2.3	Allocations tab	80
12.4.3	Parameters tab	84
12.4.4	Config tab	86
12.4.4.1	Standard configuration parameters	87
12.4.5	Env.Map tab	89
12.4.6	Logfile Pattern tab	90
12.5	Resource Links	91
12.5.1	Resource Details tab	92
<b>13</b>	<b>Batches and Jobs</b>	<b>93</b>
13.1	View	93
13.2	Concept	93
13.2.1	In short	93
13.2.2	Detailed description	94
13.2.3	Recommended convention for batch objects	95
13.3	Navigator	96
13.3.1	Pinning	96
13.3.2	Folder bookmarks	97
13.3.3	Folder search	98
13.4	Folder editor	98
13.4.1	Properties tab	98
13.4.2	Content tab	100
13.4.3	Parameters tab	101
13.4.4	Resources tab	101
13.5	Editor for Job Definitions	103
13.5.1	Properties tab	103
13.5.2	Run tab	108
13.5.3	Restart tab	112
13.5.4	Children tab	113
13.5.4.1	Child Details tab	114
13.5.5	Parents tab	120
13.5.6	Dependencies tab	121
13.5.6.1	Dependency Details tab	122
13.5.7	Dependents tab	126
13.5.8	Required Resources tab	127
13.5.8.1	Resource Details tab	130
13.5.9	Defined Resources tab	133

13.5.10 Parameters tab . . . . .	134
13.5.10.1 Parameter Details tab . . . . .	135
13.5.10.2 Predefined default parameters for the runtime system	139
13.5.11 References tab . . . . .	142
13.5.12 Triggers tab . . . . .	143
13.5.12.1 Trigger Details tab . . . . .	144
13.5.13 Triggered by tab . . . . .	149
13.6 Job Hierarchy navigation . . . . .	150
13.7 Batch in Folder Convention . . . . .	152
<b>14 schedulix!Web Users</b>	<b>155</b>
14.1 View . . . . .	155
14.2 Concept . . . . .	155
14.2.1 In short . . . . .	155
14.2.2 Detailed description . . . . .	155
14.3 Navigator . . . . .	156
14.4 Editor . . . . .	156
14.4.1 schedulix!Web Connection . . . . .	156
14.4.2 schedulix Server Connections . . . . .	158
<b>15 schedulix Server Users</b>	<b>159</b>
15.1 View . . . . .	159
15.2 Concept . . . . .	159
15.2.1 In short . . . . .	159
15.2.2 Detailed description . . . . .	159
15.3 Navigator . . . . .	160
15.4 Editor . . . . .	160
<b>16 Groups</b>	<b>163</b>
16.1 View . . . . .	163
16.2 Concept . . . . .	163
16.2.1 Description . . . . .	163
16.3 Navigator . . . . .	163
16.4 Editor . . . . .	164
16.4.1 Properties tab . . . . .	164
16.4.2 Grants . . . . .	165
<b>17 Time Scheduling</b>	<b>167</b>
17.1 View . . . . .	167
17.2 Concept . . . . .	168
17.2.1 In short . . . . .	168
17.2.2 Detailed description . . . . .	168
17.3 Navigator . . . . .	169

17.4 Editor . . . . .	170
17.4.1 Sub Schedule Details sub-area . . . . .	173
<b>18 Intervals</b>	<b>181</b>
18.1 Image . . . . .	181
18.2 Concept . . . . .	182
18.2.1 Brief description . . . . .	182
18.2.2 Detailed description . . . . .	182
18.3 Navigator . . . . .	182
18.4 Editor . . . . .	182
18.4.1 Sub-area Interval Details . . . . .	183
<b>19 Submit Batches and Jobs</b>	<b>187</b>
19.1 View . . . . .	187
19.2 Concept . . . . .	187
19.2.1 In short . . . . .	187
19.2.2 Detailed description . . . . .	187
19.3 Navigator . . . . .	188
19.4 Editor . . . . .	188
<b>20 Bookmarks</b>	<b>191</b>
20.1 View . . . . .	191
20.2 Concept . . . . .	191
20.2.1 In short . . . . .	191
20.2.2 Detailed description . . . . .	191
20.3 Navigation . . . . .	193
<b>21 Running Master Jobs</b>	<b>195</b>
21.1 View . . . . .	195
21.2 Concept . . . . .	195
21.2.1 In short . . . . .	195
21.2.2 Detailed description . . . . .	195
21.3 Master Navigator . . . . .	196
21.3.1 Master Navigator Query mask . . . . .	203
21.4 Detail Navigation . . . . .	208
21.4.1 Tree view . . . . .	208
21.4.2 Search results . . . . .	208
21.4.3 Detail navigation query mask . . . . .	209
21.5 Details mask for jobs . . . . .	210
21.5.1 Buttons . . . . .	210
21.5.2 Properties tab . . . . .	215
21.5.3 Run tab . . . . .	218
21.5.4 Timestamps & Statistics tab . . . . .	221
21.5.5 Dependencies tab . . . . .	223



21.5.6	Dependents tab . . . . .	227
21.5.7	Resource(Req) tab . . . . .	230
21.5.8	Resource(Def) tab . . . . .	232
21.5.9	Parameters tab . . . . .	233
<b>22</b>	<b>Search Running Jobs</b>	<b>235</b>
22.1	View . . . . .	235
22.2	Concept . . . . .	235
22.2.1	In short . . . . .	235
22.2.2	Detailed description . . . . .	235
22.3	Navigator . . . . .	236
22.4	Navigator Details query mask . . . . .	236
<b>23</b>	<b>Calendar</b>	<b>239</b>
23.1	View . . . . .	239
23.2	Concept . . . . .	239
23.2.1	In short . . . . .	239
23.3	Detailed description . . . . .	239
23.3.1	Query tab . . . . .	240
23.3.2	List tab . . . . .	241
23.3.3	Day tab . . . . .	241
23.3.4	Week tab . . . . .	242
23.3.5	Month tab . . . . .	242
<b>24</b>	<b>System Information</b>	<b>243</b>
24.1	View . . . . .	243
24.2	Concept . . . . .	244
24.2.1	In short . . . . .	244
24.2.2	Config tab . . . . .	244
24.2.3	System tab . . . . .	253
24.2.4	Worker tab . . . . .	255
24.2.5	Sessions tab . . . . .	256
24.2.6	SME/Q tab . . . . .	258
<b>25</b>	<b>Shell</b>	<b>259</b>
25.1	Image . . . . .	259
25.2	Concept . . . . .	259
25.2.1	Brief description . . . . .	259
25.2.2	Detailed description . . . . .	260
25.2.3	Buttons . . . . .	260
<b>Index</b>		<b>266</b>



# List of Tables

1.1	Description of the icons	4
12.1	Lock mode matrix	82
12.2	Description of the jobserver configuration parameters	88
24.1	ParameterHandling options	247
24.2	Overview of the Trace Levels	251
24.3	Session Status	257



# List of Figures

1.1	Login window	1
1.2	schedulixLauncher	2
1.3	Main Desktop	3
1.4	Example of a standard dialog	5
1.5	Busy message	6
1.6	About window	6
1.7	Navigator with a hierarchical view	7
1.8	Searching in the Navigator	8
1.9	Search results	9
1.10	Editor with tabs	10
1.11	Show Folder Path	12
1.12	Hide Folder Path	12
1.13	Show Hierarchy Path	13
1.14	Hide Hierarchy Path	13
1.15	Reverse Selection; Initial State	14
1.16	Reverse Selection; Result	14
1.17	Choosing a Resource State Profile; initial state	16
1.18	Choosing a Resource State Profile; selection list	17
1.19	Choosing a Resource State Profile; result	17
1.20	Example of how standard lists are handled	18
1.21	Handling lists; adding a line	18
1.22	Handling lists; choosing a value	19
1.23	Handling lists; result	19
1.24	Handling lists; deleting a line	20
1.25	Handling lists; editing	20
1.26	Moving objects; initial state	21
1.27	Moving objects; Content tab in the Editor	22
1.28	Moving objects; selection	22
1.29	Moving objects; cut	23
1.30	Moving objects; choosing the target folder	23
1.31	Moving objects; paste	24
1.32	Moving objects; result	24
1.33	Linking objects; initial situation	25
1.34	Linking objects; selection	25
1.35	Linking objects; paste	26
1.36	Legend for the graphical representation of schedulix objects	27

1.37	Legend for the graphical representation of parent-child relationships	27
1.38	Legend for the graphical representation of dependency relationships	28
1.39	Recording comments	29
2.1	Exit State Definitions	31
3.1	Exit State Mappings	33
3.2	Exit State Mapping Editor	34
4.1	Exit State Profiles	37
4.2	Exit State Profile Editor	38
5.1	Exit State Translations	41
5.2	Exit State Translation Editor	42
6.1	Resource State Definitions	43
7.1	Resource State Profiles	45
7.2	Resource State Profile Editor	46
7.3	State chart for a resource	47
8.1	Resource State Mappings	49
8.2	Example for a Resource State Mapping	50
8.3	Resource State Mapping Editor	50
9.1	Named Resources	53
9.2	Categories; Properties tab	54
9.3	Categories; Content tab	55
9.4	Named Resources; Properties tab	55
9.5	Named Resources; Parameters tab	57
9.6	Named Resources; Parameter Details	57
9.7	Named Resources; instances	58
9.8	Named Resources; Job Definitions tab	60
9.9	Named Resources; selector	61
10.1	Environments	63
10.2	Environment navigator	64
10.3	Environment properties	65
10.4	Environment references	66
11.1	Footprints	67
11.2	Footprint properties	68
11.3	Footprint references	69
12.1	Jobservers and resources	71
12.2	Jobservers and Scope Navigator	73

12.3	Scope properties	73
12.4	Jobserver properties	74
12.5	Jobserver and scope resources	76
12.6	Resource Details	77
12.7	Resource parameters	79
12.8	Resource allocations	80
12.9	Jobserver and scope parameters	84
12.10	Jobserver and scope configuration	86
12.11	Jobserver environment mapping	89
12.12	Jobserver log file name patterns	90
12.13	Copying and creating Resource Links	91
12.14	Information about a Resource Link	92
13.1	Batches and Jobs	93
13.2	Navigator for batches and jobs	96
13.3	Batches and Jobs; view without a pin	97
13.4	Batches and Jobs; view with a pin	97
13.5	Folder properties	98
13.6	Folder content	100
13.7	Convention Support on Paste	100
13.8	Folder parameters	101
13.9	Folder resources	101
13.10	Job Definition properties	103
13.11	Examples of batches	104
13.12	Example of a milestone	104
13.13	Run Job tab	108
13.14	Job Restart tab	112
13.15	Batches and Jobs; Children tab	113
13.16	Batches and Jobs; Child Details tab	114
13.17	Static and dynamic children	115
13.18	Example of Merge Mode	118
13.19	Example of Ignored Dependencies	120
13.20	Batches and Jobs; Parents tab	120
13.21	Batches and Jobs; Dependencies tab	121
13.22	Example of Dependency Modes	122
13.23	Batches and Jobs; Dependency Details	123
13.24	Example for Unresolved Handling	124
13.25	Batches and Jobs; Dependents tab	127
13.26	Jobs; Required Resources tab	127
13.27	Batches and Jobs; Resource Requirement details	130
13.28	Example of Sticky Handling	131
13.29	Example of load distribution with Sticky Handling	132
13.30	Batches and Jobs; Defined Resources	133

13.31	Batches and Jobs; Parameters tab . . . . .	134
13.32	Batches and Jobs; Parameter Details . . . . .	135
13.33	Example using Expression parameters . . . . .	136
13.34	Batches and Jobs; References tab . . . . .	142
13.35	Batches and Jobs; Triggers tab . . . . .	143
13.36	Batches and Jobs; Trigger Details tab . . . . .	144
13.37	Example of an Immediate Merge trigger . . . . .	145
13.38	Example of an Immediate Local trigger . . . . .	146
13.39	Example of a Before Final Trigger . . . . .	146
13.40	Example of an After Final Trigger . . . . .	147
13.41	Example of a Finish Child Trigger . . . . .	147
13.42	Jobs and batches; Triggered By tab . . . . .	149
13.43	Batches and Jobs; hierarchy view . . . . .	150
13.44	Hierarchy view with dependencies . . . . .	151
13.45	Hierarchy view with multiple dependencies . . . . .	151
13.46	Cyclic dependencies . . . . .	152
14.1	User management for the Web front end . . . . .	155
15.1	schedulix Server user management window . . . . .	159
16.1	Group management . . . . .	163
16.2	Group properties . . . . .	164
16.3	Object privileges for a group . . . . .	165
17.1	Time Scheduling . . . . .	167
17.2	Time Scheduling Navigator . . . . .	169
17.3	Time Scheduling Editor . . . . .	170
17.4	Repeat Driver . . . . .	174
17.5	Time Of Day Driver . . . . .	174
17.6	Range of Day Filter . . . . .	175
17.7	Day of Week Filter . . . . .	175
17.8	Day of Month Filter . . . . .	176
17.9	Week of Month Filter . . . . .	176
17.10	ISO Week of Month Filter . . . . .	177
17.11	Week of Year Filter . . . . .	177
17.12	Month of Year Filter . . . . .	178
17.13	Calendar Driver . . . . .	178
17.14	Calendar Filter . . . . .	179
18.1	Intervals . . . . .	181
19.1	Submit Batches and Jobs . . . . .	187
19.2	Submit navigation . . . . .	188
19.3	Submit editor . . . . .	188



20.1	Bookmarks	191
21.1	Running Master Jobs	195
21.2	Running Master Jobs Navigator	196
21.3	State chart for batches and jobs	202
21.4	Running Master Jobs Query mask	203
21.5	Running Master Jobs "Details" window	208
21.6	Navigation Details query mask	209
21.7	Details mask for jobs	210
21.8	Confirm mask	211
21.9	Job and batch properties	215
21.10	Batch und Job Run information	218
21.11	Batch and job timestamps	221
21.12	Batch und job dependencies	223
21.13	Batches and Jobs Ignore Dependencies confirmation mask	224
21.14	Example of a Recursive Ignore	225
21.15	Example of a Job Only Ignore	225
21.16	Batch and Job Dependents tab	228
21.17	Batch and Job Ignore Dependencies	228
21.18	Job Resource Requirements	231
21.19	Job Ignore Resource Requirement	231
21.20	Batch and Job Defined Resources	232
21.21	Batch and job parameters	233
22.1	Search Running Jobs	235
22.2	Search Running Jobs query mask	236
23.1	Calendar	239
23.2	Calendar Query tab	240
23.3	Calendar List tab	241
23.4	Calendar Day tab	241
23.5	Calendar Week tab	242
23.6	Calendar Month tab	242
24.1	System Information	243
24.2	System configuration	244
24.3	System runtime environment	253
24.4	Worker thread information	255
24.5	Session information	256
25.1	Shell	259



# 1 General

## 1.1 Overview

This chapter will introduce you to the schedulix!Web user interface. General actions and procedures that are not just to be found in a particular dialog are explained and illustrated here in more detail.

The schedulix!Web user interface is a browser-based dialog system. There is no need to install an additional client software.

To start the schedulix!Web interface you have to open a web browser window (e.g. Microsoft Internet Explorer). Please enter the URL you have been given to get to the schedulix!Web interface. We recommend saving this URL in your browser as a bookmark so you can start the schedulix!Web interface with one click.

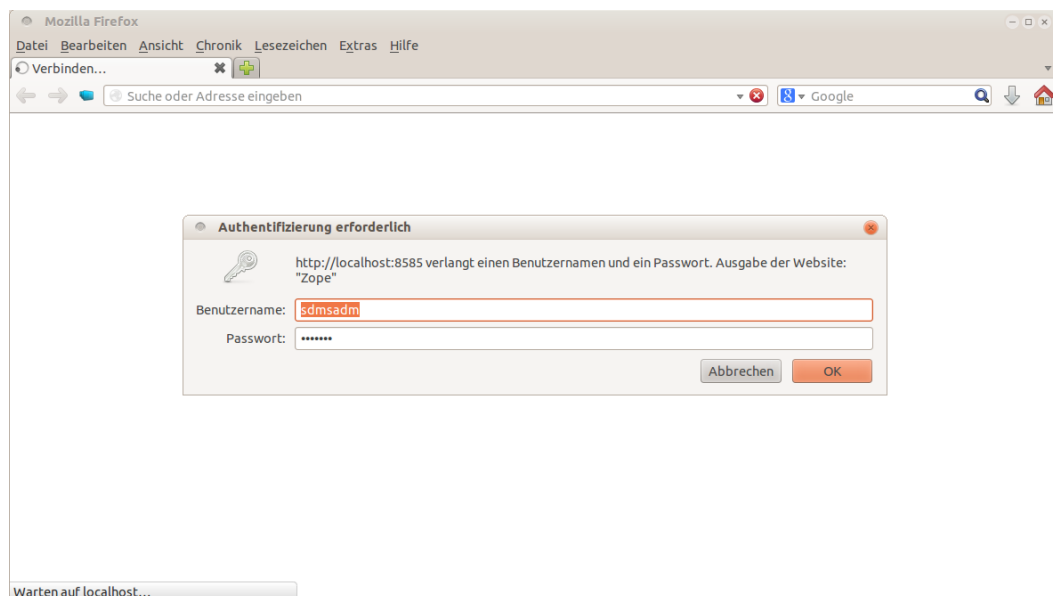


Figure 1.1: Login window

## 1.2 Login

After entering the URL, a login window opens for verifying your password. All users have to identify themselves with a user name and password to work with schedulix.

The input mask for the password prompt is shown in Figure 1.1, whereby the exact appearance may vary slightly depending on the browser.

Here you need to enter your personal user name and matching password and confirm them by clicking *OK*. Your login details can be obtained from your system administrator.

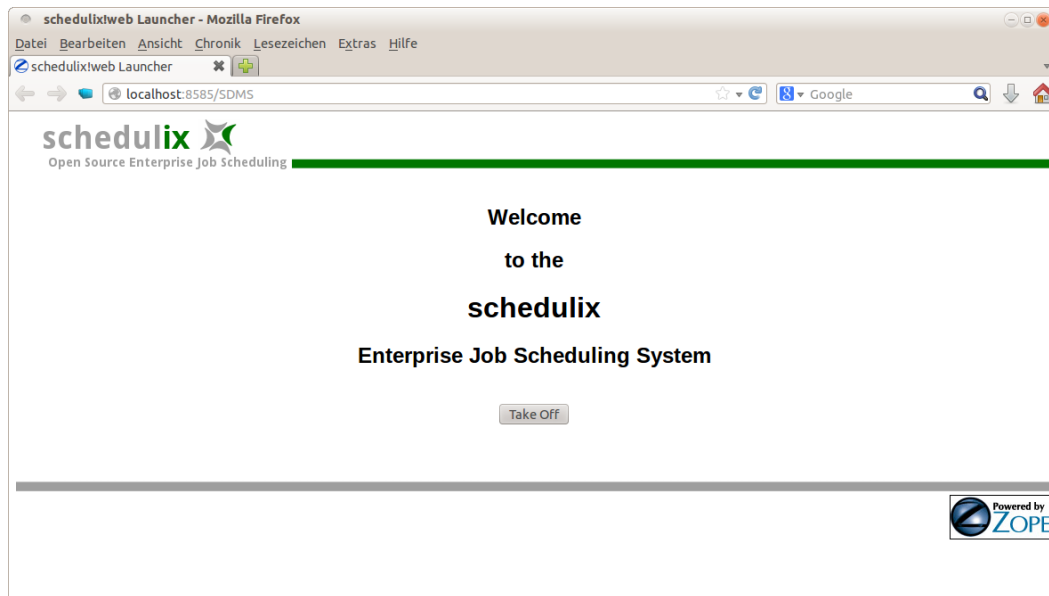


Figure 1.2: schedulixLauncher

After you have successfully logged in, the schedulixLauncher opens. Now open the main menu by clicking the *Take Off* button.

The launcher may be closed automatically depending on your browser. Your browser's security settings may prevent this or trigger a warning message.

After clicking the *Take Off* button appears the main menu called the *Main Desktop*.

### 1.3 Main Desktop

The *Main Desktop* is the main window of the schedulix!Web user interface. You can launch all the function dialogs in schedulix directly from this window. From the *Connection* options box in the header you can select which server connection you want to use for the next function dialog that is to be opened.

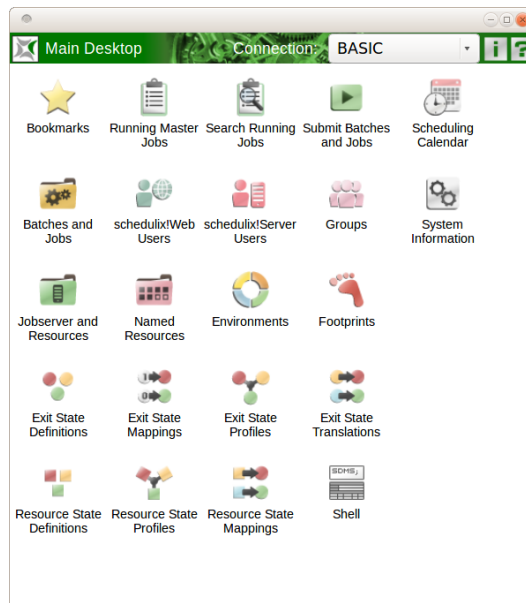







Figure 1.3: Main Desktop

The *Main Desktop* is shown in Figure 1.3.

The dialog boxes for maintaining or monitoring different objects can be opened by clicking the icons shown below:

Icon	Description
	The <i>Bookmarks</i> dialog is used to create, edit and delete views of currently running jobs.
	The <i>Running Master Jobs</i> dialog gives a clear view of the current Master Jobs.
	The <i>Search Running Jobs</i> dialog is used to search for running jobs according to specific criteria.
	The <i>Submit Jobs</i> dialog is used for starting workflows.
	The <i>Batches and Jobs</i> dialog is used to create and manage folders, batches, jobs and milestones.

*Continues on next page*

## Main Desktop

*Continued from previous page*








Icon	Description
	The <i>schedulix!Web Users</i> dialog is used for the administration of schedulix!Web users.
	The <i>schedulixServer Users</i> dialog is used for the administration of schedulixServer users.
	The <i>Groups</i> dialog is used for the administration of schedulixuser groups.
	The <i>Job Servers and Resources</i> dialog is used for the administration of job-servers, scopes and associated resources.
	The <i>Named Resources</i> dialog is used for creating, modifying and deleting Named Resources.
	The <i>Environments</i> dialog is used to define runtime environments for jobs.
	The <i>Footprint</i> dialog is used for the administration of footprints (resource profiles) in the system.
	The <i>Exit State Definition</i> dialog is used for defining and administering Exit State names.
	The <i>Exit State Mappings</i> dialog is used for the administration of translations between return values from the system processes and logical Exit States.
	The <i>Exit State Profile</i> dialog is used for the administration of profiles, i.e. a summary and prioritisation of Exit States.
	The <i>Exit State Translation</i> dialog is used for the administration of translations of Exit States to Exit States.
	The <i>Resource State Definition</i> dialog is used for defining and administering Resource State Names.
	The <i>Resource State Profile</i> dialog is used for the administration of profiles, i.e. a summaries of Resource States.
	The <i>Resource State Translation</i> dialog is used for the administration of translations of Exit States after Resource States have been changed.
	The <i>Calendar</i> dialog shows an overview of the pending submits.
	The <i>SysInfo</i> dialog shows information about the configuration and current state of the system.
	<i>Shell</i> allows the execution of API Commands.

Table 1.1: Description of each icon in the main menu

## 1.4 Layout of the object window

When a dialog box is called from the *Main Desktop*, it is opened in a new window. Each window is independent of other dialog windows and any number of dialogs can be open at the same time. A standard dialog looks like this:

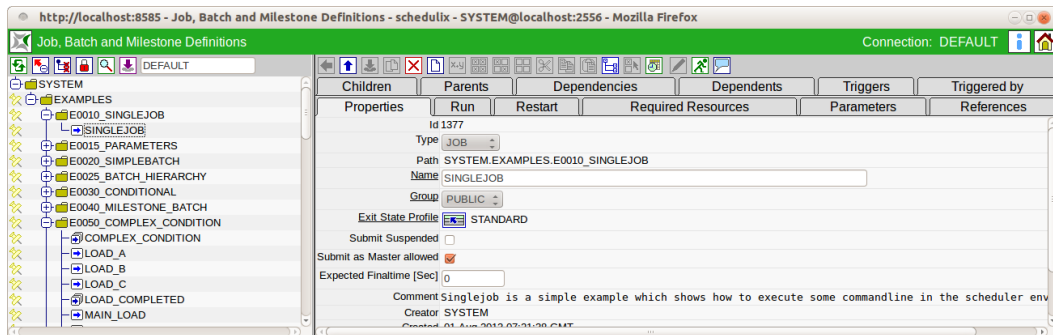


Figure 1.4: Example of a standard dialog

### 1.4.1 Title bar

Each dialog has a title bar at the top. The title bar contains the elements described below.

#### independIT icon/hourglass



The independIT icon is located in the upper left corner of the window. Clicking this icon opens a new window with the website of independIT GmbH. When you perform an action, the new or modified data has to be sent to the schedulix server or data has to be downloaded from it. This activity is indicated by the hourglass icon. No further actions can be performed in the window for as long as the hourglass icon is active. If you do try to perform an action, this will trigger an error message as shown in figure 1.5 In this case, the box has to be confirmed by clicking OK and you have to wait until all the data has been transferred to or from the server. You can continue entering data as soon as the hourglass disappears and the IndependIT icon is displayed again.

#### Name of the dialog box

The name of the active dialog box is shown next to the independIT icon.

#### About

Version and system information can be displayed by clicking this button. The following information is shown in this window:

## Layout of the object window

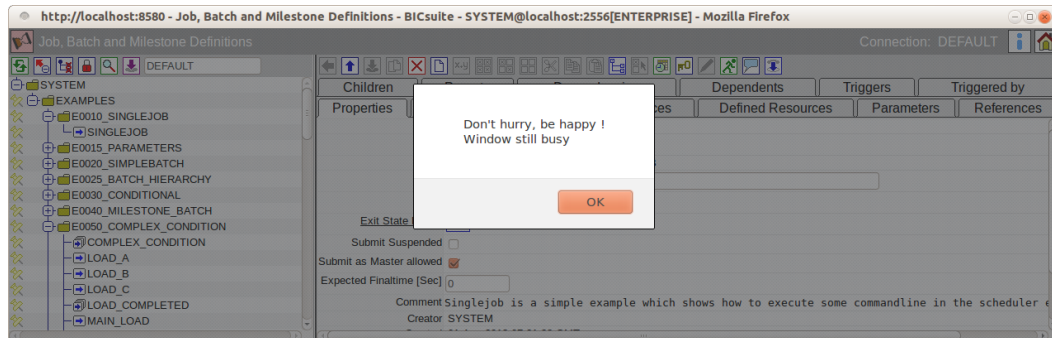


Figure 1.5: Busy message

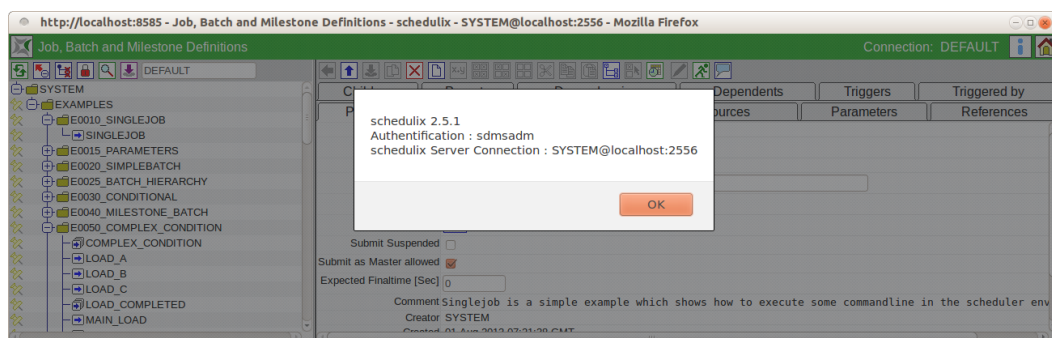


Figure 1.6: About window

*schedulix!Web n.n.n*: Current version of the schedulix!Web user interface.

*Authentication*: Name of the user who is currently logged in.

*schedulix Server Connection*: Connection data for the schedulix server.

### **Help**

The *Help* icon is only visible on the *Main Desktop* instead of the *Home* icon. Clicking the *Help* icon takes you straight to the online documentation.

### **Home**

The *Home* icon is used to call the *Main Desktop*. If the *Main Desktop* window has been minimised or it is in the background, it is restored and brought into focus. If it has been closed, clicking the icon opens a new *Main Desktop* window.



## 1.4.2 Navigator

The left pane of a dialog window is the Navigator. It shows the objects that have already been defined. These are displayed either in a hierarchy (if the type of objects can be arranged hierarchically, for example [named resources](#)) or as a simple list if there is no hierarchical order. The hierarchical view is a tree structure.

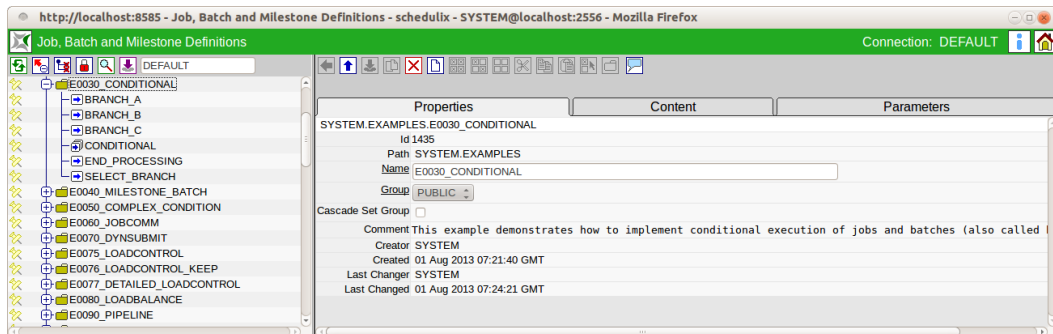






Figure 1.7: Navigator with a hierarchical view

If an object displayed in the hierarchical view is a container that can contain other objects, this is indicated by a folder icon like e.g.



These are different objects (folders, scopes, named resources, etc.) depending on their colour and how they are displayed. If a container holds sub-objects as well, a  or  is displayed. By clicking this icon, the contents of the container can be displayed  or hidden .

### 1.4.2.1 Standard buttons

Clicking the buttons above the Navigator as described below controls the window view and whether contents are displayed or hidden:




#### Refresh

The *Refresh* button is used to refresh the list of objects in the navigation. After clicking this button, all the available objects are read out from the server again.



#### Collapse All

With this button, an expanded hierarchy (i.e. a hierarchy in which the sub-folders are displayed and the parent folder has a ) is completely closed again. This means that only the objects at the top level of the hierarchy are visible.

## Layout of the object window

### **Expand All**

This button is used to fully open a closed or partially open hierarchy. This means that all the top level containers are scanned for sub-containers or sub-objects and these are displayed. The folder icon is preceded by the  $\oplus$  character. If the sub-containers hold further sub-objects, these are also displayed until all the objects throughout the entire hierarchy are shown on the screen. With a large hierarchy containing a large number of objects and sub-objects, this can take a relatively long time. For this reason, the button is not available for every hierarchical object type.

### **Show Leaves**, **Hide Leaves**

These two buttons allow non-container objects to be displayed and hidden in hierarchies in the Navigator. Only one button is alternately always visible.

### **Show Locked**, **Hide Locked**

These two buttons allow the names of objects without read privileges to be displayed and hidden. Only one of these two buttons is alternately always visible.

### **Save View**

The *Save View* button is used to save the current state of the hierarchy in the Navigator.

### **Search**

This button activates the search mask in the navigation. This enables you to very easily find an object with a known name even in a complex hierarchy. Clicking the *Search* button opens the search mask:

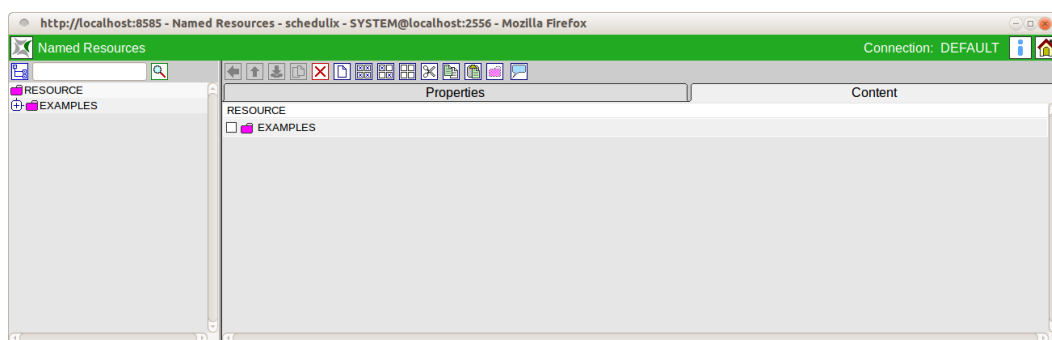


Figure 1.8: Searching in the Navigator

## Layout of the object window

### Start Find

This starts the search. In the example shown here, a search is being run in the entire hierarchy for *Named Resources* that contain the search string.

Hits are shown as a list in the Navigator. Hierarchies are not taken into consideration. The result could look like this:

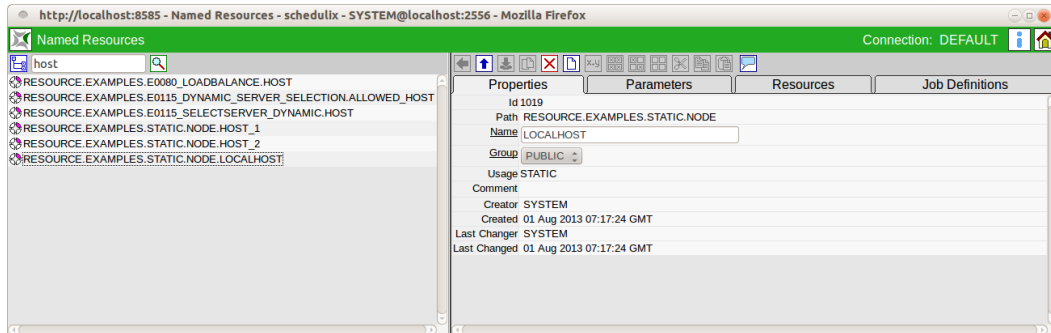


Figure 1.9: Search results

Here we were searching for the term "host", and three *Named Resources* were found that contain this term. The search is not case-sensitive with regard to the search term.

### Exit Find

Clicking this button closes the search mask and the standard button bar is displayed above the Navigator again.

## 1.4.3 Editor

The Editor is the dialog window for modifying or creating objects. When an object is selected in the Navigator, its data is displayed in the Editor and can be modified here. A selected object can be saved, deleted or duplicated using the buttons at the top of the Editor. A new object can be created by clicking the *New* button.

The data in the Editor is either displayed in full in one window or it is logically grouped in different tabs.

The number of tab sheets in a dialog varies depending upon the type of the object selected in the navigation. During an action (modifying or creating an object), you can switch between the single tabs as often as you need to and in any order. You do not need to save your changes in between because all the data (including the data in hidden tabs) is kept in the memory. Only when you click the *Save* button is the data in all the tabs sent to the schedulix server and saved there. If any changes have been undone by clicking the *Cancel* button, all the modified data (including in any hidden tabs) is discarded.

## Layout of the object window

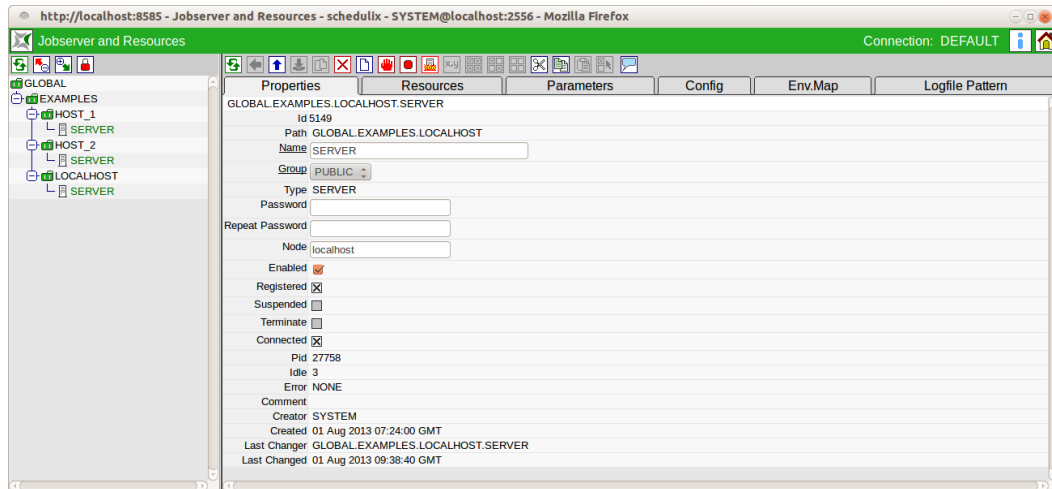


Figure 1.10: Editor with tabs

A tab sheet is selected simply by clicking the respective tab on which the name of the tab sheet is shown. The data from the selected tab is then displayed in the mask and the tab is graphically moved into focus and highlighted in a light colour. The inactive tabs are dark in colour.

The Editor button bar shows different enabled and disabled buttons depending upon which tab is open.

### 1.4.3.1 Standard buttons

The following standard buttons are shown in many dialog boxes and are used in an identical or comparable way in each dialog.

#### Cancel

This button is used to undo actions. All changes that have not yet been saved can be revoked. This means that all changes made since the data was loaded or last saved are discarded.

The *Cancel* button is only enabled if a change has been made. Changes are recognised only when you leave the first field in which a change was made. If there is only one field in the mask, you have to press the keyboard tabulator key to accept your changes. This then enables the *Cancel* button.

To prevent entered data from being lost, a confirmation query is displayed after you have clicked the button. If you really want to discard your changes, you have to confirm this query by clicking *OK*. Otherwise, you can click *Abort* to return the previous dialog and preserve the changes you have made.



Clicking the *Up* button takes you up one level in the hierarchy.



A change that has been made is saved with this button. The *Save* button is only enabled if data has actually been changed. Changes are only recognised once you have left the changed input field.



This button is used to create a new object which possesses all the properties of the currently selected object. To enable the button, you have to enter a new name for the object. When you click the button, a new object is created with this name and displayed in the navigation window.

Important: If you click *Save* instead of the *Clone* button, only the name is changed, i.e. a new object is not created.



This button is used to delete an entire object. A confirmation query is displayed before the object is displayed.

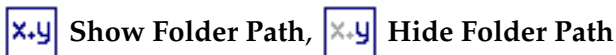
If you really want to delete the object, confirm the query *OK*. If you want to cancel the operation, press the *Abort* button.

The object is deleted from the server after the action has been confirmed. There is now no way to undo the deletion. If the deletion was unintentional, you will have to create the deleted object again.

In the lists, you can also delete selections. The procedure here is as follows. First, you have to select the rows you want to delete either individually or by clicking the *Selection* button. Now click *Drop*. All the selected entries are deleted after you have confirmed the prompt.



This button is used to create a new object. If different objects can be created in a dialog, first you have to select the type for the new object. The input fields and tabs for the corresponding object type are then displayed. If a selection is not necessary, the Editor opens immediately after you click the button. If an Editor window still contains any unsaved changes, the button is disabled to prevent the changed values from being accidentally deleted.



These buttons are switches. When you click the *Show Folder Path* button, the scopes, submitted entities, scheduling entities and folders are displayed together with the

## Layout of the object window

complete parent hierarchy. The levels are each separated by a full stop. The view looks this after you have clicked the button:

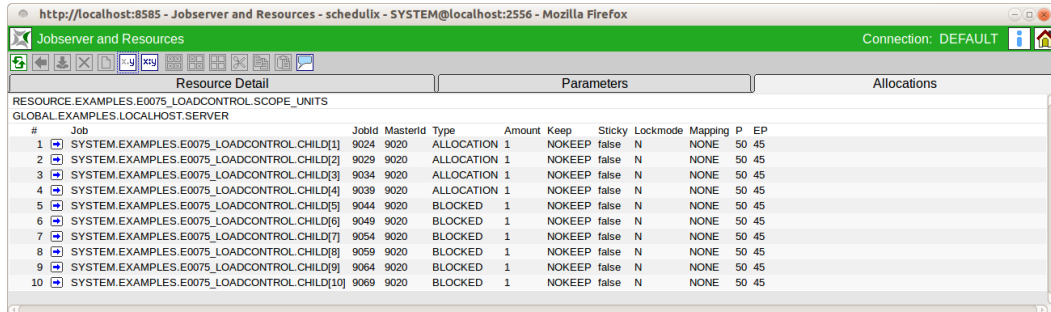


Figure 1.11: Show Folder Path

When you click *Hide Folder Path*, the previously visible parent folders are not displayed. The window looks like this:

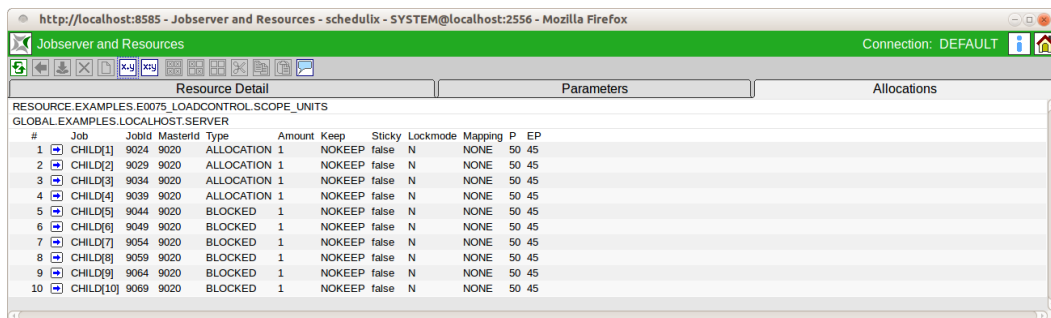


Figure 1.12: Hide Folder Path

### Show Hierarchy Path, Hide Hierarchy Path

These buttons are switches. Clicking the *Show Hierarchy Path* button displays the task with the complete superordinate parent-child hierarchy. If a task has a parent, this is shown in the name and so on until the top level has been reached. The levels are each separated by a colon (Figure 1.13).

When you click *Hide Hierarchy Path*, the previously visible hierarchy is no longer displayed. The view looks this after you have clicked the button (Figure 1.14):

## Layout of the object window

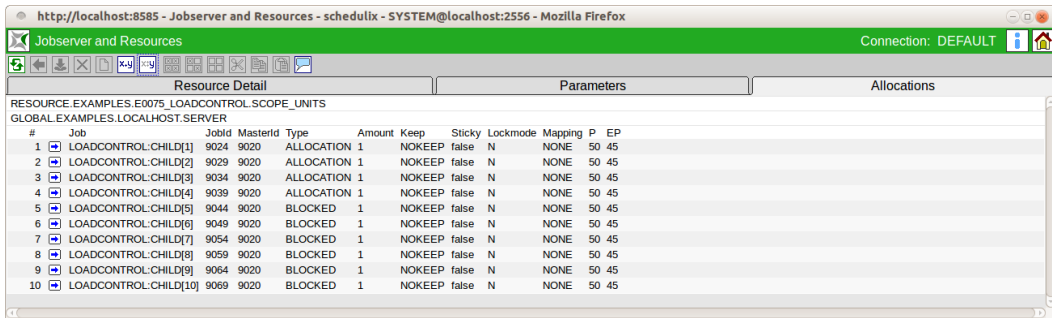


Figure 1.13: Show Hierarchy Path

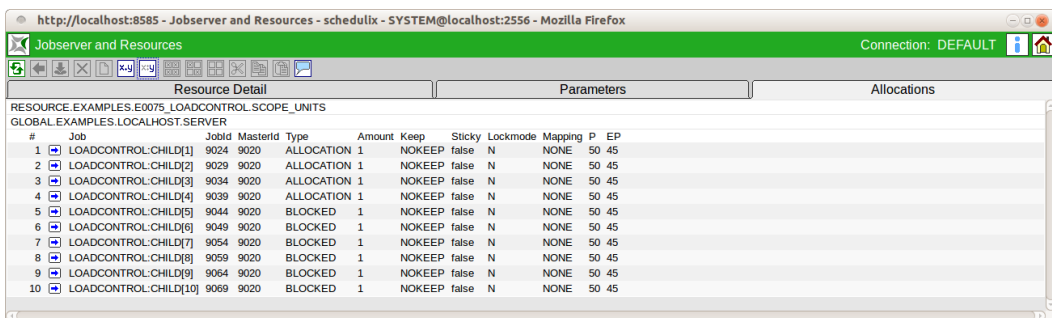


Figure 1.14: Hide Hierarchy Path



### Select All

This button is for selecting all the objects. This means that all the objects currently displayed in the Navigator or the Editor window (depending on the button bar where the button is) are selected. All the objects are then marked with a cross  in front of their name.

Once all the objects have been selected, they can then be cut, copied or deleted.



### Toggle Selection

This button is for reversing the current selection. This means that all the selected objects are deselected and all the objects that have not yet been selected are selected. The selected objects can now be deleted, cut or copied.

This behaviour is illustrated in the following two screenshots.

Your selection is reversed when you click the Toggle Selection button.

## Layout of the object window

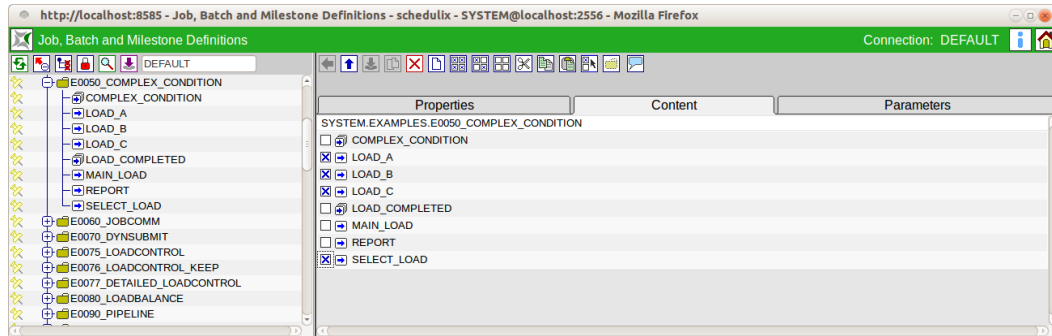


Figure 1.15: Reverse Selection; Initial State

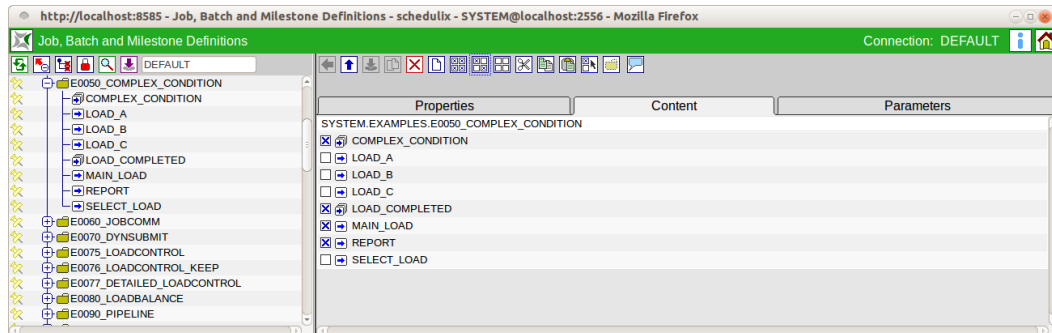


Figure 1.16: Reverse Selection; Result



 **Deselect All**

This button reverses your selection and the markings on the selected entries are removed. This allows you to undo a selection that you have made.

 **Cut**

This button is used to cut out objects out of a list or a tree. The objects are copied to the schedulix clipboard. The objects are only marked for cutting, but they disappear from the current view. They can be inserted somewhere else by clicking the *Paste* button.

 **Copy**

With the *Copy* button, objects can be copied to the schedulix clipboard for inserting them somewhere else by clicking *Paste*.

 **Paste**

This button is used to paste previously selected and suitable objects into a list. The objects have to be selected beforehand in another dialog or pop-up window.

 **Select**

With this button you can open a window where you can select objects that are suitable for a list and copy them to the clipboard by clicking the *Copy* button. The objects in the clipboard can then be inserted into the list by clicking the *Paste* button.

 **Time Scheduling**

This button opens the [Time Scheduling](#) dialog window. It is only displayed if you have selected a Master Submittable Object in the navigation window.

 **Grants**

This button opens the dialog for administering the access privileges.

 **Edit**

The *Edit* button activates an edit mode. What data is edited depends upon the context.

## 1.5 Selecting values

### 1.5.1 Selecting values from drop-down lists

With some input fields it is necessary to make a selection from a range of default values. In these cases, a *drop-down* box is displayed containing a list of all the possible values.

A *drop-down* box looks like this:



A single value is selected from the drop-down list by expanding it by clicking the button next to the box or using the keyboard shortcut ALT-DOWN ARROW.

The expanded drop-down list can look like this:



Clicking the required value places it into the field. Alternatively, you can select the values using the DOWN ARROW or UP ARROW keys.

### 1.5.2 Choosing values using the Select button

If the list of available values is only read from the server at runtime you have to use the *Select* button. In the next screenshot, a Resource State Profile can be chosen by clicking *Select*:

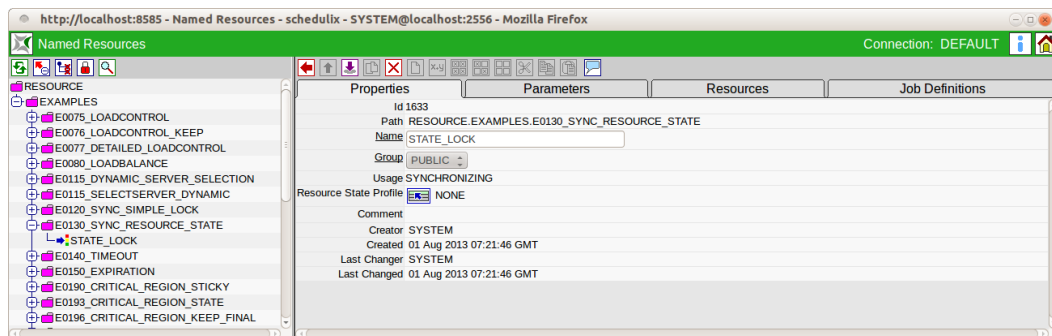


Figure 1.17: Choosing a Resource State Profile; initial state

When you click the *Select* button,



a list of possible objects is displayed in the Editor.

Clicking one of these objects places the element into the field. You can jump back from the selection mask to the editing window without making a selection by clicking the *Cancel* button.

## Selecting values

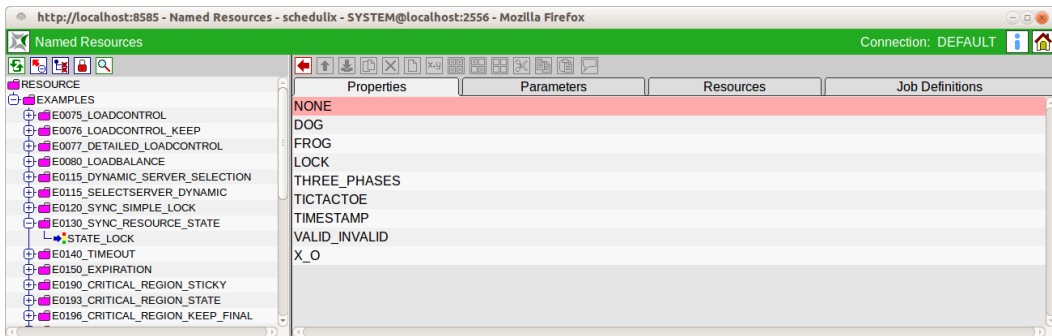


Figure 1.18: Choosing a Resource State Profile; selection list

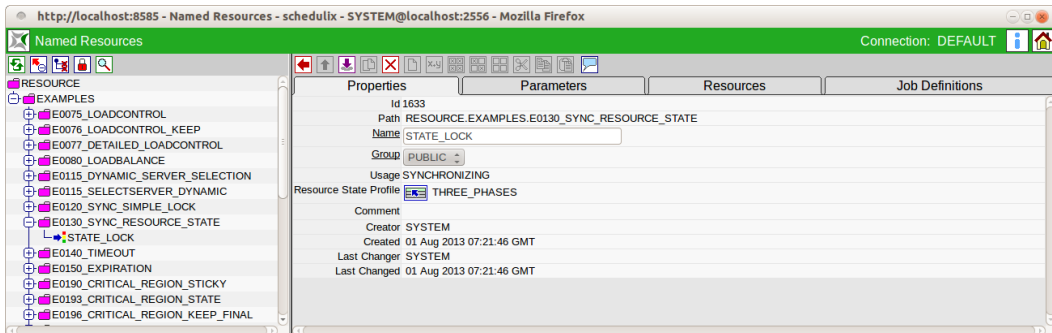


Figure 1.19: Choosing a Resource State Profile; result

## 1.6 Handling standard lists

In many dialog windows it is possible to not only enter values into the fields, but you can also create lists with links to additional objects. This way, for example, a list of Resource States belonging to a Resource State Profile is administered in that profile. Here is an example:

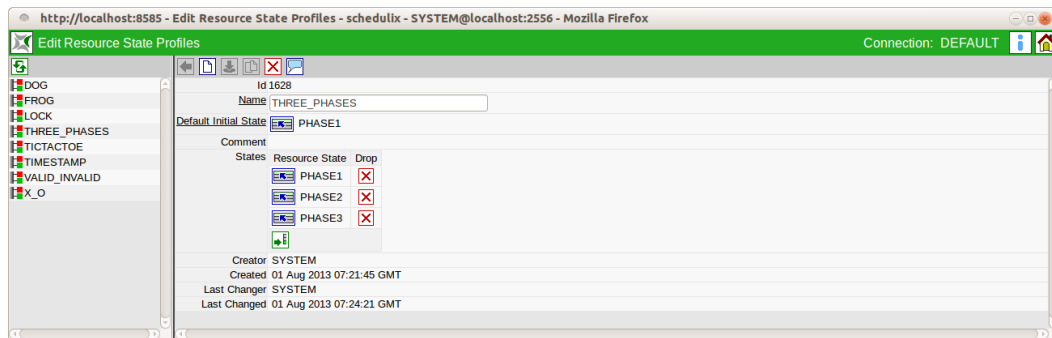


Figure 1.20: Example of how standard lists are handled

Here, to the standard fields *Name* and *Default Initial State* we want to append a list of states.

The following actions can then be performed in such a list.

### 1.6.1 Adding a line



**Add**

A new line is added to the open list by clicking the *Add* button. An empty line appears after you have clicked the button.

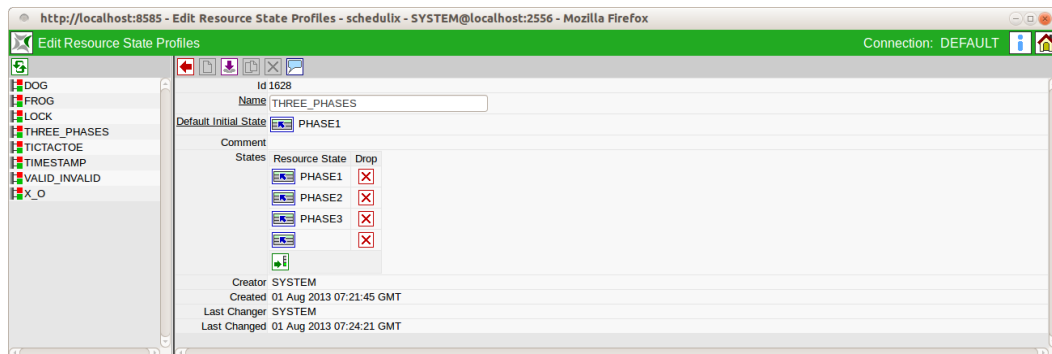


Figure 1.21: Handling lists; adding a line

In our example, the dialog looks like in figure 1.21.

## Handling standard lists

A value can then be chosen using the Select button.

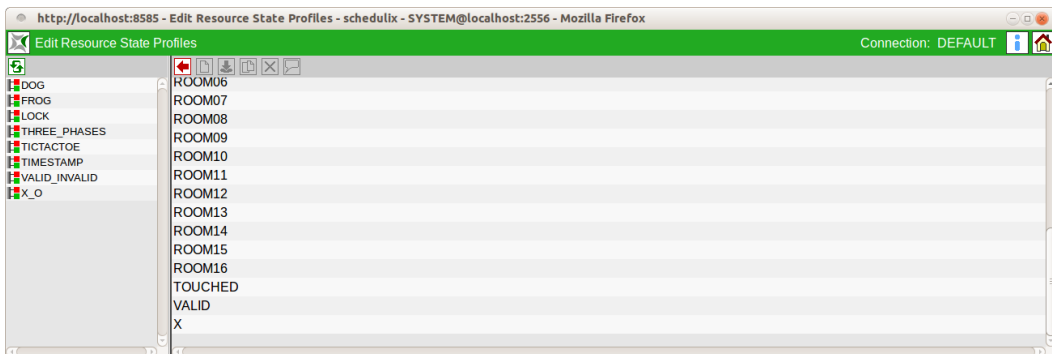


Figure 1.22: Handling lists; choosing a value

In this example, the value "Valid" was chosen.

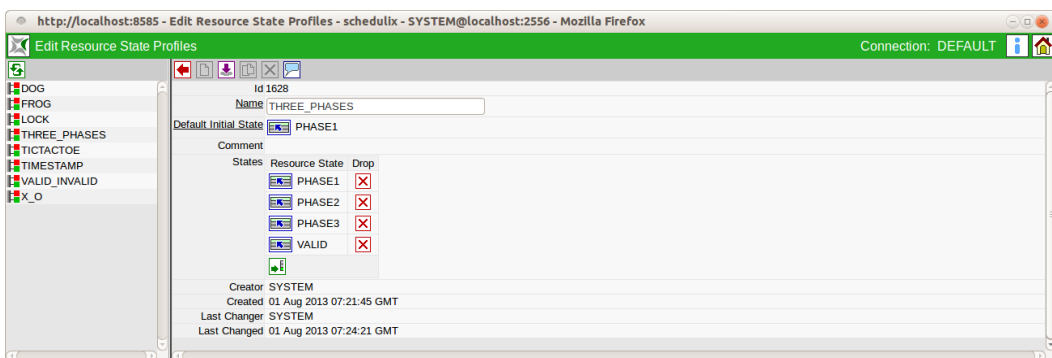


Figure 1.23: Handling lists; result

You now have to save the change with the *Save* button.

You can also modify an existing line by choosing a dependent object using the *Select* button. In this case, the link to the old object is deleted and a new link is created to the selected object.

### 1.6.2 Deleting a line

#### Drop (delete)

When you click the *Drop* button, this removes a line and the link between the object loaded in the Editor and the selected object in the line is deleted after you have confirmed the prompt. In our example, clicking the *Drop* button will delete the line "PHASE1".

## Handling standard lists

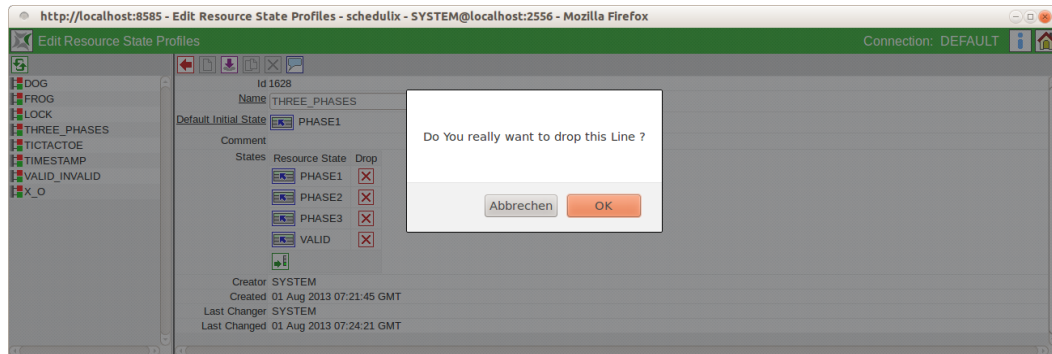


Figure 1.24: Handling lists; deleting a line

The line with "PHASE1" in it no longer exists once you have clicked *OK*. The object in the line (i.e. the Resource State "PHASE1") still exists, but it is no longer allocated to this profile.

The change now has to be saved on the server by clicking the *Save* button.

### 1.6.3 Changing line values

In many dialogs, not only does a line hold additional properties from other objects, but additional fields also have to be maintained for this link. These are shown in the list as a field string after the linked object. In the Footprint dialog, for example, not only does a *Named Resource* have to be selected in a line, but the other fields *Amount* and *Keep* need to be maintained as well.

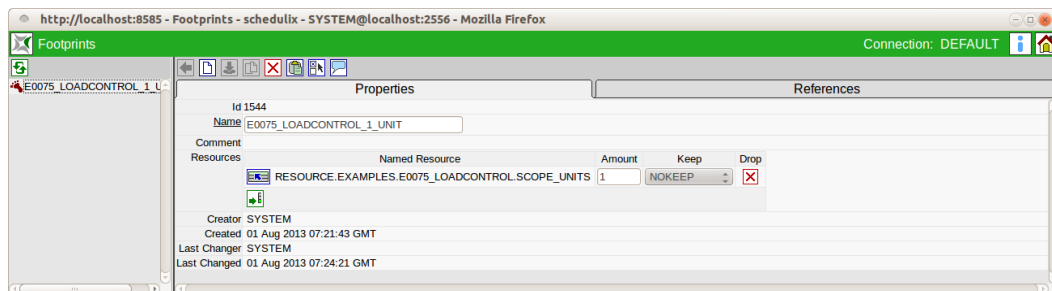


Figure 1.25: Handling lists; editing

This is done analog to how standard fields are edited.

The changes now have to be saved by clicking the *Save* button.

## 1.7 Copy & paste and the clipboard

The schedulix!Web user interface features an intelligent copy & paste functionality and an object-sensitive clipboard. This allows objects to be selected, cut, inserted and deleted between different dialog boxes or in the same dialog. So long as the action has not been performed, the data is located in the clipboard of the schedulix!Web interface system and can be used by other suitable dialogs.

The term *object-sensitive* describes the logic routine executed when inserting objects from the clipboard. Before objects are inserted from the clipboard, the system checks whether they are of the same type as the required objects in the dialog into which they are to be inserted. It is therefore not possible to insert an object of type A if the recipient dialog is expecting an object of type B. If there are no objects of type B in the clipboard, the insert operation cannot be performed.

This mechanism is not to be confused with the standard clipboard system used by Windows operating systems. The schedulix clipboard functions totally independently of the Windows clipboard and vice versa. The data stored in the schedulix clipboard is available exclusively to schedulix dialogs and cannot be used or read out by other Windows applications.

How the clipboard is used is demonstrated in two examples in the next sections.

### 1.7.1 Moving objects in the hierarchy

In a folder hierarchy, you can use the clipboard to move objects to another level in the hierarchy.

Example:

The navigation window for the *Batches and Jobs* dialog shows the following hierarchy:

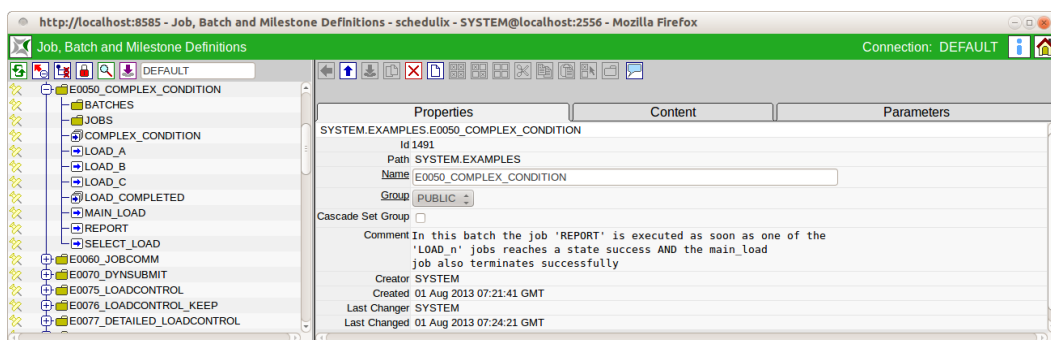


Figure 1.26: Moving objects; initial state

Below the COMPLEX\_CONDITION folder there are various jobs and batches. This folder needs to be reorganised. The batch "COMPLEX\_CONDITION" is to be moved to the BATCHES folder, and then the jobs "LOAD\_A", "LOAD\_B" and

## Copy & paste and the clipboard

“LOAD\_C” are to be moved to the “JOBS” folder.  
This can be done as follows using the clipboard:

1. Select the objects to be moved

First you select the batch COMPLEX\_CONDITION in the Navigator. The list of all the objects in the COMPLEX\_CONDITION folder are now displayed in the CONTENT tab in the Editor.

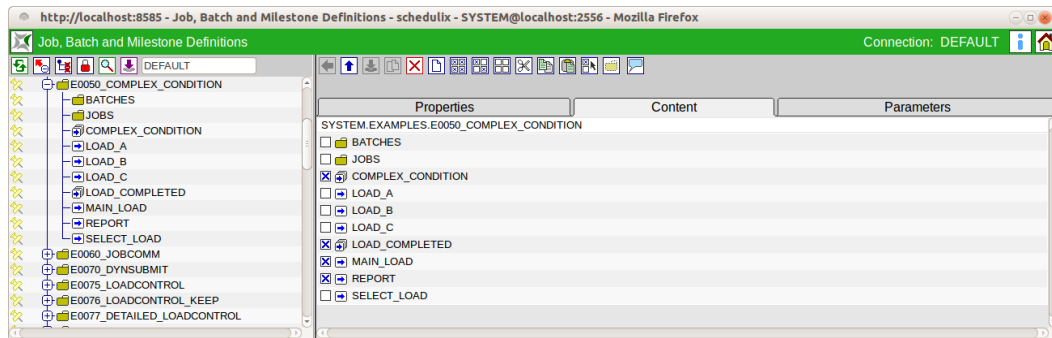


Figure 1.27: Moving objects; Content tab in the Editor

The batch is now marked by clicking the check box in front of the name and the icon.

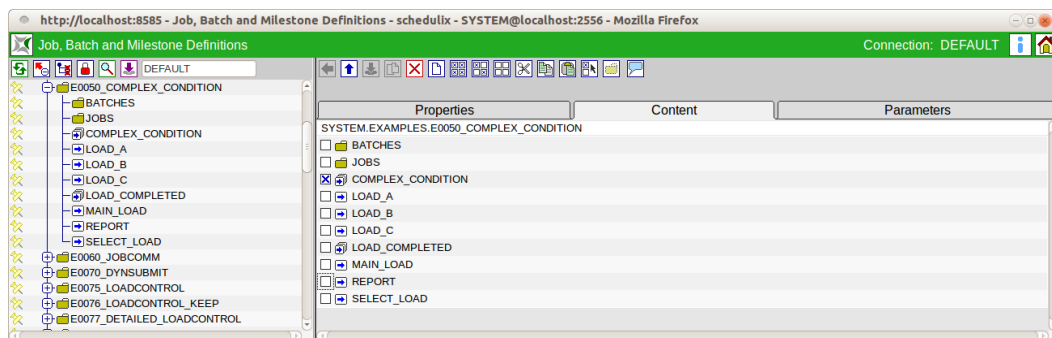


Figure 1.28: Moving objects; selection

2. Cutting out the selected objects

Using the *Cut* button, the selected batch is cut out and disappears from the screen view. The object has now been saved to the clipboard. The dialog looks like this after the batch has been cut out:

The important to remember here is that although the object that has been cut out is no longer visible, it is still in the folder. So if you close the window now, nothing has happened. It is only actually moved during the subsequent paste operation.



## Copy & paste and the clipboard

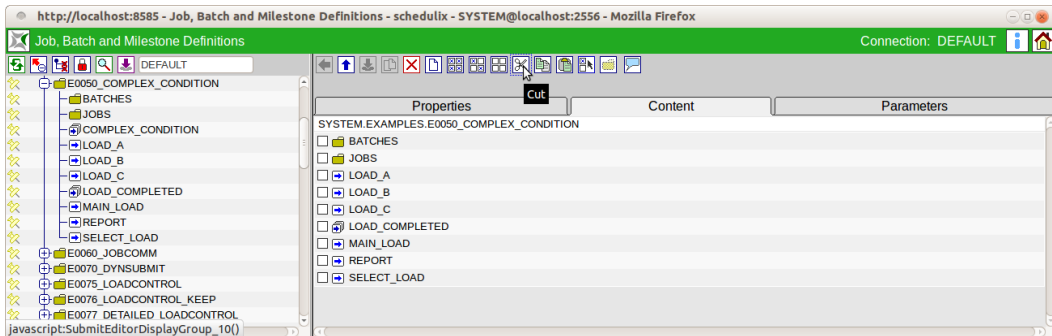


Figure 1.29: Moving objects; cut

### 3. Selecting the target folder

We now need to choose the target dialog and target folder for the move operation. This can be done simply by clicking the respective folder in the navigation pane of the target dialog. As the clipboard functions between different windows, the target folder can also be located in another dialog.

In our example, we have chosen the "BATCHES" folder and the "Content" tab opens.

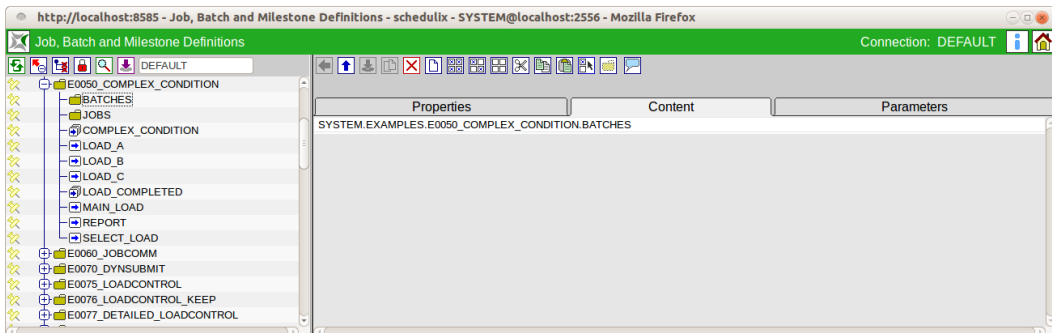


Figure 1.30: Moving objects; choosing the target folder

### 4. Inserting the selected objects

To complete the move, the object is inserted from the clipboard. This is done using the *Paste* button. Pasting the object now triggers the move action on the server. This action has now been completed and cannot be undone. The result looks like this:

Performing the action with the aforementioned jobs and the JOBS folder produces the following hierarchy:

## Copy & paste and the clipboard

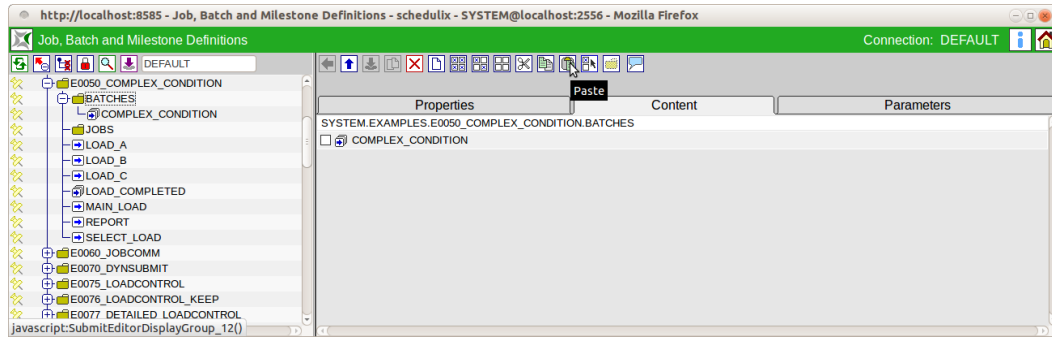


Figure 1.31: Moving objects; paste

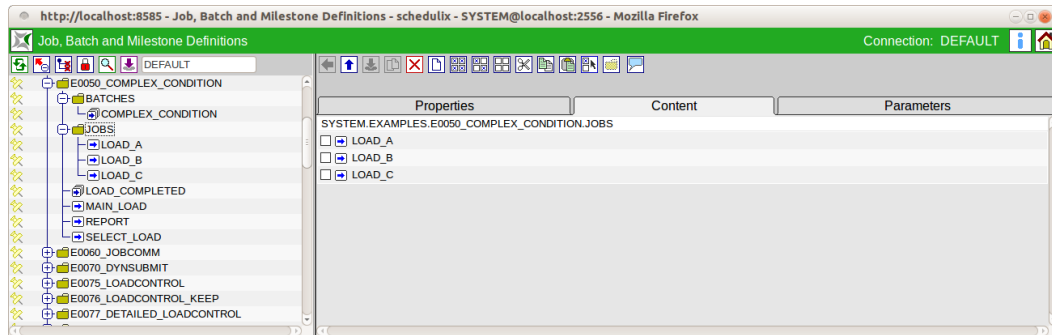


Figure 1.32: Moving objects; result

Copy & paste and the clipboard

## 1.7.2 Linking objects

In some dialogs, objects are linked to other objects or lists of objects. An example of this is the definition of dependencies. The clipboard can also be used to fill such a list.

To illustrate this and demonstrate the procedure for such an action, we will define the dependencies for a job.

Here you can see the job "END\_PROCESSING", which does not yet have any dependencies:

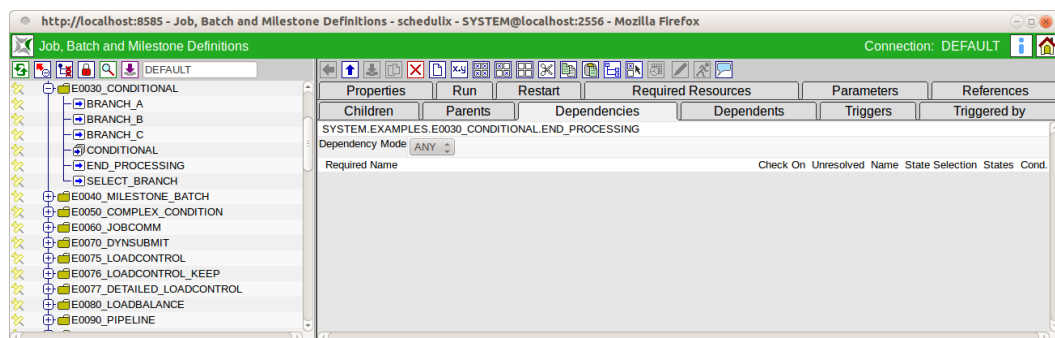


Figure 1.33: Linking objects; initial situation

Additional jobs have to be added at this point to make the execution of this job dependent upon other processing steps. These can be marked beforehand in other dialogs and copied to the clipboard. We now take the following steps:

1. Selecting the objects

Jobs can be marked in the "Content" tab of a folder, for example. In the next dialog, we have selected the three processing steps A, B and C.

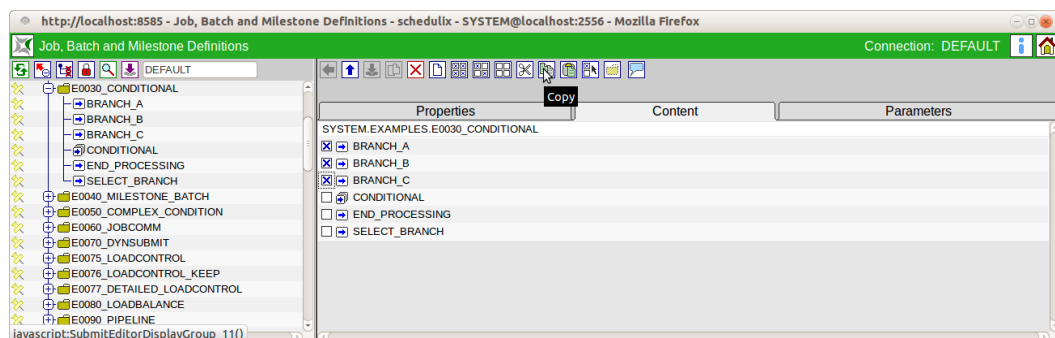


Figure 1.34: Linking objects; selection

2. Copying the objects to the clipboard

## Copy & paste and the clipboard

Clicking the *Copy* button copies the selected jobs to the clipboard.

### 3. Inserting the objects

In the Dependency Editor window of the "END\_PROCESSING" job, we now click the *Paste* button and the cached jobs are added to the list. The list now looks like this:

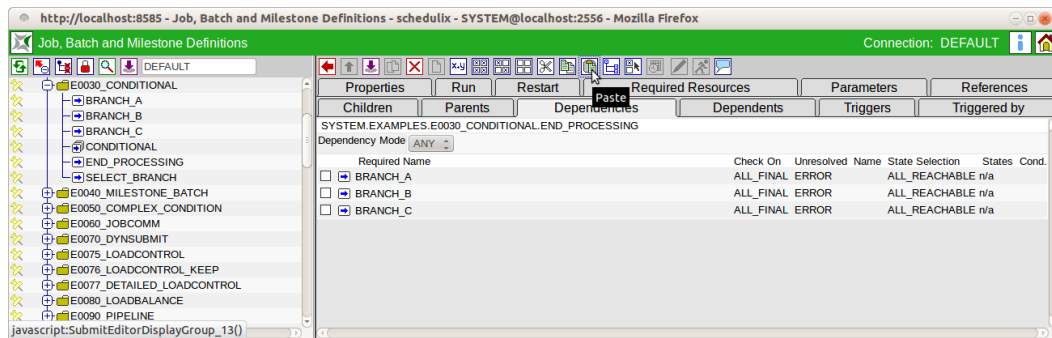


Figure 1.35: Linking objects; paste

### 4. Saving the change

In contrast to the action for moving objects, which does not have to be additionally saved, copy actions in an editor window do need to be saved by clicking the *Save* button.

Important: The cached data remains in the clipboard. If the selected jobs are to be added to other dialogs, you just have to choose the appropriate target and then click the *Paste* button.

## 1.8 Graphic Legend

Different graphics are used in this documentation to illustration correlations.

### 1.8.1 How schedulix objects are represented

The following symbols and their meanings are used in the graphics for schedulix objects:

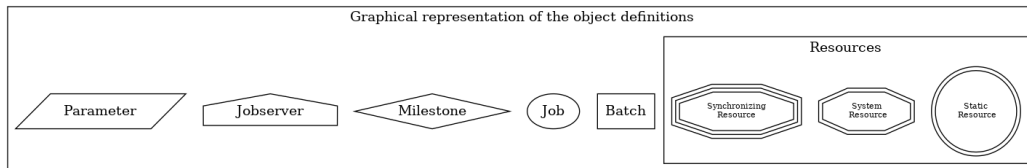


Figure 1.36: Legend for the graphical representation of schedulix objects

### 1.8.2 Representation of parent-child relationships in workflows

Relationships between parents and children are represented in the graphics as follows:

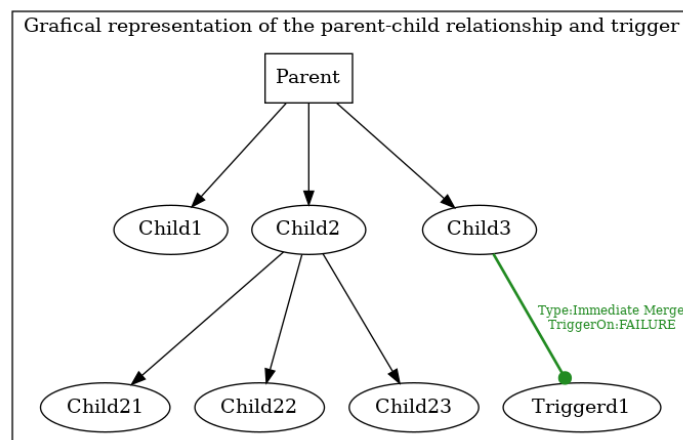


Figure 1.37: Legend for the graphical representation of parent-child relationships

The family tree is usually read from top to bottom from the highest ranking parent down to the respective children. The arrow always points from the parent to the children.

More details about parent-child relationships between tasks can be found in Chapter [Batches and Jobs](#).

### 1.8.3 Representation of dependencies between Scheduling Entities

Dependencies between individual Scheduling Entities are represented in the graphics as follows:

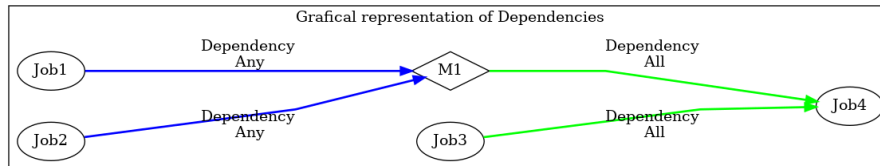


Figure 1.38: Legend for the graphical representation of dependency relationships

Dependencies are displayed from right to left. This means that the first job is shown on the far left, then comes the next one time-wise, etc.

The arrows always point from the required Scheduling Entity to the dependent Scheduling Entity. This thus shows the timeline for the Scheduling Entities. For example,  $A \rightarrow B \rightarrow C$  means that A is run first, and is followed by B first and then C. A is the required entity of B and B is the required entity of C. B is accordingly dependent upon A. C is dependent upon B (and so implicitly also dependent upon A).

The colour of the arrows corresponds to the interlinking of multiple dependencies. If the colour is green, all the required tasks are linked with "AND" (the dependency ALL).

If the colour is blue, all the required tasks are linked with "OR" (the dependency ANY).

If a task is only dependent upon precisely one other task, the arrow is green since in this case the links AND and OR are equivalent.

More details about dependency relationships between Scheduling Entities can be found in Chapter [Batches and Jobs](#).

## 1.9 Comments

Comments can be saved for most objects.



### Comment

Clicking the *Comment* button opens the *Edit Comments* mask. Here you can enter any text comments relating to the object. Text comments consist of a tag and a comment text. On display tags will be rendered as headlines. If the tag is left empty, the comment text will be concatenated with the preceding text comments comment text. Apart from entering the comment, no other action is performed on the object.

As an alternative to text comments, a link can be set to an intranet page instead. To do this, select "URL" as the type and enter the URL in the comment box. The links are also activated in the schedulix!Web interface. This allows any number of complex documents to be linked with the schedulix objects.

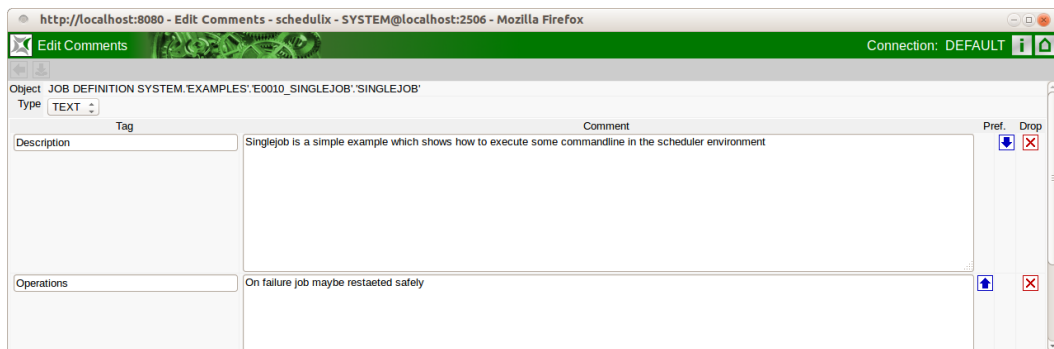


Figure 1.39: Recording comments

## 1.10 Standard fields

The following fields are defined for all objects:

**Creator** Name of the user who created the object.

**Created** Date and time at which this object was created.

**Last Changer** Name of the last user who changed the object.

**Last Changed** Date and time at which this object was last changed.





## 2 Exit State Definitions

### 2.1 View

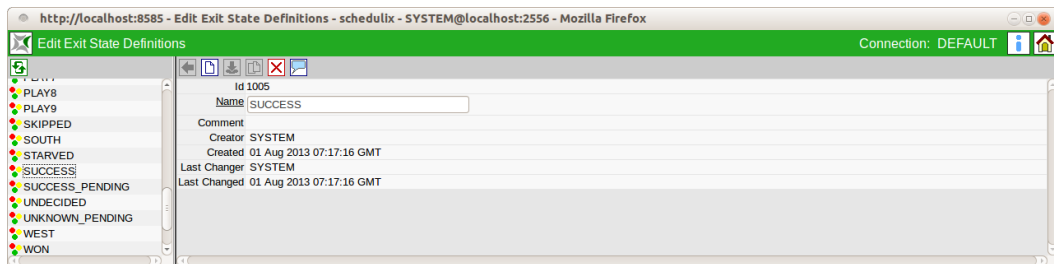


Figure 2.1: Exit State Definitions

### 2.2 Concept

#### 2.2.1 In short

The Exit State Definitions dialog is used to create logical names for exit states of a Submitted Entity.

#### 2.2.2 Detailed description

Exit State Definitions are logical names for exit states of a Submitted Entity. These logical names do not have any other properties.

### 2.3 Editor

The properties of Exit State Definitions are maintained in this tab.

Exit State Definitions can only be edited by users who belong to the "ADMIN" group.

All the input fields are "read only" for all other users.

## Editor

The fields have the following meanings:

**Id** System-wide unique number for identifying the object.

**Name** Unique name of the Exit State Definition. This name can be freely chosen.

## 3 Exit State Mappings

### 3.1 View

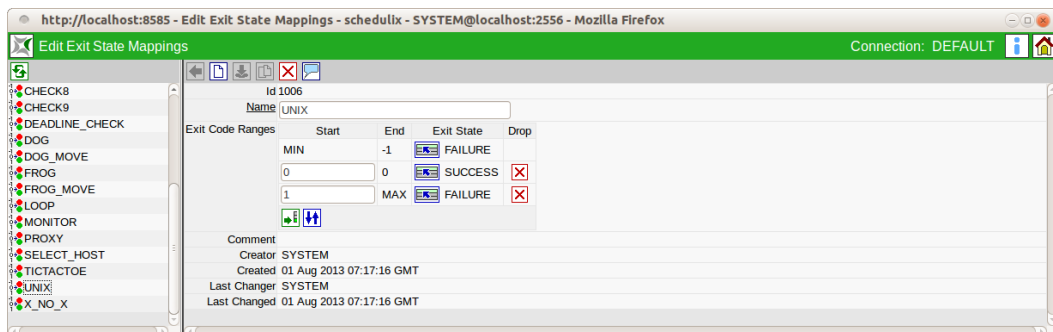


Figure 3.1: Exit State Mappings

### 3.2 Concept

#### 3.2.1 In short

Exit State Mappings are used for translating numerical Exit Codes into logical Exit States.

#### 3.2.2 Detailed description

Exit State Mappings are used to assign numerical value ranges of exit codes to logical Exit States. Each program that is run returns a numerical value to the calling program when it has finished. On UNIX systems, a value of 0 usually means that the program has been completed without any errors (SUCCESS). Values unequal to 0 indicate an error (FAILURE). However, deviating values with other meanings are also possible. For a better understanding of this logical meaning within schedulix Exit State Mappings can be used to assign value ranges to a logical name. The values range from  $-2^{31}$  to  $2^{31} - 1$ .

### 3.3 Editor

Names are assigned to numerical value ranges in this editor.

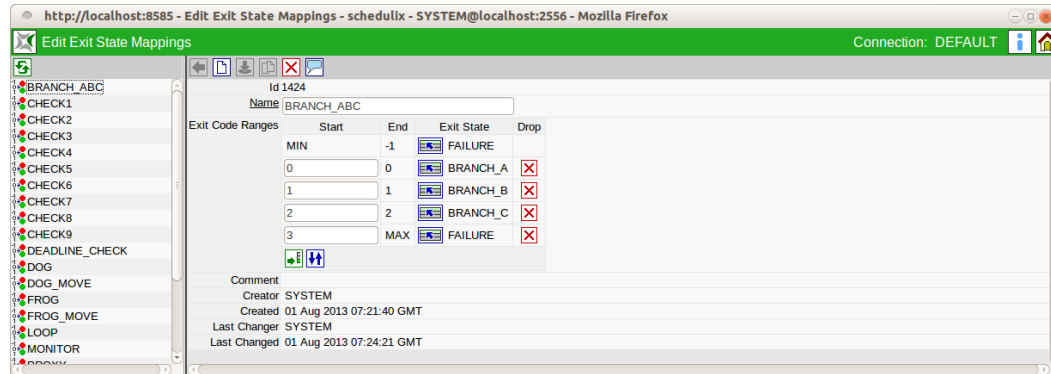


Figure 3.2: Exit State Mapping Editor

Exit State Mappings can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

Descriptions of all the standard buttons can be found in [Chapter 1.4.3.1 Standard buttons](#).

The buttons in the list shown above have the following meanings:



#### Insert new value range

This button adds another row to the table of Exit Code Ranges. The row is inserted at the end of the table and allows a new start value to be entered. This new end value is determined automatically when you save or sort the data.



#### Resort list

An unsorted list that is created by adding new intervals can be arranged in the correct order with this button.



#### Select Exit State

This button is for selecting a previously defined [Exit State](#) from a list.



#### Delete value range

This button deletes a range of values together with the Exit State from the list. The start value of the next interval is replaced by the start value of the deleted interval, i.e. the next interval is enlarged automatically.

## Editor

The fields have the following meanings:

**Id** System-wide unique number for identifying the object.

**Name** Unique name of the Exit State Mapping. This name can be freely chosen.

**Exit Code Ranges** The assignment of the respective value range is shown in a table. An Exit State can be assigned to a range of values in each row. The first row contains the start value  $-2^{31}$ , while the last row has the end value  $2^{31} - 1$ .



# 4 Exit State Profiles

## 4.1 View

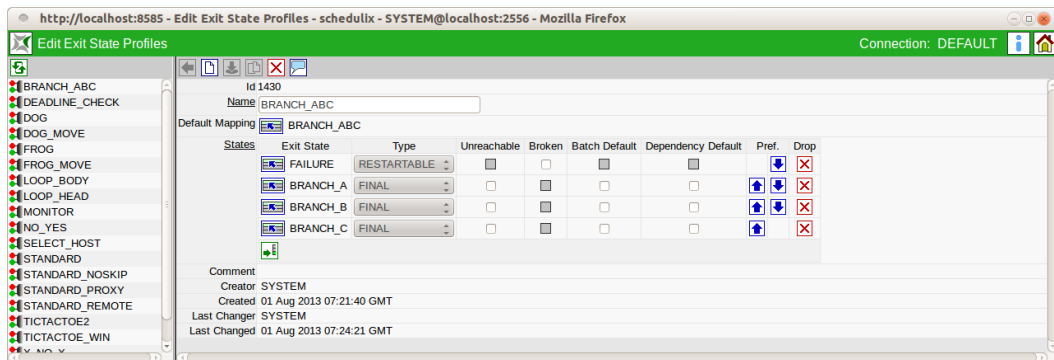


Figure 4.1: Exit State Profiles

## 4.2 Concept

### 4.2.1 In short

An Exit State Profile describes which Exit States can be achieved by a Submitted Entity. The Exit State Profile also defines whether a Submitted Entity can change its Exit State or if the Exit State is conclusive (FINAL).

### 4.2.2 Detailed description

Each Submitted Entity can finish with varying Exit States. The set of all valid Exit States for a Submitted Entity is described in an Exit State Profile. This description also defines the preferred Exit State that is crucial in a hierarchical workflow structure for determining the correct Exit State for a parent element. The Exit State with the highest preference is used.

## 4.3 Editor

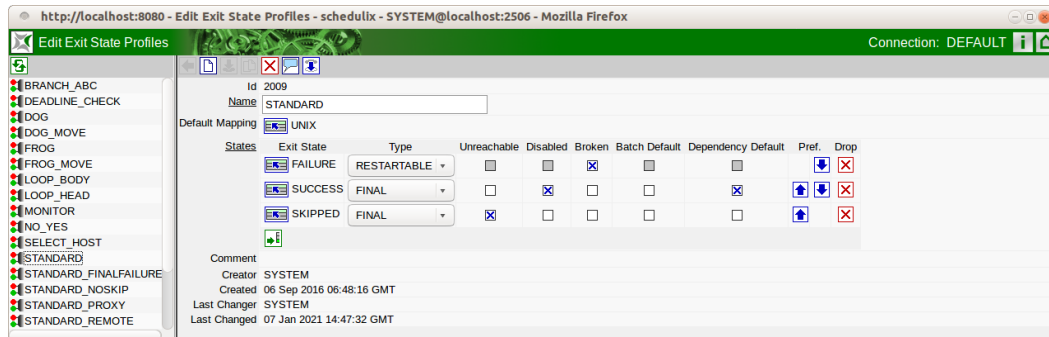


Figure 4.2: Exit State Profile Editor

In this editor, the Exit States belonging to a profile can be defined and the properties *Type*, *Unreachable*, *Broken* and *Preference* can be added to them.

Exit State Profiles can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

The fields shown above have the following meanings:

**Name** The *Name* field is used for assigning a unique name to an Exit State Profile.

**Default Mapping** The *Chooser* button is for selecting an [Exit State Mapping](#) from a list that is then used by default in conjunction with the Exit State Profiles provided that no other mapping has been defined in the job definition. An Exit State Mapping is used to translate numerical Exit Codes into Exit States.

The profile must contain an entry for each of these Exit States. However, not all Exit States in an Exit State Profile need to be accessible via the mapping.

**Type** One of three types can be assigned to each Exit State: FINAL, RESTARTABLE and PENDING.

The Exit State can no longer be changed once a task has attained its Final State. The result is final. Dependencies between jobs and batches can only be based on such FINAL Exit States.

If a task is to be capable of being restarted after it has been run, for example because an error has occurred, the Exit State must be of the type RESTARTABLE.

PENDING describes a state type that allows the Final State to be set externally via the API. A task in a PENDING state can be neither restarted nor are any dependencies fulfilled by it.

An example of how PENDING states are used:

A job sends an e-mail to an employee and requests the release of a result. After sending the e-mail, the job stops with a PENDING state. The employee can now



manually set the state to FINAL. Alternatively, a process evaluating the response e-mail can change the status. Dependent tasks can only be started once the state is FINAL.

**Unreachable** A maximum of one Exit State in the list can be marked as being Unreachable. This Exit State is set when a job in a process can no longer be executed because the dependencies cannot be fulfilled.

The Unreachable State is not attained if one or more predecessors have been cancelled. Because the cancel operation is a manual intervention, the consequences of the intervention have to be handled manually as well.



The Unreachable State must be of the type FINAL.

**Disabled** A maximum of one Exit State in the list can be marked as being Disabled. This Exit State is set when a job in a process is disabled and thus not executed. If Disabled has not been set, the FINAL state with the lowest priority preference is used which (if present) has not been set as being UNREACHABLE.

**Broken** A maximum of one Exit State in the list can be marked as being Broken. This state is set when a job is put into an Error State due to a failure. Such an error occurs, for example, if the run program cannot be started. The Error State must be a RESTARTABLE State. The Broken flag allows a trigger to be used to automatically respond to such error situations.

**Batch Default** A final Exit State in the list can be labelled as the Batch Default. This state is set when a batch or milestone with this profile has no children and therefore also has no defined Exit State. If a Batch Default has not been set, the state with the lowest priority preference is used which (if present) has not been set as being UNREACHABLE.

**Dependency Default** One or more final Exit States can be labelled as the Dependency Default. If a dependency between batches, jobs and milestones is created by selecting the DEFAULT state, the Exit States of a profile that are marked as being the Dependency Default fulfil the dependency condition regarding the Exit State.

**Preference**   The *Preference* buttons are used to set the preference of the individual Exit States. The rows are shifted up or down by clicking the buttons. A higher position indicates a higher preference.

To determine the resultant Exit State of a Submitted Entity, the Exit State with the highest preference is selected from the list of Exit States of the children and its own Exit State. In doing so, an Exit State is only taken into account if it is present in the Exit State Profile of the parent or if the state is translated into an Exit State of the parent using an Exit State Translation.

## Editor

Example: A batch has three subordinate child jobs. The batch is to display an error when at least one of its children returns an error. This means that the FAILURE state must precede the SUCCESS state.

The SUCCESS state will usually have the lowest priority, warnings have a higher priority, while Error states have the highest priority.

If an empty batch (one without any children) becomes FINAL, then there is no Exit State from which the FINAL Exit State of the batch can be determined. In this case, the Exit State from the Exit State Profile with the lowest preference is used. This is usually the SUCCESS state. The system attempts to use a FINAL state which is not the Unreachable State if one actually exists.

# 5 Exit State Translations

## 5.1 View

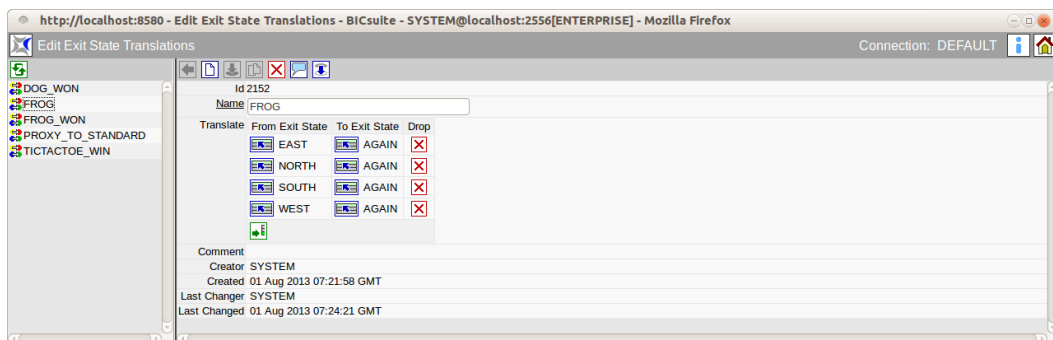


Figure 5.1: Exit State Translations

## 5.2 Concept

### 5.2.1 In short

Exit State Translations are required to translate the Exit State of a child into an Exit State of its parent.

### 5.2.2 Detailed description

Exit State Translations are used to translate a child's Exit States into Exit States of its parent. In the absence of a translation, the identity is used or the Exit State of the child is ignored by the parent. If the translation is available, all the Exit States have to be either implicitly or explicitly translated.

For example, if a child uses an Exit State Profile with the Exit States SUCCESS, WARNING, and FAILURE and the parent's Exit State Profile only knows SUCCESS and FAILURE, the final Exit States SUCCESS and WARNING can be translated into the parent's SUCCESS Exit State. The restartable FAILURE Exit State is translated into the parent's FAILURE Exit State.

## 5.3 Editor

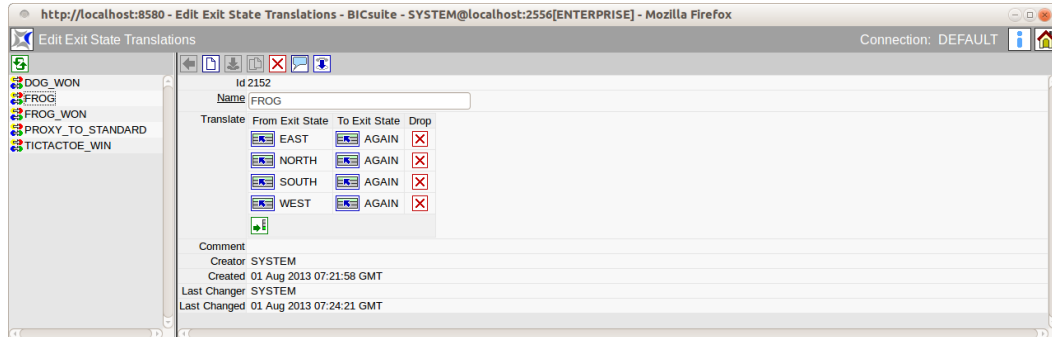


Figure 5.2: Exit State Translation Editor

Exit State Translations are defined and modified in this editor.

Exit State Translations can only be edited by users who belong to the "ADMIN" group or another group that has been granted the "manage exit state translation" privilege. All the input fields are "read only" for all other users.

The fields shown above have the following meanings:

**Id** The Id contains a system-wide unique number.

**Name** The *Name* field is used for assigning a unique name to an Exit State Translation.

**Translate From/To Exit State** Which state of a child is to be translated into which state of the parent is entered in each row.

# 6 Resource State Definitions

## 6.1 View

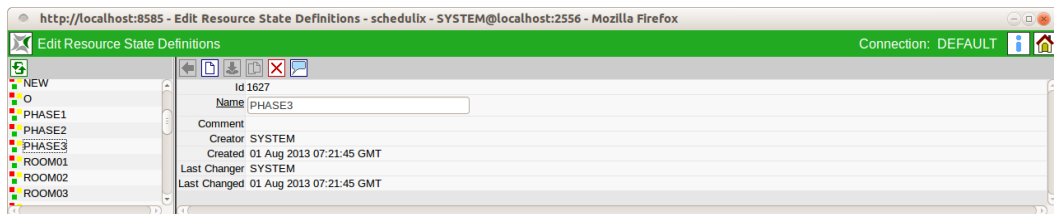


Figure 6.1: Resource State Definitions

## 6.2 Concept

### 6.2.1 In short

Resource State Definitions are used to create names for Resource States.

### 6.2.2 Detailed description

Resource State Definitions are logical names for Resource States. Each Synchronizing Resource can take on several states that can be set when a job has been completed.

## 6.3 Editor

The properties of Resource State Definitions are maintained in this tab.

Resource State Definitions can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

The fields have the following meanings:

**Id** System-wide unique number for identifying the object.

**Name** Unique name of the Resource State Definition. This name can be freely chosen.



# 7 Resource State Profiles

## 7.1 View

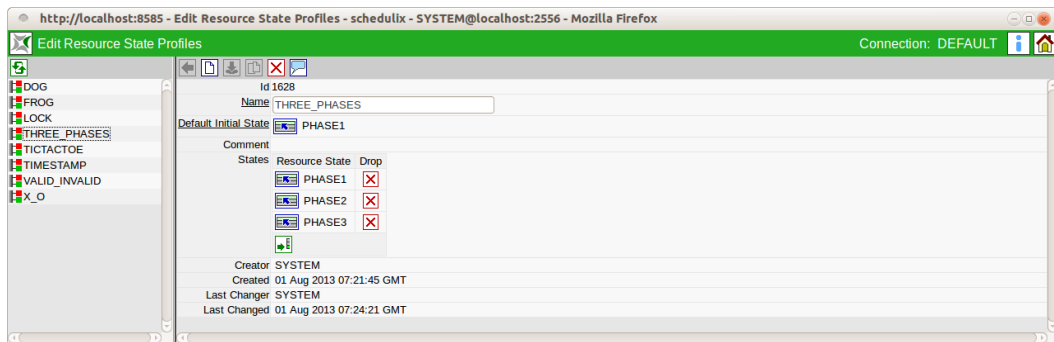


Figure 7.1: Resource State Profiles

## 7.2 Concept

### 7.2.1 In short

A Resource State Profile describes which states a resource can take on.

### 7.2.2 Detailed description

Each Synchronizing Resource can take on different Resource States. The set of all valid Resource States for a resource is described in a Resource State Profile. The initial state of a resource is also defined in this profile, whereby the state does not necessarily have to be present in the list of states.

## 7.3 Editor

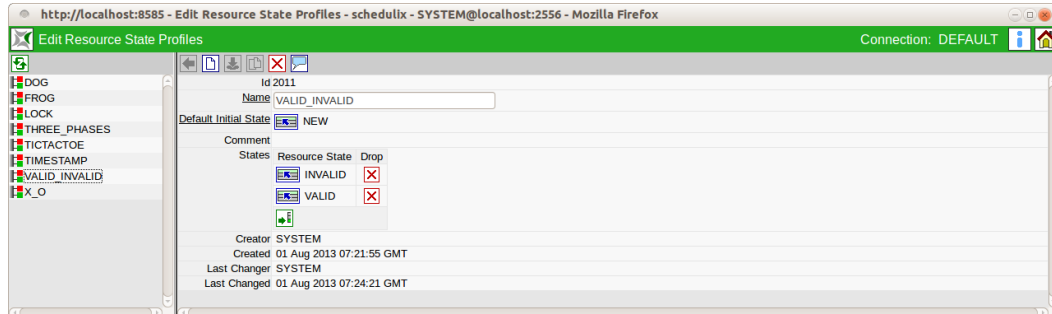


Figure 7.2: Resource State Profile Editor

The Resource States that belong to a Resource State Profile can be defined in this editor.

Resource State Profiles can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

The fields shown above have the following meanings:

**Name** The *Name* field is used for assigning a unique name to the Resource State Profile.

**Default Initial State** The *Default Initial State* field defines the initial state of the resource. This Resource State does not have to be present in the list of valid Resource States.

**Resource State** The valid Resource States are shown in the *Resource State* column in the *States* table.

### 7.3.1 Example

A database table for a data mart is shown in the scheduling system as a Synchronizing Resource with a State Model. The state chart looks like the diagram in Figure 7.3

The definition of a suitable Resource State Profile is shown in the screenshot above.



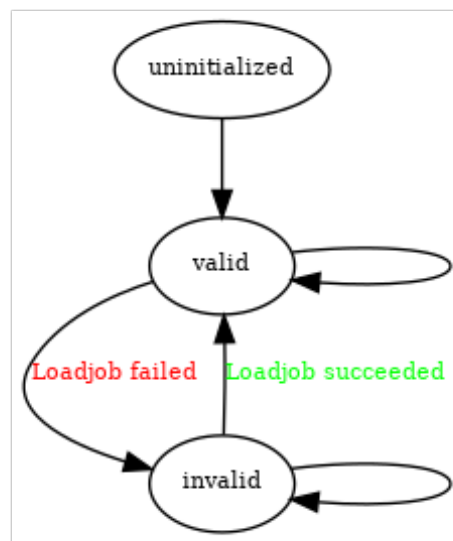


Figure 7.3: State chart for a resource



# 8 Resource State Mappings

## 8.1 View

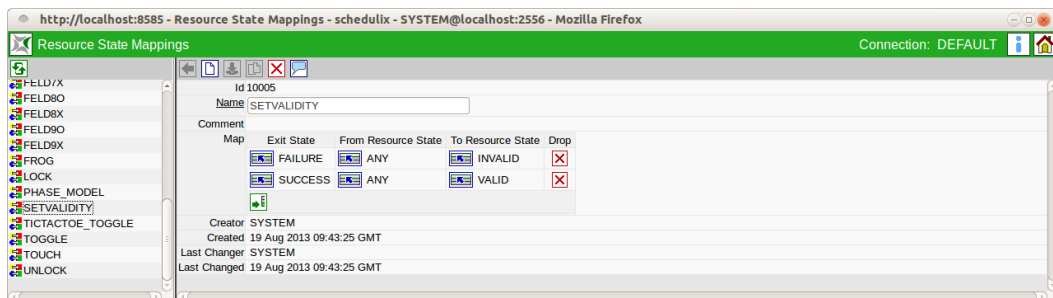


Figure 8.1: Resource State Mappings

## 8.2 Concept

### 8.2.1 In short

Resource State Mappings describe the state transition of a resource with certain Exit States.

### 8.2.2 Detailed description

Resources can change their Resource State after completion of a job depending upon the job's Exit State. Resource State Mappings describe these state transitions. Respectively one Exit State and one Resource State determine the new state of the resource.

For example, a "TABLE" resource can take on the state "VALID" or "INVALID". If the loading process belonging to the table is completed successfully (SUCCESS), the Resource State "VALID" is to be set. In the event of an error (FAILURE), the table is to be marked as being "INVALID". The corresponding definition of the Resource State Mapping would look like in Figure 8.2.

If no mapping exists for the combination of 'Exit State' and 'From Resource State', the status of the resource remains unchanged. This means that it is not necessary to define a mapping from SUCCESS/VALID to VALID.

## Editor

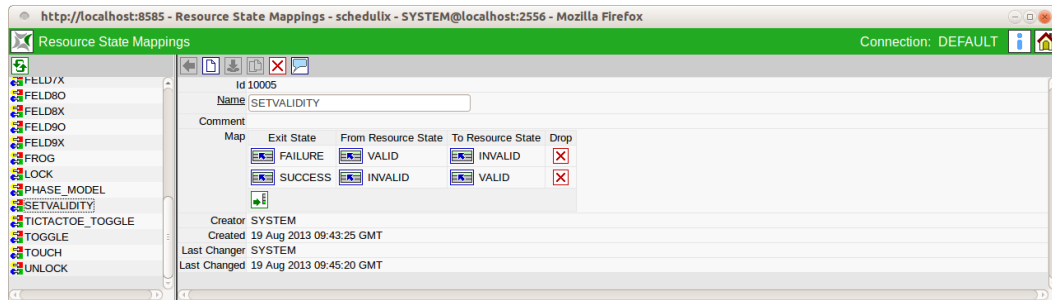


Figure 8.2: Example for a Resource State Mapping

If a state transition is to take place regardless of a Resource State, the value "ANY" needs to be selected in the line "From Resource State".

When implementing the status for a resource, the time of implementation is saved. Reference can be made to this time in the resource requirements, and so it may be practical to define the transition from "VALID" to "VALID".

## 8.3 Editor

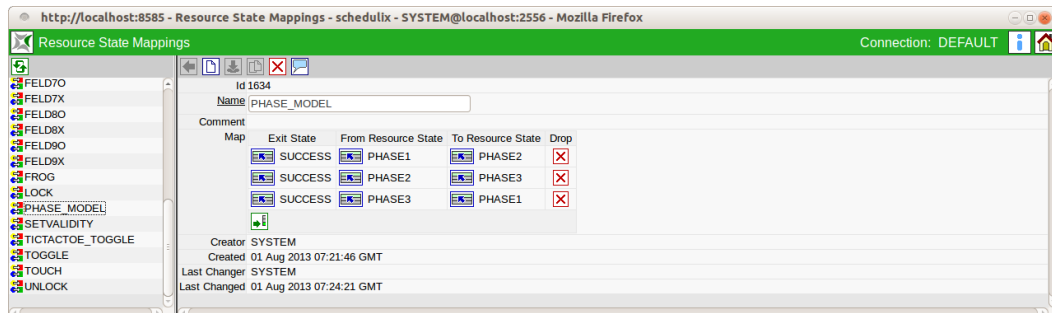


Figure 8.3: Resource State Mapping Editor

The Exit States are assigned to the resource state transitions in this editor. Resource State Mappings can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users. The fields have the following meanings:

**Id** System-wide unique number for identifying the object.

**Name** Unique name of the Resource State Mapping. This name can be freely chosen.

**Exit State** The Exit State of the job is given in the field *Exit State*.

**From Resource State** Either a specific Resource State or ANY is given in the field *From Resource State*. If the current state in combination with the Exit State is explicitly specified in the table, this rule is applied to determine the new state when implementing it. Otherwise, the ANY rule (if present) is used.

**To Resource State** The field *To Resource State* is the resultant state of the resource.



# 9 Named Resources

## 9.1 View

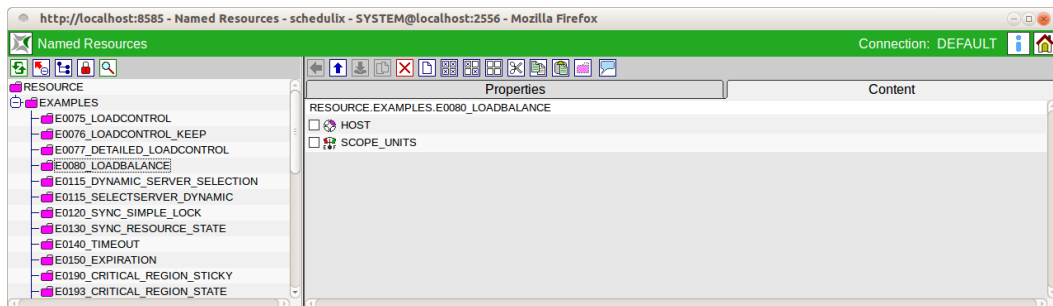


Figure 9.1: Named Resources

## 9.2 Concept

### 9.2.1 In short

This window is used to create and administer Named Resources. A Named Resource is the definition of a class of resources.

### 9.2.2 Detailed description

A Named Resource defines a class of resources. Named Resources can be instantiated as resources within a scope, folder or in an Submitted Entity.

Named Resources belong to a group. The right to use (i.e. instantiation and requirement), edit or delete the Named Resource is primarily reserved for users belonging to that group. These privileges can be granted to other groups.

Named Resources are saved in a hierarchy of categories. Categories can be nested as required, but Named Resources cannot be nested. This categorisation serves solely to facilitate the administration of Named Resources, and otherwise has no influence on how the system functions.

## 9.3 Editor

Named Resources and categories are maintained in the Editor menu. When an entry is selected in the Navigator, the details for this Named Resource or category are displayed here. New Named Resources and categories are created here as well.

### 9.3.1 Properties tab for categories

This tab is used for maintaining the category properties. It looks like this:

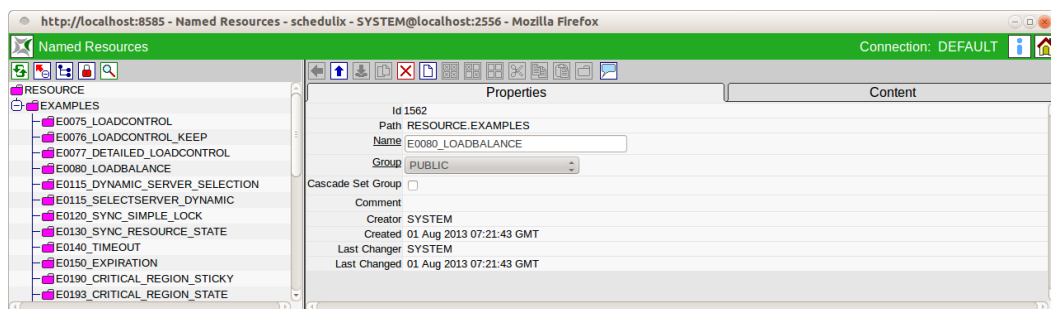


Figure 9.2: Categories; Properties tab

The "Properties" tab for a category and Named Resource has the following fields:

**ID** The ID is used to unequivocally identify the category or Named Resource Definition. It is automatically assigned by the system and is unique system-wide.

**Path** The path is the entire hierarchy of the viewed entry. The single hierarchy levels are separated by a period.

**Name** Any name can be chosen for the category or Named Resource. It must, however, be unique within the parent category.

**Group** The group to which the category is assigned can be selected from the "drop-down" list.

**Cascade Set Group** If this box is checked, the group is also set in all the Named Resources and categories below this category.

**Comment** The *Comment* gives a more detailed explanation about the object.



### 9.3.2 Content tab

The "Content" tab is only displayed for categories and looks like this:

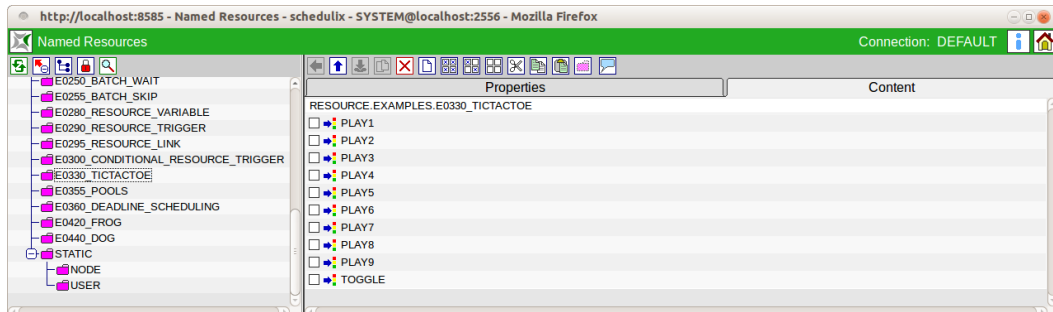


Figure 9.3: Categories; Content tab

This tab contains a list of all the Named Resources in the selected category. The following values are shown in the list:

**Name of the Named Resource** The name of the Named Resource is shown here. Selecting the Named Resource's name in this dialog opens the selection window and display of the "Details" data for this Named Resource. In this tab, you can move resources using the standard cut, copy and paste operations.

### 9.3.3 Properties tab for Named Resource Definitions

The "Properties" tab for a Named Resource Definition looks like this:

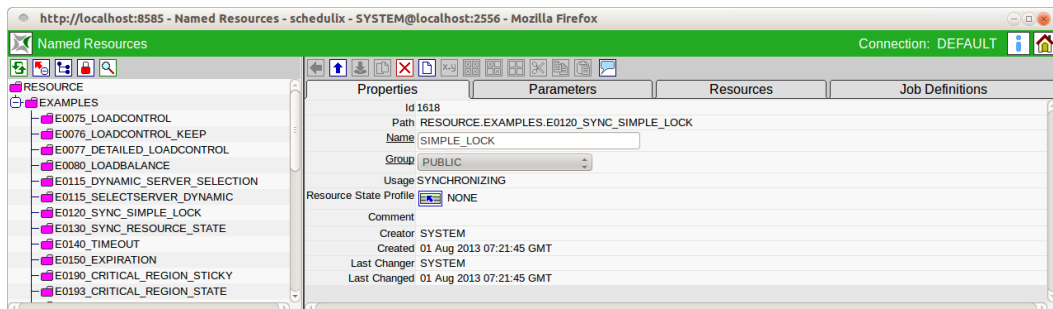


Figure 9.4: Named Resources; Properties tab

When you define a Named Resource, you have to fill in some other fields in addition to those mentioned above.

**Usage** The *Usage* field specifies the Named Resource type. List of options for the usage parameters:

1. *Category*

Categories behave like folders and can be used to arrange the Named Resources in a clearly organised hierarchy.

2. *Static*

Static resources describe workflow environments such as those offered by installed software packages or user environments. An instance of this Named Resource must exist and be available in order to be able to start a Submitted Entity that conditionally requires this resource.

3. *System*

The Named Resource is a resource that maps a system parameter. This can be, for example, a CPU unit, main memory unit or a unit of disk space. This resource also has to be quantified when it is used, i.e. the creator of a job needs to qualify and quantify the requirements that have to be met by the system resources for this object. When defining a job he will have to specify, for example, that 3 CPU units, 3 units of 512 MB main memory and 10 units of 1 GB disk space are required.

4. *Synchronizing*

Synchronizing Resources are special resources that are used to synchronise concurrently running jobs. Whether a resource requirement can be fulfilled and the resource can be allocated can be made dependable upon the current status of the resource, the last status update time, and the contending allocation of the resource by other tasks. The requirement can be quantified analogue to System Resources.

**Resource State Profile** A Resource State Profile that defines the status that the Synchronizing Resource can take on can be assigned to that Synchronizing Resource. No status or timestamp-related requirements can be defined if a Resource State Profile has not been specified. Further information can be found in Chapters [6](#) and [7](#).

### 9.3.4 Parameters tab

Additional information about a resource that can be evaluated by jobs or Resource Triggers can be saved in the "Parameters" tab.

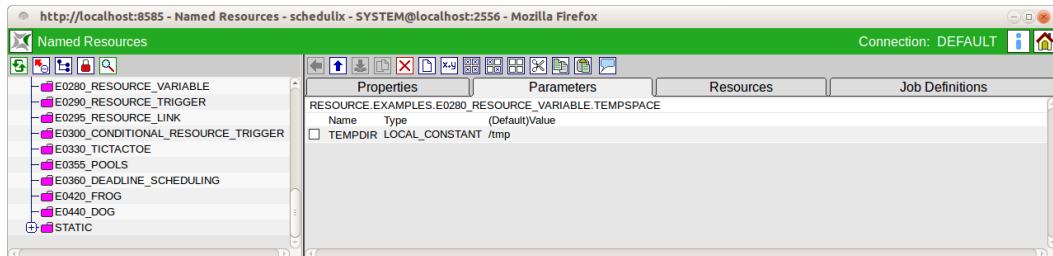


Figure 9.5: Named Resources; Parameters tab

The "Parameters" tab looks like in Figure 9.5.

The "Parameter Details" tab is opened by clicking the parameter name.

#### 9.3.4.1 Parameter Details tab

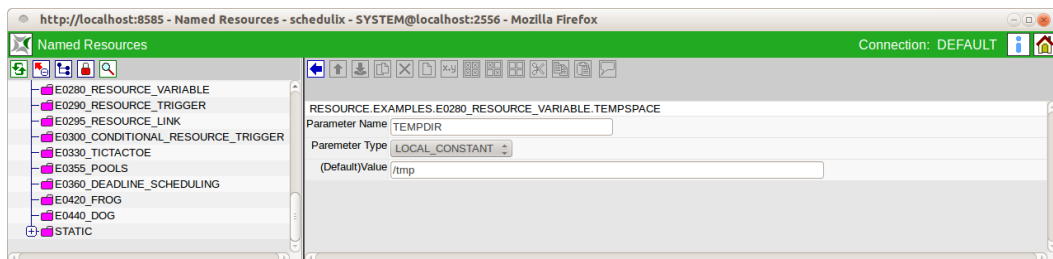


Figure 9.6: Named Resources; Parameter Details

The fields in the "Parameter Details" tab have the following meanings:

**Parameter Name** Name of the parameter.

**Parameter Type** Type of the parameter.

The following options are available:

- Constant: The Constant has a fixed value and applies for all resources.
- Local Constant: The Local Constant has a fixed value which may vary from resource to resource.

**Default Value** With the Default Value, we differentiate between Constants and Local Constants. It is the value of the parameter for Constants and the default value for Local Constants.

### 9.3.4.2 Standard parameters

Just as there are standard parameters for jobs and batches, standard parameters are also available for resources.

To be able to reference them, a slight trick has to be used analogue to the situation with batches and jobs. If you want to use the contents of the standard variable "STATE" for a processing operation, for instance, you have to create another variable (a constant) with the value "\$STATE". Due to the recursive parameter resolution, the constant is given the value of the parameter "STATE".

The following standard parameters are available:

**STATE** The state of the resource. This content is empty for non-Synchronizing Resources.

**AMOUNT** The total available amount of the resource. This value is empty for static resources.

**FREE\_AMOUNT** The available free amount of the resource. This value is empty for static resources.

**REQUESTABLE\_AMOUNT** The maximum amount of the resource that can be requested. This value is empty for static resources.

### 9.3.5 Resources tab

When you select a Named Resource in the navigation, its instances are displayed in the "Resources" tab. All the scopes, folders, Submitted Entities, Scheduling Entities or jobserverns that offer instances of this Named Resource are shown here.

The "Resources" tab looks like this:

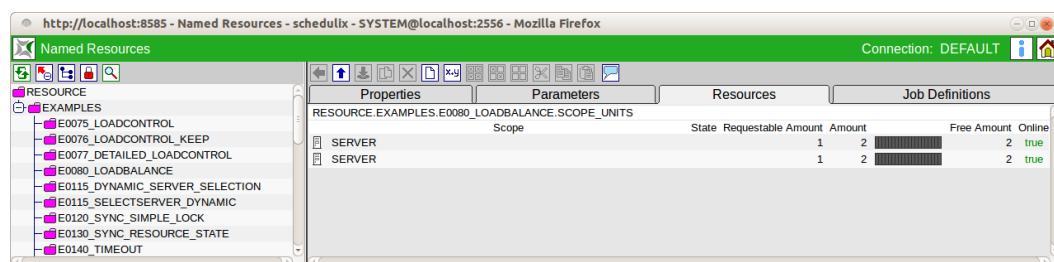


Figure 9.7: Named Resources; instances

The instantiations of the Named Resource are described by the following fields:

**Scope** The names of the scopes, Submitted Entities, Scheduling Entities or folders that offer instances (resources) of the respective Named Resource are shown here.

**State** If the selected Named Resource has a [Resource State Profile](#), the current state of the resource is displayed here. This is only possible with Synchronizing Resources.

**Requestable Amount** The maximum amount of resources that can be requested by a job.

**Free Amount** The number of free instances of this resource is shown here. A bar graphically indicates the current allocation rate.

**Amount** The amount is the number of available units of the resource.

**Online** The availability status of the resource.

### 9.3.6 Job Definitions tab

The “Job Definitions” tab shows all the Job Definitions that request this Named Resource. This tab is for information purposes only; no changes can be made here. The tab looks like this:

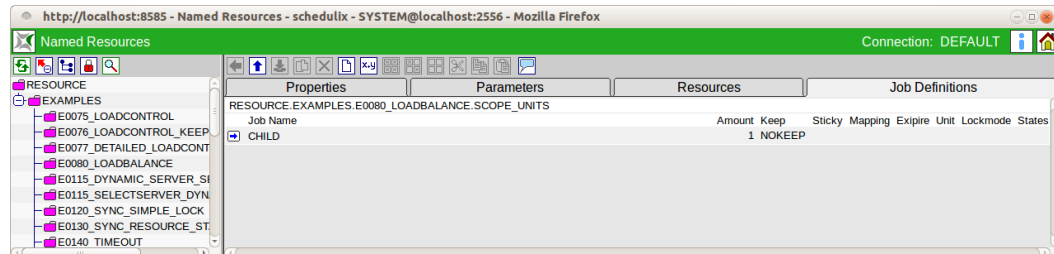


Figure 9.8: Named Resources; Job Definitions tab

The list of jobs is described in the following fields:

**Job Name** The names of the Submitted Entities that require the Named Resource are shown here.

Clicking the name opens a [Job Editor window](#).

**Amount** The amount of the resource that is required by the job.

**Keep** The value of the Keep parameter for the resource request from the job.

**Sticky** The value of the Sticky Flag for the resource request from the job.

**Mapping** If a Resource State Mapping was specified in the resource request, it is displayed here.

**Expire** The Expire value specifies the maximum length of time that may elapse since the last state transition of the resource for it to be regarded as being occupiable. A negative Expire value means that a resource must be at least as “old” as given here.

**Lock mode** The Lock mode describes the mode for accessing this resource (exclusive, shared, etc.).

**States** All the Resource States that are accepted by this job are shown here. Multiple states that are acceptable for this job are separated by commas.

## 9.4 Named Resource selector

This navigator is used for selecting single or multiple Named Resource Definitions and is called by other dialogs like [Footprint](#) as a search and selection mask. Which resources are displayed varies depending upon the call dialog. If the selector is called from the Footprint editor, only the system resources are displayed. If the selector is called from the Environment dialog, only static resources are displayed.

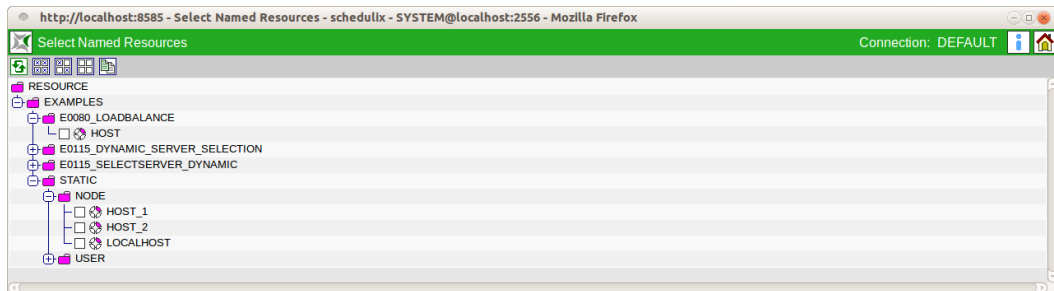


Figure 9.9: Named Resources; selector





# 10 Environments

## 10.1 View

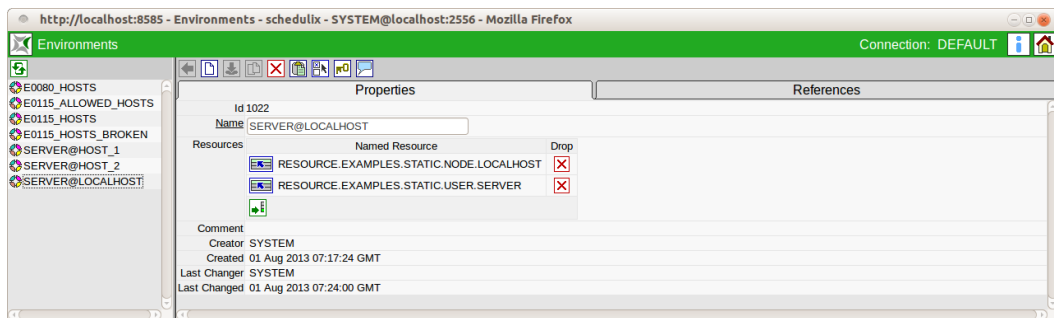


Figure 10.1: Environments

## 10.2 Concept

### 10.2.1 In short

The "Environment" dialog is used for administering environment definitions. An environment collates multiple requirements for static resources under one name.

### 10.2.2 Detailed description

Environments serve two purposes. On the one hand, they simplify the maintenance of resource requirements since they no longer have to be individually added in the Job Definition. All you have to do is to specify the respective environment. On the other hand, environments are used to determine which users or user groups are permitted to use specific runtime environments.

The Environment Definition therefore states which runtime environment is required by a job. For example, which host is to run a job or which users and programs must be available can be controlled with an environment.

The environment is a mandatory parameter in a Job Definition.

Environments can also be entered as a Folder Environment for a folder. This means that all the Job Definitions below such a folder "inherit" the resource requirements from the Folder Environment.

## Navigator

Where separate jobserver are in use for Development and Production, for example, an environment can be used for the job which allows it to be run on both jobserver. Creating two folders for Development and Testing (to which a "Development" and "Production" environment are respectively assigned whose resource requirements can only be fulfilled by development or production jobserver) allows the Development and Production environments to be easily kept separate. If such a job is located somewhere below the "Development" folder, the combination of resource requirements for Job Environment and Folder Environment ensures that the correct Development jobserver is selected. If the job is moved or copied to the "Production" folder and then executed, the Production jobserver is selected automatically. The Job Definition does not have to be modified.

### 10.3 Navigator

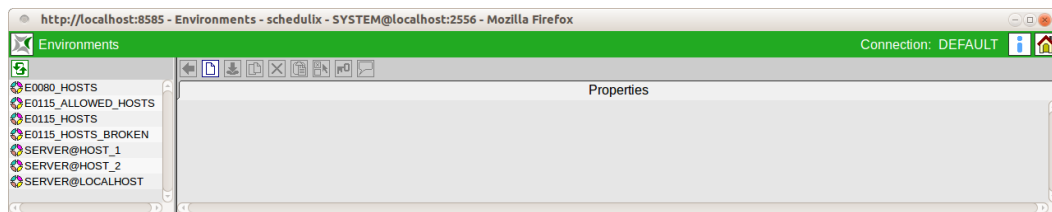


Figure 10.2: Environment navigator

The "Environment" dialog navigation window shows all the existing environments. If the logged in user is a member of the "ADMIN" group, all the environments are displayed. If this is not the case, then only those environments that can be used by the user who is logged in are shown. This means that the logged-in user must be a member of a group that holds the "use" privilege for that particular environment.

## 10.4 Editor

### 10.4.1 Properties tab

This tab is used for maintaining the environment properties. Environments can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

It looks like this:

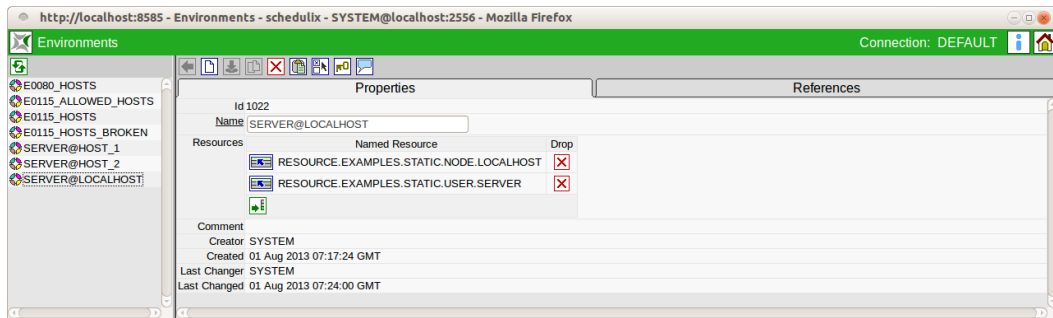


Figure 10.3: Environment properties

The "Properties" tab for environments has the following fields:

**ID** The ID is assigned automatically and is used for unequivocal, system-wide identification of the object.

**Name** The name of the environment. Any name can be chosen, although it must be unique within the environment.

The "Resources" list shows all the resources that belong to the environment. These are exclusively static resources. The *Named Resource* fields have the following meanings:

**Named Resource** The name of the Named Resource is shown here.

**Condition** A condition is entered in the *Condition* field that must be fulfilled to ensure that the resource is recognised as being valid. The condition is evaluated in the context of the requesting job.

**Drop** The *Drop* button can be used to delete a specific row.

## 10.4.2 References tab

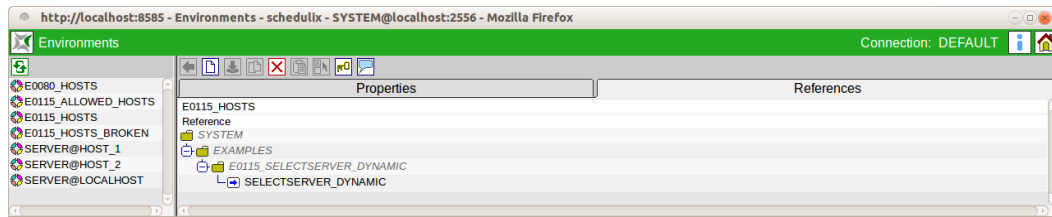


Figure 10.4: Environment references

This tab displays any references to the environment.

The "References" tab shows a tree view that only contains the parent folder of objects that reference the environment.

Folder names are usually shown in italics unless the folder references the environment as a Folder Environment.

Clicking the name of a non-recursively displayed folder or a job opens an editor window for the folder or job.

# 11 Footprints

## 11.1 View

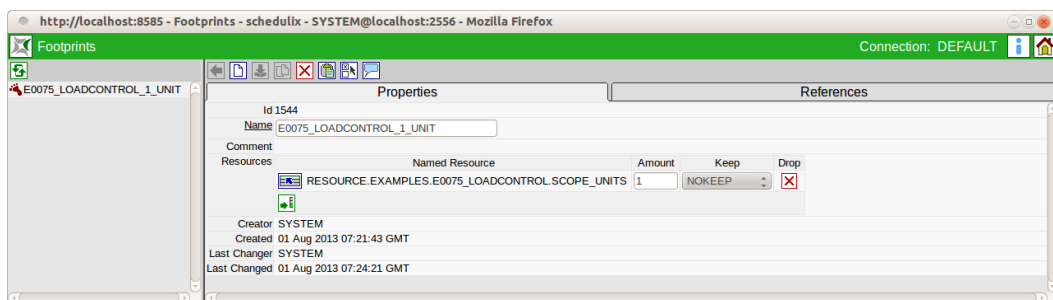


Figure 11.1: Footprints

## 11.2 Concept

### 11.2.1 In short

This dialog is used for creating a footprint definition. A footprint collates a group of System Resource requirements under one name.

### 11.2.2 Detailed description

Collating the resource requirements for footprints makes it easier to maintain the requirements for jobs in the runtime environment because they do not have to be individually specified.

The footprint can be overridden for each job. This is done by using an explicit requirement for a resource (contained in the footprint) with a deviating amount or by changing the setting for the [Keep parameter](#). A resource requirement cannot be removed, i.e. all the resources defined in the footprint are requested. The amount can be reduced to 0, although resource must still be present.

Example:

A System Resource CPU\_UNIT with the amount 4 (CPU units) is defined in the footprint. The footprint can be overridden by adding a requirement for the same System Resource (CPU\_UNIT) within the Job Definition ([Resources tab](#)) for an

amount of 2 (CPU units). It is not possible to completely remove the resource CPU\_UNIT, however.

## 11.3 Editor

### 11.3.1 Properties tab

This tab is used for maintaining the footprint properties.

Footprints can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

It looks like this:

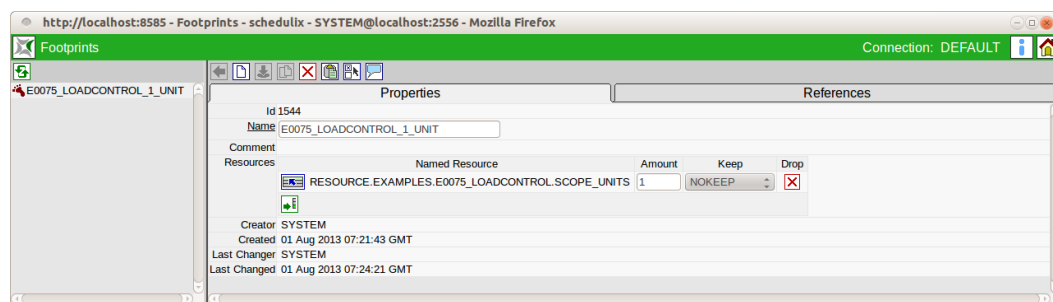


Figure 11.2: Footprint properties

The Properties tab for footprints has the following fields:

**ID** The ID is assigned automatically and is used for unequivocal, system-wide identification of the footprint.

**Name** The name of the footprint. This can be freely chosen. A list of the resources contained in the footprints can be entered in the "Resources" list.

The fields have the following meanings:

**Named Resource** The name of the Named Resource is shown here. It is selected with the *Chooser* button.

**Amount** This is the amount of the resource required by a job.

**Keep** The Keep parameter determines whether the Named Resource is retained after a job has been completed or if it can be released again. The following selection options are available:

## Editor

### 1. *No\_Keep*

The Named Resource is released after completion of the job. Whether the job has been completed successfully or aborted due to an error is irrelevant as far as the release is concerned.

### 2. *Keep*

The resource is not released until the job has attained a final state. The resource is retained in the event of an error (Restartable State).

### 3. *Keep\_Final*

The resource is not released until the job and all its children have attained a final state.

## 11.3.2 References tab

This tab displays the references to the footprint. It looks like this:

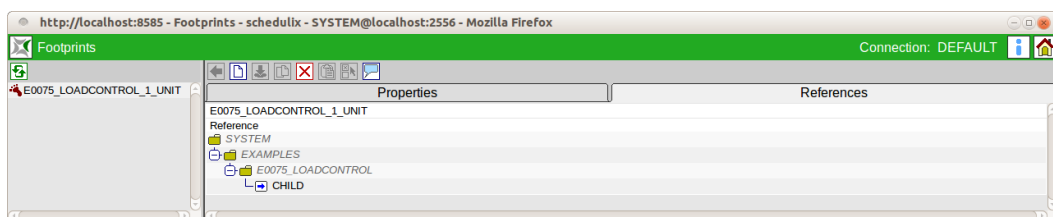


Figure 11.3: Footprint references

The References tab shows a tree view that only contains the folder below which there are jobs that reference the footprint.

Folders are displayed in italics.

Clicking the name of a job opens a window where it can be edited.





# 12 Jobserver and resources

## 12.1 View

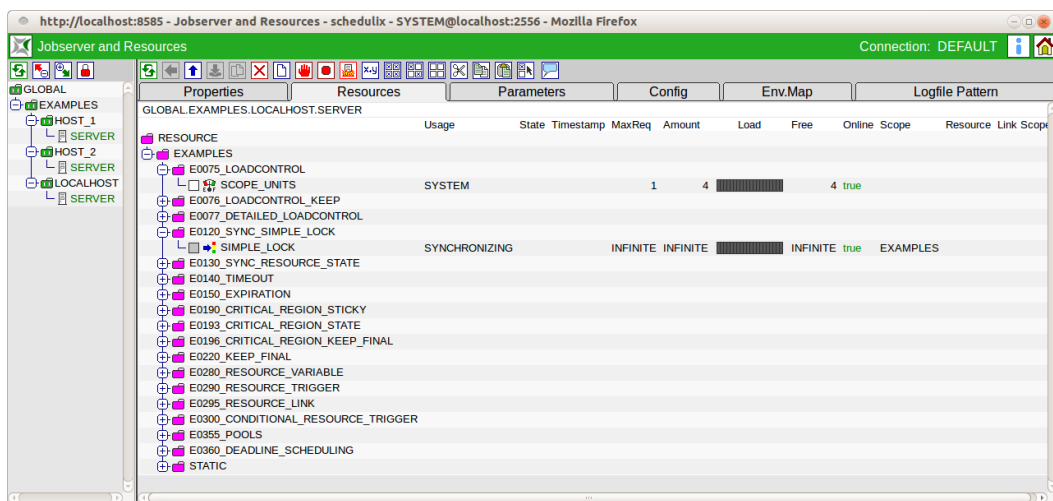


Figure 12.1: Jobserver and resources

## 12.2 Concept

### 12.2.1 In short

A jobserver is a schedulix system process that has to run on all the computers that have been defined as executive units of the schedulix Scheduling System. If a jobserver is assigned to perform a task by the schedulix Server, the jobserver executes the script or program specified in the job definition in the required environment and with the necessary parameters. The jobserver monitors the program and sends the return state of the script or program back to the schedulix Server.

### 12.2.2 Detailed description

The physical distribution (i.e. which jobserver run on which computers and which resources are made available by these jobserver) can be defined and maintained in the "Job Servers and Resources" dialog.

## Concept

These jobserver must be set up and started at system level.

A differentiation is made between two types of objects in this dialog. On the one hand there are so-called scopes, which can serve as a container for the respective jobserver or child scopes.

On the other hand, those jobservers that reflect the executive system processes are administered in this dialog.

The hierarchy can be subdivided as required and is largely dependent upon the complexity of the system environment and administration structure (e.g. in the data centre).

For example, it is possible to define each physical host as a scope, and all the workflow environments required on this computer can be defined as jobservers below this scope. If the system landscape is more complex, it is practical to use multiple levels for departments, system architectures (Unix, Windows) or similar entities.

The scopes and jobservers can be assigned to resources at every level of this hierarchy. These resources can then be used by all the children jobservers.

### **Example:**

A value of 5 is entered for the System Resource CPU\_UNITS at host system level. If 2 jobservers have been configured on the computer, a total of 5 CPU units can be shared between them. If a jobserver occupies 3 CPU units when executing a job, only 2 CPU units remain available to the other jobserver. However, this behaviour is not additive. If the resource CPU\_UNITS is also created on one of the aforementioned jobservers (for example with a value of 3), only these 3 CPU\_UNITS will be available to that jobserver. Accordingly, 5 CPU\_UNITS from the parent scope will be available to the other jobserver.

## 12.3 Navigator

The Navigator provides a hierarchical structural view in which scopes are defined as folders. The jobserver are located within the scopes.

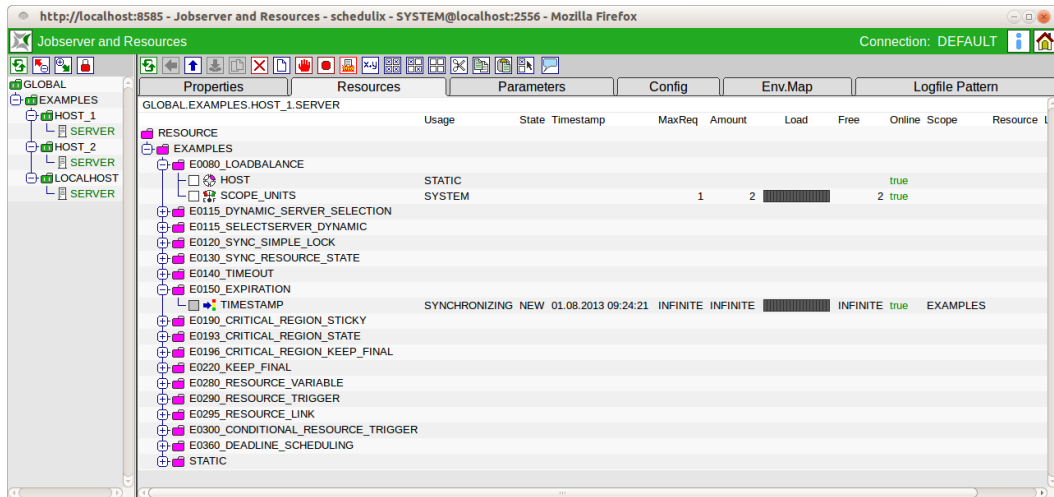


Figure 12.2: Jobserver and Scope Navigator

## 12.4 Editor

### 12.4.1 Properties tab

The "Properties" tab is used for maintaining the properties of scopes and jobserver.

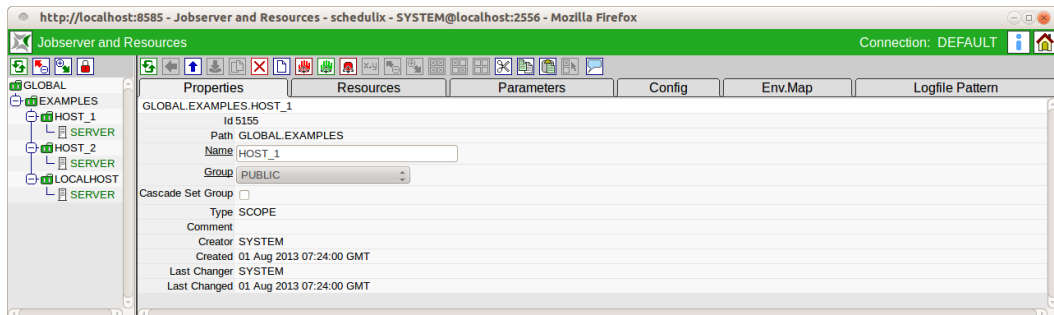


Figure 12.3: Scope properties

"Properties" tab for scopes is shown in Figure 12.3.

"Properties" tab for jobserver is shown in Figure 12.4.

## Editor

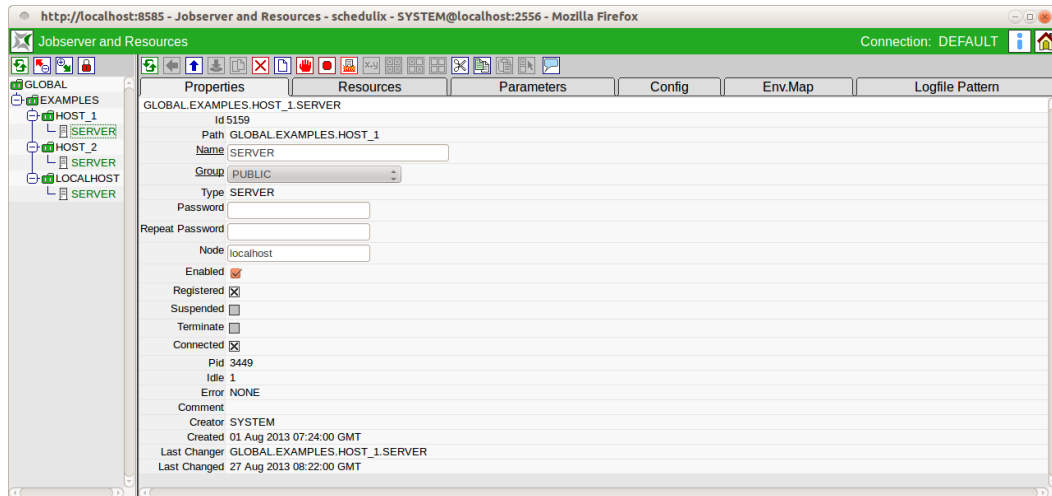


Figure 12.4: Jobserver properties

The fields in the "Properties" tab have the following meanings:

**ID** The ID is used for unequivocal, system-wide identification of the object.

**Path** This is the complete path to the object within the hierarchy.

**Name** This is the name of the scope or the jobserver. The name can be freely chosen, but it must be unique within the hierarchy level.

**Group** The group can be selected from a drop-down list. It designates the scope's owner group.

**Type** The *Type* field shows the type of the selected object. The type is selected when creating a new server or scope and cannot subsequently be changed.

### OPTIONS FOR TYPE

#### 1. *Scope*

A scope is a category that can be grouped hierarchically organised below the jobserver. This could be a host computer, for example, or, at an even higher level, all the computers in a department or all the Windows-based PCs. The structure and depth of the hierarchy can be modified depending upon the system landscape and should reflect this.

## 2. Server

This object is a physical jobserver. The physical process has to log in to the schedulix system under the selected name and in doing so set up the connection to this jobserver definition.

**Password** The *password* used by the jobserver to log on to the schedulix Server. Each jobserver requires a name and password to log on to the schedulix Server. A job can only be handed over to the jobserver to be executed with a valid login. The password is hidden on the screen and must be identical to the field *Repeat password*.

Since the password is not displayed in the dialog, it is not clear whether a password has already been entered or not. The password does not have to be specified again when changing other properties. If the jobserver is logged in, it is automatically notified about any password changes.

**Repeat Password** The *Repeat Password* must be the same as the password you entered in the *Password* field. It is necessary to enter it twice to detect any accidental typing mistakes.

**Node** The *Node* specifies the computer on which the jobserver is running. This field has a purely documentary character.

**Enabled** If the jobserver has set the Enabled flag, the jobserver process can log onto the schedulix Server. The login will fail if the flag has not been set.

**Suspended** If the Suspended flag has been set, although the jobserver process can log on it will not be given any jobs to run. State changes for previously allocated jobs are accepted by the schedulix Server.

**Terminate** The Terminate flag indicates whether the jobserver is to terminate or not. If this flag has been set, the jobserver will receive this message at the next communication step and behave accordingly.

It may take a few seconds before the jobserver terminates. The termination can be verified by clicking the [Refresh button](#). Once the jobserver has finished, the value displayed in the *Connected* field changes to "FALSE".

**Connected** The *Connected* field shows the connection status of an external jobserver process. If the jobserver process was able to successfully log on to the schedulix system, this value is set to "TRUE".

**PID** The PID is the process identification number of the jobserver process on the respective host system. It is communicated to the schedulix system during the registration.

The PID and the *Node* field allow the process to be located in the operating system running on the respective server.

**Error** An error message is displayed here if an error occurred while the jobserver process was running.

### 12.4.2 Resources tab

The "Resources" tab looks like this:

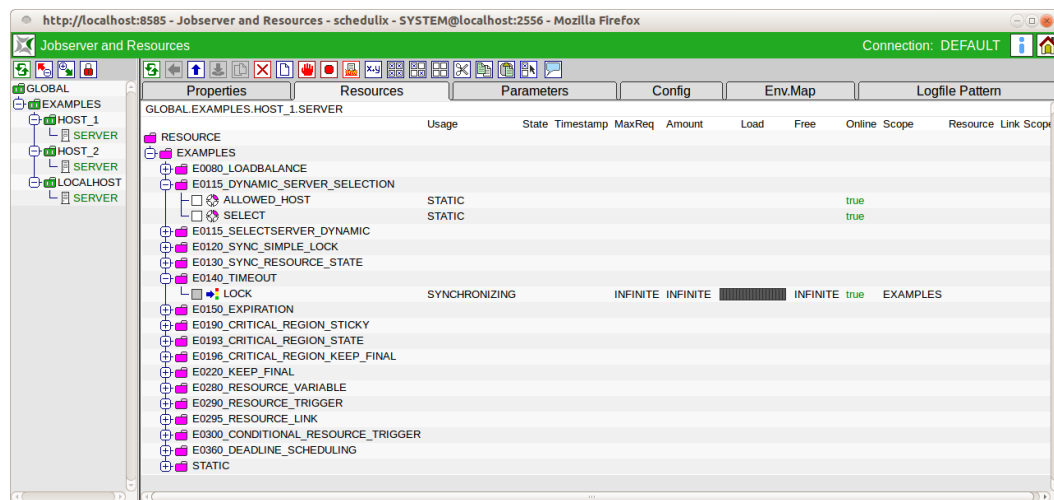


Figure 12.5: Jobserver and scope resources

All the resources offered by the current scope or jobserver are shown in the Resources tab. If any resources (especially of the type [Synchronizing](#)) are to be awarded system-wide, this can be done in the "GLOBAL" scope.

If multiple instances of a resources are entered in different scopes or jobserver, these are respectively a separate instance of that resource.

**Example:**

If a Synchronizing Resource A is entered in Scope X and Scope Y, these two instances of Resource A are mutually independent (in X and Y). This means that a possible synchronisation will only take place among all the jobs in Scope X and Scope Y respectively. If a global synchronisation is to take place here, Resource A must be defined in the GLOBAL scope instead.

The resources are displayed as a map of the hierarchy from the navigation in the [Named Resources](#) dialog instead together with additional details. The first field in

the list shows the name of the Named Resource or category. If this is a category, a folder icon stands here as well. The category can be opened and closed by clicking this icon. If it is a Named Resource, clicking the name opens the "Resource Details" tab.

The other fields are explained in the next section.

### 12.4.2.1 Resource Details tab

The "Resource Details" tab is displayed as soon as a resource is selected. The instance information for this Named Resource is entered and maintained in this tab. Here, an instance is the expression of the Named Resource in the selected scope or jobserver. All instance parameters can be entered and maintained here.

The "Resource Details" tab looks like this:

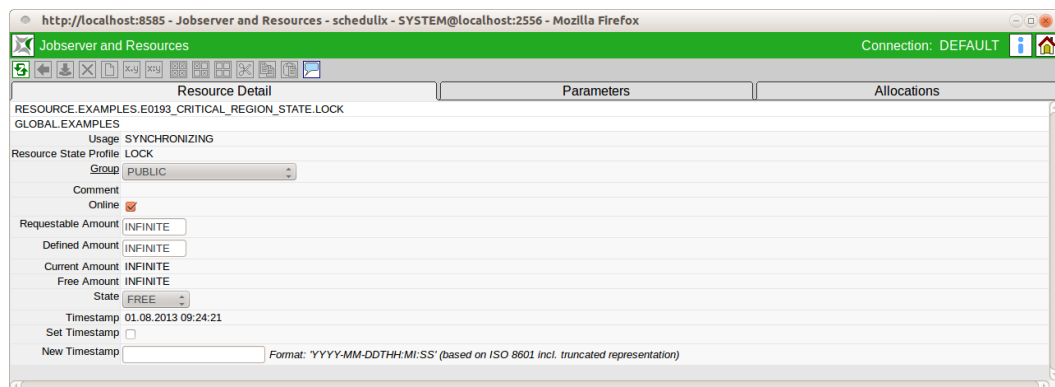


Figure 12.6: Resource Details

The fields in the "Resource Details" tab have the following meanings:

**Usage** The *Usage* field specifies the "Resource" type. More details about resource types can be found in the [Named Resource](#) dialog.

**Resource State Profile** This field is only visible if the resource usage has been set to "Synchronizing".

This is the [Resource State Profile](#) assigned to the resource.

**Online** *Online* can be used to switch a resource in this scope or jobserver online or offline. An offline resource is temporarily unavailable. Example:  
A scheduled computer or database downtime is pending. If the computer or database has been mapped as a resource in the system, the resource can be set as being offline as soon as it is physically no longer available. All jobs that require this resource are automatically no longer executed. If the resource is to be defined in another scope

as well (as a separate instance), all the jobs that would require the resource would deviate to this scope or jobserver.

An offline resource does *not* trigger the error message "Job cannot run in any scope because of resource shortage".

**Requestable Amount** The *Requestable amount* is the maximum amount that can be requested.

**Defined Amount** This field is only visible if the resource usage has been set to "Synchronizing" or "System".

The *Defined Amount* states the current number of instances of the Named Resource for this scope or jobserver. If the number of instances for this scope changes, you can edit this field accordingly.

Example:

This may become necessary if the hardware has been modified. If the resource represents a CPU unit and 2 CPU units are installed in the current host, the amount is 2. If 2 more CPU units are then installed, this value must be increased to 4.

**Current Amount** The *Current Amount* and *Defined Amount* are identical.

**Free Amount** This field is only visible if the resource usage has been set to "Synchronizing" or "System".

The *Free Amount* designates the total number of instances of a resource in the selected scope or jobserver that have not yet been allocated to jobs.

**State** The *State* field is only visible if the resource usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned. It designates the current status of the resource in this scope or jobserver. The state can be set or changed with this value. The drop-down list contains all the valid Resource States of the Resource State Profile, which can be selected here. This may be necessary when manually fixing an error to restore a resource to a state that allows further processing by follow-on jobs.

If you manually change this value, the value in the *State* field in the "Resources" tab is not automatically refreshed. Click the *Refresh* button if this is required and necessary.

**Timestamp** This field is only visible if the resource usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned. The *Timestamp* indicates the time of the last change made to a Resource State.

Timestamps play a role if an expiration interval has been defined in an expiry object. If this is the case, the timestamp must not be older than the specified interval. More details about expiration times can be found in [Chapter 13.5.8.1](#).



## Editor

If the completion of a job causes the Resource State to change, the timestamp is automatically updated by the schedulix Server.

**Set Timestamp** This field is only visible if the resource usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned. The *Set Timestamp* flag is a so-called action flag. If you set this flag and then click the *Record* button, the time shown in the *Timestamp* field is set to the current time.

Example:

This may be necessary when manual intervention is required to fix an error and the state has been changed manually in the *State* field .

### 12.4.2.2 Parameters tab

Additional information about a resource that can be evaluated by jobs or Resource Triggers can be saved in the Parameters tab. The parameters are defined in the Named Resources.

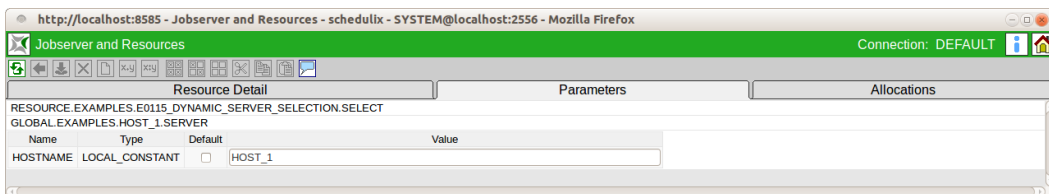


Figure 12.7: Resource parameters

The columns in the list shown above have the following meanings:

**Name** The "Name" column shows the name of the parameter.

**Type** This is the parameter type. The following options are available:

- **Constant:** The constant has a fixed value and applies for all resources.
- **Local Constant:** The Local Constant has a fixed value which may vary from resource to resource.

The values of the Local Constants can be changed here.

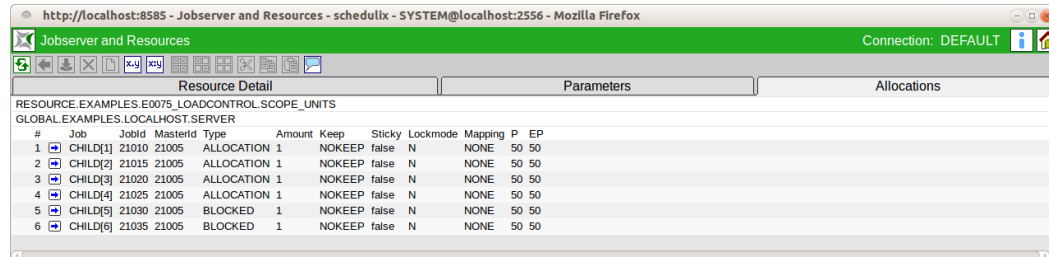
**Default** The *Default* field indicates whether the displayed value is the default value. The flag has to be set to restore the value to the default value.

**Value** This is the value of the parameter.

### 12.4.2.3 Allocations tab

The "Allocations" tab provides information about which tasks have been currently allocated the resource and which tasks require resources but cannot obtain them because they have the incorrect state or due to a lock.

The Sheet tab looks like this:



#	Job	JobId	MasterId	Type	Amount	Keep	Sticky	Lockmode	Mapping	P	EP
1	CHILD[1]	21010	21005	ALLOCATION	1	NOKEEP	false	N	NONE	50	50
2	CHILD[2]	21015	21005	ALLOCATION	1	NOKEEP	false	N	NONE	50	50
3	CHILD[3]	21020	21005	ALLOCATION	1	NOKEEP	false	N	NONE	50	50
4	CHILD[4]	21025	21005	ALLOCATION	1	NOKEEP	false	N	NONE	50	50
5	CHILD[5]	21030	21005	BLOCKED	1	NOKEEP	false	N	NONE	50	50
6	CHILD[6]	21035	21005	BLOCKED	1	NOKEEP	false	N	NONE	50	50

Figure 12.8: Resource allocations

The columns in the list shown above have the following meanings:

**Job** Depending on the setting, either the name of the job or the full path of the folder hierarchy for this job is shown here.

**Job Id** This is the ID of the job instance that was started with either a direct submit in the [Submit Jobs](#) dialog or a submit of the Master Batch or job.

**Master Id** This is the ID of the job instance or batch that was started as a Master Job in the [Submit Jobs](#) dialog and contains the current job as a child. If the job is itself a Master Job, then its own Job Id is shown here.

**Type** The *Type* refers to the method used by the current job instance to access or attempt to access the resource. The following options are available:

1. Requested
 

The job asks the server if the resource is available.
2. Reserved
 

The resource is reserved by the job. This has the same effect as Allocated, except that reserved resources can be released again when the job is started. Allocated resources are released at the earliest when the job has finished.
3. Allocated
 

"Allocated" indicates that the job is currently running and accessing the resource. Other jobs that want to access this resource can, depending upon the type of access lock, be allocated this resource as well or be blocked.

4. Blocked

If a job is labelled as being "Blocked", it cannot be run at the moment because it is waiting for access to the selected resource but is not able to obtain it due to a non-matching lock mode or other criteria.

The reason why the resource is not available is indicated by the red lettering. As soon as the resource becomes available again, the type changes from "Blocked" to "Requested".

5. Available

The resource is available.

6. Ignored

The user has told the system to ignore the resource request.

7. Master Reservation

A Master Reservation is a reservation of Sticky Resources.

**Amount** The *Amount* is the number of resource instances allocated to or requested by the current job. If the value of the amount exceeds the currently available number that can be maintained in the *Amount* field in the "Resource Details" tab and there is no alternative scope available with a sufficient number, the job cannot be executed.

**Keep** The *Keep* parameter defines the Keep state of the current resource. More information about the Keep state can be found in Chapter [11.3.1](#).

**Sticky** The *Sticky* parameter defines whether the job is given the current "Sticky" resource or not. More information about the Sticky Flag can be found in Chapter [13.5.8.1](#).

**Lock mode** The *Lock mode* defines the access mode being used in allocating the resource to the current job. The following access modes are available:

1. Exclusive (X)

No other job has access to this resource.

Example: When loading a database table.

2. Shared Exclusive (SX)

These access operations are mutually compatible, but they are not compatible with shared access operations.

Example: An application performs numerous small write operations on a database table.

## 3. Shared (S)

Shared access operations are mutually compatible.

Example: Evaluations of complete database tables or large parts of them.

## 4. Shared Compatible (SC)

These access operations are both mutually compatible as well as being compatible with S and SX access operations.

Example: An application performs numerous, brief read transactions on a database table. Such applications do not impede one another. Small write transactions or large-scale evaluations are not an impediment either.

## 5. NoLock (N)

The NoLock lock mode indicates that the resource is neither being locked nor is a lock to be heeded. A job that locks a resource with NoLock can always access it regardless of any other lock options.

Here is a compatibility matrix summarising all the lock options and their interactions.

(+) means that the lock modes are mutually compatible. Two jobs that request the two lock modes can run concurrently.

(-) means the lock modes exclude one another. A job stays blocked until the other job releases the resource again.

	Exclusive (X)	Shared (S)	Shared Exclusive (SX)	Shared Compatible (SC)	NoLock (N)
Exclusive	-	-	-	-	+
Shared	-	+	-	+	+
Shared Exclusive	-	-	+	+	+
Shared Compatible	-	+	+	+	+
NoLock	+	+	+	+	+

Table 12.1: Lock mode matrix

**Mapping** The *Mapping* indicates the current Resource State Mapping that is using the resource.

**P (Priority)** The *Priority* defines which start priority the process had at the time of the submit in the [Submit Jobs](#) dialog. The values for the priority range from 100 (lowest priority) to 0 (highest priority).

**EP (Effective Priority)** The longer a job waits to be executed in the system, the higher the effective priority of its execution in the schedulix system. This is to make sure that previously started jobs are taken into account before the later jobs (depending upon the settings in the [Priority field in the Batches and Jobs dialog](#) and the start time).

How the priority rises over time can be set in the server configuration. The default setting causes the priority to rise by one point for every half an hour that elapses since the job was submitted.

Example: If a job was started at 12 p.m. with a priority of 50, the effective priority at 12:30 p.m. is 49, at 1 p.m. it is 48 and so on until the highest level has been reached.

**Blocked Processes** If the current job is marked in the list as being blocked, the reason why the job cannot be executed is shown in red.

Example: If the resource is currently exclusively allocated to another job and the current job also wants that resource to be exclusively allocated to it, the *Lock mode* field is shown in red.

### 12.4.3 Parameters tab

All the parameters defined in the jobserver or scope are administered in the "Parameters" tab. These parameters can be used for the jobs and those programs that are executed in the jobs.

The parameters are inherited in scope hierarchies. A parameter can be defined at the highest level and is then automatically made available to every scope and jobserver located below this level.

The "Parameters" tab looks like this:

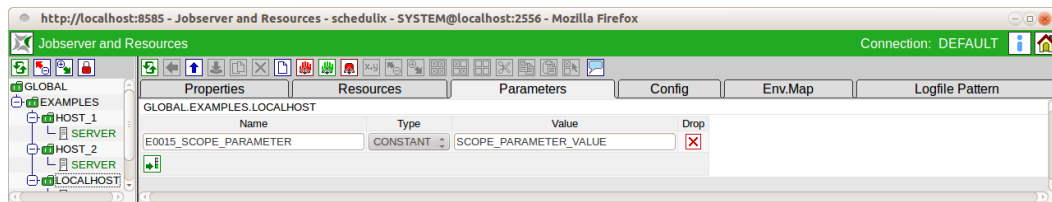


Figure 12.9: Jobserver and scope parameters

A list of all the parameters is displayed here. If a jobserver has been selected in the navigation, all the parameters inherited from the parent scope hierarchy are visible as well in addition to the current parameters. However, they cannot be changed at this level. To change an inherited parameter at this level, it has to be recreated under the same name. This value overwrites the value of the inherited part. The displayed value can now be changed.

The columns in the list shown above have the following meanings:

**Name** The name of the parameter is shown here.

**Type** This is the type used for the parameter. The following options are available:

1. Dynamic

This designates a dynamic parameter. The value of this parameter is derived from the value of an environment variable that has to be maintained in the system on which the current scope or jobserver is running.

This value is only determined when the jobserver is started. If the jobserver is currently running on the computer, the actual value of the identically named environment variables on the computer and the environment in which the jobserver is running is displayed in the *Value* field.

*Important: The environment variable must have been created on the computer and in the jobserver's start environment.*

## 2. Constant

If the parameter type is "Constant", it is a constant expression that needs to be entered in the *Value* field. If a job or its program uses this value, the parameter is replaced by this constant expression when the job or program is executed.

### **Example:**

If a script is to be run independently of the current database, the database connection can be mapped as a parameter. If the database connection parameter is simultaneously defined with respectively different values in both Scope 1 (for example the test database) and Scope 2 (for example the production database), the script can run in both environments without having to modify it because the database connection is defined in the parameter.

**Value** This is the value of the parameter. If the parameter type is "Constant", the parameter value has to be entered. If it is a dynamic parameter, the current value of the environment variables in the system on which the jobserver is running is displayed here.

## 12.4.4 Config tab

The configuration of the jobserver is described in the “Config” tab. A table of key/-value pairs is used for the configuration. Please refer to the jobserver documentation for the meanings of the individual values.

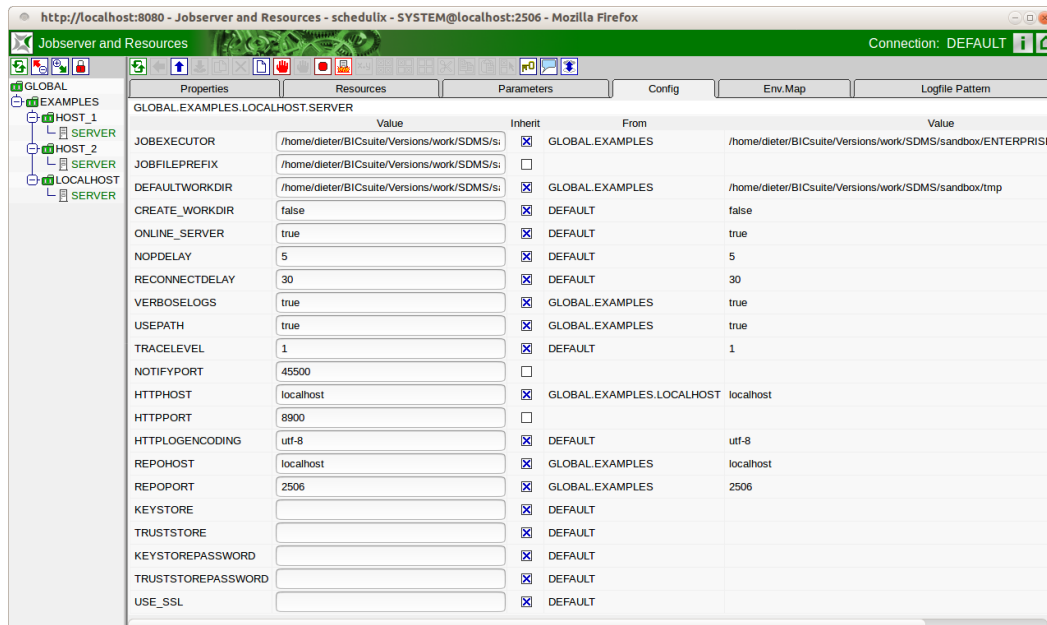


Figure 12.10: Jobserver and scope configuration

The fields in the “Config” tab have the following meanings:

**Value** The *Value* field shows the respective key/value pair.

**Inherit** The *Inherit* field shows whether the key/value pair value is being inherited from a parent scope. The current entry is deleted by setting the flag in the *Inherit* field. In this case, the value of the key/value pair is determined via the inheritance mechanism. The entry “DEFAULT” means that this is preset by the system.

**From** The *From* field shows from which scope the value is being inherited. If *Default* is displayed for the scope, these are the default configuration values.

**Value** The inherited value of the key/value pair.



#### 12.4.4.1 Standard configuration parameters

The following table shows the names of the configuration parameters for a jobserver and their meanings:

Name	Meaning
JOBEXECUTOR	The fully qualified path of the job executor.
JOBFILEPREFIX	Path and file prefix of the task files.
DEFAULTWORKDIR	Working directory of the jobserver and the default working directory for the jobs executed using the jobserver.
ONLINE_SERVER	Boolean value. If this is set to "False", the jobserver logs off when there is no work to be done.
NOPDELAY	Time between two requests for jobs sent to the scheduling server.
RECONNECTDELAY	Time between two attempts to log on to the scheduling server.
VERBOSELOGS	If this is set to "True", the start and end times of the jobs are written to their log files in addition to the process output.
USEPATH	This parameter defines whether the jobserver uses the path setting when starting processes. If "USEPATH" is set to "False", all the program calls have to be fully qualified.
TRACELEVEL	Defines how "communicative" the jobserver is. <ul style="list-style-type: none"> <li>• 0: Errors are logged</li> <li>• 1: Errors and warnings are logged</li> <li>• 2: Errors, warnings and information are logged</li> <li>• 3: Debug level; all messages are logged</li> </ul>
NOTIFYPORT	The Notify port is the port to which a UDP packet is sent if a job is ready for executing by the jobserver.
HTTPHOST	The jobserver can send log files using the HTTP protocol. The two parameters "HTTPHOST" and "HTTPPORT" have to be set for this.
HTTPPORT	Port for sending log files.
HTTPLOGENCODING	Character set for the log files. When displaying log files, a meta/charset tag with this character set is created in the HTML header.

*Continues on next page*

---

*Continued from previous page*

---

<b>Name</b>	<b>Meaning</b>
REPOHOST	Host name or IP address of the scheduling server.
REPOPORT	Port for the scheduling server.

---

Table 12.2: Description of the jobserver configuration parameters

## 12.4.5 Env.Map tab

When a job is started, a number of default parameters for the job are sent to the jobserver as key/value pairs. These parameters can be set as environment variables before the process is started. Whether and under what name these variables are made visible is configured in the "Env.Mapping" mask.

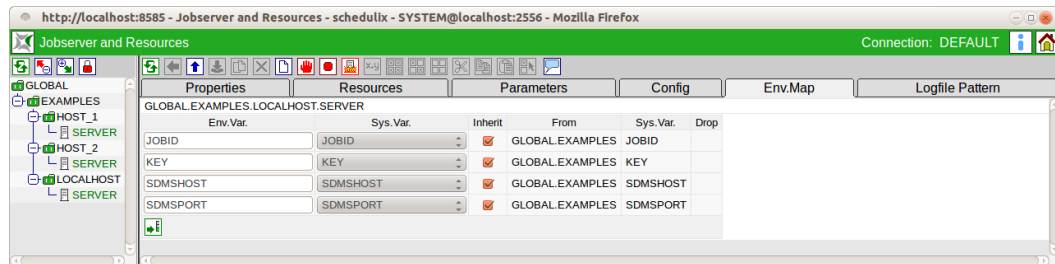


Figure 12.11: Jobserver environment mapping

The columns in the list shown above have the following meanings:

**Env.Var.** The name of the environment variable is shown in the *Env.Var.* field.

**Sys.Var.** The name of the parameter is shown in the *Sys.Var.* field.

**Inherit** The *Inherit* field shows whether the key/value pair value is being inherited from a parent scope. The current entry is deleted by setting the flag in the *Inherit* field. In this case, the value of the key/value pair is determined via the inheritance mechanism.

**From** The *From* field shows from which scope the value is being inherited. If *Default* is displayed for the scope, these are the default configuration values.

## 12.4.6 Logfile Pattern tab

The jobserver can send the contents of log files using the HTTP protocol. To prevent the uncontrolled reading of arbitrary files, only files with names conforming to certain patterns can be requested.

Permitted patterns have to be entered by the administrator. If he does not do so, no files can be requested.

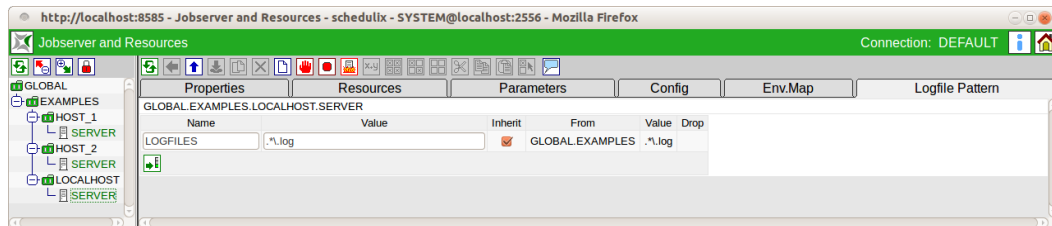


Figure 12.12: Jobserver log file name patterns

The log file patterns are regular expressions with a twist. The character string `"/. ./"` or `"\ . \"` (Directory up) cannot usually occur in the name of a requested file. This rule makes it easier for the administrator to define the log file patterns.

It is generally advisable to test the whole name. A pattern such as `".*\.log"` allows all files to be read which contain this string somewhere in their name. The pattern `".*\.log$"`, on the other hand, is an important limitation, since the name has to end with this string.

The pattern `"^/tmp/.*\\.log$"` is being used in the screenshot. All files below the directory `"/tmp"`

that end with `"/.log"` can be requested.

Requests for log files that do not match any of the patterns are logged in the jobserver's log file as errors. The messages look something like this:

```
ERROR [Jobserver] 01-12-2009 16:43:43 CET [HttpThread] ERROR: Illegal file request: /etc/passwd
```

together with some additional information (which is logged for each file request):

```
...[HttpThread] Got Request from 1.2.3.4 : GET /?FNAME=/etc/passwd HTTP/1.1
...[HttpThread] Got Request from 1.2.3.4 : Host: localhost:8881
...[HttpThread] Got Request from 1.2.3.4 : User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.0.15)
    Gecko/2009102815 Ubuntu/9.04 (jaunty) Firefox/3.0.15
...[HttpThread] Got Request from 1.2.3.4 : Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
...[HttpThread] Got Request from 1.2.3.4 : Accept-Language: en-us,en;q=0.5
...[HttpThread] Got Request from 1.2.3.4 : Accept-Encoding: gzip,deflate
...[HttpThread] Got Request from 1.2.3.4 : Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
...[HttpThread] Got Request from 1.2.3.4 : Keep-Alive: 300
...[HttpThread] Got Request from 1.2.3.4 : Connection: keep-alive
...[HttpThread] Got Request from 1.2.3.4 : Cookie: _ZopeId='55722729A4JSGHnrljM'; tree-s='...'
...[HttpThread] Got Request from 1.2.3.4 : Cache-Control: max-age=0
```

An error message is displayed to the user:

```
ERROR: The requested filename doesn't match any of the configured patterns
```

## 12.5 Resource Links

Resource Links allow resources to be allocated in a different scope to the one in which a job is being executed.

For example, if operations are being performed on a database server in a client/server environment, the resources required by the database operation are located not on the client computer on which the job is being executed, but on the host computer of the database server.

Resource Links can be used to map a resource for one scope in another scope.

A Resource Link is created by copying a resource with the usage *STATIC*, *SYSTEM* or *SYNCHRONIZING* from a scope or server to the clipboard and pasting it into another scope.

If a resource and/or a Resource Link is being pasted from the clipboard, the following mask opens for choosing how the paste operation is to be performed.

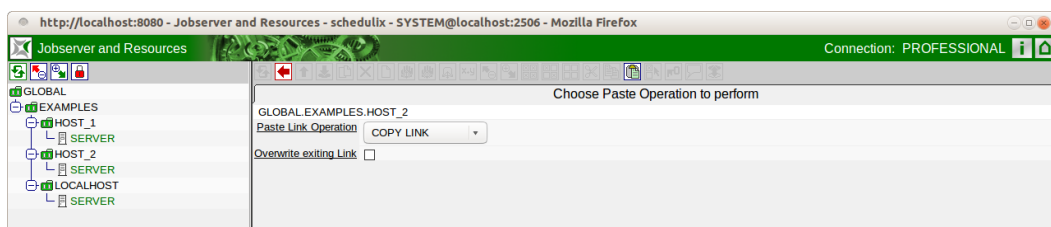


Figure 12.13: Copying and creating Resource Links

If a resource is being pasted from the clipboard, the field *Paste Resource Operation* is displayed.

The following selections are possible:

- **COPY RESOURCE:** The resource is copied.
- **LINK TO RESOURCE:** A Resource Link is created.

If a Resource Link is being pasted from the clipboard, the field *Paste Link Operation* is displayed.

The following selections are possible:

- **COPY RESOURCE:** The resource referenced by the link is copied.
- **COPY LINK:** The link is copied.
- **LINK TO LINK:** A Resource Link referencing the link is created.

If a Resource Link is being pasted from the clipboard, the field *Overwrite Existing Link* is displayed as well. If this check box is enabled, an error message is not displayed if a link with the same name already exists. Instead, the link is overwritten or modified.

Resource Links are indicated in the resource list by an arrow in the icon.

### 12.5.1 Resource Details tab

The “Resource Details” tab for a Resource Link shows the details of the actual resource that is addressed by the Resource Link. If the Resource Link is edited, the changes are applied to be addressed resource.

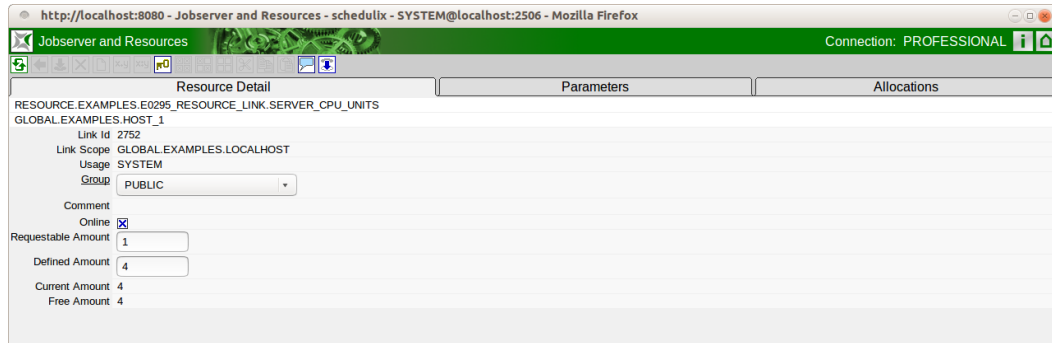


Figure 12.14: Information about a Resource Link

The following ‘read-only’ fields are also displayed in addition to the attributes of the base resource that have already been mentioned.

**Link Id** The Id of the resource referenced by the link.

**Link Scope** The scope in which the linked resource is located.

# 13 Batches and Jobs

## 13.1 View

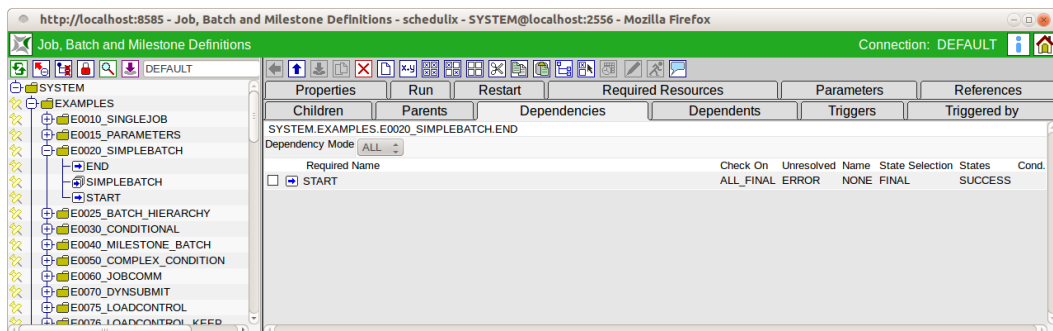


Figure 13.1: Batches and Jobs

## 13.2 Concept

### 13.2.1 In short

The Batch and Jobs dialog is where all the Job Definitions in the schedulix Scheduling System are managed.

The objects "Batches", "Jobs" and "Milestones" are used to define the units to be executed in the Scheduling System.

A job is a container for an executable script or program in the Scheduling System. When a job is submitted and started, the jobserver runs the specified program or script and sends feedback after the process has been completed as to whether it has succeeded or failed (Exit State).

A batch is a container for other objects and contains so-called children. When a batch is started, all the children are started automatically as well. A batch does not have a program or script that is executed.

The batch (or job) that is the first object started in the Submit Batches and Jobs dialog is called the Master Job.

A milestone is an object that can be used to send a notification when a certain number of completed jobs has been attained (or not attained) or for mapping complex dependencies. A milestone doesn't have an executable program or script either.

### 13.2.2 Detailed description

Each Scheduling Entity (batch, milestone or job) can belong to a batch run in a parent-child hierarchy. A job is started when its parent has been started. This means that when such an executable object hierarchy is submitted, all the jobs are submitted from top to bottom. However, this takes place within a transaction so that all the actions, as seen by an outsider, are performed simultaneously.

Each task (batch, milestone or job) can have dependencies. A dependency describes which executable object (required job) must have been executed beforehand. If only one specific Exit State is to be taken into account, the dependent job is not started until the required job has been completed in full and the correct Exit State has been reported back to the system.

This allows dependency models to be created which ensure that all the lead jobs really could be successfully executed in order to start a follow-on process (e.g. a report).

Each task can require different [resources](#). This can be defined by entering the [environment](#) and a [footprint](#) and by specifying the required resources.

This ensures that has all the required resources (e.g. sufficient main memory and CPU units, the correct database and the requisite system programs, etc.) are available to the runtime environment being used to execute the job.

Synchronizing Resources can also be used to make sure that processes do not hinder one another or that the resources have specific states and are up to date time-wise. A load distribution can also be achieved using the resources.

Each Scheduling Entity can define or use certain parameters. This enables data to be transferred between the individual jobs or from the runtime environment. The parameters can be transferred in both directions within the parent-child relationship (from parent to child, from child to parent).

However, it is also possible to access the parameters of Scheduling Entities that do not belong to the parent-child relationship. This allows data to be transferred between all the Scheduling Entities of a Master Job. More information about the parameters can be found in [Chapter 13.5.10](#).

Actions can be defined for a task to be performed when a particular event occurs (e.g. completion of the job) or when specific Exit States are attained. These actions are called triggers. A trigger defines which action is to be performed when a specific event occurs. The action that is triggered is again an executable object.

If the event occurs, this trigger is initiated and the job (or batch) is submitted. This functionality can be used to send a message to a system administrator, for example.



### 13.2.3 Recommended convention for batch objects

For the sake of clarity and to facilitate working with batch objects, we advise our users to adhere to the following convention.

A Batch Object should be located in a folder with the same name. This folder should contain the batch and its children (sub-batches and jobs). Sub-batches should, in turn, be located in a separate subfolder. Job Objects that have children should be treated like batches, i.e. they should also be located in separate subfolders.

This convention is not mandatory, but it can significantly reduce the time and effort required to grant access privileges or deploy, copy and move sub-workflows, etc., and greatly improve the orientation.

The following function extensions have been implemented in the schedulix!Web interface to make it easier to apply this convention:

- When you create a folder, a Batch Object can also be created at the same time in this folder.
- When you create a batch, you can also create a parent folder for the batch.
- When renaming batches or folders, the folder or batch with the same name can be renamed as well.
- When creating objects in a folder with a batch or job of the same name, at the same time the new object can be added to the batch or job as a child.
- When cloning jobs, the new job can be added as a child to the same batch or job in the folder.
- When cloning folders, the batch or job of the same name in the new folder can be automatically renamed.
- When cloning folders, the batch or job of the same name can be added as a child to the same batch or job in the parent folder.

## 13.3 Navigator

All objects of the types "batch", "job" and "milestone" are hierarchically managed in the Navigator in containers (so-called folders). These folders are used for clearly organising objects.

In addition, parameters and a default [environment](#) can be defined in them. All the parameters for the folder are available to all the jobs in that folder. The folder environment behaves additively to the environment of a job.

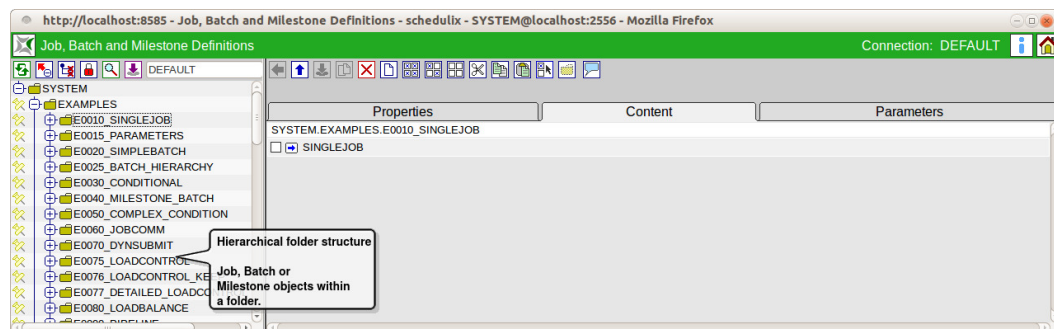



Figure 13.2: Navigator for batches and jobs

### 13.3.1 Pinning

For greater clarity in the Navigator folder, objects can be "pinned" by clicking the

*Pin* icon .

A pin can be removed again by clicking the *Unpin* icon .

Rows with a pin are always displayed regardless of whether the parent folders are expanded or not. This allows you to scroll significantly more quickly in the Navigator.

The "Expand" and "Collapse" icons in the tree view are coloured yellow if there is a pinned object below a folder. This indicates that collapsing a folder will not close it completely, but that the path to the pinned object will always remain visible.

## Navigator

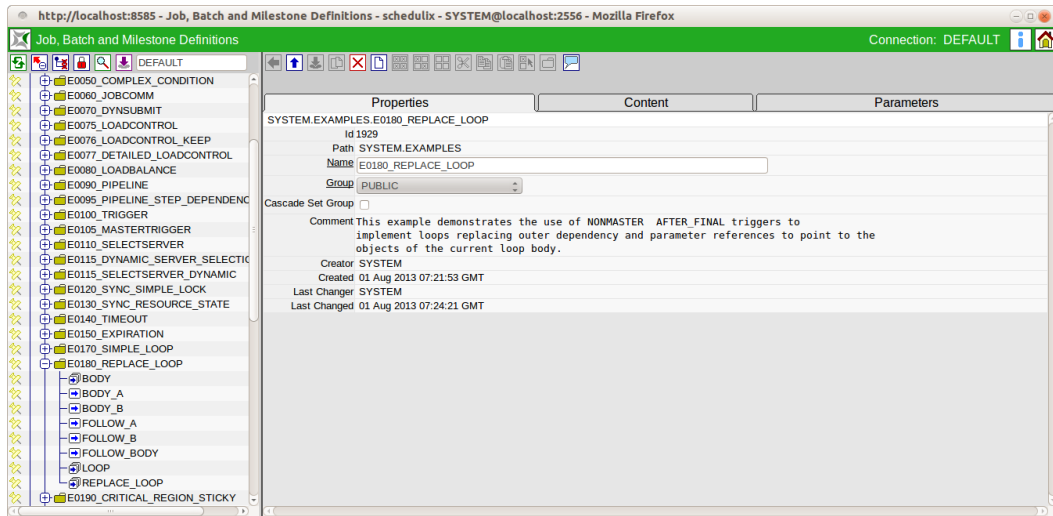


Figure 13.3: Batches and Jobs; view without a pin

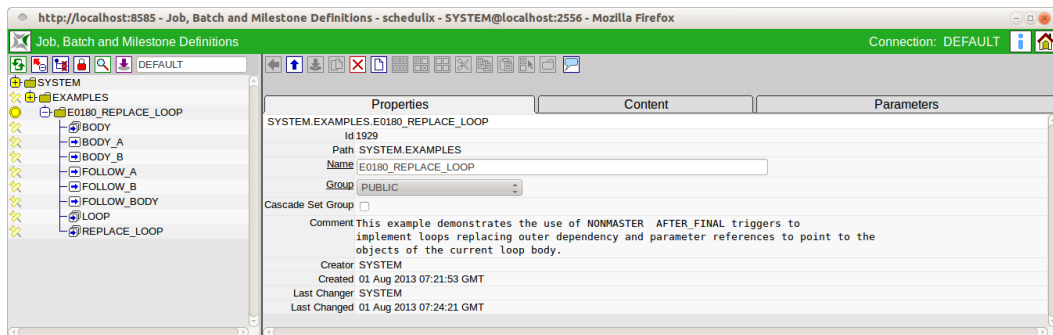


Figure 13.4: Batches and Jobs; view with a pin

### 13.3.2 Folder bookmarks



#### Store Bookmark

Clicking the *Store Bookmark* button creates a bookmark for the current Navigator setting.

The navigation settings contain information about how the jobs, batches and locked objects are displayed, the current expansion state of the tree and the pinned objects. The name for the saved bookmark is entered in the box to the right of the *Store Bookmark* button.

The "DEFAULT" bookmark is used when calling the "Batches and Jobs" dialog from the schedulix!Web desktop.

Other bookmarks can be opened by clicking [Bookmark](#) (Folder).

## Folder editor

If the current user is a “Web GUI Admin”, an options field is displayed next to the name of the bookmark.

Whether the bookmark is to be saved as a System Bookmark (visible to all users) or a User Bookmark (only visible to the current user) is set in this field.

### 13.3.3 Folder search



Find

Clicking the *Find* button switches the Navigator to search mode.

In the input field, you can enter a search pattern corresponding to the syntax selected in the options field to the right of the input field. The SQL LIKE syntax and regular expressions are supported.

In the next options field, you can define whether this pattern is to apply to the entire path (FULL) or only the end of it (TAIL).

The hits are displayed in the Navigator by clicking the *Find* button in search mode. Clicking a hit opens the object in the editor pane, the object is pinned, and the search mode closes again.

## 13.4 Folder editor

### 13.4.1 Properties tab

All information pertaining to the general properties of the folder is recorded in the “Properties” tab.

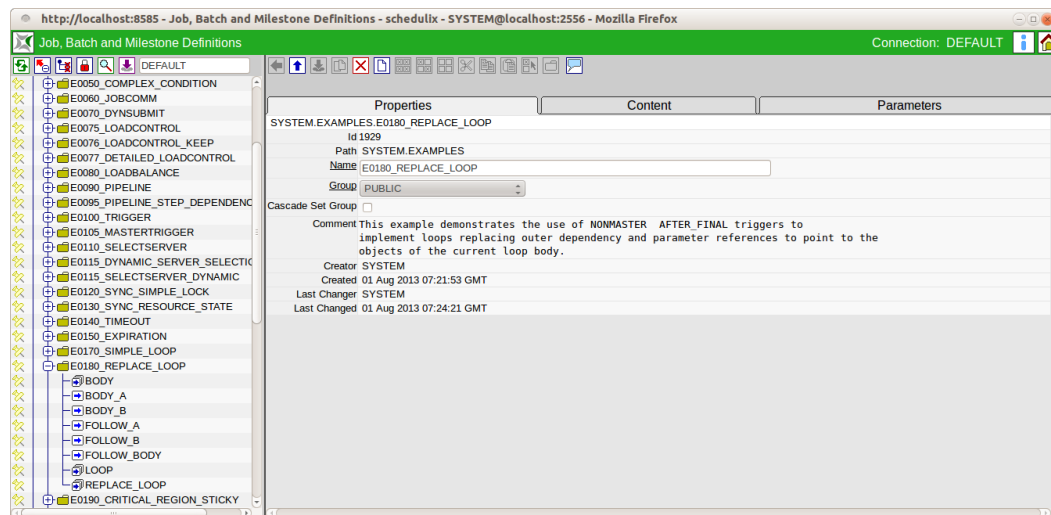


Figure 13.5: Folder properties

**ID** The *Id* is the unique identification number of the object.

**Path** The *Path* describes the parent folder hierarchy. All parent folders are separated by periods.

**Name** The *name* serves as a comprehensible and unambiguous identification of the object in the current context. It must be unique in the folder. Identical names in different folders are allowed.

**Rename in Content** If the name of a folder is changed and the folder conforms to the "Batch in Folder" convention (contains a batch or job object of the same name), the *Rename to Content* check box is displayed. If this option is selected and the folder is renamed or cloned, the batch or job object in the folder is renamed as well.

**Group** The *Group* field defines the schedulix user group that owns the object.

**Cascade Set Group** The *Cascade Set Group* is used to set the user group for the folder and all the subordinate objects.

**Create Batch in Folder** The *Create Batch in Folder* check box is displayed when creating a new folder and is used in support of the recommended convention for folders as well as batch and job objects. If a check mark is set here, a batch object of the same name is also created in the new folder.

**Batch Exit State Profile** The *Batch Exit State Profile* field is only displayed if the *Create Batch in Folder* check box has been enabled. It is used to select the Exit State Profile for the new batch.

**Add as Child** The *Add as Child* check box is displayed in the following cases. When creating a new folder, the *Create Batch in Folder* check box is enabled and the parent folder of the newly created folder conforms to the "Batch in Folder" convention (it contains a batch or job object of the same name). An existing folder is renamed is changed and both the folder itself as well as its parent folder conform to the "Batch in Folder" convention (they both contain a batch or job object of the same name). In this case, the *Add as Child* check box only has an effect when cloning the folder. The *Add as Child* check box is ignored when the folder is saved (renamed). The *Add as Child* check box can be used to select whether the newly created batch or a copied (cloned) batch or job is also to be added as a child of the parent batch or job folder.

## Folder editor

**Environment** The *Environment* field is a selection box for choosing a valid [Environment Object](#). This selection defines the additional requirements for all the jobs in the folder that have to be fulfilled by their execute environment.

**Comment** A more detailed explanation about the object can be entered in the *Comment* field.

### 13.4.2 Content tab

The content of the folder is shown in the "Content" tab.

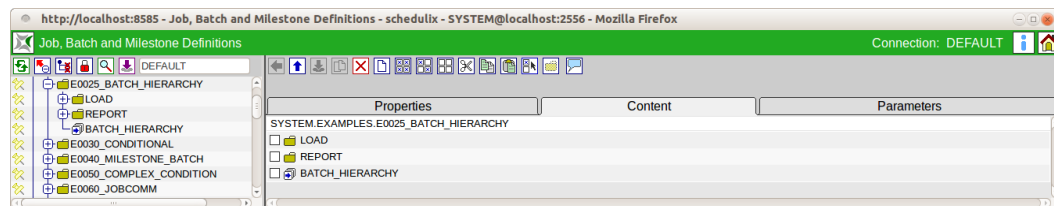


Figure 13.6: Folder content

When copy/cut with a following paste is done in "content" tabs of folders, depending on whether source and target folders and selected objects are following the batch in folder [convention](#), a list is displayed on paste. This list shows the actions on parent/child relationships, which will be taken to follow the convention. These actions can be (de)selected individually before the paste operation is executed.

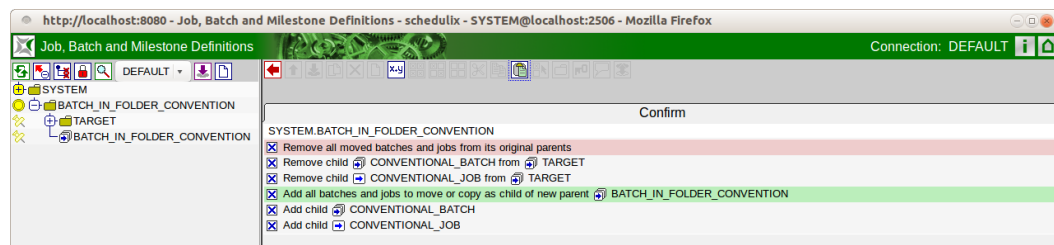


Figure 13.7: Convention Support on Paste

### 13.4.3 Parameters tab

The parameters can be defined in the "Parameters" tab.

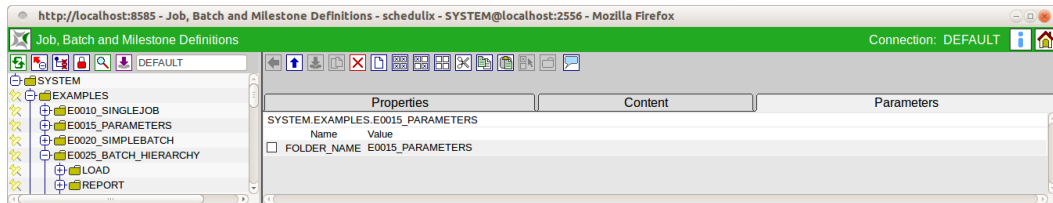


Figure 13.8: Folder parameters

**Name** This is the name of the parameter. All the parameter values are exchanged between the individual jobs using this name.

**Value** The *Value* field shows the parameter value.

### 13.4.4 Resources tab

The resources are shown in the "Resources" tab. Clicking the name of the resource opens the [Resource Details](#) tab.

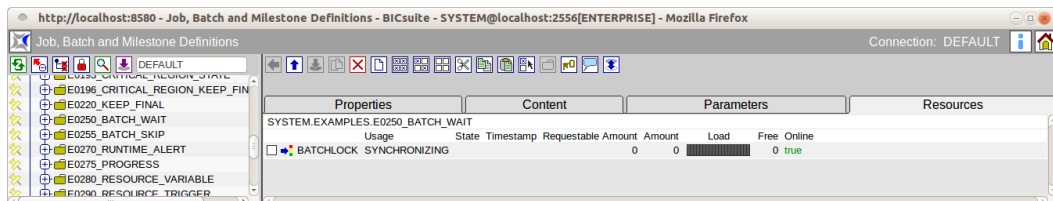


Figure 13.9: Folder resources

**Usage** The *Usage* field specifies the "Resource" type. More details about resource types can be found in the [Named Resource](#) dialog.

**State** The *State* field is only visible if the resource usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned.

It designates the current status of the resource in this scope or jobserver. The state can be set or changed with this value. The drop-down list contains all the valid Resource States of the Resource State Profile, which can be selected here.

This may be necessary when manually fixing an error to restore a resource to a state that allows further processing by follow-on jobs.

If you manually change this value, the value in the *State* field in the "Resources" tab is not automatically refreshed. Click the *Refresh* button if this is required and necessary.

**Timestamp** This field is only visible if the resource usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned. The *Timestamp* indicates the time of the last change made to a Resource State.

Timestamps play a role if an expiration interval has been defined in an expiry object. If this is the case, the timestamp must not be older than the specified interval. More details about expiration times can be found in [Chapter 13.5.8.1](#).

If the completion of a job causes the Resource State to change, the timestamp is automatically updated by the schedulix Server.

**Requestable Amount** The maximum amount of resources that can be requested by a job.

**Amount** The *Amount* is the number of resource instances allocated to or requested by the current job. If the value of the amount exceeds the currently available number that can be maintained in the *Amount* field in the "Resource Details" tab and there is no alternative scope available with a sufficient number, the job cannot be executed.

**Load** run The *Load* shows the graphical load of the resource.

**Free** This field is only visible if the resource usage has been set to "Synchronizing" or "System".

The *Free Amount* designates the total number of instances of a resource in the selected scope or jobserver that have not yet been allocated to jobs.

**Online** *Online* can be used to switch a resource in this scope or jobserver online or offline. An offline resource is temporarily unavailable.



## 13.5 Editor for Job Definitions

### 13.5.1 Properties tab

All information pertaining to the general properties of job, batch, or milestone objects is recorded in the "Properties" tab. Only the respectively required information is displayed in the "Properties" tab depending on the type you selected in the navigation or while creating the new object.

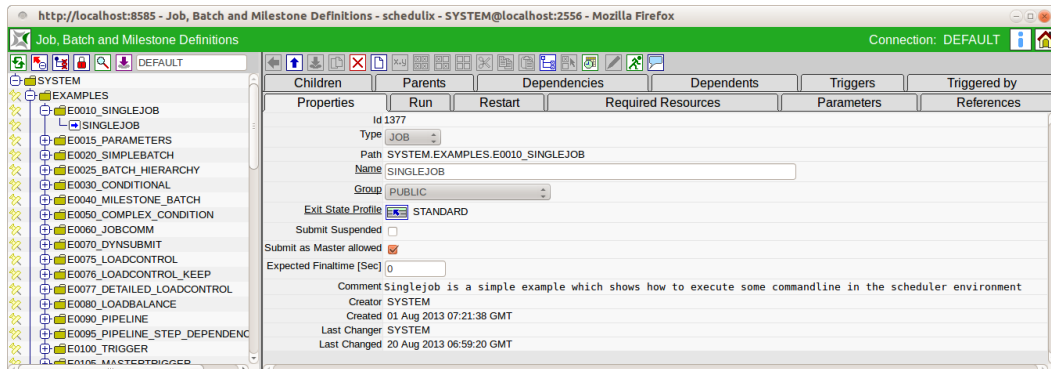


Figure 13.10: Job Definition properties

The fields in the "Properties" tab have the following meanings:

**ID** The *Id* is the unique identification number of the object.

**Type** This is the *type* of object. One of the following types can be selected here:

#### 1. Batch

The object is a batch. A batch is used as a startable container for a number of jobs, batches, or milestones. It does not itself have an executable program, but is only used as the start object for its child objects.

An example of modelling as a batch is a number of jobs that have to be executed on a daily basis. These can all be grouped under a batch called "Daily". All weekly jobs are likewise grouped under the batch "Weekly", which also includes the "Daily" batch as this obviously has to be executed as a batch at the same time. A "Monthly" batch can additionally be created that contains all the monthly executable jobs. This means that only these three batches have to be created in this system in the [Time Scheduling](#) and all the child jobs, batches, or milestones are started automatically at these times.

The legend for the graphic can be found in [Chapter 1.8](#).

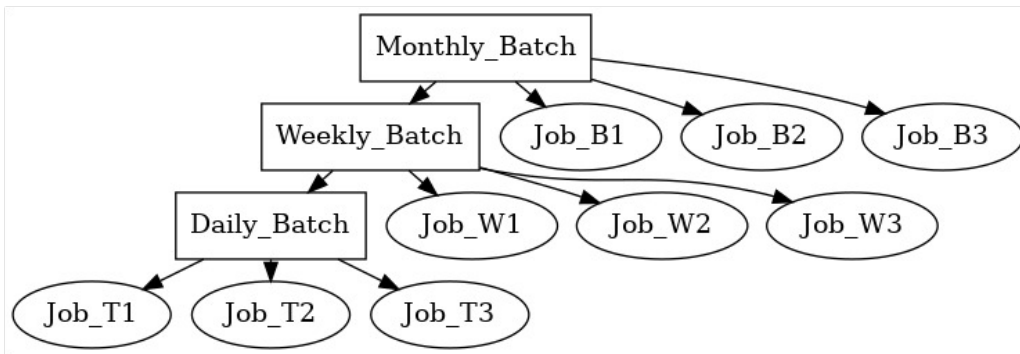


Figure 13.11: Examples of batches

## 2. Milestone

A milestone is required for several reasons. Firstly, it serves as a placeholder for a particular time during the course of a complex batch run. When this milestone is reached, a trigger can be used to perform an action (for example to send a text message or e-mail).

Secondly, milestones are required to define complex dependencies between single jobs. Since a job can only be dependent upon either all the required objects or at least one required object, it is not possible to directly define more complex dependencies (upon Job1 and Job2 or Job3, etc.).

These options can now be defined using a milestone. By modelling a Milestone1 that is reached when Job1 and Job2 have been completed, the dependency of our job upon Job3 or Milestone1 can be modelled and the condition can now be mapped.

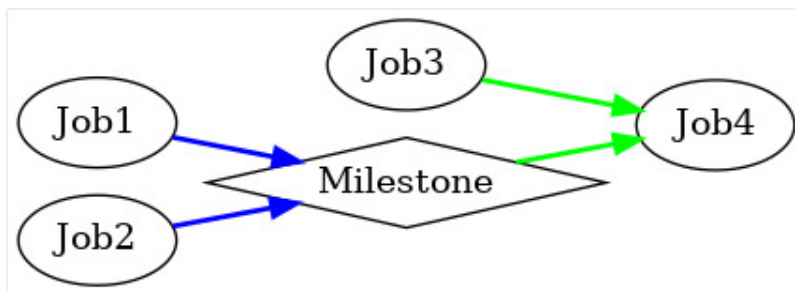


Figure 13.12: Example of a milestone

The legend for the graphic can be found in Chapter 1.8.

Milestones also have a unique characteristic in that dynamic children can be

defined for them. Every time a job which is also a child of a milestone is dynamically submitted, that milestone also receives this job as a child.

This functionality allows for "pipeline" processing. Let's suppose, for example, that data from a (partitioned) table needs to be processed. Due to the partitioning it has been decided to parallelise the work. The relevant data is extracted for each partition and then processed in some form of other in parallel operations. As soon as all the extraction jobs have finished, the data is to be processed yet again. If the extraction job is defined as the dynamic child of a milestone, then the milestone will attain the final state at precisely the time all the extraction jobs have been completed successfully. Further processing of the table can be made dependent upon the milestone and in turn take place parallel to the processing of the extracted partition data.

### 3. Job

All executable objects (programs, scripts, etc.) that are to be executed in the Scheduling System on a [jobserver](#) must be defined as a job.

### 4. Folder

A folder is a folder within the folder hierarchy for the executable objects. It is used for organising the other objects because in a normal scheduling system there are many different jobs, batches and milestones that have to be managed, and with a flat hierarchy it is very easy to lose track of them. How the hierarchy is structured is decided by each user. It could be logically organised (all objects on the same theme in one folder, etc.) or it could map the departmental or human resource structure (all objects relating to user/department X are placed in the same folder). The user should just be able to quickly find his way around the hierarchy and easily find the required objects.

Folders cannot be executed nor can they have dependencies. It is possible to define an [environment](#) for a folder. Furthermore, parameters can be defined which can be used by all the jobs located in the folder.

**Path** The *Path* describes the parent folder hierarchy. All parent folders are separated by periods.

**Name** The name serves as a comprehensible and unambiguous identification of the object in the current context. It must be unique in the folder. Identical names in different folders are allowed.

**Decomposite** The *Decomposite* check box is displayed if an object type is changed from "job" to "batch". If a check mark is set here, a so-called decomposition is carried out. Not only is the object type changed to "batch", but a job is also created as a child of this batch which then takes over the job properties of the original job

object. The batch object produced by the decomposition (with a job as a child) behaves, in the context of the Scheduling System, as the previously existing batch object. All the time schedules, dependencies, parameter references, etc. are retained. The result is an ideal starting point for decomposing jobs into smaller single steps (process decomposition).

**Job Name** The *Job Name* is only displayed if the *Decomposite* check box has been enabled. The name of the Job Object to be created during the decomposition can be entered here.

**Rename Parent Folder** If the name of a batch is changed and the batch conforms to the "Batch in Folder" convention (it is located in a folder of the same name), the *Rename Parent Folder* check box is displayed. If this check box is enabled, when you save (rename) the batch the parent folder of the same name is renamed as well. The *Rename Parent Folder* check box is ignored when cloning a batch.

**Create Batch Folder** The *Create Batch Folder* check box is displayed when a new batch is created or the type of an object is changed to "batch". It serves to support the recommended convention for folder, batch and job objects. If a check mark is set here, a folder with the same name as the batch is created for it and the new batch is created in this folder. The *Create Batch Folder* check box is also displayed if an object type is changed from "job" to "batch" (decomposition).

**Add as Child** The *Add as Child* check box is displayed in the following cases. When creating a new batch, job or milestone object where the parent folder of the new object to be created conforms to the "Batch in Folder" convention (contains a batch or job object of the same name). In the case of an existing job or milestone where the name is changed and the parent folder conforms to the "Batch in Folder" convention. In this case, the *Add as Child* check box only has an effect when cloning the job or milestone. The *Add as Child* check box is ignored when the object is saved (renamed). The *Add as Child* check box can be used to select whether the newly created object or a copied (cloned) job or milestone is also to be added as a child of the parent batch or job folder.

**Group** The *Group* field defines the schedulix user group that owns the object.

**Exit State Profile** An [Exit State Profile](#) can be selected in this field.

**Submit Suspended** The *Submit Suspended* switch can be used to delay the actual start of a workflow, for example where external authorisation by a third person is necessary for a job. Only when this authorisation has been given can the job be started.

**Resume** This field is also only displayed if the option *Submit Suspended* has been enabled. Here you can choose whether a workflow should be resumed automatically.

The following options are available:

- NO: disables this function.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time** This field is only displayed if "AT" was selected for *Resume*. The required resume time is entered here in the format YYYY-MM-DDTHH:MI:SS. This format is based on the ISO standard 8601 and also permits an incomplete entry. If you enter 'T16:00', the job will be resumed at 16:00 hours (starting from the current set time).

**Resume In** This field is only displayed if "IN" was selected for *Resume*. Here you can specify how many time units (see *Unit*) the system is to wait for until the resume action is triggered.

**Unit** This field is only displayed if "IN" was selected for *Resume*. This field is used to define whether the entry in *Resume In* is in minutes (MIN), hours (HOUR) or days (DAY).

**Nice Value** This field is only visible if the object is a batch. The *Nice Value* indicates the priority to be applied for running children of this object. It is an offset value which is added to the priority of the children.

**Example:**

If the process has a Nice Value of -50 and the child has a priority of 100, then the priority of the child process of the batch is 50. If the child process were to be started on its own, the priority would be 100.

**Submit as Master Allowed** This field is only visible if the object is a batch or a job.

This switch specifies whether it is possible to submit this job as a Master Job in the [Submit Batches and Jobs](#) dialog. If the switch is not enabled, this is not possible and the job or batch cannot be executed on its own, but only as a child of any parent job.

**Comment** A more detailed explanation about the object can be entered in the *Comment* field.

### 13.5.2 Run tab

The "Run" tab is only displayed if an object in the navigation has the type "Job" or if, in the case of a new object, "Job" was chosen in the selection box.

All the information about the executable command line for a job object is managed in the "Run" tab.

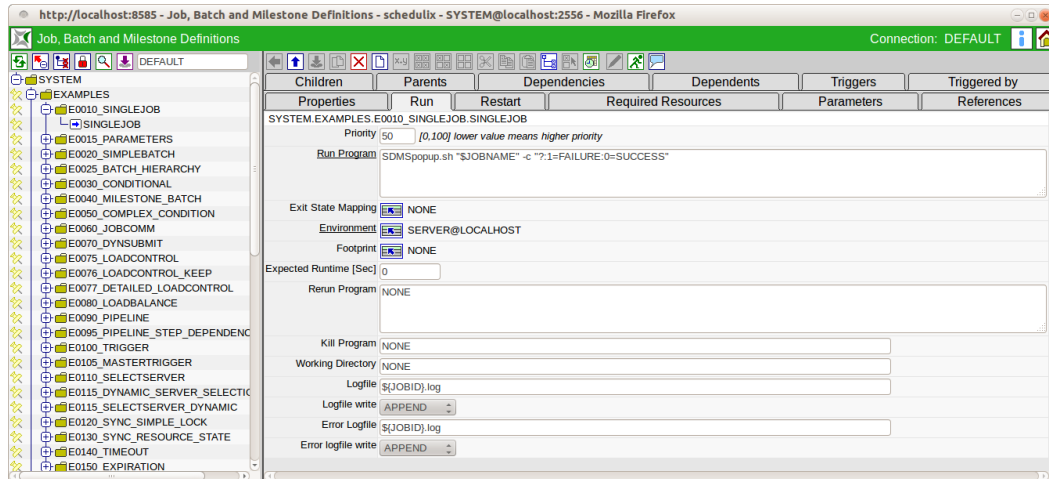


Figure 13.13: Run Job tab

The fields in the "Run" tab have the following meanings:

**Priority** The *Priority* field indicates the urgency with which the process, if it is to be started, is to be considered by the Scheduling System. The range of values for priorities ranges from 100 (very low) to 0 (very high). In the system, all the startable jobs with their priorities are taken into account and the job with the highest priority is started. If two jobs have an identical priority, the job with the lowest ID will be started.

The longer the waiting time for a job before it is started in the Scheduling System, the higher its priority. This means that if a process is started with a priority of 50, its priority increases per time unit. This is set in the Scheduling System configuration. For example, the priority of a job can increase by one point every half an hour. This means that the priority is 49 after half an hour, then 48 after one hour, etc. Consequently, older jobs are accorded a higher priority and taken into account before more recently submitted jobs.

**Run Program** A command line to be executed by a suitable jobserver is entered in the *Run Program* field. This command line is first interpreted by the Scheduling System and then decomposed into the program call and single parameters. The Bourne shell rules with regard to quoting are followed here.

If any environment variables or parameters are to be specified, these have to be quoted conformant to the respective execute environment (depending on the shell or command line interpreter of the environment; see [Environment](#)). A list of all the usable standard parameters can be found in the paragraph [13.5.10.2](#). Please refer to the documentation on your command line interpreter (shell) for more information about the quoting to be used.

If the target system is a Windows system, some other difficulties are very likely to arise. When a table of executables and parameters is made available for execution under Windows, first of all Windows compiles a command line from the table which it then interprets. The use of quoting under Windows also causes problems because the called programs either remove the quotes or don't remove them depending on the type of executable (.exe or .bat).

If the executable program (the first element of the command line) is a valid integer, the command line is not run by the jobserver. Instead, the job is treated as if it had completed itself with the integer as the Exit Code. Dummy jobs with 'true' or 'false' as the program can now be implemented as '0' instead of 'true' or '1' instead of 'false' and are therefore processed much more efficiently and quickly by the system.

Should it really be necessary to run an executable with a number as the name, this can be achieved by using a path prefix (e.g. './42' instead of '42').

**Exit State Mapping** An existing [Exit State Mapping](#) can be entered in the *Exit State Mapping*. This defines the mapping of the Exit Code for a job after an Exit State. In the absence of a specified mapping, the default mapping from the Exit State Profile is used. The mapping must be compatible with the profile.

**Environment** In the *Environment* field it is necessary to select an existing [Environment](#). All the Named Resources in this environment must be present on a jobserver so that the job can be run on that server.

**Footprint** An existing [footprint](#) can be selected in the *Footprint* field. All the Named Resources for this footprint must be present on a jobserver in a sufficient quantity so that the job can be run on that server.

**Expected Runtime** The *Expected Runtime* describes the anticipated time that will be required to execute a job. This is entered manually and can be used to monitor the runtime, for example.

**Expected Finaltime** The *Expected Finaltime* describes the anticipated time required to execute a job (from the submit until it attains a final state). This has to be entered manually and is used for the display in the calendar. There is obviously a standard parameter for this as well so that the expected value is made available to the job.

**Rerun program** The *Rerun Program* field specifies the command that is to be executed when repeating the job following an error (rerun). This can be a different command to Run program as it may be necessary to perform some cleanup work prior to a restart before the original command can be executed again.

If no value was entered in the Rerun program, the Run program is called when restarting the job.

See 'Run program' above for details regarding processing (quoting, treating integers as a program name).

**Kill program** The *Kill program* field determines which program is to be run to terminate a currently running job.

If no value is entered in this field, it will not be possible to prematurely terminate the job from the schedulix interface.

An example of a Kill program in a Unix environment is the **"kill"** command. The process ID of the current job can also be accessed in the Kill program using the predefined system variable "PID" (example: "kill -9 \$PID").

**Working Directory** The *Working Directory* specifies the directory in which the Run program is to be started. The jobserver switches to the specified directory before the Run program is started.

If no value is entered in this field, the default directory of the [jobserver](#) is used as the working directory.

A parameter or a folder, scope or jobserver parameter can also be specified as the working directory.

**Logfile** The *Logfile* field specifies the file in which all the normal outputs of the Run program are to be returned. These are usually all the outputs that use the standard output channel (stdout under UNIX).

Parameters can also be replaced in the *Logfile* field. For example, the parameter JOBID can be used to allocate a job a unique name or a TIMESTAMP can be used for the date.

If no value is entered, all the outputs will be discarded.

If no path is specified in the log file, it is written to the working directory.

**Logfile Write** The *Logfile Write* field tells the system where a log file is to be created. The following options are available:

- Append

The Append option appends all the created log files to an existing log file. If no log file exists yet, a new one is created.

- Truncate



With the Truncate option, the log file is deleted and a new one is created for each new action.

**Error Logfile** The *Error Logfile* field specifies the file in which all the error outputs from the Run program are to be returned. Error outputs are all the outputs that use the error output channel (stderr under UNIX).

Parameters can also be replaced in the *Error Logfile* field. For example, the parameter JOBID can be used to allocate a job a unique name or a TIMESTAMP can be used for the date.

If no value is entered, all the outputs will be discarded.

If no path is specified in the log file, it is written to the working directory.

If the same name as the one in the *Logfile* field is being used, these outputs will also be written to this file. The sequence of the outputs (error or normal) is random due to the buffering, however. No outputs from the normal output channel (stdout) are overwritten regardless of the content of the "Error Logfile Write" field.

**Error Logfile Write** The *Error Logfile Write* field tells the system where the error log file is to be created. The following options are available:

- Append

The Append option appends all the created error log files to an existing error log file. If no error log file exists yet, a new one is created.

- Truncate

With the Truncate option, the error log file is deleted and a new one is created for each new action.

### 13.5.3 Restart tab

The restart behaviour for a job can be defined in the "Restart" tab.

A job will only be restarted if its Job State is "FINISHED".

If a job gets a "RESTARTABLE" Exit State because of an "ERROR" Job State, it will NOT be restarted!

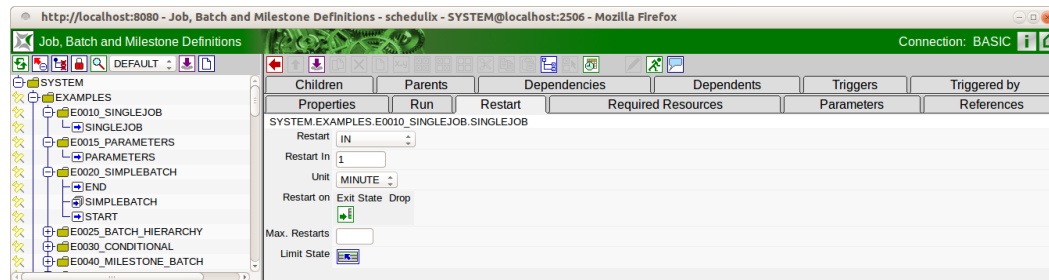


Figure 13.14: Job Restart tab

The fields in the "Restart" tab have the following meanings:

**Restart** This determines whether the job is to be automatically restarted if it attains a "RESTARTABLE" Exit State.

The following selections are possible:

- NO: No restart. No other fields are displayed in the "Restart" tab.
- IMMEDIATE: Immediate restart.
- AT: Restart at a specified time (e.g. "T16:00" at 16:00 hours).
- IN: Restart after a delay (e.g. 10 minutes)

**Restart Time** This field is only displayed if "AT" was selected under *Restart*.

Time when the job is to be restarted using the format "YYYY-MM-DDTHH:MI:SS". This format is based on the ISO standard 8601 and also permits an incomplete entry. If you enter 'T16:00' here, the job restart will be suspended and resumed at 16:00 hours (based on the submit time of the job).

**Restart In** This field is only displayed if "IN" was selected under *Restart*.

Number of time units (see *Unit*) for the delay until the job restarts.

**Unit** This field is only displayed if "IN" was selected under *Restart*.

Units for the delay interval (see *Restart In*).

**Restart On** Table of Exit States that are to trigger a restart.

If the table is empty, jobs are only started for all "RESTARTABLE" Exit States.

**Condition** Optional condition (from schedulix PROFESSIONAL) for triggering a restart.

**Max. Restarts** Maximum number of times that the job is restarted.

The system parameter TriggerSoftLimit applies if nothing is entered here. Entering 0 will restart the job any number of times.

**Limit State** Here an Exit State can be set, which will be the Exit State of the job, if the maximum number of restarts was reached.

### 13.5.4 Children tab

All the executable objects that have been defined as a child of the current job are displayed as a list in the "Children" tab. Other objects can be added as children using [copy and paste](#).

The sequence in which the children are run is not fixed, but it can be defined in the dependencies.

The "Children" tab looks like this:

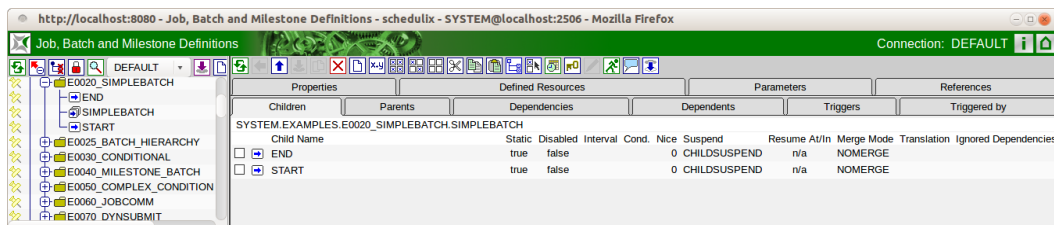


Figure 13.15: Batches and Jobs; Children tab

A list of all the job, milestone and batch objects that are children of the selected job is shown here. The columns in the list shown above have the following meanings:

**Child Name** This is the name of the child. Clicking the name opens the "Child Details" tab.

The other columns are explained in the next section.

### 13.5.4.1 Child Details tab

The “Child Details” tab is displayed after selecting the name of a child object in the “Children” tab. Here, all the data already displayed in the table as elements is listed again and can also be modified here. The tab looks like this:

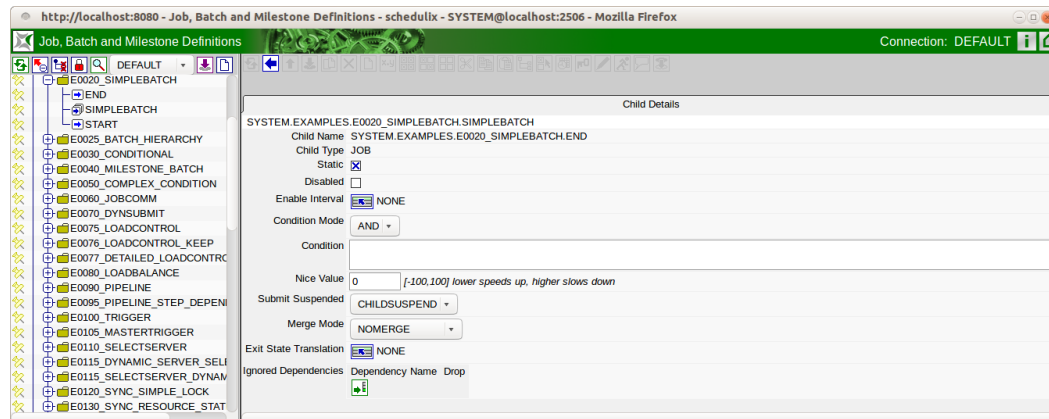


Figure 13.16: Batches and Jobs; Child Details tab

The fields in the “Child Details” tab have the following meanings:

**Child Name** The *Child name* is always displayed with the full path to the parent folder hierarchy.

**Child Type** The *Child Type* specifies the type of the child (batch, job or milestone).

**Alias** A child can be assigned a new logical name by entering it in the *Alias* field. This name must be unique among all the children of the selected job. The alias is only significant for Dynamic Children. The Parent Job can create additional instances of the children by stating the alias in the submit. This means that the parent does not need to know where the child has been saved. This abstraction makes it considerably easier to reorganise the folder structure.

**Static** The *Static* switch indicates whether a child is to be submitted statically or dynamically. In the case of a static child, the child instance is instantiated when the parent is submitted. If it is a dynamic Child (the Static switch has not been set), one or more instances of the child can be dynamically created at runtime.

Children are often used for dynamic parallelisation tasks. Here you only decide during the runtime of the Run program whether and how many dynamic children are to be created. These are then created by the schedulix API. This allows you to split up a process into a large number of parallel working sub-processes.

**Disabled** If the *Disabled* switch is set, the batch or job defined as child will be submitted as being "disabled". All children of the "disabled" batch or job will also be submitted as being "disabled". "Disabled" Jobs are not executed but go straight to their default Exit State if all their dependencies have been fulfilled.

This way it is possible to skip a batch or job, which is part of a processing chain without losing the order of processing of the chain.

**Enable Interval** Optionally, an "Enable" interval can be set. If this interval is set, a check is run when the child is submitted to determine whether the submit time for the master is a valid time for the "Enable" interval. If this is not the case, the child is submitted as being "disabled".

**Condition Mode** If a Condition is entered, the Condition Mode defines whether the Condition shall be combined with a set Enable Interval by using OR or AND.

**Condition** If a condition is set, it will check on submit of the child. If the condition is not met, the Child Job will be disabled. The Condition Mode controls how the Condition is combined with a given Enable Interval.

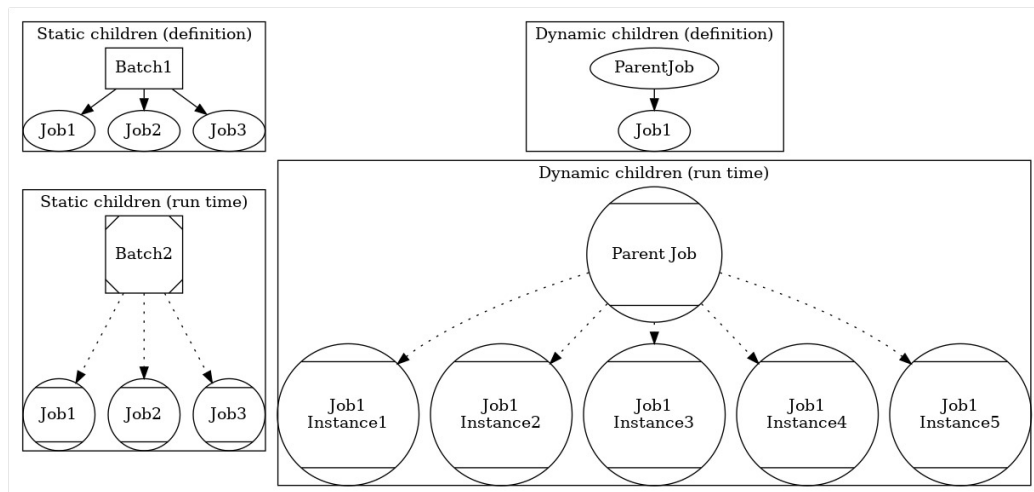


Figure 13.17: Static and dynamic children

The legend for the graphic can be found in Chapter 1.8.

**Nice Value** The *Nice Value* indicates the priority to be applied for running the selected child. It is an offset value which is added to the priority of the child.

**Example:** If the process has a Nice Value of -50 and the child has a priority of 100, the job instance started as a child of this job has a priority of 50. If the child process is started on its own, it has a priority of 100.

If the parent is a batch, the Nice Value of the child is added to that of the parent.

**Submit Suspended** The *Submit Suspended* parameter specifies the form in which the child object is delayed when being started or if it can be started immediately. The following options are available:

1. YES

The child is created as being suspended when it is submitted and it has to be started by being explicitly released.

2. NO

The child is not suspended when it is submitted and can be started immediately.

3. CHILDSUSPEND

Whether a delay takes place or not depends on the *Suspend* field for the child job. This means that the setting defined in the child job is applied.

**Resume** This field is also only displayed if the option *Submit Suspended* has been set to "YES". Here you can choose whether a workflow should be resumed automatically.

The following options are available:

- NO: disables this function.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time** This field is only displayed if "AT" was selected for *Resume*.

The required resume time is entered here in the format YYYY-MM-DDTHH:MI:SS. This format is based on the ISO standard 8601 and also permits an incomplete entry. If you enter 'T16:00', the job will be resumed at 16:00 hours (starting from the Job Submit time).

**Resume In** This field is only displayed if "IN" was selected for *Resume*.

Here you can specify how many time units (see *Unit*) the system is to wait for until the resume action is triggered.

**Unit** This field is only displayed if "IN" was selected for *Resume*.

This field is used to define whether the entry in *Resume In* is in minutes (MIN), hours (HOUR) or days (DAY).

**Merge Mode** Merge Mode indicates whether a child object is to be started multiple times within a "Master Job" run or not.

The legend for the graphic can be found in Chapter 1.8.

The following options are available:

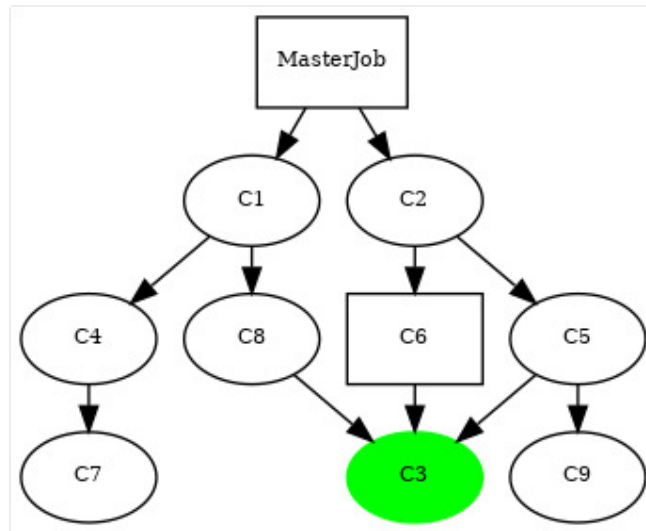


Figure 13.18: Example of Merge Mode

1. No Merge

Jobs are not merged. The child job is started without taking into consideration whether this job has already been run in the current Master Batch.

In the example shown in Figure 13.18, the job C3 is started three times if No Merge is set in each parent-child relationship.

2. Failure

When the job is submitted, the system checks whether the child job has already been started. If this is the case, an error message is returned and the start of the Master Job is aborted.

In the example shown in Figure 13.18, the job cannot be submitted.

3. Merge Local

A merge is performed in the child hierarchy of the parent object. This means that if the job in the child hierarchy has already been started once, it will not be started again. If the job has already been started within the Master Job but outside the child hierarchy of the parent, this fact is ignored and the job is submitted again.

The dependencies of this job are taken into account just as if this job had been started in this child relationship.

If in the example above Merge Local Mode has been set for the child relationship C2→C6 and C2→C5,



the job C3 is only started twice because the merge took place in the child hierarchy of C2. As the relationship C8→C3 is not in this child hierarchy, C3 is executed because no merge has been performed here.

#### 4. Merge Global

The system verifies within the entire Master Job whether this job has already been started before. If this is the case, the job will not be restarted.

This may be necessary if a job should only run once, but it is needed as a requirement by several jobs. It can be defined as a child of multiple parent processes. The parent process that is run first starts the process. The other parent processes use Merge Global to check whether this job has already been run and they don't start it again.

The dependencies of this job are taken into account just as if this job had been started in this child relationship.

In the example above, the job C3 is only started once because a global merge takes into account all the relationships in the Master Job and therefore performs the merge for the relationships (C8→C3, C6→C3 and C5→C3).

Which relationship is used to actually start the job is immaterial.

**Exit State Translation** In this selection mask, you can select the Exit State Translation to be used to report the Exit State of the child object to the parent object. More details about Exit State Translations can be found in Chapter 5.

**List of Ignored Dependencies** Here you can add a list of dependencies which are to be ignored by the child in this parent-child relationship. All of these dependencies are no longer taken into consideration for the child and it can be started without the dependency on the parent being fulfilled.

An example of Ignored Dependencies is shown in Figure 13.19. The legend for the graphic can be found in Chapter 1.8.

Only those dependencies that have a value entered in the *Dependency Name* field can be ignored. Unnamed dependencies cannot be chosen in the selection box and are therefore not ignored.

## Editor for Job Definitions

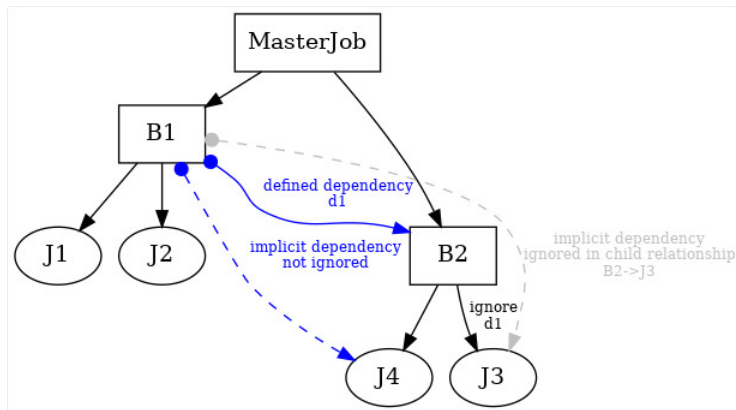


Figure 13.19: Example of Ignored Dependencies

### 13.5.5 Parents tab

The "Parents" tab shows a list of higher-ranking parent objects which use the selected object as a child. This therefore visually represents the upward view, whereas the children represent the downward view since all the children of the currently selected object are displayed there. The tab looks like in Figure 13.20.

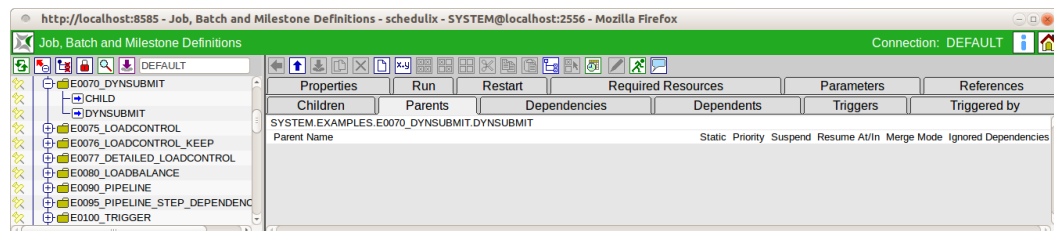


Figure 13.20: Batches and Jobs; Parents tab

The tab is for information purposes only. Values cannot be changed nor can you add any new list entries. This is only possible in the direction parent-to-child.

The columns in the list above are explained under "Children" tab.

### 13.5.6 Dependencies tab

The “Dependencies” tab shows a list of all the objects upon which the current object is dependent. A dependency exists if the current object is not allowed to start until all the objects upon which the job is dependent have been executed beforehand and completed with a specific [Exit State](#).

Additional objects can be added as required Scheduling Entities using [copy and paste](#).

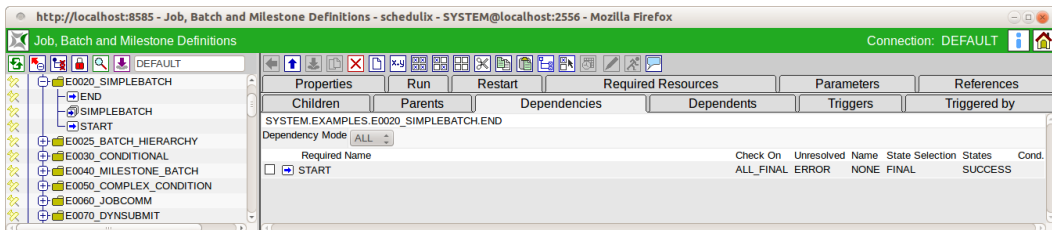


Figure 13.21: Batches and Jobs; Dependencies tab

The tab looks like in [Figure 13.21](#)

The field in the list shown above has the following meaning:

**Dependency Mode** “Dependency Mode” states the context in which the list of dependencies has to be viewed. The following options are available:

1. ALL

All the dependencies must be jointly fulfilled before the object can start. This corresponds to linking all the conditions with “**AND**”.

**Example:** Object C is dependent upon Job A and Job B (that is, it should only start once A and B have finished) and so the Dependency Mode ALL needs to be selected.

2. ANY

At least one of the dependencies must be fulfilled before the object can start. This corresponds to linking all the conditions with “**OR**”.

**Example:** Object C is dependent upon Job A or Job B (that is, it should only start once A or B has finished) and so the Dependency Mode ANY needs to be selected.

The dependencies of a job are either logically linked by the ANY operator or by the ALL operator. A batch or milestone object is used where it is necessary to map complex dependency links (for example, dependent upon A or B and C or D like in [Figure 13.22](#)). The legend for the graphic can be found in [Chapter 1.8](#).

## Editor for Job Definitions

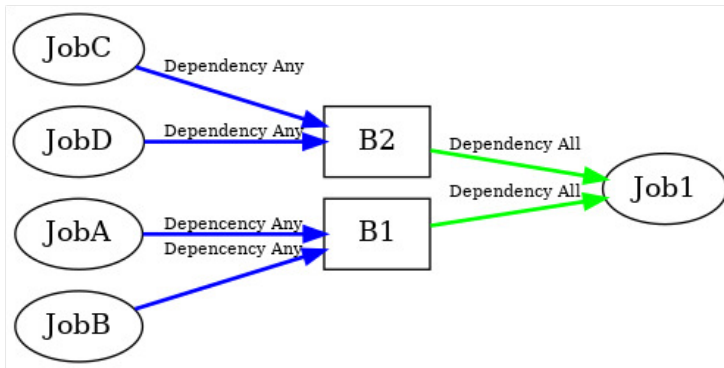


Figure 13.22: Example of Dependency Modes

In the example above, batches B1 and B2 were generated. B1 has linked the dependencies A and B with ANY and B2 has linked the dependencies C and D with ANY as well. Our object (Job1) can now link the two batches as a dependency using ALL, and in doing maps the complex condition (dependent upon A or B and C or D). The "Dependencies" list shows all those objects upon which the currently selected object is dependent. All the elements in the list are linked using Dependency Mode. The list columns are explained in the next section.

### 13.5.6.1 Dependency Details tab

The "Dependency Details" tab opens when an entry has been selected in the "Dependencies" tab by clicking the Required Name.

All the aspects of a dependency between the two jobs can be configured here. The tab looks like this:

The fields in the "Dependency Details" tab have the following meanings:

**Dependency Name** The *Dependency Name* is optional and is the prerequisite for ignoring dependencies.

**Required Name** This is the name of the object upon which the currently selected job is dependent. It corresponds to the full name with the parent folder hierarchy.

**Required Type** The type of the required object is displayed in the *Required Type* field.

**Check On** The type of integrity check to be used for the required object is described in the *Check On* field. The following options are available:

## Editor for Job Definitions

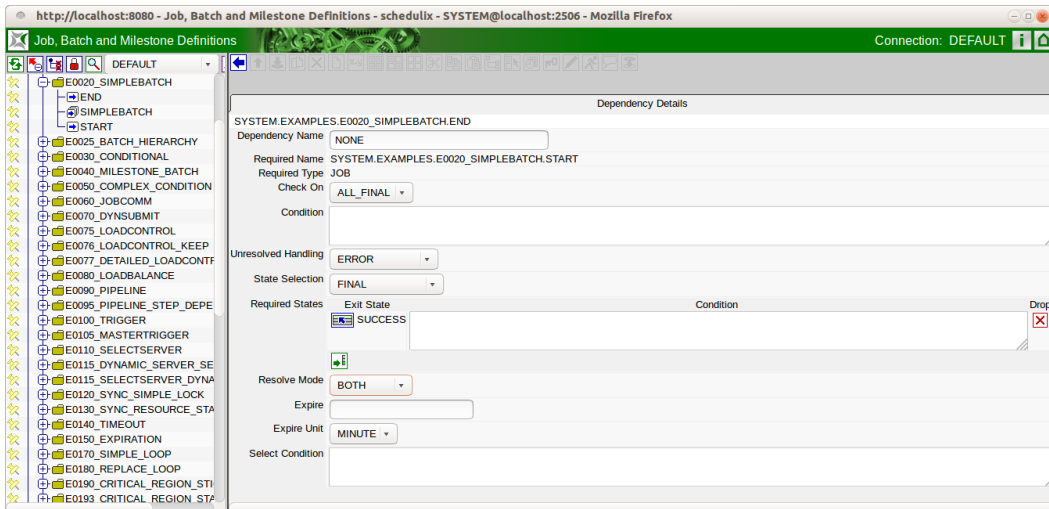


Figure 13.23: Batches and Jobs; Dependency Details

### 1. ALL\_FINAL

If this option is selected, both the required job and all its children must have attained a Final State.

### 2. JOB\_FINAL

If this option is selected, the system only checks whether the job has attained a Final State. The state of the child objects is not verified.

**Condition** This *condition* has to be additionally fulfilled before the dependency can be regarded as having been fulfilled. Parameters of the type "Resource Reference" cannot be used because no resources have been allocated at the time the condition is evaluated.

**Unresolved Handling** The *Unresolved Handling* selection field describes what to do if a dependent object instance is not present in the current Master Batch.

The legend for the graphic can be found in Chapter 1.8.

The following options are available:

### 1. IGNORE

The dependency is ignored if the required job does not exist. If the job is present, it is handled in the normal way.

An IGNORE may be necessary where smaller batches have been merged to form larger ones. If the small batch is running on its own, it can ignore all the

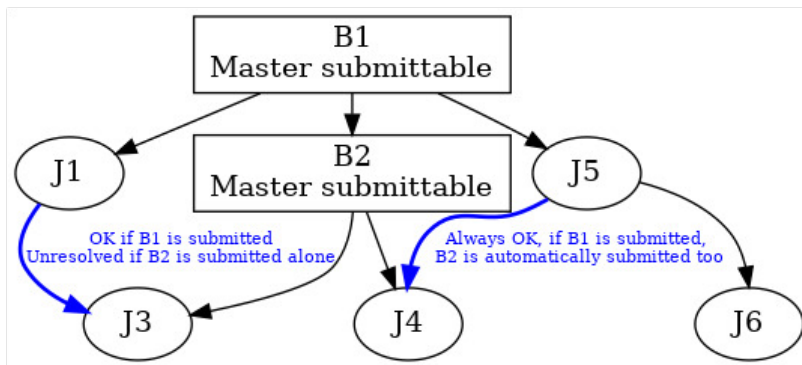


Figure 13.24: Example for Unresolved Handling

dependencies that only have to be taken into account with larger batch runs. If it is running as part of the large batch, the required jobs are present and the dependencies have to be taken into consideration.

In the example above, an Ignore has to be entered in the dependency J1→J3 if the batch B2 can be started on its own as well.

2. ERROR

If the required job is not present, the submit action is terminated with an error.

3. SUSPEND

The current job switches to the Suspend State. This means that it has to be started either by an external program or manually as soon as the required job is available.

In the example above, Job J3 would switch to a Suspend State were J1 to be a dynamic child of B1 and it is not clear when the job is to be submitted.

The idea behind SUSPEND is that neither an unattended start nor an "ERROR" occur.

4. DEFER

The dependency will be resolved when the required job is submitted later during the master run, due to a trigger or dynamic submit.

If a master run gets inactive and only jobs in state DEPENDENCY\_WAIT are left, jobs only waiting on DEFER dependencies will become UNREACHABLE.

5. DEFER\_IGNORE

The dependency will be resolved when the required job is submitted later due to a trigger or dynamic submit.

If a job is only waiting on DEFER dependencies, those dependencies will be ignored.

**State Selection** The *State Selection* field defines which Exit States for the required jobs fulfil the dependency and the dependent job can start. The following options are available:

1. FINAL

If this option is selected, the required job must have attained a Final State. This option allows the required Exit States to be selected in the following *Required States* table.

2. ALL\_REACHABLE

If this option is selected, all the Final Exit States that are not defined as being Unreachable in the Exit State Profile of the required job are valid.

3. UNREACHABLE

If this option is selected, only the Exit State that is defined as being Unreachable in the Exit State Profile of the required job is valid.

4. DEFAULT

If this option is selected, all the Final Exit States that are defined as being Dependency Default in the Exit State Profile of the required job are valid.

**Required States** This is a list of all the valid [Exit States](#) which the required object must have for the dependency to be fulfilled and so the dependent job can start. This list is only visible if the option FINAL has been selected in the *State Selection* field.

**Example:**

A Job A required by Job B can take on two Exit States (SUCCESS or WARNING). As Job B should only start if Job A has been completed with the SUCCESS Exit State (i.e. it has finished successfully), only a SUCCESS entry is allowed in the list of Required States. Once Job A has been completed and has received the SUCCESS Exit State, Job B can then start. If Job A finishes with WARNING, Job B cannot start. This list is necessary because a dependent job can accept multiple Exit States as a requirement. In our example, this means that Job B would have entered both Exit States (SUCCESS and WARNING) as Required States if it is to start in any case regardless of whether Job A was completely successful or it finished with a warning.

**Resolve Mode** How the dependency is to be resolved can be selected in the *Resolve Mode* field. The following options are available:

1. INTERNAL

If this option is selected, the dependency is only resolved within the master batch of the dependent job or batch.

2. BOTH

This option is used to first attempt to resolve the dependency within the master batch of the dependency job or batch. If this does not succeed, the dependency is resolved externally.

3. EXTERNAL

This option defines an external resolution in other master batches. If all non-external dependencies are not fulfilled, an attempt is made to determine the requested job. The respective jobs are determined in other master batches, and it is verified whether the expiration and select conditions are fulfilled. In the case of multiple candidates, the most recently submitted job or batch is selected and assigned.

**Expire** The *Expire* and *Expire Unit* fields define the duration of the submit time of the dependent job or batch after the submit time of the requested job or batch in order to be able to perform a resolution of the dependency. The *Expire* and *Expire Unit* fields are only visible if BOTH or EXTERNAL have been selected in the *Resolve Mode* field.

**Select Condition** The *Select Condition* defines a condition that must be fulfilled for a resolution of the dependency to be possible. Parameters of the requested job or batch can be addressed with \$REQUIRED.<parametername> and \$DEPENDENT.<parametername> for parameters of the job or batch to be executed. The *Select Condition* field is only visible if BOTH or EXTERNAL have been selected in the *Resolve Mode* field.

### 13.5.7 Dependents tab

The "Dependents" tab shows all the Scheduling Entities that are dependent upon the selected Scheduling Entity. This is the reverse view of the "Dependencies" tab as the dependent jobs are displayed here and not the required jobs.

The tab looks like this:

The list shows all the objects (jobs, milestones, batches), which require the selected Scheduling Entity. The list is purely informative and can neither be edited nor can the field values in this view be changed. To make any changes, you have to



## Editor for Job Definitions

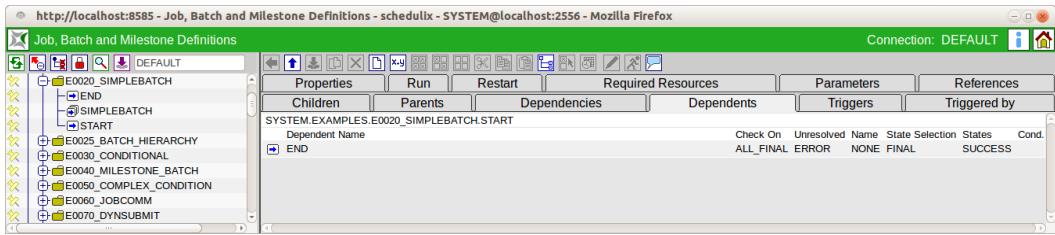


Figure 13.25: Batches and Jobs; Dependents tab

navigate to the list of displayed objects and open the “Dependencies” tab, where you can make the necessary changes.

The list columns are explained in the “Dependency Details” tab.

### 13.5.8 Required Resources tab

The “Resources” tab shows all the [Resources](#) that are required to execute this object. Additionally required resources can be added here.

All the resources that have already been defined by the [environment](#) or a specified [footprint](#) are displayed here. These can only be removed by changing the respective environment or footprint.

The “Resources” tab looks like this:

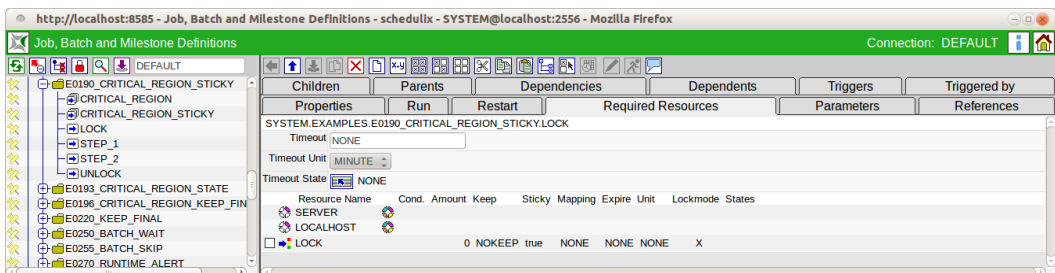


Figure 13.26: Jobs; Required Resources tab

The fields and columns in the “Required Resources” tab have the following meanings:

**Timeout** The *timeout* specifies how long the job is to wait for all the resources after it has been submitted. If no value is entered here, the job will wait for them indefinitely.

The value in this field must be regarded in conjunction with the *Timeout Unit* field. The value for the timeout is entered in the *Timeout* field. The unit is entered in the *Timeout Unit* field. **Example:** If the value 5 has been entered in the *Timeout* field and the timeout unit is *MINUTES*, a job will wait 5 minutes for a resource. If it

does not become available within this time period, this job cannot be started and it terminates with the Exit State defined in the *Timeout State* field.

**Timeout Unit** The timeout unit is the unit that is to be used for the selected value in the *Timeout* field. If no value is entered in the *Timeout* field, the value in the *Timeout Unit* field is irrelevant.

**Timeout State** Here you can select the Exit State which the job is to take on should a timeout occur. By clicking the selection button you can choose those Exit States from the list that are to be made available to the *Exit State Profile*.

For example, you could define a separate Timeout State or use the Failure State. If no value is entered in the *Timeout* field, the value in the *Timeout State* field is irrelevant.

The "Resources" list shows all the resources that are currently required by the selected job for it to be executed. The list contains the following columns:

**Resource Name** This is the name of the resource that is needed to start the current job.

Next to the field name is an icon which indicates the context from which the resource originates. The following icons are used:

- Footprint



The resource originates from the *Footprint* field. An individual request can be created for each resource which overwrites the original request.

- Environment



The resource originates from the *Environment* field. This is only used for informational purposes and cannot be changed in this tab.

- No icon

If the resource name is not followed by an icon, the resource was added in this tab and can also be modified here. By clicking the name, you can switch to the "Resource Details" tab and change the values in the list. You can also click the *Selection* button in front of the name to mark this row and delete or move it.

**Cond.** A condition can be entered in the *Cond.* field that must be fulfilled to ensure that the resource is recognised as being valid. The other columns are explained in the next section.

### 13.5.8.1 Resource Details tab

The “Resource Details” tab opens when you select a list entry in the Required Resources tab.

All the details for an allocated resource can be entered and changed in this tab. The tab looks like this:

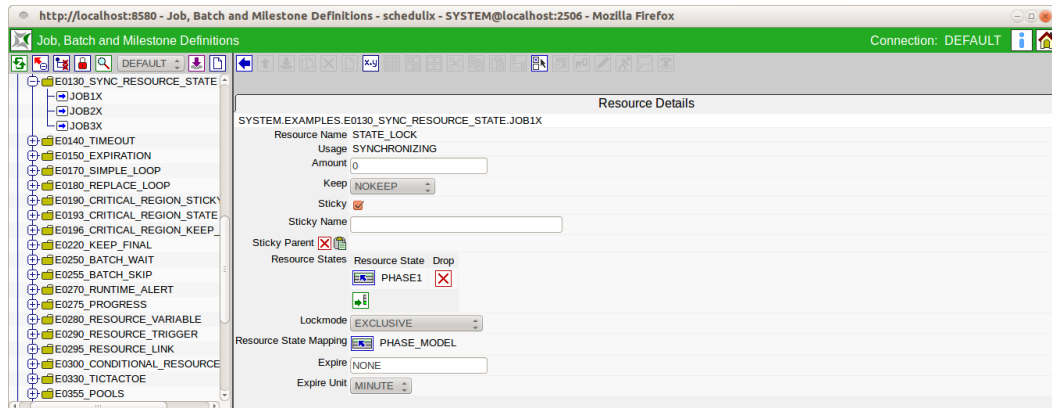


Figure 13.27: Batches and Jobs; Resource Requirement details

The fields in the “Resource Details” tab have the following meanings:

**Resource Name** This is the name of the required resource. The full name together with the parent folder hierarchy is displayed here.

**Usage** This is the usage of the resource. More information about the usage can be found in Chapter 9.3.3.

**Amount** This field is only displayed if the usage type is either “SYSTEM” or “SYNCHRONIZING”.

This is the amount that is required by the resource in this job. Maximum amounts of the resource are made available on the jobserver (or in the parent scopes, see [Jobserver](#)). Every job that requires an amount of this resource reduces the number of free amounts.

**Example:**

A jobserver makes the amount ‘5’ available of Resource A (for example 5 CPU units on this server).

Job A starts now and requires the amount ‘3’. The jobserver has now occupied 3 CPU units, leaving 2 still available.

Job B wants to start, but it also requires an amount of ‘3’. The job cannot be handed over to the jobserver to be started because at this moment only 2 CPU units are available.

Job B can only start when Job A has finished and at least three CPU units are available.

**Keep** The value of the Keep parameter is set here. More information about the Keep parameter can be found in Chapter 11.3.1.

**Sticky** This switch is used to control how resources are returned. If this switch is set, the resource is retained within a Master Job until the last job that requires this resource with an activated Sticky Flag has been completed.

This functionality is for keeping reserved resources within a Master Job while preventing them from being used in the interim by other jobs.

**Example:**

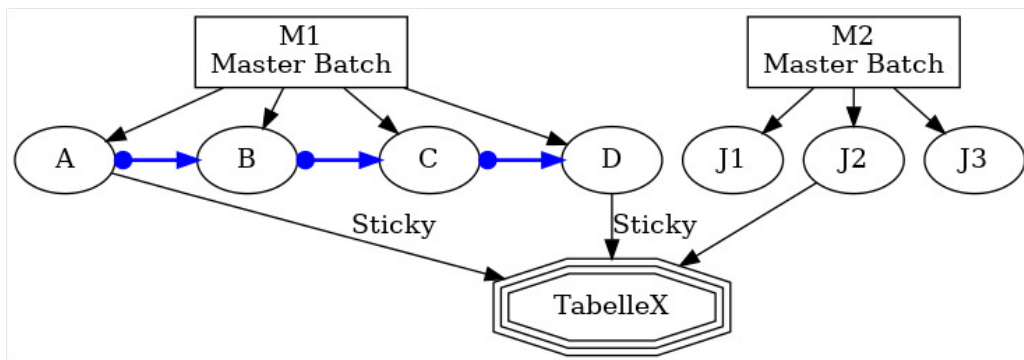


Figure 13.28: Example of Sticky Handling

Master Job M1 comprises the jobs A, B, C and D, which are executed in succession. Job A generates Table X, which is required by Job D. Jobs B and C don't need this Table X. Another Master Job (M2) also wants to access the table and modify it. However, this is to be prevented as long as Job D has not been executed because these actions would corrupt the results from Job A and make them unusable. Table X is now mapped as a Synchronizing Resource.

If a Sticky Flag is not set, the resource would be returned after Job A has been completed and the table could be modified by J2. With a set Sticky Flag (which has to be done in Job A and Job D), the resource is not returned following completion of Job A, but is retained in the Master Job as being allocated.

Other jobs cannot allocate this resource now. Only once Job D (the last job in this Master Job that requires the resource with a set Sticky Flag) has been completed is the resource released again and it can be used by other jobs.

With the Sticky Flag, all the necessary processes can be run on the same scope or jobserver using a Sticky Resource.

## Editor for Job Definitions

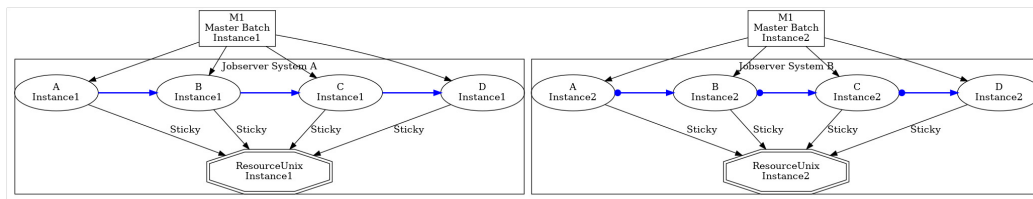


Figure 13.29: Example of load distribution with Sticky Handling

Here's another example to illustrate this:

In a system environment there are 2 UNIX workstations with an identical configuration and environment that are used for load distribution. A Master Job can run on one of these two workstations. However, since the intermediate results are to be transferred as data files between the individual jobs in the Master Job, the Master Job must run on the workstation after it has been selected.

This can be achieved by means of the Sticky Flag. Jobservers (System A and System B) are installed respectively on these workstations and the environment is mapped as a Synchronizing Resource Definition (Resource Unix). This Resource Definition is now allocated within the two jobservers (twice, once on each jobserver). Now all the children of the Master Job have to be defined so that they require the Resource Definition with the Sticky Flag.

The first job (Job A) in the Master Job now allocates one of the two available resources (on one or the other jobserver). Since this sticky has been allocated, it is not returned following completion of the first job. All the other jobs in the Master Job are defined as belonging exclusively to the selected jobserver, and therefore to its workstation, by virtue of this Sticky Resource.

In the diagram, one Master Job (Instance 1) is running entirely on jobserver System A and a second Master Job (Instance 2) is running entirely on System B.

**Sticky name** If the sticky flag described above is set, the *Sticky Name* field is displayed. If the *Sticky name* is set, the resource is only available to those jobs in the same master run that want to allocate the resource with the same sticky name, but not to the other jobs.

**Sticky parent** If the sticky flag described above is set, the *Sticky Parent* field is displayed. If the *Sticky parent* is set, the resource is only available to those jobs in the same master run that want to allocate the resource with the same sticky parent, but not to the other jobs.

**Resource States list** The *Resource States list* is only displayed if the resource has the type "Synchronizing" and a Resource State Profile has been allocated.

A list of required Resource States can be entered here. The job can only be started if the required resource is in one of these states.

**Lock mode** The *Lock mode* indicates the access mode that is to be used to access the required resource. More information about the *Lock mode* can be found in Chapter 12.4.2.3.

**Resource State Mapping** This field is only displayed if the resource has the type "Synchronizing" and a Resource State Profile has been allocated. The *Resource State Mapping* defines how and whether the state of the resource is to be changed after the job has finished. More information about Resource State Mappings can be found in Chapter 8.

**Expire** Gives the value of the Expiration interval. More information about Expire can be found in Chapter 9.3.6.

**Expire Unit** This is the unit used for the Expiration interval. More information about the Expire Unit can be found in Chapter 9.3.6.

**Ignore Expiration On Rerun** If this flag is set, the expire condition will not be checked again on rerun. For a job, which has been executed the first time, the expire condition was met already. The job can be restarted now, despite of the expire condition is not longer fulfilled at the rerun time.

### 13.5.9 Defined Resources tab

The "Defined Resources" are instantiated in that moment when the job starts. These resources are only visible for jobs in the same workflow.

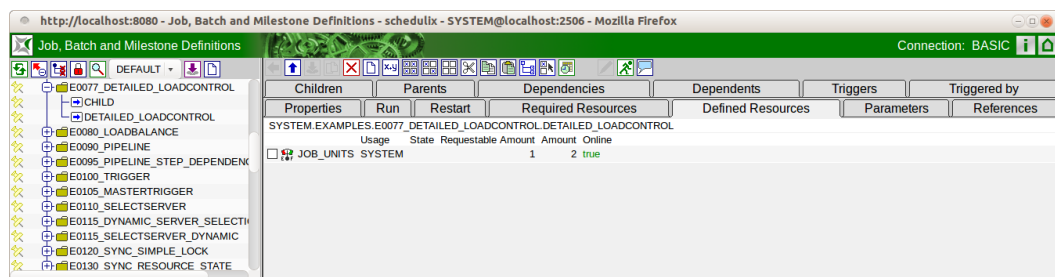


Figure 13.30: Batches and Jobs; Defined Resources

The fields in the list shown above have the following meanings:

**Usage** The *Usage* field specifies the "Resource" type. More details about resource types can be found in the [Named Resource](#) dialog.

**State** The *State* field is only visible if the resource usage has been set to “Synchronizing” and a [Resource State Profile](#) has been assigned. It designates the current status of the resource in this scope or jobserver. The state can be set or changed with this value. The drop-down list contains all the valid Resource States of the Resource State Profile, which can be selected here.

This may be necessary when manually fixing an error to restore a resource to a state that allows further processing by follow-on jobs.

If you manually change this value, the value in the *State* field in the “Resources” tab is not automatically refreshed. Click the *Refresh* button if this is required and necessary.

**Requestable Amount** The maximum amount of resources that can be requested by a job.

**Amount** The *Amount* is the number of resource instances allocated to or requested by the current job. If the value of the amount exceeds the currently available number that can be maintained in the *Amount* field in the “Resource Details” tab and there is no alternative scope available with a sufficient number, the job cannot be executed.

**Online** *Online* can be used to switch a resource in this scope or jobserver online or offline. An offline resource is temporarily unavailable.

### 13.5.10 Parameters tab

In the “Parameters” tab, you can define the parameters that can be used for communication and data transfer between jobs. Parameters that originate from the environment or from the jobserver’s scope or folder environment and the job environment can be defined here for use within the current job.

The “Parameters” tab looks like this:

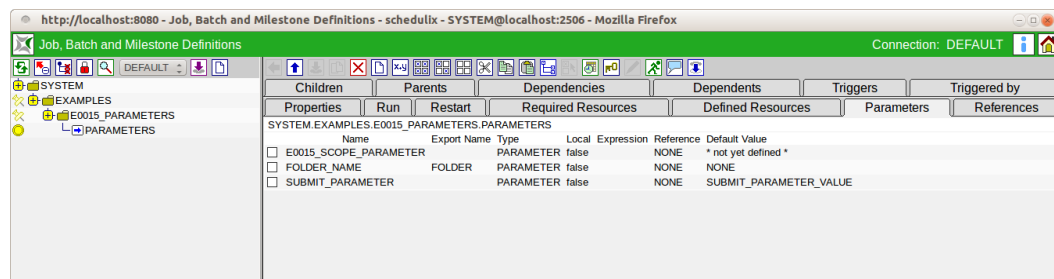


Figure 13.31: Batches and Jobs; Parameters tab

All the parameters that have been currently defined in this Scheduling Entity are shown in the list above. A new parameter can be created by clicking the *New* but-



ton. This new parameter is then shown in the list after you have entered the data and return variable. Selected parameters can be deleted as well by clicking *Delete*. You can switch to the "Parameter Details" tab by clicking the parameter name. The columns in the "Parameters" tab are explained in the next section.

### 13.5.10.1 Parameter Details tab

The "Parameter Details" tab opens when you select a parameter in the Parameters tab or you want to create a new parameter by clicking the *New* button. All the details for a parameter can be entered and modified here.

The "Parameter Details" tab looks like this:

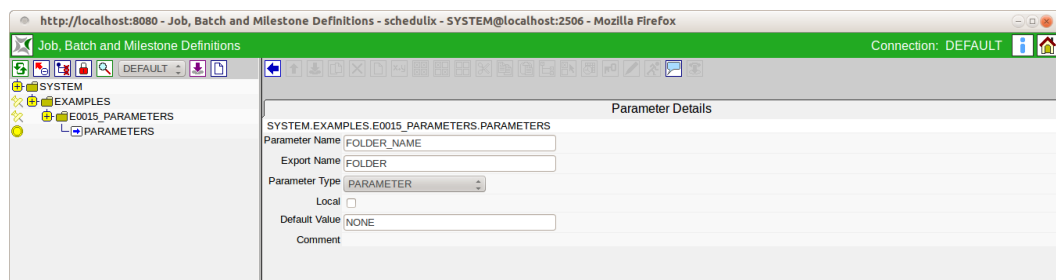


Figure 13.32: Batches and Jobs; Parameter Details

The fields in the "Parameter Details" tab have the following meanings:

**Parameter Name** This is the name of the parameter. All the parameter values are exchanged between the individual jobs using this name. The parameter value is set by the first job and can be used in the second job under the same name.

**Parameter Type** This is the parameter type and indicates how the parameter is used.

Options for "Parameter Type"

1. PARAMETER

Parameters of the type "Parameter" can optionally be specified at the time of the submit if a default value has been defined. They have to be specified if a default was not defined. This only applies for the Master Batch parameters, however. All other parameters behave as if they were of the type "Import".

2. IMPORT

The value must only be resolvable at runtime. The usage of this variable is explicitly documented.

### 3. RESULT

This is the result of a job that can be exported to a follow-on job.

### 4. CONSTANT

This parameter is a constant that can no longer be changed. The value of the constant is entered using the field *Default Value*. This value is mandatory for parameters of the type "Constant".

### 5. EXPRESSION

An Expression parameter is a special form of parameter for parallel processing using dynamic child jobs. To obtain an aggregate of all the children in such cases, an aggregate function can be selected in the supplementary field that is displayed, and the child parameter that is to be aggregated is entered in the field "Expression Parameter Name".

If the Expression parameter is used at runtime, all the instances of the child job are searched for the value of this parameter and the corresponding aggregate function is performed.

Here is an example illustrating this:

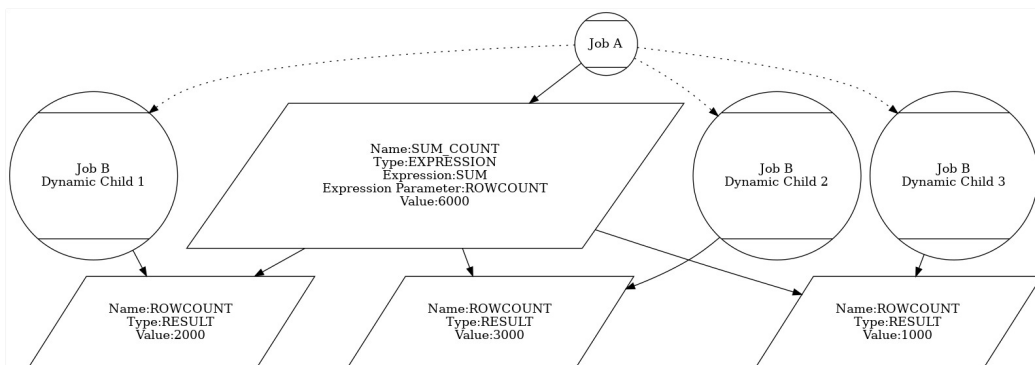


Figure 13.33: Example using Expression parameters

Job A has a dynamic Child Job B, which is to be simultaneously submitted at runtime. The Child Jobs B write all the x-rows in a table in parallel. Child Job A now wants to know how many rows have been written in total by all the child jobs.

To achieve this, Job B must have defined a parameter (type Result) for instance called LINECOUNT, which contains the number of processed rows on completion of the job run.

In Job A, a parameter TOTALCOUNT is now created with the type "Expression", the expression option "SUM" and the parameter name "LINECOUNT".

Job A starts now and submits 3 Child Jobs B (B1, B2, B3). B1 writes 2000 rows, B2 writes 3000 rows, and B3 writes 1000 rows. When the children have finished, Job A (or a follow-on job) queries the TOTALCOUNT parameter and receives the result 6000 rows, which corresponds to the sum total of all the children.

#### 6. REFERENCE

The type "Reference" allows a parameter to be allocated as a reference to another parameter. This parameter is performed with a cross-search of all the jobs on the same level as the current job. This enables you to obtain parameters for jobs which are not immediate parents of the current job. The parameter that is to be referenced must be inserted into the *Reference* field by clicking *Paste (Copy from the Parameters tab of another Job Definition)*.

#### 7. CHILDREFERENCE

The type "Childreference" allows a parameter to be allocated as a reference to another parameter. This referenced parameter is determined by running a search in all the children of the current job. This enables parameters from the job's own children to be identified.

#### 8. RESOURCEREFERENCE

The type "Resourcereference" allows a parameter to be allocated as a reference to a Resource Parameter. The parameter that is to be referenced must be inserted into the *Reference* field by clicking *Paste (Copy from the Parameters tab of a Named Resource)*.

**Expression** This field is only visible if the value "Expression" has been selected in the field *Parameter type*.

The type of aggregate function can be selected here.

The following rule applies for all aggregate functions: Should the parameter or parameter value not exist or be of the wrong type (string instead of a number) in some child objects, these values are then ignored and only the available and correct values are aggregated.

The following aggregates are available:

##### 1. SUM

The sum of all the values for the referenced parameter for all the child elements.

##### 2. AVG

The average of all the values for the referenced parameter for all the child elements which contain valid and existing values.

3. MIN

The minimum of all the values for the referenced parameter.

4. MAX

The maximum of all the values for the referenced parameter.

5. COUNT

The number of child instances that have defined this parameter.

6. NONE

An aggregate function has not been selected yet. None is not a valid value.

**Expression Parameter Name** This field is only visible if the value "Expression" has been selected in the field *Parameter type*.

This is the name of the child parameter for which the aggregate is to be calculated. More information about aggregates can be found in the *Expression* field.

**Local** If a check mark is set here, this parameter is only visible to the job itself.

**Default Value** The *Default Value* specifies the default value. This value is returned when the parameter is resolved if an explicitly set value could not be found during the search.

This field is optional. A value must be entered here in the case of a constant.

Variables are evaluated recursively. This means that if a variable contains a string such as \$NAME or \${NAME}, the variable NAME is resolved and the value is inserted in its place. This means it has now also become possible - albeit indirectly - to access system variables for other jobs. The \$ character is represented by \\$, and \\ is used for a backslash.

**Comment** A more detailed explanation about the object can be entered in the *Comment* field.

**Reference** This field is only visible if the value "Reference", "Childreference" or "Resourcereference" has been selected in the field *Parameter type*.

A parameter can be entered here with *Copy and Paste* which is referenced using the selected name.

### 13.5.10.2 Predefined default parameters for the runtime system

Predefined default parameters in the runtime system can be used by all Submitted Entities in the field *Run Program* as a transfer parameter or in the run program itself. The predefined default parameters contain information about the Submitted Entity and its environment.

The following default parameters are provided by the runtime system:

**Job environment parameters** These are all the parameters that relate to the Submitted Entity itself. Job environment parameter names:

1. JOBNAME

This is the name of the job that was entered in the *Name* field.

2. JOBID

This is the Id of the Submitted Entity. It enables the current instance of the entity to be unequivocally identified.

3. KEY

The *Key* is the password for the respective job. It is used to log onto the server.

4. MASTERID

The *MASTERID* is the ID of the Master Job that is executing the current job.

5. PID

The *PID* is the process ID of the Submitted Entity at operating system level. This can be used for the kill program, for instance.

6. LOGFILE

The LOGFILE parameter specifies the log file defined in the *Logfile* field in the [Batches and Jobs dialog](#) that is to be used.

7. ERRORLOG

The ERRORLOG parameter specifies the error log file defined in the *Error Logfile field* in the [Batches and Jobs dialog](#) that is to be used.

8. SDMSHOST

The SDMSHOST parameter specifies the current host of the SDMS server.

9. SDMSPORT

The SDMSHOST parameter specifies the current port of the SDMS server on the host.

10. JOBTAG

The JOBTAG specifies the Child Tag for the Submitted Entity. This enables a dynamically created child entity to be unequivocally identified.

11. SUBMITTIME

The SUBMITTIME specifies the time the Master Job is submitted.

12. STARTTIME

The STARTTIME is the time when the Submitted Entity was actually started on the jobserver (if the entity is a job).

13. SEID

The SEID is the ID of the Scheduling Entity.

14. WORKDIR

WORKDIR is the path to the working directory.

15. JOBSTATE

The JOBSTATE is the Exit State of the job, batch or milestone.

16. MERGEDSTATE

The MERGEDSTATE is the resultant Exit State of a job and its children.

17. EXPRUNTIME

The EXPRUNTIME is the Expected Runtime as stored in the Job Definition.

18. EXPFINALTIME

The EXPFINALTIME is the Expected Final Time as stored in the Job Definition.

19. PARENTID

The PARENTID is the ID of the parent job, milestone or batch.

20. STATE

The STATE is the current state of the job.

21. ISRESTARTABLE

ISRESTARTABLE states whether the job is restartable or not.

22. SYNCTIME

The SYNCTIME is the time for Synchronize Wait.

23. RESOURCETIME

The RESOURCETIME is the time for Resource Wait.

24. RUNABLETIME

RUNABLETIME is the time for Runnable.

25. FINISHTIME

FINISHTIME is the end time for the process.

26. SYSDATE

The variable SYSDATE is the current date and time.

27. LAST\_WARNING

This variable contains the text from the last audit entry that resulted in the warning count being raised.

28. RERUNSEQ

This variable states how often the job has already been restarted.

29. SCOPENAME

This variable delivers the full path name to the jobserver that is executing or has executed the job.

**Trigger parameters** There are several predefined parameters that can be used in scripts for trigger jobs. These parameters are only available if the Submitted Entity was started as part of a trigger. If the entity was submitted in the normal way, these parameters are not fulfilled.

Trigger environment names:

1. TRIGGERNAME

This is the name of the trigger.

2. TRIGGERTYPE

This enables a job to perform different actions based on the trigger type.

3. TRIGGERBASE

The TRIGGERBASE parameter contains the name of the Triggering Object, i.e. of the job that sends a trigger to a parent by means of an Exit State Translation, for example.

4. TRIGGERORIGIN

The TRIGGERORIGIN parameter specifies the name of the entity that contains the trigger. This is the job that is shown the trigger in its Triggers tab.

## Editor for Job Definitions

### 5. TRIGGERREASON

The TRIGGERREASON parameter specifies the name of the object that activates the trigger.

### 6. TRIGGERBASEID, TRIGGERORIGINID, TRIGGERREASONID

The ID of the Submitted Entity is saved in this parameter for each of these three objects (TRIGGERBASE, TRIGGERREASON, TRIGGERORIGIN).

### 7. TRIGGERBASEJOBID, TRIGGERORIGINJOBID, TRIGGERREASONJOBID

The ID of the Job Definition is saved in this parameter for each of these three objects (TRIGGERBASE, TRIGGERREASON, TRIGGERORIGIN).

### 8. TRIGGEROLDSTATE

This parameter specifies the Job State prior to the activating moment that is relevant to the trigger.

### 9. TRIGGERNEWSTATE

This parameter specifies the Job State after the activating moment that is relevant to the trigger.

### 10. TRIGGERSEQNO

TRIGGERSEQNO is the number of times the trigger was activated.

## 13.5.11 References tab

All references (parameters of the types REFERENCE and CHILDREFERENCE) from other job or batch objects to parameters for the current object are shown in the "References" tab.

The "References" tab looks like this:

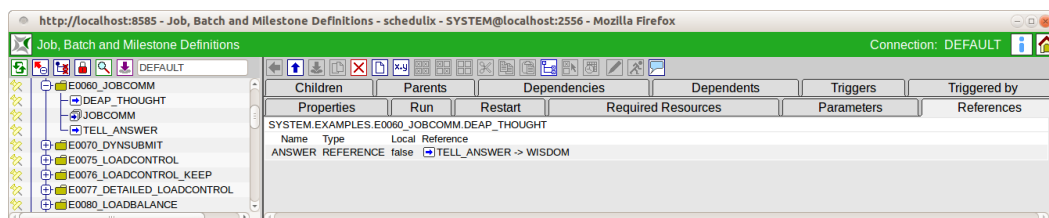


Figure 13.34: Batches and Jobs; References tab



### 13.5.12 Triggers tab

The “Triggers” tab describes all the triggers that are defined for this Scheduling Entity. A trigger is used to start another executable object if a configurable event has been reached in the currently selected job (or one of its children, see below). For example, messages could be sent when a job has been completed. Follow-up processing or error handling routines can also be started automatically.

The “Triggers” tab looks like this:

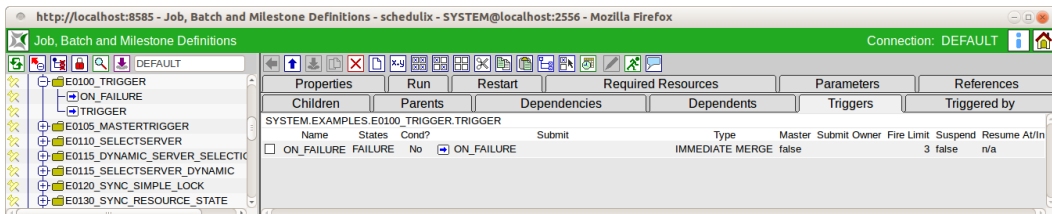


Figure 13.35: Batches and Jobs; Triggers tab

The “Triggers” dialog shows a list of all the triggers assigned for the selected task. The job to be started by the trigger can be selected and added to the list by copy and paste. The name for the trigger is taken from the name of the job when it is copied. The trigger details can then be viewed and modified by clicking the trigger’s name. The trigger is deleted by clicking the *Drop* button. The columns in the list have the following meanings:

**Name** This is the name of the trigger. When you add a new job to the list of triggers, the trigger name is taken over from the job name. The “Trigger Details” tab is opened by clicking the trigger name. All the data (including the trigger name) can be viewed and modified here.

**States** This is a comma-separated list of all the states that are to be taken into account for the trigger.

**Cond?** The *Cond?* field indicates whether a condition has been specified.

**Submit** This is the name of the job or batch that is to be submitted using the trigger. Important: This field can also be selected with a mouse click. Doing so opens the edit mask for the respective job. If you have added or modified any data and confirmed the confirmation query, all the modified data is lost.

**Param?** The *Param?* field indicates whether trigger parameters have been specified.

The other columns are explained in the next section.

### 13.5.12.1 Trigger Details tab

The “Trigger Details” tab is opened by selecting a trigger in the “Triggers” tab. All the detailed information regarding the trigger can be modified or entered here.

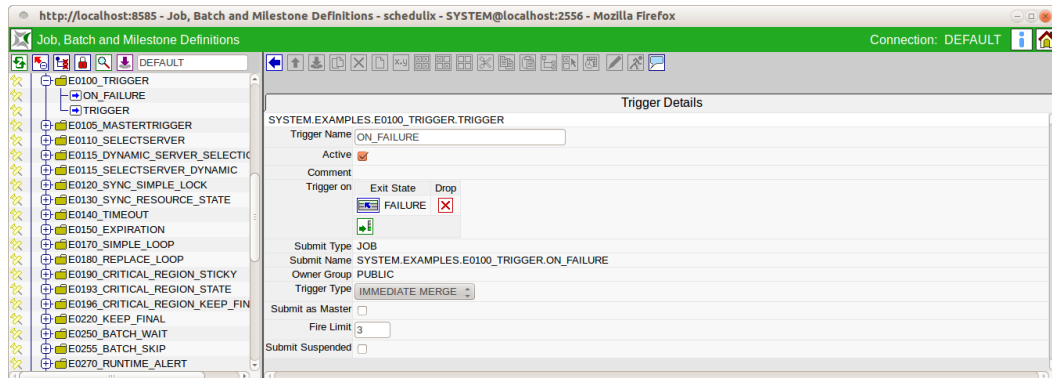


Figure 13.36: Batches and Jobs; Trigger Details tab

The tab contains the following fields:

**Trigger Name** This is the name of the trigger. When you create a new trigger (i.e. when you add a job to the trigger list), the job name is taken over for it automatically. The name can then be changed as required in the “Trigger Details” tab.

**Active** States whether the trigger is active. If this switch is not set, the trigger will not be activated when the trigger event occurs. An inactive trigger is indicated in the trigger list by a black hand (Suspend symbol).

**Trigger On list** A list of Exit States for the selected process (not the job that is to be triggered) for which the trigger is to be activated can be defined here. If no list is specified, the job’s Exit State is irrelevant and the trigger is activated whenever the achievable event occurs.

**Example:**

A trigger is to be activated when a job finishes with an error (EXIT\_STATE: FAILURE). In this case the Trigger On list must have selected the state FAILURE. Other states (SUCCESS, etc.) are not listed because they are to be ignored when the trigger is activated.

If a trigger is always to be activated when the process ends (regardless of the state, which is perhaps only differentiated in the job that initiates the trigger), the “Trigger on” list can be left empty or filled with any states that the job can take on.

**Condition** The *condition* that has to be fulfilled is entered in the *Condition* field.

**Submit Type** This is the type of the object (batch or job) that is started when the trigger is activated.

**Submit Name** This is the path and name of the object that is started when the trigger is activated. The full name of the object with all the parent folder hierarchies is used here.

**Parameters** The *Parameters* table makes it possible to specify which parameters are handed over when submitting the batch or job. The parameter name and an expression for determining the value that are to be handed over are defined in the columns (Name) and *Expression*. Further details can be found in the documentation of the command language.

**Owner Group** The owner of the trigger is defined in the field *Owner Group*.

**Trigger Type** This is the type of event that is to activate the trigger. The following events are used:

1. Immediate Merge

The job itself or one of its children has taken on a state in the 'Trigger On' list. If no value is entered, the trigger is activated with each state transition.

**Example:**

A text message is to be sent if the job or one of its children contains an error.

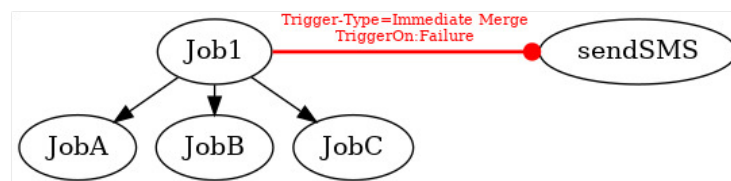


Figure 13.37: Example of an Immediate Merge trigger

2. Immediate Local

The job itself has taken on a state in the 'Trigger On' list. If no value is entered, the trigger is activated when the job finishes. The state of the children is ignored.

**Example:**

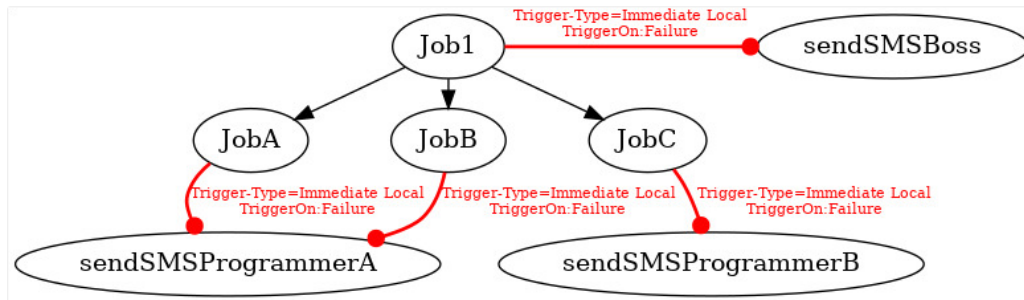


Figure 13.38: Example of an Immediate Local trigger

A text message is to be sent to the project manager if the job contains an error. If one of the job’s children contains an error, a second trigger is implemented in the child which sends a text message to the responsible programmer.

3. Before Final

The trigger condition is evaluated shortly before the Scheduling Entity would reach a Final State. Here, the Trigger On list only needs to contain Final States (or none, in which case the trigger is activated for every Final State instance), as otherwise the trigger will not be activated. If the Scheduling Entity that is to be triggered is not submitted as a master, the Scheduling Entity can only attain a Final State after the Scheduling Entity started by the trigger has also attained a Final State.

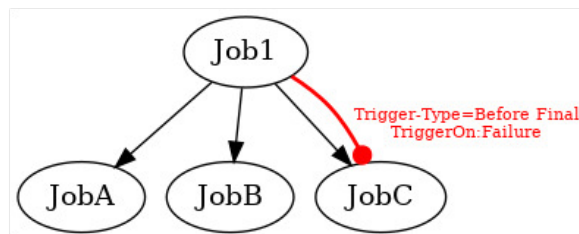


Figure 13.39: Example of a Before Final Trigger

In the example above, Job C will be called again if the job fails. Since this is a Before Final trigger, Job1 cannot attain a final state for as long as JobC has been terminated without a FAILURE.

4. After Final

The trigger is activated shortly after the job has reached a Final State. Here, the Trigger On list should only comprise Final State (or none, in which case

the trigger is activated for every Final State instance), as otherwise the trigger will not be activated.

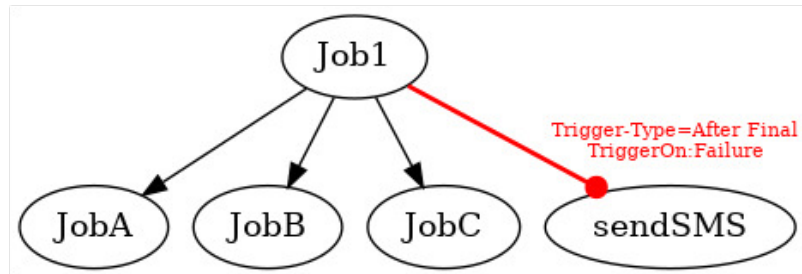


Figure 13.40: Example of an After Final Trigger

The legend for the graphic can be found in Chapter 1.8.

### 5. Finish Child

The trigger is activated when a direct or indirect child of the triggering job has attained a state in the Trigger On list. This allows a single point of monitoring to be implemented.

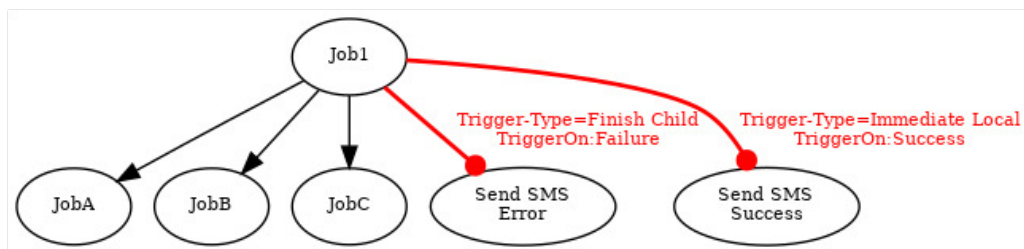


Figure 13.41: Example of a Finish Child Trigger

In the example above, a text message (inform programmer) would be sent if one of the child jobs takes on a Failure State. If one's own job (Job 1) is successful, the text message (well done!) is sent.

## 6. Until Finished

The Until Finished trigger periodically checks the condition. If the Condition is "True", the trigger continues to be activated until the "Finished" state has been achieved. The condition is then evaluated for the last time and, if it is evaluated to "True", the trigger is activated.

## 7. Until Final

The Until Final trigger periodically checks the condition. If the Condition is "True", the trigger continues to be activated until the "Final" state has been achieved. The condition is then evaluated for the last time and, if it is set to "True", the trigger is activated.

**Submit as Master** If the "Submit as Master" flag has been set, the job that is started by the trigger is submitted as its own Master Job and does not have any influence on the current Master Job run of the triggering job. If the flag is not set, the triggered job is started as a child of the job to be triggered. This also means that the job in which the trigger is activated is not completed (cannot attain a Final State) as long as the triggered job is still running. The only exception is the trigger type "After Final", where the trigger is only activated after the triggered job has attained a Final State. If this is the case, the triggered job is started as a child of the parent of the triggering job. If the triggered job is the same Scheduling Entity as that of the triggering job, the triggered job takes the place of the triggering job.

**Fire Limit** This defines the maximum number of trigger activations for this trigger within a workflow. When the maximum number of trigger activations is reached, no more triggers are activated and no more new jobs are started.

**Submit Suspended** Defines whether the triggered Scheduling Entity is to be started as being suspended.

**Resume** This field is also only displayed if the option *Submit Suspended* has been enabled. Here you can choose whether a workflow should be resumed automatically.

The following options are available:

- NO: disables this function.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time** This field is only displayed if "AT" was selected for *Resume*. The required resume time is entered here in the format YYYY-MM-DDTHH:MI:SS. This format is based on the ISO standard 8601 and also permits an incomplete entry. If you enter 'T16:00', the job will be resumed at 16:00 hours (starting from the time the trigger was activated).

**Resume In** This field is only displayed if "IN" was selected for *Resume*. Here you can specify how many time units (see *Unit*) the system is to wait for until the resume action is triggered.

**Unit** This field is only displayed if "IN" was selected for *Resume*. This field is used to define whether the entry in *Resume In* is in minutes (MIN), hours (HOUR) or days (DAY).

### 13.5.13 Triggered by tab

The "Triggered by" tab lists all the triggers that submit this Scheduling Entity. The "Triggered by" tab looks like this:

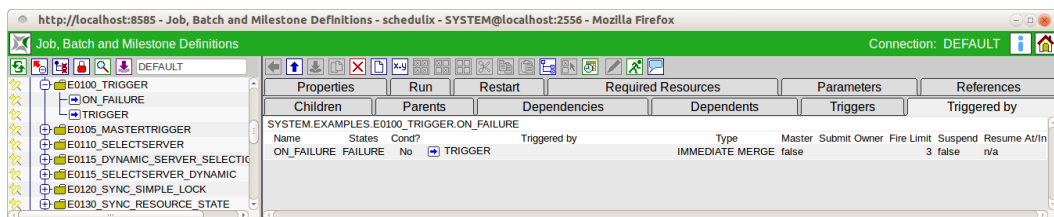


Figure 13.42: Jobs and batches; Triggered By tab

The columns in the list have the following meanings:

**Name** This is the name of the trigger.

**States** This is a comma-separated list of all the states that are to be taken into account for the trigger.

**Cond?** The *Cond?* field indicates whether a condition has been specified.

**Triggered by** This is the name of the job, batch or milestone that defines the trigger.

Important: This field can also be selected with a mouse click. Doing so opens the edit mask for the respective job. If you have added or modified any data and confirmed the confirmation query, all the modified data is lost.

The meanings of the other columns can be found in the chapter "Trigger Details tab".

## 13.6 Job Hierarchy navigation

The Job Hierarchy navigation window can be viewed for each batch and job in the "Batches and Jobs" dialog. It looks like this:

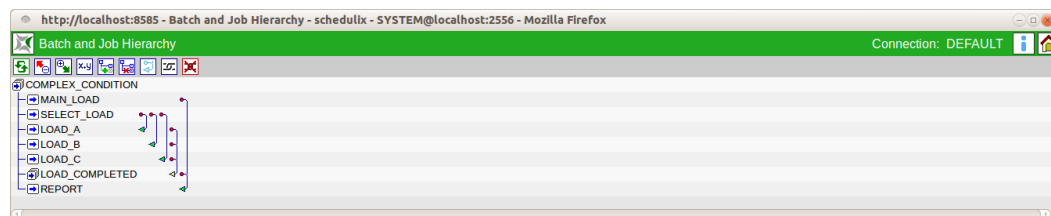


Figure 13.43: Batches and Jobs; hierarchy view

All the children of the current entity are displayed in the navigation window. If the children also have a child hierarchy, this can be expanded as well until the entire hierarchy of the object is visible.

The standard buttons are supplemented by the following buttons:



### Add Children

Children can be inserted directly in the Job Hierarchy navigation window by clicking the *Add Children* button. To do this, click the button followed by the job or batch that is to acquire the children. A mask opens where you can selected one or more children. Having selected the children, close the mask and then click the parent again. The children are added to it automatically.



### Remove Child

Children can be deleted by clicking the *Remove Child* button. To do this, click the button and then the parent. The children are deleted automatically after you have clicked "OK".



### Show Dependencies

The *Show Dependencies* button has two functions. By default, the dependencies are



## Job Hierarchy navigation

not displayed in the dialog window and the *Show Dependencies* button appears in the button bar.

The mutual dependencies between the Scheduling Entities can be graphically displayed by clicking the *Show Dependencies* button. Each dependency is indicated by an arrow. The required Scheduling Entity is represented by the start of the arrow (the round part of it) and the dependent Scheduling Entity is the arrowhead.

**Example:**

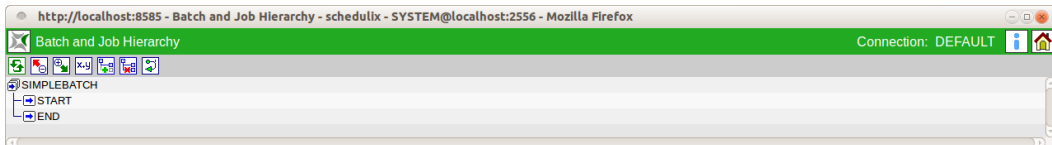


Figure 13.44: Hierarchy view with dependencies

In the example above, the job END is dependent upon the job START.

If a job is dependent upon multiple objects, an arrow is displayed with several shafts (the round parts of the arrow) representing all the required Scheduling Entities. The arrowhead remains the dependent Scheduling Entity.

**Example:**

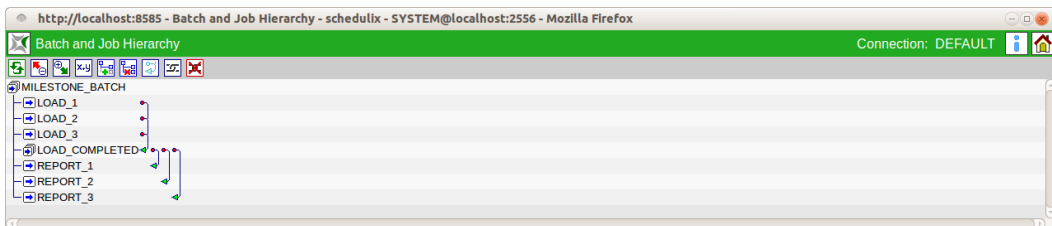


Figure 13.45: Hierarchy view with multiple dependencies

In this example, the batch LOAD\_COMPLETED is dependent upon the LOAD\_1, LOAD\_2 and LOAD\_3.



### Hide Dependencies

A *Hide Dependencies* function is the second state of this button. This appears if the *Show Dependencies* button was clicked causing the display of the dependencies to be activated. The dependencies view can be disabled again by clicking the *Hide Dependencies* button.



### Chain

With the *Chain* button you can switch to the Chain mode, where job dependencies

## Batch in Folder Convention

can be added directly in the Job Hierarchy navigation window. To do this, first click the *Chain* button followed by the Dependent Job and the Required Job. This creates a dependency relationship between the two jobs.

You can exit the Chain mode by clicking the *Cancel* button.

### Unchain

The *Unchain* button is used to revoke dependency relationships. To do this, first click the *Unchain* button followed by the dependency that is to be revoked.

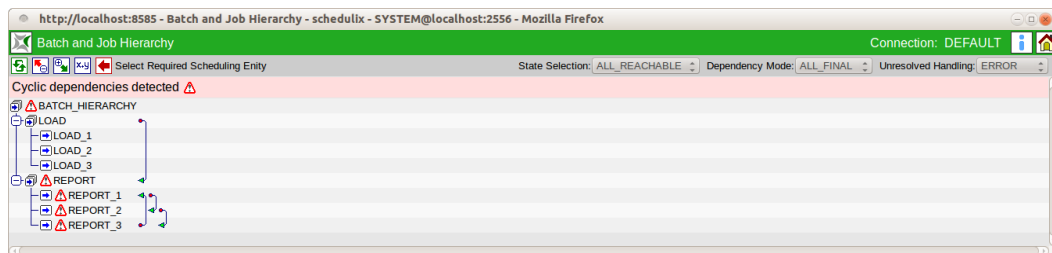


Figure 13.46: Cyclic dependencies

Cyclic dependencies can be easily created with the *Chain/Unchain* buttons. These are identified and reported by the front end. See Figure 13.46.

## 13.7 Batch in Folder Convention

To enhance the comfort and overview when working with the system, we recommend to follow the following convention:

For every batch a folder with identical name should be created to contain the batch object and all immediate children. If a child itself has children (like a sub batch), according to convention, another folder should be created in the parents folder, to contain the child and its direct children. Children which are reused in more than one parent can reside in different folders.

The schedulix Web GUI supports this convention in a number of ways if the *Enable Batch In Folder Convention Support* checkbox of the schedulix!Web User settings is selected.

1. On creation of a folder, the checkbox *Create Batch in Folder* controls whether a batch with the same name should be created inside the new folder.
2. On creation of a batch, the checkbox *Create Batch Folder* controls whether a folder with same name should be created to contain the new batch.
3. On creation of objects in a folder, which follows the convention (containing a batch or job with identical name), the checkbox *Add as Child* controls whether

## Batch in Folder Convention

the new object should be added as a child of the batch with the same name as the folder.

4. On changing the name of a folder, the checkbox *Rename in Content* whether to rename the batch/job with identical name inside the folder.
5. On changing the name of batch or job in folder with identical name, the checkbox *Rename Parent Folder* controls whether the parent folder is renamed too.
6. On changing the name of an object, the button *Clone* allows to create a copy of this object with a new name. In this case all relevant checkboxes to control the batch in folder convention will be displayed as well.
7. When copy/cut with a following paste is done in "Content" tabs of folders, depending on whether source and target folders and selected objects are following the batch in folder convention, a list is displayed on paste. This list shows the actions on parent/child relationships, which will be taken to follow the convention. These actions can be (de)selected individually before the paste operation is executed.



# 14 schedulix!Web Users

## 14.1 View

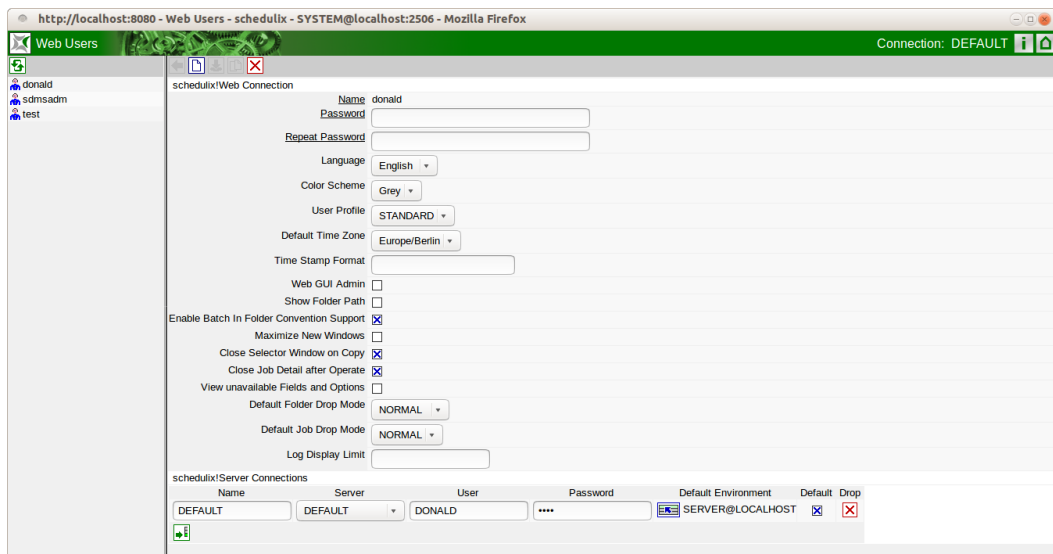


Figure 14.1: User management for the Web front end

## 14.2 Concept

### 14.2.1 In short

The user management module is used to define schedulix interface users. All users who want to work with the schedulix interface should have their own user ID. The user must log in with this user ID and a personal password in the [Login window](#).

### 14.2.2 Detailed description

In addition to the display options and security mechanisms, the connection to the server can also be maintained in this dialog window. The schedulix Server uses its own user management functionality which is independent of the user management module for the interface system. For this reason, it is possible for all interface users

to utilise just one server user, or the interface users can be mapped 1:1 on the server as well. Always notify your system administrator when making changes to your user account.

## 14.3 Navigator

All the user IDs are displayed in the navigation window.

## 14.4 Editor

The editor is used to maintain all the data for a selected user.

### 14.4.1 schedulix!Web Connection

The fields under Web-GUI-Connect have the following meanings:

**Name** The user's login name.

The login name is entered as the user name in the [Login](#) dialog.

**Password** The *Password* authenticates each user who logs into the schedulix System and has to be entered in the [Login](#) dialog. The password is hidden on the screen and must be identical to the *Repeat password*.

Since the password is not displayed in the dialog, it is not evident whether a password has already been entered or not. It only has to be entered once, however.

**Repeat Password** The *Repeat Password* must be the same as the password you entered in the Password field.

It has to be entered twice because a typing error when entering it just once can cause problems. Since it is unlikely that the user will make the same typing error twice, it is assumed that if the password is twice entered correctly, then the intended term has been used.

**Colour Scheme** The colour scheme for the user interface is selected here.

**User Profile** The user's profile is entered here.

**Default Time Zone** This parameter defines the default time zone when creating new time schedules. All time stamps continue to be displayed in this time zone.

**Time Stamp Format** A format other than the default format (%Y.%m.%d %H:%M:%S example 2014.06.18 14:06) for displaying the time stamps can be set here. The format corresponds to the strftime C standard function.

**Web-GUI-Admin** This check mark must be set if this user is an administrator for the web interface.

**Show Folder Path** If this flag is set, all the folders and Scheduling Entities with their full path names are displayed when opening new windows.

**Maximize New Windows** New dialog windows are always maximized when opening them if this check mark has been set. If it is not set, the window always opens in the normal size.

**Close Selector Window on Copy** If this check mark is set, windows that are opened for selecting objects are closed automatically after clicking the *Copy* button.

**View unavailable Fields and Options** If this check mark is set, the unavailable fields and options are displayed.

**Default Folder Drop Mode** This is the mode that is used for deleting a folder in the [Batches and Jobs](#) dialog window. The following options are available:

1. NORMAL

In this mode it is only possible to delete a folder if it is empty, i.e. it does not have any subfolders and no jobs, batches or milestones.

This is the "default" value for the system.

2. CASCADE

In this mode, the folder is deleted in a cascade. All the subfolders and all the jobs, batches and milestones in them are deleted.

Important: This setting is to be used with extreme caution because it is possible to delete a large number of objects at once without actually intending to do so.

**Default Job Drop Mode** This is the mode that is used for deleting a job (or a batch or milestone).

OPTIONS

1. NORMAL

In this mode, it is only possible to delete a job, batch or milestone if no other jobs are dependent upon it. The Scheduling Entity must not have any more child relationships either.

This is the "default" value for the system.

## 2. FORCE

In this mode, the scheduling entity is deleted even though other Scheduling Entities are dependent upon it and it still has some child relationships.

### 14.4.2 schedulix Server Connections

One or more server connections can be configured here.

On the Main Desktop, the server connection that is to be used for the next window to be opened can be selected in an Options field in the window header.

After editing the server connections, it may be necessary to reload the Main Desktop in the browser to make sure that the server connections are displayed correctly in this field.

**Name** The name of the server connection for selecting it in the header of the Main Desktop.

**Server** The name of the server as entered by the schedulix!Web system administrator under /Custom/SDMSServers (in Zope).

**User** The name of the schedulix Server user account. This does not have to be the same as the Web user's name.

**Password** This user ID and password are used to connect the schedulix interface with the schedulix Server.

The password is hidden on the screen.

**Default Environment** The Default Environment is entered by default for jobs.

**Default** If this check mark is set, this server connection is used as the default setting. It must be set for precisely one row.

**Server Default** If this check mark is set, this server connection is used as the default setting only if the server is known. It must be set for each server in precisely one row.



# 15 schedulix Server Users

## 15.1 View

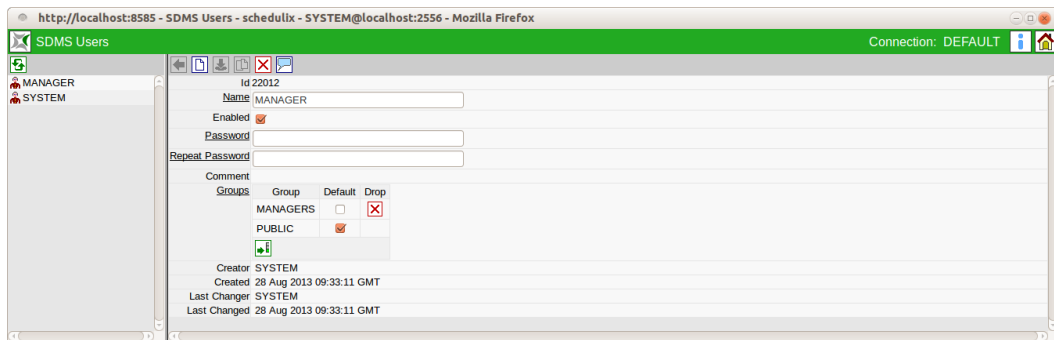


Figure 15.1: schedulix Server user management window

## 15.2 Concept

### 15.2.1 In short

The schedulix Server user is required for legitimisation toward the Scheduling Server.

### 15.2.2 Detailed description

Since the Scheduling Server is not just operated using the web interface described in this documentation, but also with the aid of other interfaces such as *sdmsh*, access controls are required on the server. The users and their group memberships can be maintained using the dialog described here.

## 15.3 Navigator

All the users are displayed in the navigation window.

The Navigator pane is only displayed if the active user is a member of the "ADMIN" group. All other users will only see the editor that can be used to change their own password and the default group they need to choose.

## 15.4 Editor

The editor is used to maintain all the data for the selected or new user. SDMS users can only be edited by users who belong to the "ADMIN" group. All the input fields are "read only" for all other users.

The editor contains the following fields:

**Name** The user's login name.

The login name is entered as the user name in the [Login](#) dialog.

**Enabled** If the active user is a member of either the "ADMIN" group, this option is used to define whether the selected user is permitted to log in.

**Password** The *Password* authenticates each user who logs into the schedulix System and has to be entered in the [Login](#) dialog.

The password is hidden on the screen and must be identical to the *Repeat Password*. Since the password is not displayed in the dialog, it is not evident whether a password has already been entered or not. It only has to be entered once, however.

The maximum length for a password is 64 characters. All characters (i.e. including spaces) are allowed.

The password can always be changed if the active user is a member of the "ADMIN" group or if the user to be modified is the active user (you can obviously change your own password).

**Repeat Password** The *Repeat Password* must be the same as the password you entered in the Password field.

It has to be entered twice because a typing error when entering it just once can cause problems. Since it is unlikely that the user will make the same typing error twice, it is assumed that if the password is twice entered correctly, then the intended term has been used.

**Groups** The groups to which the user belongs are listed in the "Groups" table. The table features the following columns:

**Group** Groups to which the specified user member belongs.

## Editor

**Default** The default group is the one that is joined by a new object in the absence of a defined group.

**Drop** This button is used to delete the group membership. This action has to be confirmed.



# 16 Groups

## 16.1 View

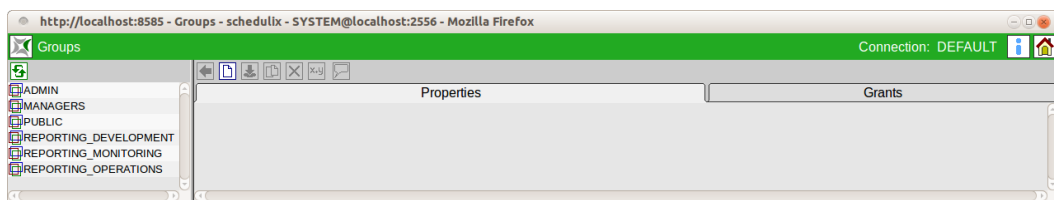


Figure 16.1: Group management

## 16.2 Concept

### 16.2.1 Description

The groups are the 'legal entity'. The individual users belong to groups and they have the same privileges held by the groups. A user can belong to several groups. The following groups always exist:

- Public: Every user is always a member of the *Public* group.
- ADMIN: Each user in the *ADMIN* group holds all privileges throughout the system.

### 16.3 Navigator

All visible groups are shown in a list in the Navigator. Visible in this context means that the user is either a member of the group or he has been granted "view" privileges for it.

## 16.4 Editor

### 16.4.1 Properties tab

This tab is used for maintaining the group properties.

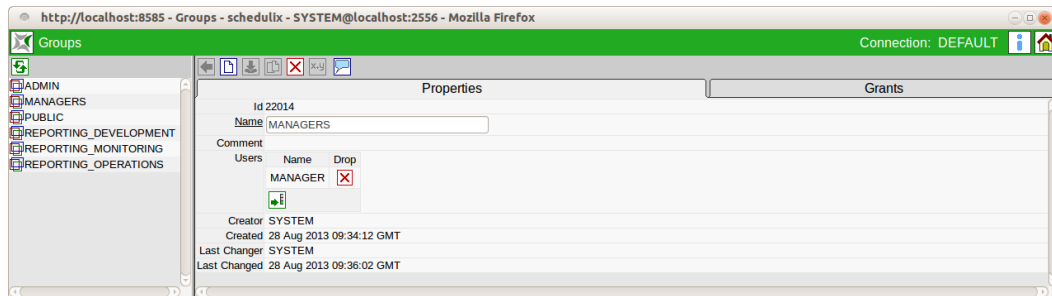


Figure 16.2: Group properties

This window contains the following fields:

**ID** The Group Id is shown in this field.

**Name** The name of the group is shown in the *Name* field.

**Users** The following field is in the Users list.

**Name** The group user is shown in the *Name* field.

## 16.4.2 Grants

The “Grants” tab shows all the objects to which group privileges have been assigned. The object type is indicated by the respective icon. The list is always sorted by the object names.

Moving the mouse pointer over the privileges field in a row opens a popup window with an explanation of the privileges string.

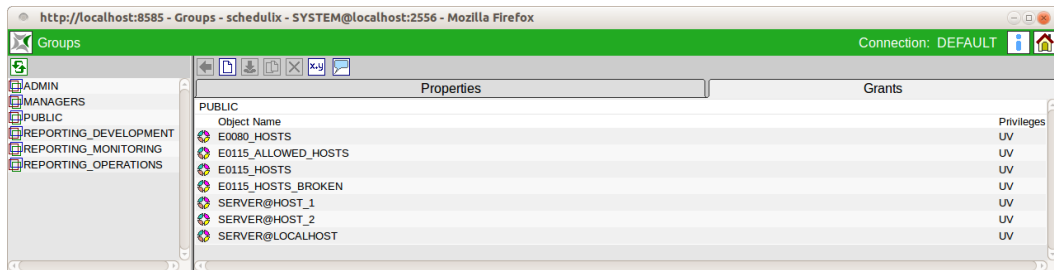


Figure 16.3: Object privileges for a group





# 17 Time Scheduling

## 17.1 View

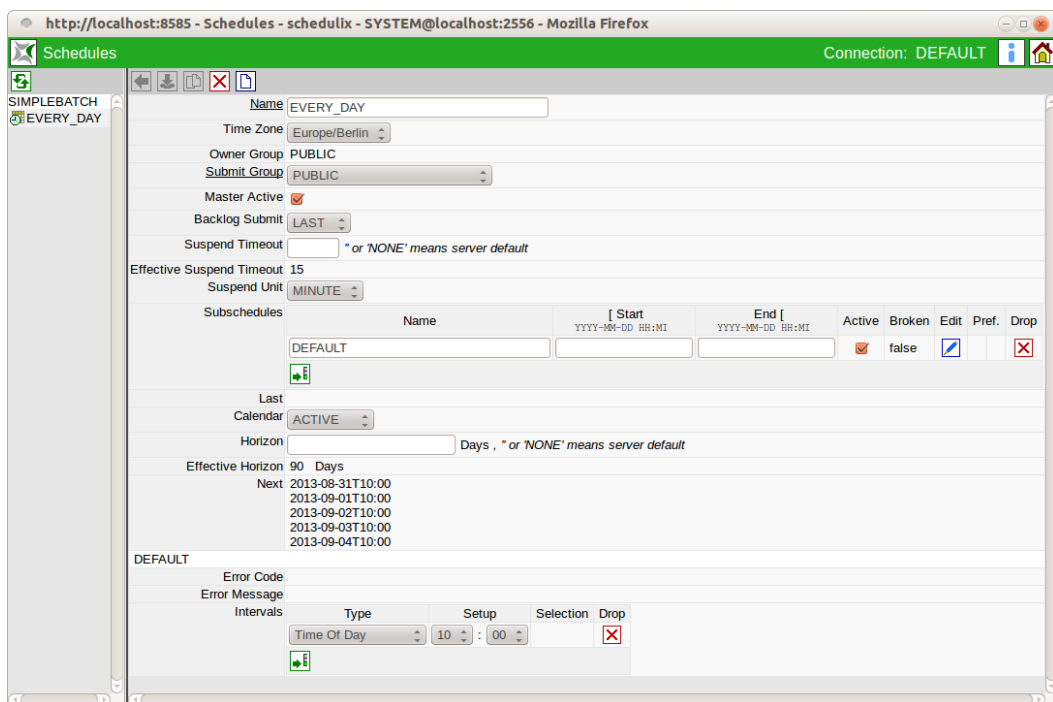


Figure 17.1: Time Scheduling

## 17.2 Concept

### 17.2.1 In short

The Time Scheduling module allows batches and jobs to be started at specific times and can be used to easily define complex schedules for jobs and batches.

A number of points in time which determine when a job is to be started is called a schedule.

Schedules always apply to just one Master Job.

### 17.2.2 Detailed description

In the "Time Scheduling" dialog window, a differentiation is made between Main Schedules, which are shown in the navigation as elements, and Sub Schedules, which can be administered within a Main Schedule.

Main Schedules are mutually independent and no interaction takes place.

Sub Schedules can be defined as being separated from the time interval or as overlapping. This allows different downtimes to be simulated or varying schedules can be set which are therefore always active.

If the schedules are to be changed on demand, this can be entered as an interim schedule. The highest priority has to be assigned to the interim plan to activate it. The original schedule is retained. The original schedule is reactivated by reassigning the highest priority to it.

## 17.3 Navigator

The Main Schedules for this Scheduling Entity are displayed in the navigation. A Scheduling Entity can have multiple Main Schedules. These Main Schedules are taken into account independently of one another. This means that the start times of all the Main Schedules are factored in.

If multiple Main Schedules have identical start times, the Scheduling Entity is started several times.

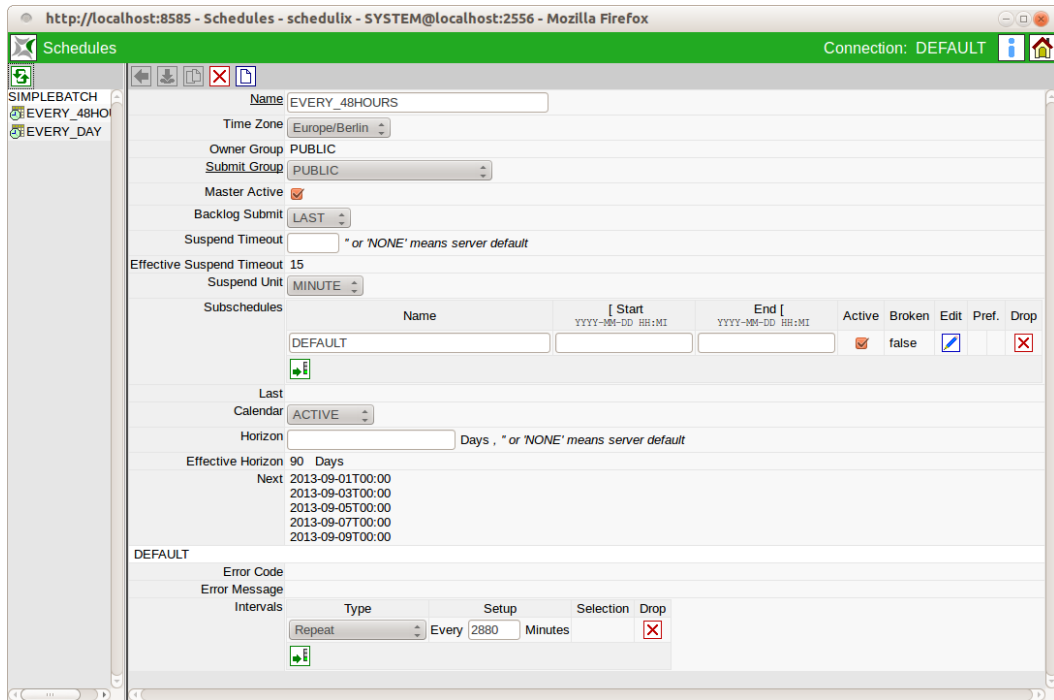


Figure 17.2: Time Scheduling Navigator

## 17.4 Editor

All the "Detail" elements for the Main Schedule selected in the navigation are displayed in the editor. Here is where Sub Schedules can be created and a list of times maintained for when the selected job or batch is to be started. The "Editor" mask looks like this:

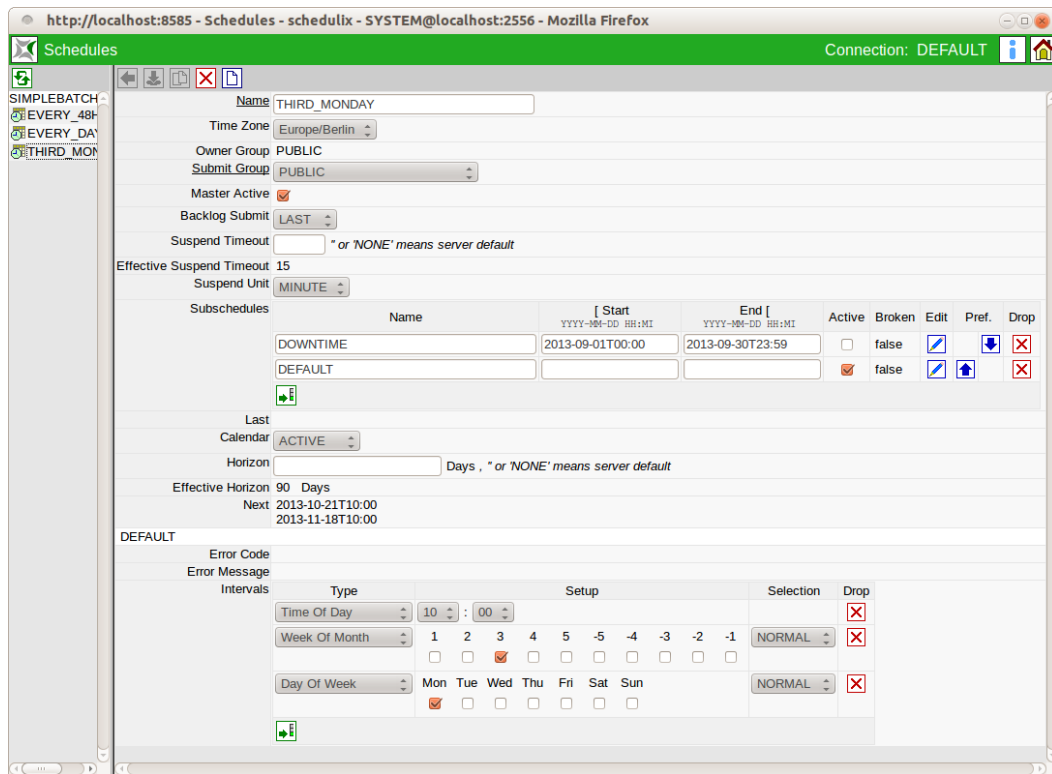


Figure 17.3: Time Scheduling Editor

The fields shown above have the following meanings:

**Name** This is the name of the Main Schedule. When you first create a new Main Schedule, it is automatically assigned the name "MASTER", but this can be arbitrarily changed. Each Main Schedule that you create has to have a different name.

**Time Zone** The time zone for which the schedule is to be calculated. Which time zones are made available here can be configured by the schedulix!Web administrator.

**Owner Group** The owner of the schedule is defined in the field *Owner Group*.

**Submit Group** The *Submit Group* determines the owner group of the submitted job.

**Master Active** If the flag for *Master Active* is set, the schedule is taken into account by the Time Scheduling System. If the flag is not set, the schedule is ignored.

**Backlog Submit** The backlog determines how the Time Scheduling System behaves after a server downtime:

- None: The "missed" jobs are no longer executed.
- Last: Only the last of the "missed" jobs is executed.
- All: All the "missed" jobs are executed.

**Suspend Timeout, Suspend Unit** Following a server downtime, a suspended job is submitted if the scheduled submit time is longer than the time entered in the fields *Suspend Timeout* and *Suspend Unit*. If no time is entered, the server-wide default time applies.

**Sub Schedules list** All the Sub Schedules of the Main Schedule are shown in the "Sub Schedules" list. The schedules are only valid within the stated timeframe. The Sub Schedules with the highest preference (at the top of the list) are taken into account first. These 'cover' the Sub Schedules further down the list for their timeframe. This means that only one Sub Schedule can be valid at any one time. The fields in the list have the following meanings:

**Name** The name of the Sub Schedule is entered in the *Name* field.

**Start** The date and time from which this Sub Schedule is valid are entered in the *Start* field. If this field is left empty, the Sub Schedule is considered to be "always valid".

**End** The date and time from which this Sub Schedule is invalid are entered in the *End* field. If this field is left empty, the Sub Schedule is considered to be "indefinitely valid".

The following buttons are available in the Sub Schedules list:

**Active** Jobs are only submitted if this flag is set. Consequently, submits can be prevented by deleting the flag. Downtimes are usually implemented as high-preference Sub Schedules with a non-set Active Flag.

**Broken** The *Broken* field is used to check whether an error occurred when the job was submitted. If this is the case, TRUE is entered in the *Broken* field for this Sub Schedule.

**Edit** Clicking the *Edit* button opens the interval description belonging to the Sub Schedule, where you can precisely define when the job is to be started.

**Pref.** The *Preference* buttons are used to set the preference of the individual Sub Schedules. The rows are shifted up or down by clicking the buttons.

**Drop** This button is used to delete the object.

**Last** The last time the job is to be executed by the Scheduling System is shown in the *Last* field. This field is empty if the job has never been started by the Scheduling System.

The date format looks like this:

YYYY:MM:DDTHH:MM

The placeholders correspond to those in the *Start* field.

**Calendar** This field determines whether the scheduled execution times for the job are to be noted in the calendar. If this is the case, the next two fields are also important.

**Horizon** The *Horizon* field determines for how far in advance entries are to be made in the calendar. The time period is entered in days.

**Effective Horizon** The configured default value applies as the horizon if the *Horizon* field is left empty. The *Effective Horizon* field shows the currently valid value.

**Next** The next scheduled time(s) for the task to be executed by the Scheduling System is/are shown in the *Next* field. If no schedules or times have been defined, this field is left empty. Once a schedule has been created and times have been defined, the closest possible date after this schedule is displayed. If no time is possible because mutually exclusive times have been defined, a value is not displayed here either.

When calendar entries are created, both the next execution time and the following four execution times (providing they lie within the horizon) are displayed.

The date format is identical to the *Start* field.

### 17.4.1 Sub Schedule Details sub-area

This sub-area shows all the "Details" information for the first Sub Schedule or a Sub Schedule you have selected by clicking the *Edit* button. The Sub Schedule Details begin in the mask with a grey line showing the name of the currently selected Sub Schedule followed by the following fields:

**Error Code** If an error occurred while the job was being executed in the Time Scheduling, the returned error code is displayed in the *Error Code* field. If no error occurred, this field remains empty.

**Error Message** If an error occurred while the job was being executed in the Time Scheduling, the returned error message is displayed in the *Error Message* field. If no error occurred, this field remains empty.

**Intervals list** If no interval is entered here (the list is empty) and the Sub Schedule is marked as being active, a job is submitted precisely once at the start time of the Sub Schedule.

A list that is not empty must contain exactly one "driving" interval of the type "Time Of Day", "Repeat" or "Calendar (Driver)". The "driving" interval defines the cycle in which a submit can potentially take place. The other "filtering" intervals determine by virtue of their filtering effect which of the cycles created by the "driving" interval actually lead to a submit.

**Type** The *Type* field defines the type of interval. Another mask is displayed in the *Setup* column depending on the interval type.

**Setup** A mask for configuring the interval is displayed here depending on the interval type.

**Selection** In the case of intervals that can be selected (e.g. "Day Of Week"), in this column you can choose either a "NORMAL" or "INVERSE" selection.

The following interval types are used:

- Repeat

This is a "driving" interval for starting the task at regular intervals according to a schedule. The number of minutes can be entered in the *Setup* field.

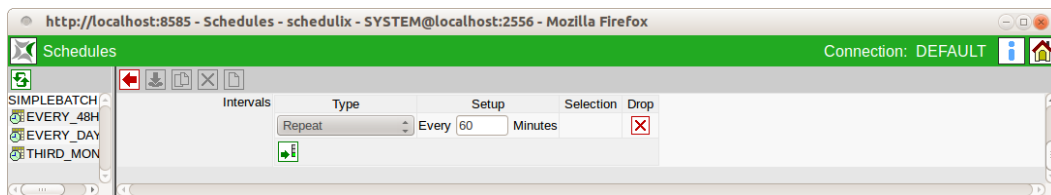


Figure 17.4: Repeat Driver

In the example, the job is started every 60 minutes.

- Time of Day

This "driving" interval type can be used to start the job at a specified time of day. The time of day is entered in the *Setup* field by selecting the hour (24H) and minutes.

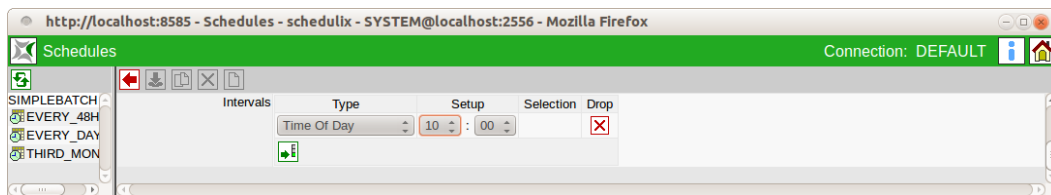


Figure 17.5: Time Of Day Driver

In the example, the job is started every day at 10:00 hours.

Several "Time Of Day" intervals can be created. This allows you to easily define multiple execution times for any one day.



## Editor

- Range of Day

This "filtering" interval type is only permitted in combination with the "driving" interval types "Repeat" and "Calendar (Driver)". It allows you to set a restriction to one or more time ranges during a day.

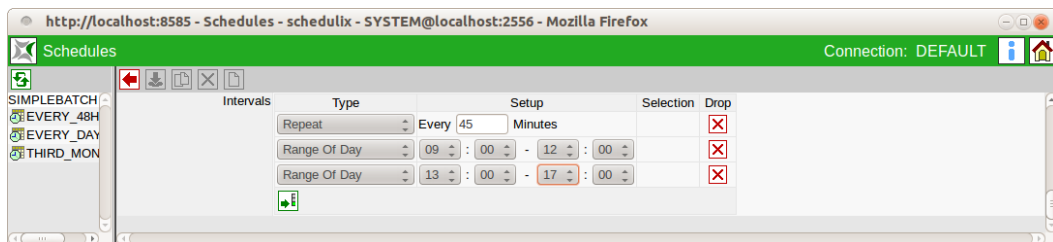


Figure 17.6: Range of Day Filter

Multiple "Range Of Day" intervals can be created. This allows multiple execution times to be easily defined.

- Day of Week

The "filtering" interval type "Day of Week" can be used to select the weekdays on which the job is to be executed. This is defined in the *Setup* field by choosing the relevant weekdays.

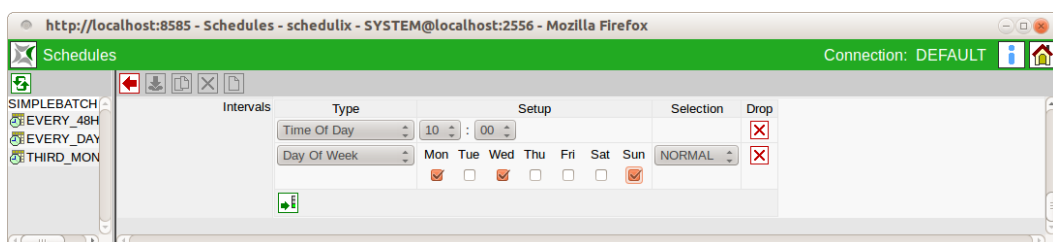


Figure 17.7: Day of Week Filter

In the example, the job is executed every Monday, Wednesday and Sunday.

## Editor

- Day of Month

The “filtering” interval type “Day of Month” can be used to select the days of the month on which the job is to be executed. This is defined in the *Setup* field by choosing the relevant days of the month.

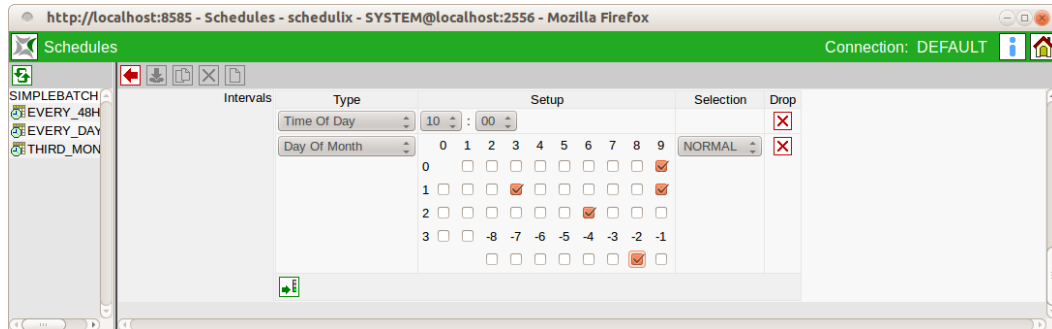


Figure 17.8: Day of Month Filter

In the example, the job is executed on the 9th, 13th, 19th, 26th and penultimate day of each month.

- Week of Month

The “filtering” interval type “Week of Month” can be used to select the week of the month in which the job is to be executed. This is defined in the *Setup* field by choosing the relevant weeks in the month.

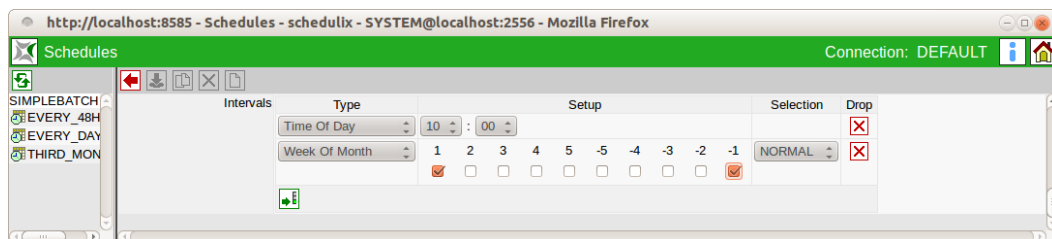


Figure 17.9: Week of Month Filter

In the example, the job is executed in the first and last weeks of each month. The first week is defined here as being the first 7 days of the month and the last week is the last 7 days of the month.

## Editor

- ISO Week of Month

The "filtering" interval type "ISO Week of Month" can be used to select the ISO week of the month in which the job is to be executed. This is defined in the *Setup* field by choosing the relevant weeks in the month.

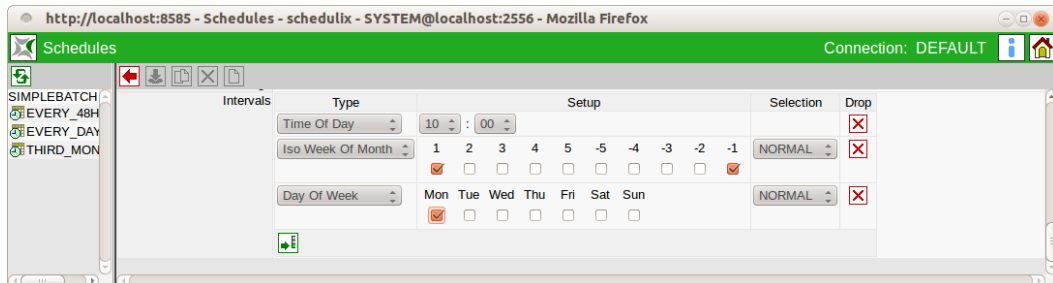


Figure 17.10: ISO Week of Month Filter

In the example, the job is executed on the first Monday of the first ISO week of each month. ISO weeks always begin on Mondays. Weeks are assigned to a months if at least 4 days of the week lie in this month. For example, if the first weekday of a month is a Wednesday, the first ISO week of the month already begins in the preceding month.

- ISO Week of Year

The "filtering" interval type "ISO Week of Year" can be used to select the ISO calendar week of the year in which the job is to be executed. This is defined in the *Setup* field by choosing the relevant weeks in the year.

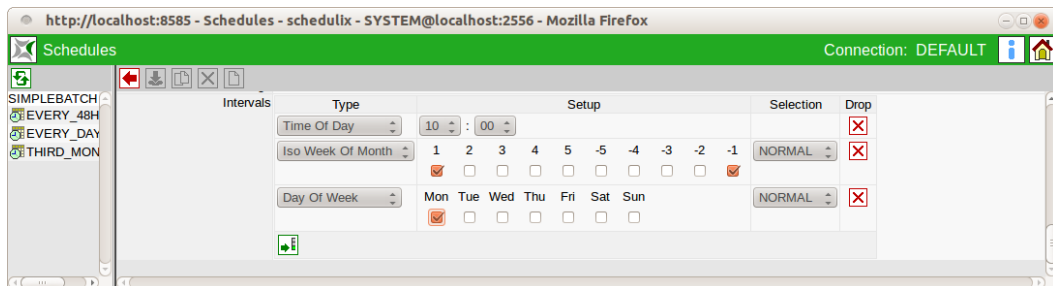


Figure 17.11: Week of Year Filter

In the example, the job is started on the Tuesday of the 31st ISO calendar week. ISO weeks always begin on Mondays. Weeks are assigned to a year if at least 4 days of the week lie in this year. For example, if the first weekday of a year is a Wednesday, the first ISO week of the month already begins in the preceding year.

## Editor

- Month of Year

The “filtering” interval type “Month of Year” can be used to select the month of the year in which the job is to be executed. This is defined in the *Setup* field by choosing the relevant months of the year.

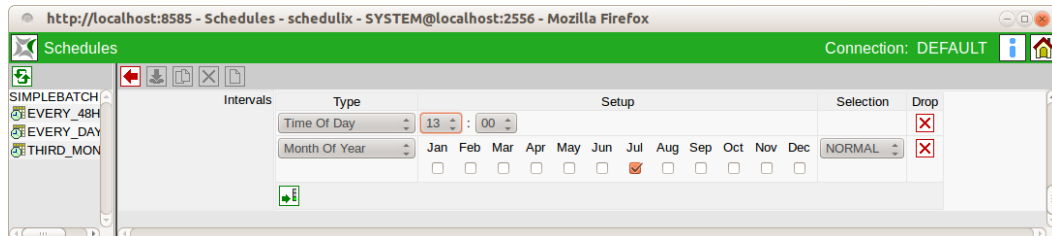


Figure 17.12: Month of Year Filter

In the example, the job is started every day in July at 13:00 hours.

- Calendar (Driver)

The “driving” interval type “Calendar (Driver)” can be used to take execution times from a predefined calendar. This option is only available only if the administrator has created such a calendar and registered it in schedulix!Web.

The calendar can then be selected in the *Setup* field. The option “with select on” determines the unit (DAY, WEEK, MONTH, YEAR) to which the subsequent selection refers.

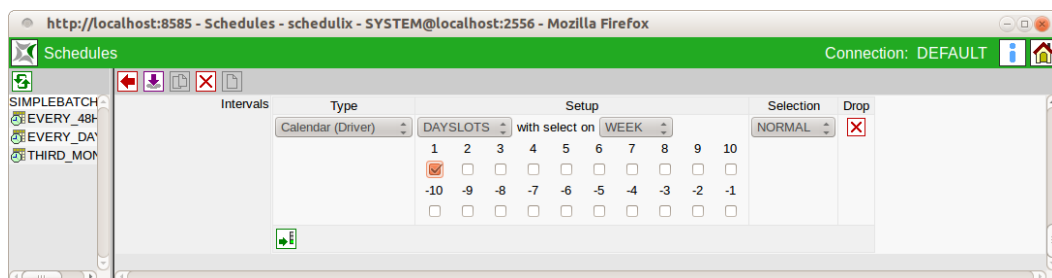


Figure 17.13: Calendar Driver

In the example above, the first entry in each calendar week is selected from a calendar called “DAYSLOTS”.

## Editor

- Calendar (Filter)

With the "filtering" interval type "Calendar (Filter)", the execution time points can be restricted by means of a predefined calendar.

The calendar can then be selected in the *Setup* field. The option "with select on" determines the unit (DAY, WEEK, MONTH, YEAR) to which the subsequent selection refers.

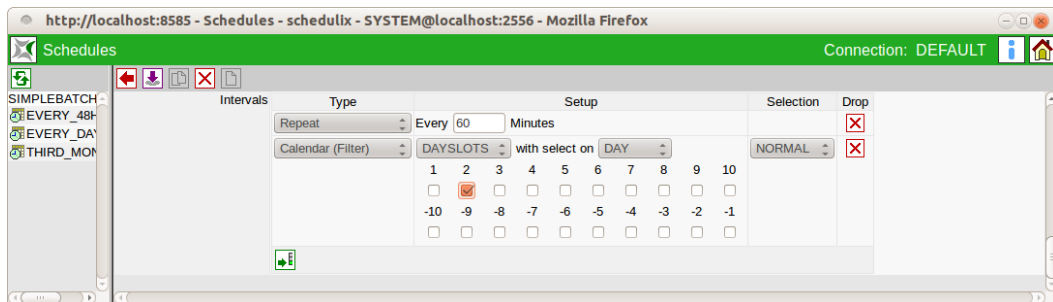


Figure 17.14: Calendar Filter

In the example above, a submit is performed every 60 minutes if this is in the second block of each day in the calendar "DAYSLOTS".

**Setup** The *Setup* field defines the interval more precisely. It varies according to the type of interval.



# 18 Intervals

## 18.1 Image

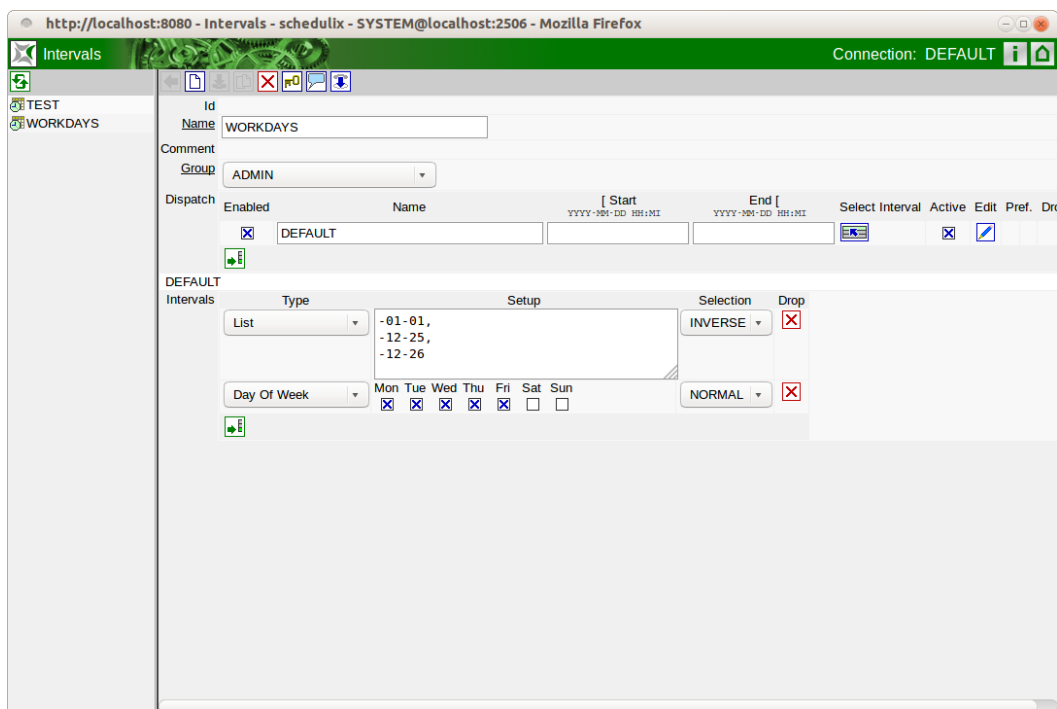


Figure 18.1: Intervals

## 18.2 Concept

### 18.2.1 Brief description

Intervals allow the definition of time intervals which can be repeated in cycles. Defined intervals can be used as "calendars" (drivers or filters) in time scheduling for batches and jobs. Whether a child job is to be "disabled" when it is submitted can likewise be controlled by using an interval. Defined intervals can also be used to define other intervals, both as "Calendar" and as "Select" dispatch intervals.

### 18.2.2 Detailed description

Intervals can be created and edited or deleted in the "Intervals" dialogue window. Intervals comprise one or more dispatchers, each with a filter chain. The order of the dispatchers determines which filter chain is used for a point in time. The functionality of the dispatcher is comparable to the sub schedules in the time scheduling for batches and jobs. For intervals, however, a different defined interval can be used as the dispatcher. Dispatchers can also be temporarily deactivated.

## 18.3 Navigator

The navigation shows all the intervals visible to the user that were created in the "Intervals" dialogue window. The interval is opened in the editor by clicking it in the navigator.

## 18.4 Editor

The editor displays the "Interval" selected in the navigation. Dispatchers can be created here, and their filter chains can be edited.

The fields in the editor mask named above have the following meanings:

**Name** This is the name of the interval.

**Owner Group** The owner of the interval is defined in the field *Owner Group*.

**Dispatcher list** All the dispatchers for the interval are shown in the "Dispatchers" list.

The dispatchers are only valid in the entered period and with the set select interval if the point in time to be examined is a valid time for the select interval.

The dispatchers with the highest priority (at the top of the list) are taken into consideration first. These 'conceal' dispatchers for their period and/or select interval that are further down the list. This means that only one dispatcher can be valid at any one time.



The fields in the list have the following meanings:

**Enabled** The *Enabled* field is a flag that determines whether the dispatcher line is to be used. If this flag is not set, the line is treated as if it did not exist.

**Name** The name of the dispatcher is entered in the *Name* field.

**Start** The date and time from which this dispatcher is valid are entered in the *Start* field. If this field is left empty, the dispatcher is considered to be "always valid".

**End** The date and time from which this dispatcher is invalid are entered in the *End* field. If this field is left empty, the dispatcher is considered to be "indefinitely valid".

**Select Interval** A "Select" interval can be set in the *Select Interval* selection field. A time is then handled by this dispatcher if it is a valid time for the "Select" interval.

**Active** This flag must be set if the dispatcher filter chain is to be used to check whether a time for the edited interval is valid. If this flag is not set, a time that is to be tested which is valid for the "Select" interval of the dispatcher is regarded as being invalid for the edited interval.

The following buttons are available in the list of dispatchers:

**Edit** Clicking the *Edit* button opens the interval description for the dispatcher in which the user can precisely define whether a time is valid, respectively the next valid time for this interval.

**Pref.** The *Preference* buttons are used to set the preference for the individual dispatchers. The rows are shifted up or down by clicking the buttons.

**Drop** This button is used to delete the object.

### 18.4.1 Sub-area Interval Details

This sub-area shows all the "Details" information for the first dispatcher or for another dispatcher selected by clicking the *Edit* button. The dispatcher details begin in the mask with a grey row containing the name of the currently selected dispatcher, followed by the following fields:

**Intervals list** If an interval is not entered here (the list is empty), and if the dispatcher is marked as being active, the edited interval will generate a time precisely once at the start time of the dispatcher, or all times which are valid for the time period and/or "Select" interval will be considered to be valid.

A list that is not empty may only contain one "driving" interval of the type "Time Of Day", "Repeat" or "Calendar (Driver)". The "driving" interval defines the cycle in which a submit can potentially take place.

The other "filtering" intervals determine by means of their filtering effect which cycles generated by the "driving" interval actually result in a submit. In contrast to time scheduling, intervals can also be created without a "driving" interval because intervals themselves can also be used as filters.

**Type** The *Type* field defines the type of interval. Another mask is displayed in the *Setup* column depending on the interval type.

**Setup** A mask for configuring the interval is displayed here depending on the interval type.

**Selection** In the case of intervals which enable a selection (e.g. "Day Of Week"), in this column it is possible to choose whether "NORMAL" or "INVERSE" is to be selected.

The following interval types are used:

- Repeat  
A "driving" interval type which enables a task to be started at controlled regular intervals. The number of minutes can be entered in the *Setup* field.
- Time of Day  
This "driving" interval type can be used to start the job at a specified time of day. The time of day is entered in the *Setup* field by selecting the hour (24 h) and minutes.  
Several "Time Of Day" intervals can be created. This allows multiple execution times to be easily defined for any one day
- Range of Day  
This "filtering" interval type allows for a limitation to one or more time periods in any one day. Multiple "Range Of Day" intervals can be created. This allows multiple validity time periods to be easily defined for any one day.
- Day of Week  
The "filtering" interval type "Day of Week" can be used to select the weekdays on which the interval is to be valid. This is defined in the *Setup* field by selecting the relevant weekdays.

- Day of Month

The "filtering" interval type "Day of Month" can be used to select the days of the month on which the interval is to be valid. This is defined in the *Setup* field by selecting the relevant days of the month.

- Week of Month

The "filtering" interval type "Week of Month" can be used to select the weeks of the month on which the interval is to be valid. This is defined in the *Setup* field by selecting the relevant weeks in the month.

- ISO Week of Month

The "filtering" interval type "ISO Week of Month" can be used to select in which ISO calendar week of the month the interval is to be valid. This is defined in the *Setup* field by selecting the relevant weeks in the month. ISO weeks always begin on Mondays. Weeks are assigned to a month if at least 4 days of the week lie in this month. For example, if the first weekday of a month is a Wednesday, the first ISO week of the month already begins in the preceding month.

- ISO Week of Year

The "filtering" interval type "ISO Week of Year" can be used to select in which ISO calendar week of the year the interval is to be valid. This is defined in the *Setup* field by selecting the relevant weeks in the year. In the example above, the job is executed on the Tuesday of the 31st ISO calendar week. ISO weeks always begin on Mondays. Weeks are assigned to a year if at least 4 days of the week lie in this year. For example, if the first weekday of a year is a Wednesday, the first ISO week of the year already begins in the preceding year.

- Month of Year

The "filtering" interval type "Month of Year" can be used to select in which month the interval is to be valid. This is defined in the *Setup* field by selecting the relevant months of the year.

- Calendar (Driver)

Another existing interval can be used by means of the "driver" interval type "Calendar (Driver)".

The calendar can then be selected in the *Setup* field. The option "with select on" determines the unit (DAY, WEEK, MONTH, YEAR) to which the subsequent selection refers.

- Calendar (Filter)

By means of the "filtering" interval type "Calendar (Filter)", the execution time points can be restricted by means of a predefined calendar. The calendar can then be selected in the *Setup* field. The option "with select on" determines the unit (DAY, WEEK, MONTH, YEAR) to which the subsequent selection refers.

- List

A list of time ranges can be entered that are to be used for filtering purposes with the "filtering" interval type "List". The time periods are separated with ",". All "selections" of intervals described in the syntax documentation are valid. Please note that a selection defined in the command line language "'2019-01-01' - '2019-01-02' " must be written as "2019-01-01 to 2019-01-02" in the input field of the "List" interval type.

**Setup** The *Setup* field defines the interval more precisely. It varies according to the type of interval.

# 19 Submit Batches and Jobs

## 19.1 View

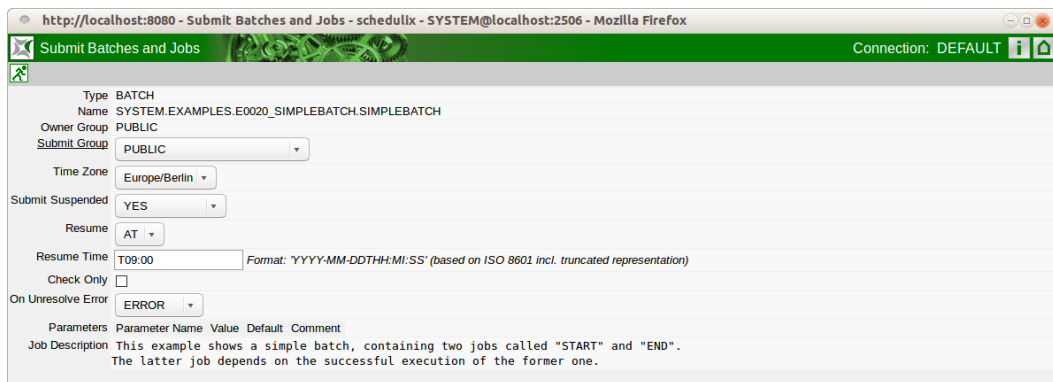


Figure 19.1: Submit Batches and Jobs

## 19.2 Concept

### 19.2.1 In short

The "Submit Batches and Jobs" dialog is used for manually starting Master Jobs defined in this dialog. The submit informs the schedulix Server that this Master Job is to be activated.

### 19.2.2 Detailed description

Manually releasing Master Jobs is one of two methods to start a job. The second method uses the job entry in the [Time Scheduling](#).

Starting a job manually also allows you to set certain parameters and determine whether you really want to start the job or just check if it can actually be started.

You can also define whether the Master Job is to be suspended straight away.

### 19.3 Navigator

The navigation shows all the available Master Jobs in a folder hierarchy. It corresponds to the navigation pane in the Batches and Jobs dialog window. One difference here, though, is that only objects for which the flag "Submit as Master allowed" has been set are displayed.

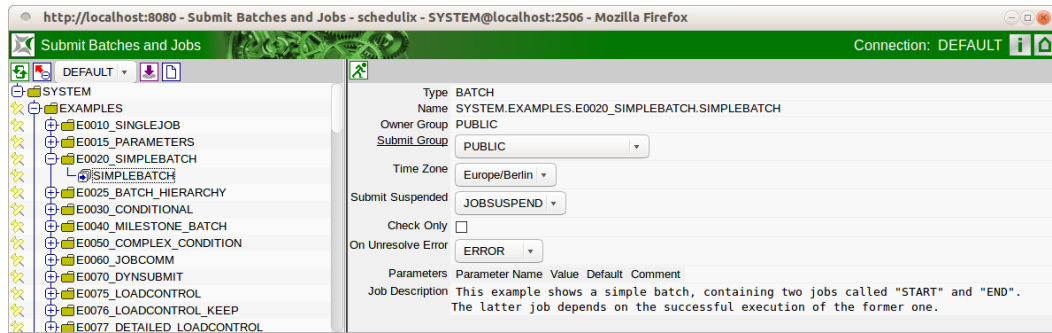


Figure 19.2: Submit navigation

### 19.4 Editor

The Editor "Frame" can be used to hand over start options and parameters and submit a Master Job.

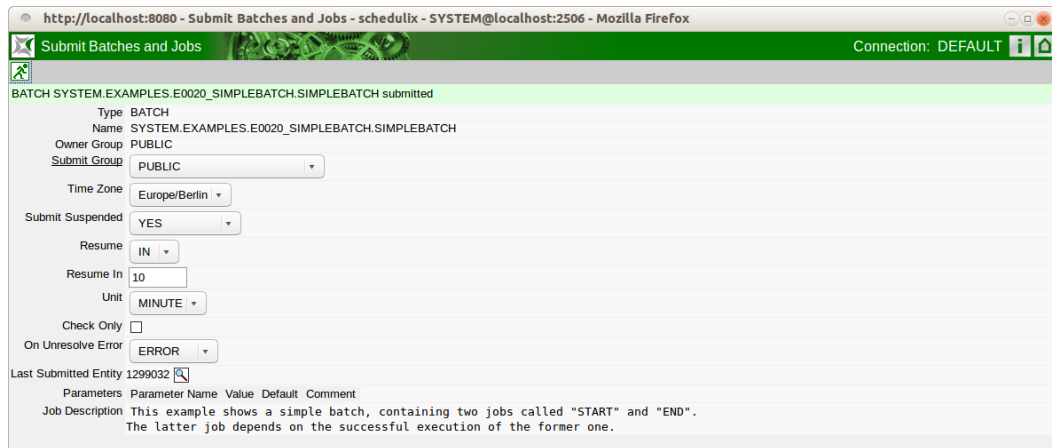


Figure 19.3: Submit editor

The following button is active in this mask:


**Submit**

The *Submit* button is used to start the job (the flag "Check Only" is not set) or check the start capability of the Master Job (the flag "Check Only" is set).

The fields shown above have the following meanings:

**Type** The type of the Master Job. This can be either a job or a batch.

**Name** This is the full name (with its path) of the Master Job.

**Owner Group** The owner of the job is defined in the field *Owner Group*.

**Submit Group** The *Submit Group* determines the owner group of the submitted job.

**Submit Suspended** This field defines whether the job is to take on a Suspend State right after the start or not.

The input fields *Resume*, *Resume Time*, *Resume In* and *Unit* are only displayed if this field has been set to 'YES'.

The following options are available:

- YES  
The child is created as being suspended when it is submitted and it has to be started by being explicitly released.
- NO  
The child is not suspended when it is submitted and can be started immediately.
- JOBSUSPEND  
Whether a delay takes place or not depends on the *Suspend* field for the job to be submitted. This means that the setting defined in the job is applied.

**Resume** Here you can choose whether a workflow should be resumed automatically.

The following options are available:

- NO: deselects this functionality and no more input fields are displayed.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time** The required resume time is entered here in the format YYYY-MM-DDTHH:MI:SS.

This format is based on the ISO standard 8601 and also permits an incomplete entry. If you enter 'T09:00', the job will be resumed at 09:00 hours (starting from the current set time).

**Resume In** Here you can specify how many time units (see *Unit*) the system is to wait for until the resume action is triggered.

**Unit** This field is used to define whether the entry in *Resume In* is in minutes (MIN), hours (HOUR) or days (DAY).

**On Unresolve Error** When a job is submitted, the *On Unresolve Error* field determines what to do in the event of unresolved dependencies.

The following options are available:

- **Error:** Dependencies that cannot be resolved are regarded as errors. The missing dependencies cause the submit to be aborted.
- **Suspend:** With Suspend, the submit takes place as if these dependencies had not been undefined. However, the job is submitted in a suspended state.
- **Ignore:** With Ignore, the submit takes place as if these dependencies had not been undefined.

**Check Only** This flag states whether the job is being submitted or its start capability is being checked.

**Parameters list** If the job requires any parameters, the parameter values have to be entered in this list prior to the submit. More information about parameters can be found in Chapter [13.5.10](#).

**Job Description** The description of the job from the Job Definition.



# 20 Bookmarks

## 20.1 View

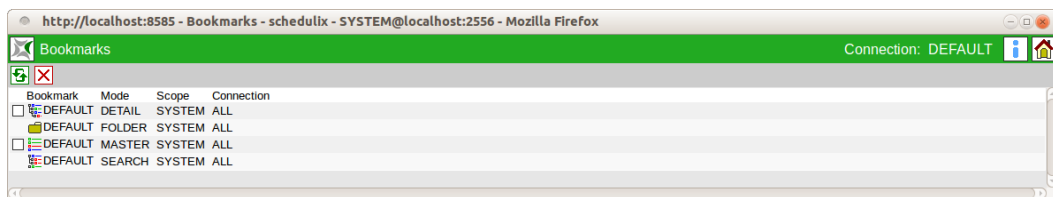


Figure 20.1: Bookmarks

## 20.2 Concept

### 20.2.1 In short

Queries from the "Running Master Jobs" and "Search Running Jobs" dialogs can be called in the "Bookmark" dialog. Recurring queries and overview windows can be easily saved and opened again here.

### 20.2.2 Detailed description

Bookmarks can be made available and modified for the current user or throughout the system. It is also possible to always make certain bookmarks immediately available after logging in to the system. These are then started automatically. The list of bookmarks contains the following fields:

**Bookmark** The name of the bookmark is shown here. Selecting the name opens a new window with the request saved as a bookmark.

**Mode** The *Mode* field shows the type of bookmark. The following types are used:

1. FOLDER

This is a folder bookmark.

## Concept

### 2. MASTER

This is a bookmark that was saved by the search mask described in Chapter 21. Using this bookmark means that only Master Jobs are visible.

### 3. SEARCH

The Search mode designates a bookmark that was saved by the search mask described in Chapter 22. This allows all jobs to be found.

### 4. DETAIL

This designates a bookmark that describes the "Detail" view (more information about this can be found in Chapter 21). This CHILD is always related to a MASTER mode bookmark (it is the child of the "Master" view).

Which bookmark is selected is determined as follows:

If the Master Job or batch defines a parameter `DETAIL_BOOKMARK`, then this is used. If this is not the case, the "Detail Bookmark" field for the Master Bookmark is evaluated. If this is not set either, the Detail Bookmark 'DEFAULT' is used.

**Scope** The Scope parameter defines whether the bookmark is visible throughout the entire system wide or just to the current user. The following options are available:

#### 1. SYSTEM

A bookmark with the scope SYSTEM is visible for all users throughout the system. It is displayed in the bookmark list of every user and can be used by all of them. Any changes made to a bookmark from the scope SYSTEM apply system-wide for all users.

Bookmarks of the type SYSTEM, however, can only be created by users with Web GUI administration privileges (see Web Users). Although a 'normal' user is allowed to modify SYSTEM bookmarks, it is then only saved as a USER bookmark. This means that local changes only overwrite the bookmark for the current user, but not for other users. If this bookmark is deleted, the original SYSTEM bookmark is displayed again.

#### 2. USER

If the scope has the value USER, then this bookmark was created by the current user and is only available to this user.

## Navigation

**Connection** The *Connection* field indicates the validity of the bookmark when using multiple server connections. The following options are available:

1. CURRENT

The bookmark is only valid for the current connection. This bookmark is no longer visible if the bookmark window is opened for another connection.

2. ALL

The bookmark is valid for all server connections.

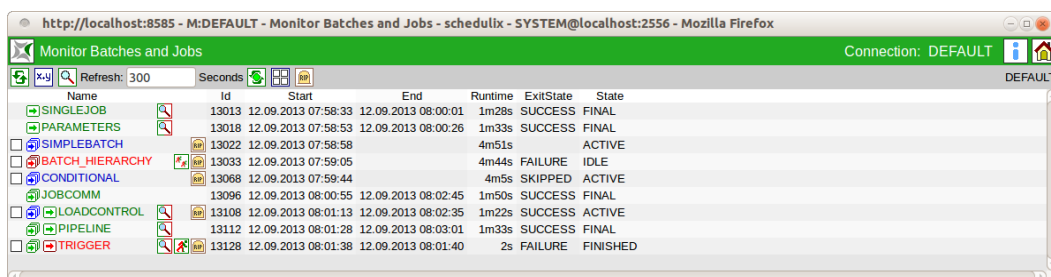
### 20.3 Navigation

The Bookmark dialog only features a navigation window. An entry selected here is displayed in a new window.



# 21 Running Master Jobs

## 21.1 View



The screenshot shows a web browser window displaying the 'Monitor Batches and Jobs' interface. The interface includes a refresh button, a refresh interval of 300 seconds, and a table of job details. The table has columns for Name, Id, Start, End, Runtime, ExitState, and State. The jobs listed are:

Name	Id	Start	End	Runtime	ExitState	State
SINGLEJOB	13013	12.09.2013 07:58:33	12.09.2013 08:00:01	1m28s	SUCCESS	FINAL
PARAMETERS	13018	12.09.2013 07:58:53	12.09.2013 08:00:26	1m33s	SUCCESS	FINAL
SIMPLEBATCH	13022	12.09.2013 07:58:58		4m51s		ACTIVE
BATCH_HIERARCHY	13033	12.09.2013 07:59:05		4m44s	FAILURE	IDLE
CONDITIONAL	13068	12.09.2013 07:59:44		4m5s	SKIPPED	ACTIVE
JOBCOMM	13096	12.09.2013 08:00:55	12.09.2013 08:02:45	1m50s	SUCCESS	FINAL
LOADCONTROL	13108	12.09.2013 08:01:13	12.09.2013 08:02:35	1m22s	SUCCESS	ACTIVE
PIPELINE	13112	12.09.2013 08:01:28	12.09.2013 08:03:01	1m33s	SUCCESS	FINAL
TRIGGER	13128	12.09.2013 08:01:38	12.09.2013 08:01:40	2s	FAILURE	FINISHED

Figure 21.1: Running Master Jobs

## 21.2 Concept

### 21.2.1 In short

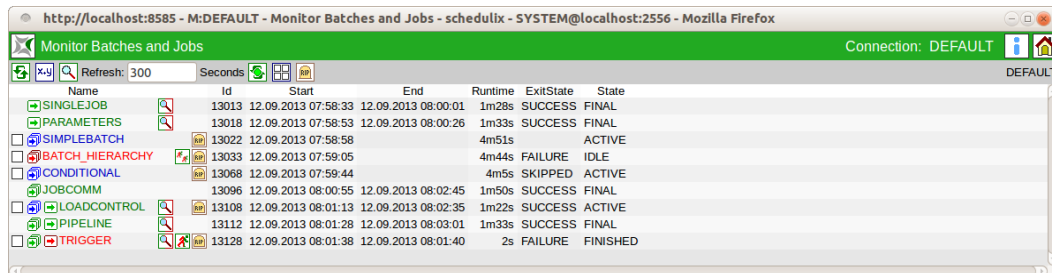
The *Running Master Jobs* dialog is used to display the currently running or (depending on the settings) the historically completed Master Jobs in the system.

### 21.2.2 Detailed description

The *Running Master Jobs* dialog is where the currently running jobs (Monitoring) are displayed and the active jobs (Operating) are maintained. The display and filter criteria for the Running Master Job dialog can be modified in the [bookmark](#) (Default, Master System). This bookmark describes how the entire dialog is displayed.

## 21.3 Master Navigator

The navigation takes up the whole dialog window. A list of all the currently running 'Master-submittable' jobs and batches is shown here.



Name	Id	Start	End	Runtime	ExitState	State
SINGLEJOB	13013	12.09.2013 07:58:33	12.09.2013 08:00:01	1m28s	SUCCESS	FINAL
PARAMETERS	13018	12.09.2013 07:58:53	12.09.2013 08:00:26	1m33s	SUCCESS	FINAL
SIMPLEBATCH	13022	12.09.2013 07:58:58		4m51s		ACTIVE
BATCH_HIERARCHY	13033	12.09.2013 07:59:05		4m44s	FAILURE	IDLE
CONDITIONAL	13068	12.09.2013 07:59:44		4m5s	SKIPPED	ACTIVE
JOBCOMM	13096	12.09.2013 08:00:55	12.09.2013 08:02:45	1m50s	SUCCESS	FINAL
LOADCONTROL	13108	12.09.2013 08:01:13	12.09.2013 08:02:35	1m22s	SUCCESS	ACTIVE
PIPELINE	13112	12.09.2013 08:01:28	12.09.2013 08:03:01	1m33s	SUCCESS	FINAL
TRIGGER	13128	12.09.2013 08:01:38	12.09.2013 08:01:40	2s	FAILURE	FINISHED

Figure 21.2: Running Master Jobs Navigator



### Find

Clicking the *Find* button opens the page for entering the search criteria as well as configuring the screen.



### Auto-Refresh On/Off

The *Auto-Refresh On* button is also a switch and can be used in two positions. In the position "Auto-Refresh On", a refresh is run automatically in the set time.

The time in seconds between two refreshes is entered in the field to the left of the button. The default value is 300 seconds. The colour of the button indicates the state of the Auto-Refresh function. Green = on, red = off.

The list in the dialog shows either all the Master Jobs that are active in the current system (Default) or, if filter options have been entered in the search mask, the correspondingly filtered results. The following columns are shown in the list:

**Name** The full path and name of the Master Job or just its name are shown here depending on the position of the Show Job button.

Clicking the name opens the Details mask if the object is a Master Submittable Batch or Job with children or otherwise the Details mask for jobs.

Icons can be displayed in front of the name which describe certain states and situations. The following icons are used:



The displayed object is a job that is either currently running or can run. The icon is blue.



The displayed object is a pending job. The icon is lilac.

- Job icon without a period: The job is in the Dependency Wait state.
- Job icon with a period: The job is in the Synchronize Wait state.
- Job icon with two periods: The job is in the Resource Wait state.



The displayed object is a job that will be submitted in the future by the Time Scheduling.



The displayed object is a job that will be submitted as suspended in the future by the Time Scheduling.



The displayed object is a job that has been cancelled. The state of this job can no longer be changed. The icon is brown.



The displayed object is a job that is in the Final state. The state of this job can no longer be changed. The icon is green.



The displayed object is a job that is in one of the states Finished and Restartable, Unreachable or Error. The icon is red.



The displayed object is a batch that is running at this moment. The icon is blue.



The displayed object is a batch that is waiting at this moment. The icon is lilac.



The displayed object is a batch that will be submitted in the future by the Time Scheduling.



The displayed object is a batch that will be submitted as suspended in the future by the Time Scheduling.



The displayed object is a batch that is in the Final state. The state of this batch can no longer be changed. The icon is green.



The displayed object is a batch that has been cancelled. The state of this batch can no longer be changed. The icon is brown.



The displayed object is a batch that is in the Unreachable state, or which has at least one child that is in one of the states Finished and Restartable, Unreachable or Error. The icon is red.



The displayed object is a pending milestone. The icon is lilac.



The displayed object is a milestone that has been cancelled. The icon is brown.



The displayed object is a milestone that is in the Final state. The icon is green.



The displayed object is a milestone that is in the Unreachable state, or which has at least one child that is in one of the states Finished and Restartable, Unreachable or Error. The icon is red.



These icons tell the user that the object has been suspended. The icon with the clock indicates that an Automatic Resume has been set for the object. The name colours have the same meanings as the matching coloured buttons.



The *Cancel Run* button is displayed if the job is waiting for resources or dependencies or it has still been allocated on a jobserver. The scheduled job run can be cancelled by clicking the *Cancel Run* button. The job then gets the state "Cancelled". Neither a rerun nor a resume are possible. The job doesn't get an Exit State either. Jobs that are waiting for a cancelled job have to ignore this job or be cancelled as well.



This button opens a new window displaying the contents of the log files. Should this not be possible, please contact your system administrator.





### Rerun

The *Rerun* button is displayed if the current job has been finished and it is in a "Restartable" state. This means that it has an Exit State which is not final. Clicking the *Rerun* button calls the [Rerun](#) program (if defined, otherwise the [Run](#) command). This allows a failed job to be restarted after troubleshooting it.



### Rerun Children

The *Rerun Children* button is displayed if children of the current job are in a Restartable state. This means they have a Restartable Exit State. Clicking the *Rerun Children* button starts the [Rerun](#) for all the Restartable children. This allows failed children to be restarted after troubleshooting them.

**Id** The *Id* is the unique identifier of the current "Master Job" runtime instance in the system.

**Start** The *Start* field shows the time or start of the submit.

**End** The *End* field shows the finish time for the job. This field stays empty if the job has not yet finished.

**Runtime** This is the current runtime of the job in days (d), hours (h), minutes (m) and seconds (s). If the job is still running, the runtime is displayed until the last refresh of the dialog and is updated after each new refresh.

**Exit State** If the job has been finished, its *Exit State* when it finished is displayed. This field stays empty if the job has not yet finished.

**State** The *Job State* is the current runtime state of the job. The runtime system of the schedulix Server system assigns and changes these states when the job (i.e. its runtime instance) has passed through the schedulix system. The job can have the following states:

## Master Navigator

### 1. Submitted

The job was submitted manually, by a parent process or by the Time Scheduling and is to be executed. This is the initial state of a job and is usually not visible.

### 2. Dependency Wait

The job is waiting for required dependencies that have to be fulfilled.

### 3. Synchronize Wait

The job is waiting for the required "Synchronizing Resources".

### 4. Resource Wait

The job is waiting until sufficient System Resources become available.

### 5. Unreachable

The job cannot be executed because one or more dependencies have not been fulfilled. This situation can be resolved by ignoring dependencies.

### 6. Cancelled

The job has been manually cancelled and is no longer running.

### 7. Error

The job cannot be executed due to a definition error. An example of this is a required resource that cannot be provided by any of the jobserver. In this case, the job can be restarted after the error has been remedied.

What also frequently happens is that a mistake is made when entering the Run program. If the jobserver is consequently unable to start the process, the job is placed in an Error state. The job concerned can be restarted in this case as well.

If a Master Submit by the Time Scheduling fails, the master is assigned an Error State in the system to identify this error. This makes the error visible for the person responsible. The job or batch is not restartable, however, and has to be submitted manually after being repaired.

### 8. Runnable

The job can be started by a jobserver. The jobserver should be checked if the job remains in this state for any length of time.

9. Starting

The job was handed over to a jobserver to be started.

10. Started

The job was handed over to a jobserver to be started. The jobserver has acknowledged the job.

11. Running

The Run command was started by the jobserver.

12. To Kill

The user has instructed the schedulix Server to run the Kill program associated with the job. The Kill program can be run by clicking the *Cancel Run* button.

13. Killed

The Kill program associated with the job has been run.

14. Broken Active

The jobserver has lost the connection to the Run program. Although the process is still running, the result of the job (Exit Code) can no longer be returned to the jobserver and correspondingly the schedulix Server.

15. Broken Finished

If a job has the state "Broken Active" and it stops, it switches to the state "Broken Finished". This means that the process is no longer running. The Exit Code for the process could no longer be determined by the jobserver. It is necessary to manually check the job results and set the Exit State. The Exit State is set with the *Set State* button.

16. Finished

The job has been finished. The Exit State is shown in the *Exit State* field.

17. Final

The job and all its children have finished and have a Final Exit State.

The [21.3](#) diagram shows the status transitions of the schedulix runtime system.

**Variables** Depending on the configuration, the state column can be followed by the variables defined there in the specified order and colour.

# Master Navigator

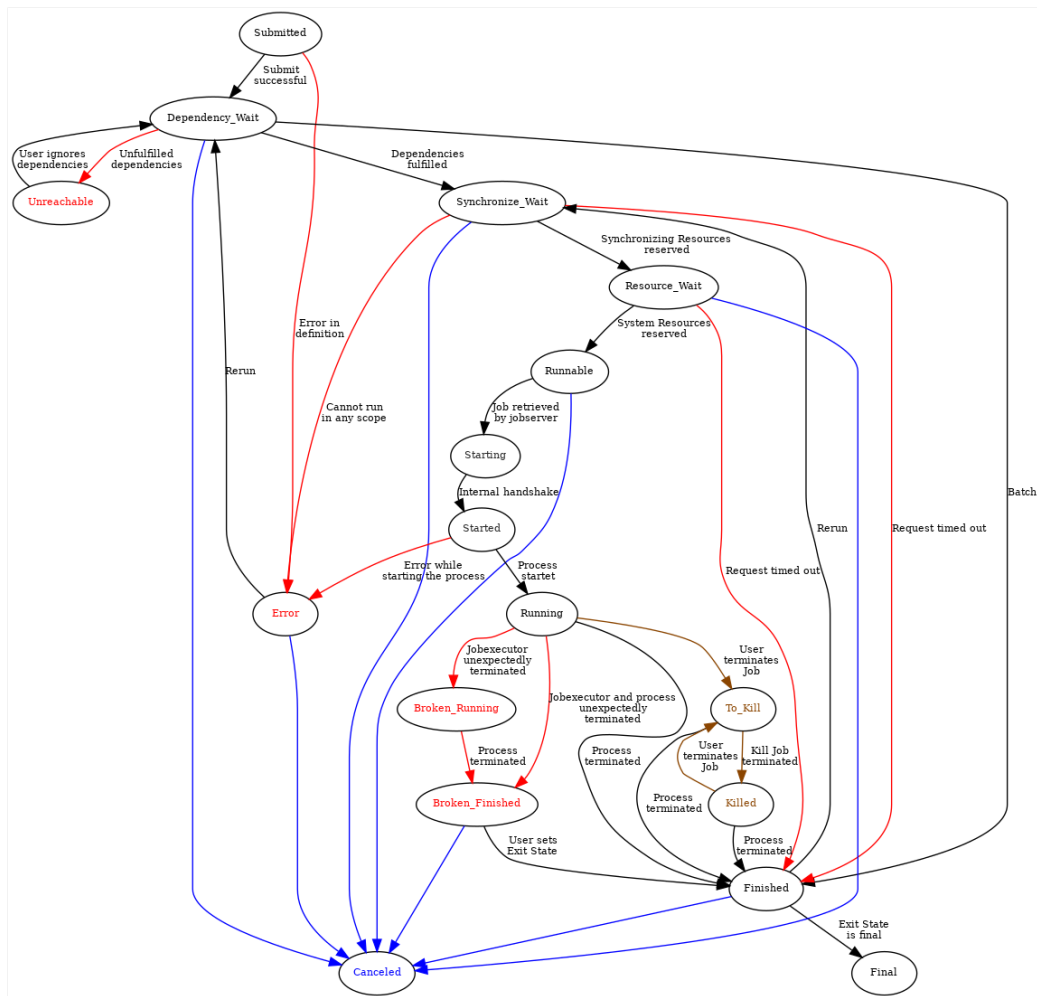


Figure 21.3: State chart for batches and jobs

### 21.3.1 Master Navigator Query mask

The Query mask is opened by clicking the *Settings* button. The query requirements for the current dialog are set in this mask. The filter and display criteria can be edited here.

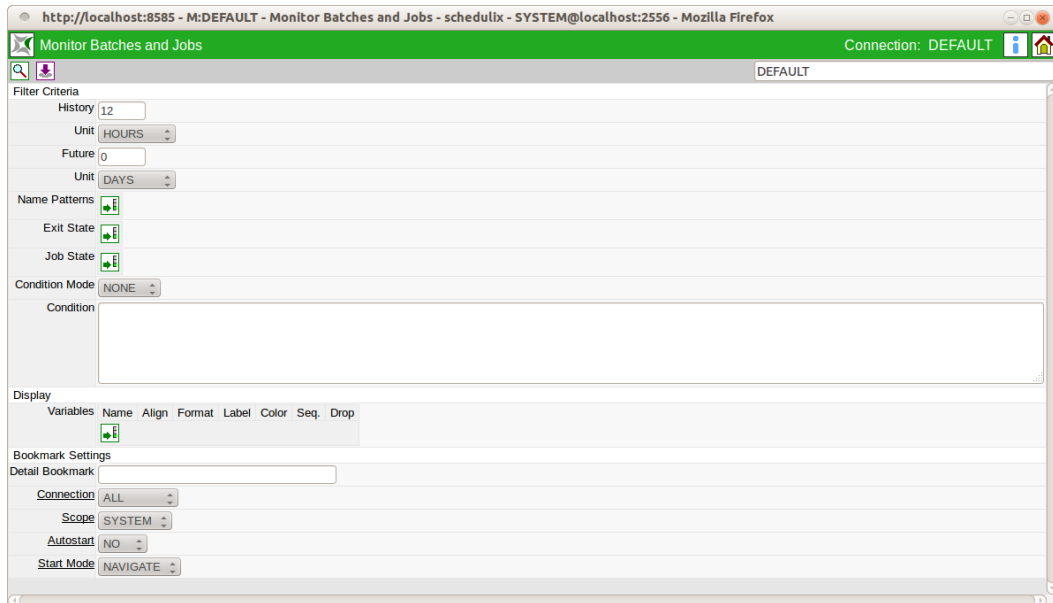


Figure 21.4: Running Master Jobs Query mask

Making changes in the mask and then saving them with the current or a changed bookmark name (entry field in the upper right of the headline) allows them to be used again at a later time. If the changes are only temporary ones, you can simply enter them and jump back to the navigation window. This means that the changes only apply for the current search. The changes will be discarded if the dialog is then closed.

The fields shown above have the following meanings:

**History/Unit** The maximum time that may have elapsed since the end of a workflow so that it appears in the display. For example, if all the currently running jobs and all the jobs that have been completed in the last hour are to be displayed, the value 1 has to be entered here and the unit Hours has to be set in the *Unit* field. If all the running jobs and all the jobs that have been completed in the last hour are to be displayed, the value 1 has to be set in the *History* field with Hours in the *Unit* field. The following options are available:

1. Minutes

2. Hours

3. Days

**Future/Unit** The setting in the *Future* field determines whether and to what extent batches or jobs that are to be submitted in the future are displayed in the Master List. Regardless of the interval selected in the *Future* field, only the next start can be displayed for schedules without an activated calendar. Only planned starts can be displayed in the calendar horizon of schedules with a calendar. The *Unit* field is handled like the History field.

**Name Patterns list** The list of Name Patterns can hold an amount of Name Patterns to be used to filter the returned results. A Name Pattern can be the full name, part of the name or a regular expression.

When searching for multiple entries in the list, all occurrences of these entries are sought. These are linked with "OR".

Example:

You want to search for all occurrences of the name "Pipeline".

The following Name Patterns are possible:

- Full: **pipeline** (the search is not case-sensitive)
- Sub-string: **eline**
- Regular expression: **P.\*line**

The list of Name Patterns can be modified using the *Add* and *Remove* buttons.

**Exit State list** Exit States can be entered in this list which the sought jobs must have if they are to be shown in the results list.

Example:

Only those jobs with the Exit State "Failure" are to be listed. By adding the Exit State "Failure" to the list of Exit States you can now search for these jobs. If the list does not contain any entries, the Exit States are ignored for the search. The list of Exit States can be modified using the *Add* and *Remove* buttons.

**Job State list** Job States in one of the defined states can be searched for in this list. If the list does not contain any entries, the Job States are ignored for the search. The list of Job States can be modified using the *Add* and *Remove* buttons.

**Condition mode** The Condition mode defines whether and how the subsequent condition is to be evaluated.

The following options are available:

- None: The condition is ignored.
- And: The search query and condition are linked with "And".
- Or: The search query and condition are linked with "Or".
- Minus: The search query and negation of the condition are linked with "And".
- Only: Only the condition is taken into account.

**Condition** The query can be refined using the condition. Please refer to the syntax documentation (List Job) for the exact syntax and the various options that are available to do this.

**Variables list** All the job variables and parameters that are to be displayed in the navigation list are defined in the Variables list. This is an easy way to get an overview of the main job parameters. They are displayed at the end of each line in the navigation list.

The following fields have to be populated for the variables:

**Name** The name of the variable or parameter has to be entered here.

**Align** The *Align* field is for defining how the variables are to be displayed in the list. The following options can be selected:

- LEFT  
The display is left-aligned.
- CENTER  
The display is centred.
- RIGHT  
The display is right-aligned.

## Master Navigator

**Format** The format for the output can be defined here. The following options are available:

- NONE  
No formatting is used.
- NUMBER  
Numerical formatting with thousand separators and decimal points (as necessary) is used.

**Label** A heading that is to be shown for the variable in the list header can be defined here. The variable name is used as a heading if a label is not defined.

**Colour** The background colour for the variables column can be configured here. A preview of each colour is shown in the field's "drop-down" menu.

**Seq.** The list order of the variables can be changed using the *Up* and *Down* buttons. These buttons are only visible if the variables list has more than one entry in it.

**Bookmark Settings** The entry fields in the *Bookmark Settings* section are only meaningful when a bookmark is saved. The following fields describe the properties of the bookmark to be saved.

**Detail Bookmark** If the name of a detail bookmark is entered in the field *Detail Bookmark*, opening a detail navigation window from a master navigation window opened by this bookmark will use the entered bookmark name for the opened detail navigation window. If this detail bookmark is not yet existing, the detail navigation window will show this bookmark name in square brackets. Those will disappear when the detail bookmark is saved in the opened detail navigation window. The field *Detail Bookmark* is only visible in the *Bookmark Settings* of master navigation windows.

**Connection** This field defines the server connections the bookmark can be used with. There are two variants:

1. ALL  
The bookmark can be used with any server connection.
2. CURRENT  
The bookmark can be used with with the current server connection only.



**Scope** This field defines the visibility of the bookmark to users. There are two variants:

1. USER

The bookmark is only visible to the current user. No other users can see this bookmark or, if it is a change made to a system-wide bookmark, only the change is locally visible to this user.

2. SYSTEM

The bookmark or current change is visible to all users. This means that all users see this new bookmark in their bookmark list. If it is a change made to an existing system-wide bookmark, the change affects all the users.

**Autostart** Autostart defines whether a bookmark is to be activated immediately after a user logs on. There are two variants:

1. YES

The bookmark is started immediately after the login. This means that a window opens on the screen after the user logs on for each bookmark where the autostart option is set to "YES".

2. NO

The bookmark must be explicitly started by clicking in the Bookmark dialog and it is not automatically activated. This is the default setting.

**Start Mode** The Start mode defines whether the query is run immediately after the bookmark has been started and the navigation window is to be opened, or whether the query dialog is to be displayed. There are two variants:

1. NAVIGATE

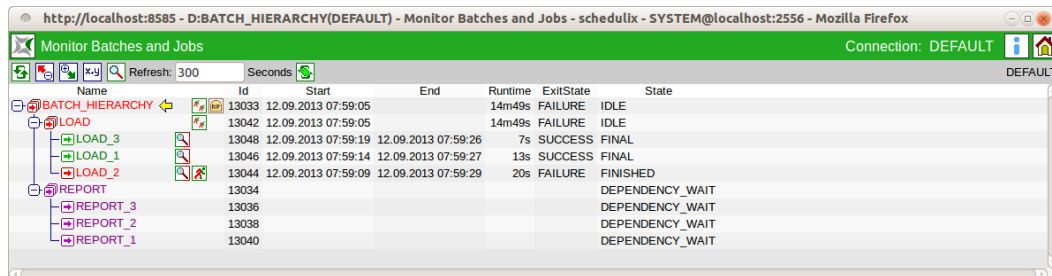
The query is run immediately and the navigation screen opens with all the results.

2. QUERY

The Query dialog is displayed first.

## 21.4 Detail Navigation

Clicking one of the Master Jobs (which has children) opens the “Details” window for the Master Job.



Name	Id	Start	End	Runtime	ExitState	State
BATCH_HIERARCHY	13033	12.09.2013 07:59:05		14m49s	FAILURE	IDLE
LOAD	13042	12.09.2013 07:59:05		14m49s	FAILURE	IDLE
LOAD_3	13048	12.09.2013 07:59:19	12.09.2013 07:59:26	7s	SUCCESS	FINAL
LOAD_1	13046	12.09.2013 07:59:14	12.09.2013 07:59:27	13s	SUCCESS	FINAL
LOAD_2	13044	12.09.2013 07:59:09	12.09.2013 07:59:29	20s	FAILURE	FINISHED
REPORT	13034					DEPENDENCY_WAIT
REPORT_3	13036					DEPENDENCY_WAIT
REPORT_2	13038					DEPENDENCY_WAIT
REPORT_1	13040					DEPENDENCY_WAIT

Figure 21.5: Running Master Jobs “Details” window

The “Details” window is similar to the Master window but with the following differences:

### 21.4.1 Tree view

Instead of the flat list in the Master window, all the children of the current batch and their own children are displayed in a tree hierarchy. The first level is expanded, but the others are not. The exception here is the search using ‘Name Patterns’, since here every found job is listed and the respective level and parent levels are also automatically expanded.

### 21.4.2 Search results

The results of a search are indicated by an arrow symbol



This enables you to differentiate between found entries that were expanded because another child entry meets the search criteria, and search hits. Only results are displayed with the arrow icon.

### 21.4.3 Detail navigation query mask

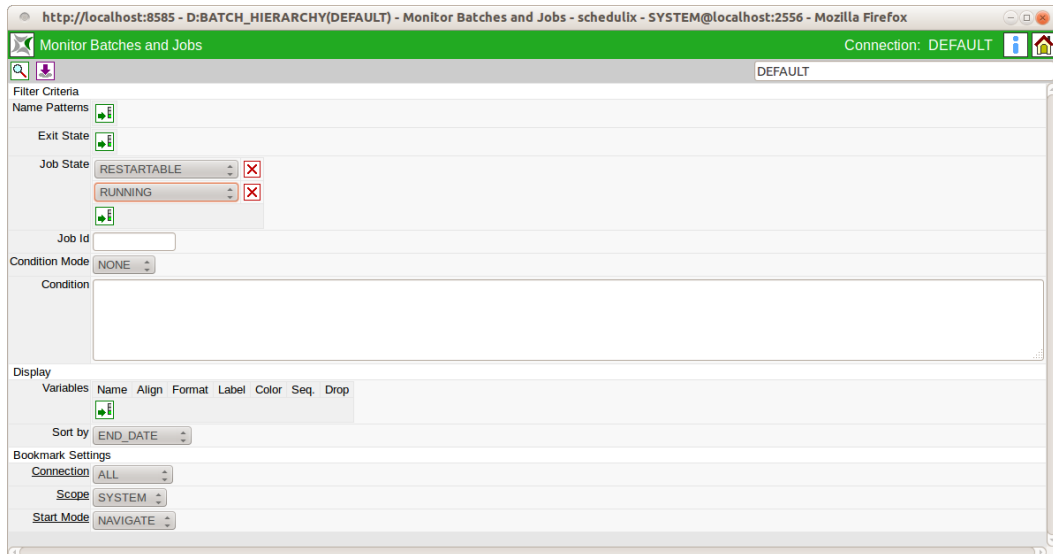


Figure 21.6: Navigation Details query mask

The query mask for the Navigation Details corresponds to the query mask for the Master Navigation, but with the following deviations:

The fields *History*, *Unit*, *Bookmark Details* and *Autostart* are not shown here and some other fields have been added. The mask contains the following fields:

**Job State list** In the Job State list, jobs which have the entered Job State can be filtered. Each of the states described in the *State* field can be selected and used as a filter.

If no Job States have been entered, they are ignored by the search.

**Job Id** A specific Job Id that is to be searched for can be entered in this field. A maximum of one job can be returned as a result.

**Sort By** The list can be sorted by various criteria with this function. The following criteria can be used:

1. Name

The results are sorted alphabetically by job name.

2. Start Date

The results list is sorted chronologically by the job start time.

## Details mask for jobs

### 3. End Date

The results list is sorted chronologically by the job finish time.

**Enable Bulk Operations** This flag whether checkboxes for selection and bulk operations will be available in the navigator view.

**Bulk Select Hits only** If this flag is set, only search hits can be selected for bulk operations.

## 21.5 Details mask for jobs

Clicking a job without children in the "Details" or "Master" window automatically opens the Details mask for jobs. It looks like this:

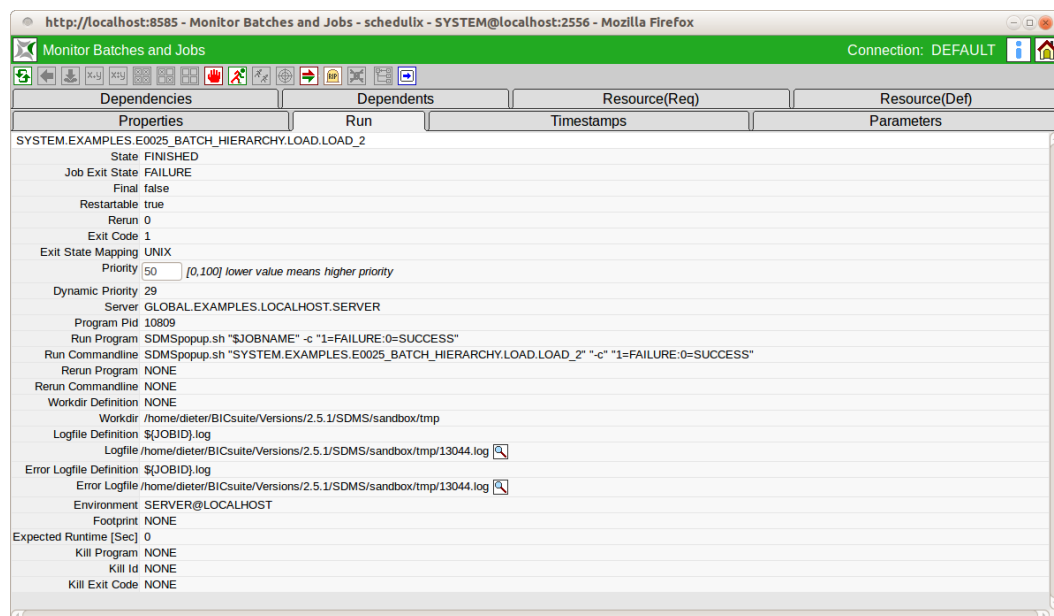


Figure 21.7: Details mask for jobs

All the runtime details are displayed in this mask and some of them can also be modified here. It also features some action buttons which allow the current objects to be manipulated.

### 21.5.1 Buttons

The following buttons are all action buttons. This means that these buttons are used to perform a certain action on the selected job. To do this, a Confirm window opens like in Figure 21.8 after you click the button.

## Details mask for jobs

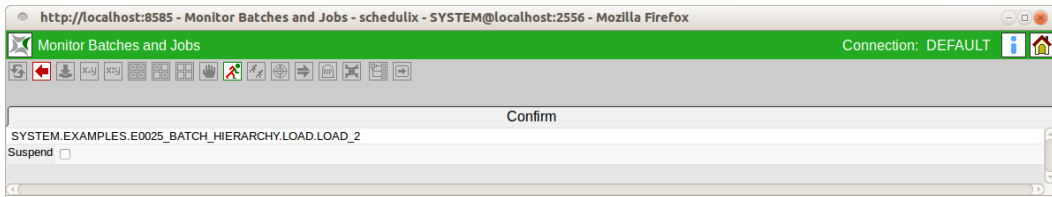


Figure 21.8: Confirm mask

You can now enter a comment in the text box to describe the reason for the action or some other remarks. This text is later shown in the Audit tab in the field *Reason/Comment*. The action is performed when you click the respective action button again. Clicking the *Cancel* button aborts the action and opens the previous screen again. The Confirm screen contains additional input fields for controlling the action (these vary according to the action).



### Suspend

Jobs can be suspended using the *Suspend* button. This is only possible if the jobs do not already have the "Suspended" state and they have not yet been completed. The jobs are placed in the "Suspended" state if this action is successful.

**Suspend Local** If the *Suspend Local* checkbox is checked, the children of the suspended job or batch will not be suspended. This way the execution of the children will continue, but the the batch or job itself will not go FINAL. Dependent batches or jobs will have to wait until the batch or job is resumed.

**Suspend Restricted** The Suspend Restricted checkbox is only editable for users of group ADMIN. If checked only ADMIN users are allowed to resume the batch or job.

Additional input fields: *Resume*, *Resume Time*, *Resume In*, *Unit*

**Resume** Here you can choose whether a workflow should be resumed automatically.

The following options are available:

- NO: deselects this functionality and no more input fields are displayed.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time** The required Resume Time in the format "YYYYMMDD HH:MM" is entered here.

**Resume In** Here you can specify how many time units (see *Unit*) the system is to wait for until the resume action is triggered.

**Unit** This field is used to define whether the entry in *Resume In* is in minutes (MIN), hours (HOUR) or days (DAY).



### Resume

Jobs in the "Suspended" state can be reactivated with the *Resume* buttons. This means that the resources and dependencies are checked again, and if all these checks are successful the job is started on a jobserver.

Additional input fields: *Resume*, *Resume Time*, *Resume In*, *Unit*

**Resume** The resume type can be selected here.

- OFF: This option is only available if the job is suspended and an Autoresume (IN/AT) is active. It disables this auto-resume function.
- NOW: A resume takes place immediately. No other input fields are displayed.
- AT: selects an automatic resume action at a fixed time. The *Resume Time* input field is displayed.
- IN: selects an automatic resume action after a time period has expired. The *Resume In* and *Unit* input fields are displayed.

**Resume Time, Resume In, Unit** See above (Suspend action)



### Rerun

The *Rerun* button is displayed if the current job has been finished and it is in a "Restartable" state. This means that it has an Exit State which is not final. Clicking the *Rerun* button calls the [Rerun](#) program (if it has been defined, otherwise the [Run](#) command). This allows a failed job to be restarted after troubleshooting it.

Additional input fields: *Suspended, Resume, Resume Time, Resume In, Unit*

**Suspended** If this flag is set, the job is suspended and the *Resume* check box is displayed. If this field is not set, no other fields are displayed either.

**Resume, Resume Time, Resume In, Unit** See above (Suspend action)



### Rerun Children

The *Rerun Children* button is displayed if children of the current job are in a "Restartable" state. This means they have a Restartable Exit State. Clicking the *Rerun Children* button starts the [Rerun](#) program for all the Restartable children. This allows failed children to be restarted after troubleshooting them.

Additional input fields: See *Rerun*



### Kill

The *Kill* button is only displayed if the current job is running at that moment (state Running) and a [Kill](#) program has been defined in the job definition. Clicking the *Kill* button calls the respective Kill program and the job's state is set to "TO KILL". If the Kill program was run successfully, the job's state changes to "KILLED". This enables the job to be terminated during the program runtime.



### Set State

## Details mask for jobs

This allows you to manually set an Exit State for a job. The *Set State* button is displayed if a job fulfils at least one of the following conditions:

- The job is in an Error State (restartable)
- The job is in a Pending State
- The job is suspended and not active and not in a FINAL State

Additional input fields: *Set Exit State, Force, Resume*

**Set Exit State** A valid Exit State can be selected here in the *Set Exit State* field.

**Force** Here you can select whether you want to allow an Exit State that cannot be attained by the job itself using its Exit Code.

**Resume** Here you can select whether a suspended job is to be resumed after the Set State.



### Cancel Run

The *Cancel Run* button is displayed if the job is waiting for resources or dependencies or it has still been allocated on a jobserver. The scheduled job run can be cancelled by clicking the *Cancel Run* button. The job then gets the state "Cancelled". Neither a rerun nor a resume are possible. The job doesn't get an Exit State either. Jobs that are waiting for a cancelled job have to ignore this job or be cancelled as well.



### Disable

The *Disable* button is displayed if the job or batch is or was not yet active. Clicking the *Disable* button disables the job or batch including all the children. All the disabled jobs are no longer executed, and take on their default Exit Status as soon as their dependencies have been fulfilled. This operation cannot be undone.



## Details mask for jobs

### **Comment**

You can enter a comment regarding the job by clicking the *Comment* button. Apart from entering the comment, no other action is performed on the job.

### **Edit Job**

The *Edit Job* button is not an action button because no changes can be made here to the current runtime object. Clicking the *Edit Job* button opens a new window with the [Batches and Jobs Definition Screen](#) for the job definition. The data shown here refers to the definition of the task at the time of the submit.

## 21.5.2 Properties tab

The “Properties” tab contains all the information regarding the job definition and the current state. It looks like this:

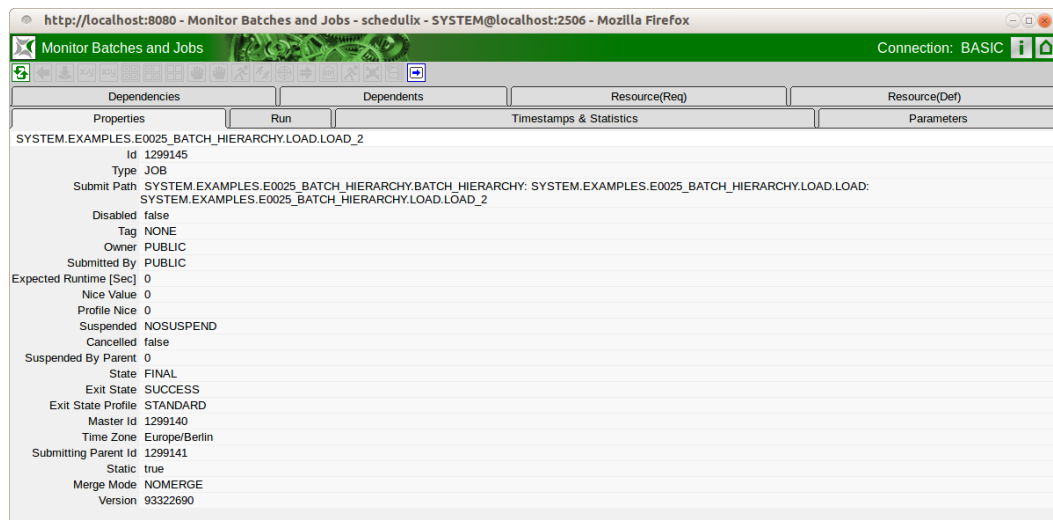


Figure 21.9: Job and batch properties

The fields in the “Properties” tab have the following meanings:

**ID** This is the unique identification number of the job runtime object.

**Type** This is the type of runtime object. More details about types can be found in [Chapter 13.5.1](#).

**Submit Path** This is the full submit path to the runtime object. This means that the comma-separated path used for the submit is given here.

## Details mask for jobs

**Tag** This is the name of the Child Tag. This is only required for a dynamic child and uniquely identifies the current child instance. More details about dynamic children can be found in Chapter [13.5.4.1](#).

**Owner** This is the name of the group that created the job definition for this runtime object.

**Submitted By** This is the name of the group that submitted the runtime instance. If the job was dynamically submitted by a parent program, this is the user who submitted the parent.

**Unresolved Handling** The *Unresolved Handling* field is only shown in the Master Job mask. It shows the setting of the *On Unresolved Error* field from the submit mask.

- Ignore: Unresolved Dependencies are always ignored.
- Suspend: If errors occur due to Unresolved Dependencies, the entire process is submitted as being "suspended".
- None: Default behaviour; in the event of Unresolved Dependencies, the behaviour is as defined in the Dependency Definition.

**Nice Value** This is an input field where the Nice Value can be entered. More information about Nice Values can be found in Chapter [13.5.4.1](#). Since this is the only input field in the dialog box, you have to press the tabulator key after entering the data to enable the *Save* button for saving your changes.

**Suspended** This field indicates whether the runtime object is currently suspended (TRUE) or not (FALSE). The Suspend State can be changed using the *Suspend* and *Resume* buttons.

**Suspend by Parent** This indicates whether the runtime object was suspended by the parent (1) or not (0).

**State** This shows the current state of the runtime object. More details about states can be found in Chapter [21.3](#).

**Error Message** If the state "ERROR" was reported in the *State* field, a detailed description of the error is displayed.

**Exit State** If the job has finished, its resultant Exit State is shown here.

## Details mask for jobs

**Exit State Profile** The Exit State Profile entered in the job definition is shown here. More details about Exit State Profiles can be found in Chapter [13.5.1](#).

**Master Id** This is the Id of the Master Job that was submitted in order to create this runtime object. If the object itself was submitted as the Master Job, this is identical to the Id.

**Submitting Parent Id** This is the Id of the parent runtime object that submitted the current job. If the job does not have a parent, "NONE" is displayed here.

**Static** *Static* is the descriptor indicating whether the current runtime object is a statically (TRUE) or dynamically submitted child (FALSE). More details about dynamic children can be found in Chapter [13.5.4.1](#).

**Merge Mode** This is the Merge Mode that is currently being used. More details about the Merge Mode can be found in Chapter [13.5.4.1](#).

**Version** This is the version number of the Job Definition current at the time of the submit.

### 21.5.3 Run tab

The "Run" tab contains all the information relating to the current details about the Run program, the runtime environment and the computer. The tab looks like this:

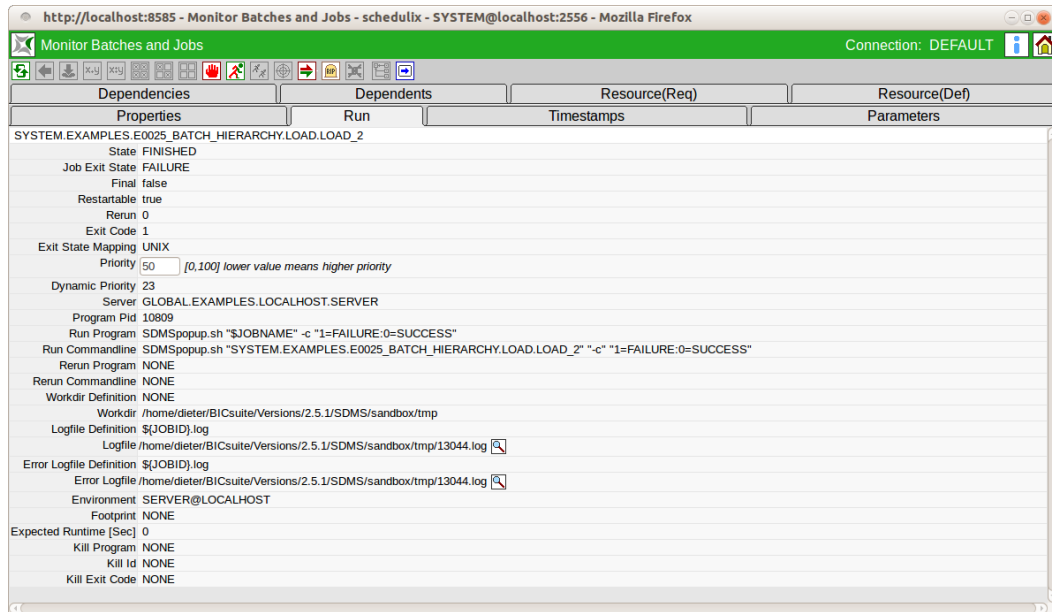


Figure 21.10: Batch und Job Run information

The fields in the "Run" tab have the following meanings:

**State** This shows the current state of the runtime object. More details about states can be found in Chapter 21.3.

**Job Exit State** The Job Exit State shows either the job's current Exit State (if it has finished) or "NONE".

**Final** If the Job Exit State is a Final State, the value in this field is "TRUE"; if it is "NONE" or not a Final State, the value is "FALSE".

**Restartable** If the job is restartable, the value in this field is "TRUE", otherwise it is "FALSE". If the value is TRUE, the *Rerun* button is enabled.

**Rerun** The *Rerun* field shows the number of times the job has been restarted.

**Exit Code** The Exit Code is the exit value that the Run program had when the process finished. More details about the Exit Code can be found in Chapter 13.5.2.

**Exit State Mapping** This is the Exit State Mapping defined for the job in the Batches and Jobs dialog that was used to convert the Exit Code into the Job Exit State. More details about Exit State Mapping can be found in Chapter [13.5.2](#).

**Priority** This is an input field It shows the current priority, which can also be changed here. The new priority is saved by pressing the tabulator key and clicking the *Save* button. More details about priorities can be found in [13.5.2](#).

**Dynamic Priority** This is the dynamic (effective) priority for the job. It rises as the number of submits increases until the job is executed. More details about the dynamic priority can be found in Chapter [12.4.2.3](#).

**Server** The *Server* field shows the current jobserver on which the process is running or has been run.

**Program PID** The *Program PID* is the process Id of the [Run program](#).

**Run Program** This is the command line for the [Run program](#) defined in the Batches and Jobs dialog. The variables and parameters are still displayed here under their names and are not substituted.

**Run Command Line** This is the command line as defined in the Batches and Jobs dialog for the [Run program](#) with substituted values. This is the view as it was handed over to the processing shell. All the variables and parameters were substituted for their current values.

**Rerun program** This is the command line as defined in the Batches and Jobs dialog for the [Rerun program](#). The variables and parameters are still displayed here under their names and are not substituted. NONE is shown here if a Rerun program has not been defined.

**Rerun Command Line** This is the command line as defined in the Batches and Jobs dialog for the [Rerun program](#) with substituted values. This is the view as it was handed over to the processing shell. All the variables and parameters were substituted for their current values. NONE is shown here if a rerun has not taken place.

**Workdir Definition** The *Workdir Definition* field shows the directory from where the process is being run. If "None" is shown here, the default working directory for the executing jobserver is used.

## Details mask for jobs

**Workdir** This is the [Workdir](#) that is being used (or is to be used) in the current job. The substituted parameters and variables are displayed here if the job has finished or it is still running (otherwise just the parameters).

**Logfile Definition** The *Logfile Definition* corresponds to the definition of the [log files](#) in the Batches and Jobs dialog.

**Logfile** The name and path of the currently used [log files](#) are shown here (if the job has finished or it is still running), otherwise NONE is displayed. Possible variables and parameters are substituted for the actual values. Clicking the button



opens a new window displaying the contents of the log files. Should this not be possible, please contact your system administrator.

**Error Logfile Definition** The Error Logfile Definition corresponds to the definition of the [error log files](#) in the Batches and Jobs dialog.

**Error Logfile** The name and path of the currently used [error log files](#) are shown here (if the job has finished or it is still running), otherwise NONE is displayed. Possible variables and parameters are substituted for the actual values. The button



has the same function as described above for log files.

**Environment** This field corresponds to the definition of the [environment](#) in the Batches and Jobs dialog.

**Footprint** This field corresponds to the definition of the [footprint](#) in the Batches and Jobs dialog.

**Expected Runtime** This corresponds to the definition in the [Batches and Jobs dialog](#).

**Kill Program** This corresponds to the definition of the [Kill program](#) in the Batches and Jobs dialog.

**Kill Id** The Kill Id contains the process Id of the [Kill program](#) if this has been started. If a Kill action has not yet been performed, the Kill Id is NONE.

## Details mask for jobs

**Kill Exit Code** The Kill Exit Code is the result returned to the jobserver by the [Kill program](#) after the kill process has finished. This can be used to determine whether a kill attempt has succeeded or failed if this is configured in the Kill program.

### 21.5.4 Timestamps & Statistics tab

The "Timestamps & Statistics" tab shows the different dates and times of the respective jobs. These times give an overview of the temporal behaviour and duration of the job runs. The tab looks like this:

#	Start	End	Runtime	Exit State	Logfile	Error Logfile	Workdir	Commandline
0	18.07.2017 12:04:44	18.07.2017 12:04:54	10s	FAILURE	873085.log	873085.log	/home/dieter/BICsuite/Versions/work/SDMS/sandbox/tmp	SDMSpopup.sh "SYSTEM.EXAMPLES.E0010_SINGLEJOB.SINGLEJOB"
1	18.07.2017 12:05:40	18.07.2017 12:06:14	34s	FAILURE	873085.log	873085.log	/home/dieter/BICsuite/Versions/work/SDMS/sandbox/tmp	SDMSpopup.sh "SYSTEM.EXAMPLES.E0010_SINGLEJOB.SINGLEJOB"
2	18.07.2017 12:06:30	18.07.2017 12:07:14	44s	SUCCESS	873085.log	873085.log	/home/dieter/BICsuite/Versions/work/SDMS/sandbox/tmp	SDMSpopup.sh "SYSTEM.EXAMPLES.E0010_SINGLEJOB.SINGLEJOB"

**Statistics**

Final	18.07.2017 12:07:14
Process Time	2m35s
Active Time	1m28s
Idle Time	1m7s
Idle Percentage	43 %
Dependency Wait Time	
Suspend Time	
Synchronize Wait Time	2s
Resource Wait Time	
Jobserver Handling Time	6s
Restartable Time	59s
Child Wait Time	

Figure 21.11: Batch and job timestamps

The tab is for information purposes only; no entries can be made here.

The fields in the "Timestamps & Statistics" tab have the following meanings:

**Submit** The submit time for the current job is shown in the *Submit* timestamp. If it is a static job this is the submit time for the Master Job; in the case of dynamic jobs it is the time when the parent job performed the dynamic submit. The submit date is the same as the time when the job switched to the "Dependency Wait" state. The *Runs* table lists every execution of the job. Normally this table only contains one line. However, if the job was restarted due to any errors, the table can contain multiple lines.

**Start** The column *Start* shows when the Run program was started by the jobserver.

**End** The column *Finish* shows when the job was completed.

**Runtime** The column *Runtime* shows the runtime of the run.

## Details mask for jobs

**Exit State** The column *Exit State* contains the Exit Status of the run.

**Logfile** The column *Logfile* contains the name of the logfile for standard out. A click on the icon (Magnifying Glass) after the Name opens the logfile for display.

**Error Logfile** The column *Error Logfile* contains the name of the logfile for standard error. A click on the Icon (Magnifying Glass) after the Name opens the error logfile for display. The icon is only displayed if the name of the logfile and error logfile differ.

**Workdir** The column *Workdir* shows the working directory of the job.

**Commandline** The column *Commandline* shows the commandline executed.

**Jobserver** The column *Jobserver* shows the name of the Jobserver agents an, which executed the job.

**Exit Code** The column *Exit Code* contains the Exit Code of each execution.

**PID** The column *PID* contains the system process id of the executed program.

**Synchronize** The column *Synchronize* indicates the time when the job switched to the "Synchronize Wait" state. That is the time when all the job's possible dependencies were fulfilled.

**Resource** The column *Resource* shows the time of the status transition to Resource Wait. This means that all the required [Synchronizing Resources](#) had been fulfilled and allocated to the job at this time.

**Runnable** The column *Runnable* contains the time when the job entered the "Runnable" state. This means that all the dependencies and resource requirements could be fulfilled. From now on the job can be processed by a server job.

**Final** The *Final* timestamp shows when the job that has created a Final State finished. If the job was finished but it has a Restartable State, NONE is entered here.

**Process Time** *Process Time* is the time, the job has spent after all dependencies have been fulfilled.

**Active Time** *Active Time* is the time, the job or one of its children was active.



## Details mask for jobs

**Idle Time** *Idle Time* is the time, the job was inactive and none of its children were active.

**Idle Percentage** *Idle Percentage* is the percentage of the *Idle Time* of the *Process Time*.

**Dependency Wait Time** *Dependency Wait Time* is the time, the jobs waited on dependencies.

**Suspend Time** *Suspend Time* is the time, the job was suspended.

**Synchronize Wait Time** *Synchronize Wait Time* is the time, the job spent waiting on synchronizing resources.

**Resource Wait Time** *Resource Wait Time* is the time, the job spent waiting on system resources.

**Jobserver Handling Time** *Jobserver Handling Time* is the time overhead of the jobserver agent.

**Restartable Time** *Restartable Time* is the time, the job spent waiting for a restart after an error.

**Child Wait Time** *Child Wait Time* is the time, the job waited for its children to become FINAL.

### 21.5.5 Dependencies tab

All the current job's dependencies on other jobs are displayed in the "Dependencies" tab. The tab looks like this:

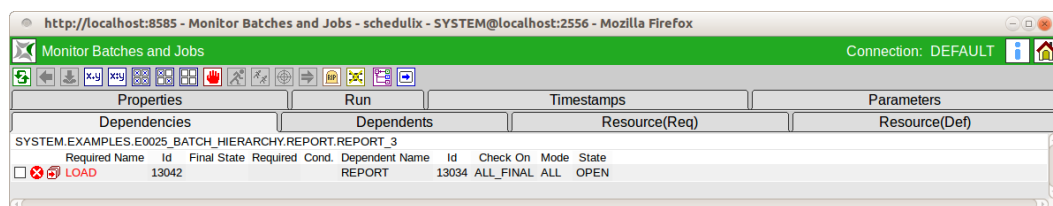


Figure 21.12: Batch und job dependencies

This dialog gives an overview of all the required dependencies, their current state and the possibility for deliberately ignoring dependencies.

The mask contains the following buttons:

### Ignore Dependencies

The *Ignore Dependencies* button is used to deliberately ignore one or more dependencies. All the dependencies that are to be ignored must have been previously selected in the mask. When you now click the button, a new window opens where you can select the Ignore mode and enter a remark. The window looks like this:

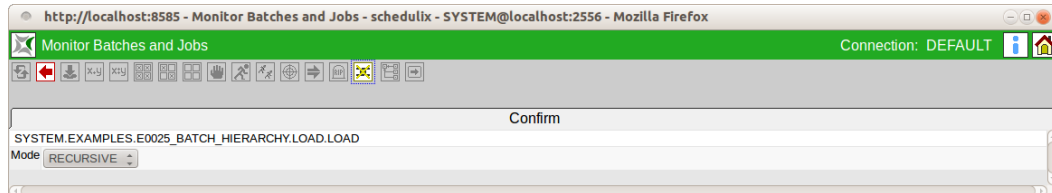


Figure 21.13: Batches and Jobs Ignore Dependencies confirmation mask

The ignore mode has to be entered in this mask. By default, all the children of a Submitted Entity inherit its dependencies: if B is required by A, each of A's children (CA) is likewise dependent on B.

The Ignore mode can now be used to determine the behaviour of the inherited dependency at the children.

The following options are available:

1. Recursive

#### **This is the default behaviour.**

The Ignore refers not only to the current job, but also to the inherited dependencies on the required job held by the children of the current job. This means that the parent job ignores this inherited dependency and all its children ignore it as well. If a child has an explicitly stated dependency on the required job which was not inherited, this is not ignored but is observed.

Figure 21.14 shows an example.

2. Job Only

Only the job itself ignores the dependency. All the children observe the inherited dependency and wait for the required job. This allows the parent job to be executed while forcing its children to wait.

Figure 21.15 shows an example.

Having entered the Ignore mode and then a comment in the Comment field, you can perform the action by clicking the *Ignore Dependency* button again. The tab list shows all the dependencies that have to be fulfilled when the job starts and their current states. In this list, you can make a selection that can be used to ignore dependencies by clicking the *Ignore Dependencies* button.

Details mask for jobs

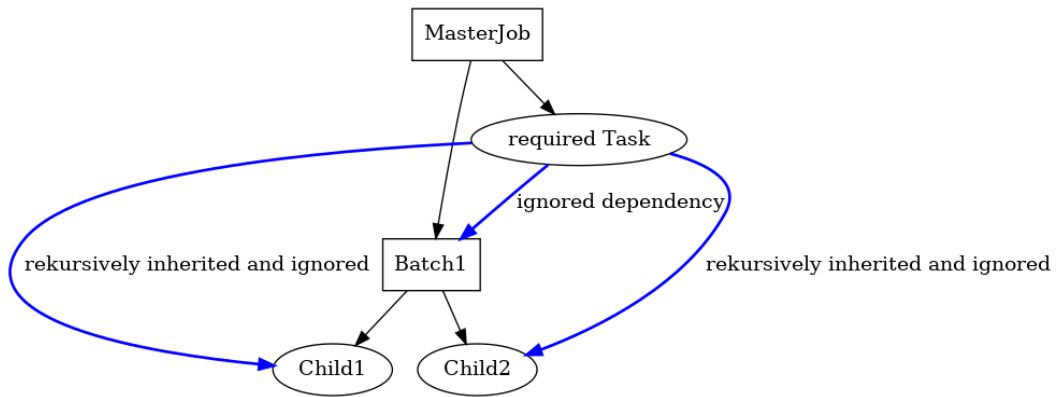


Figure 21.14: Example of a Recursive Ignore

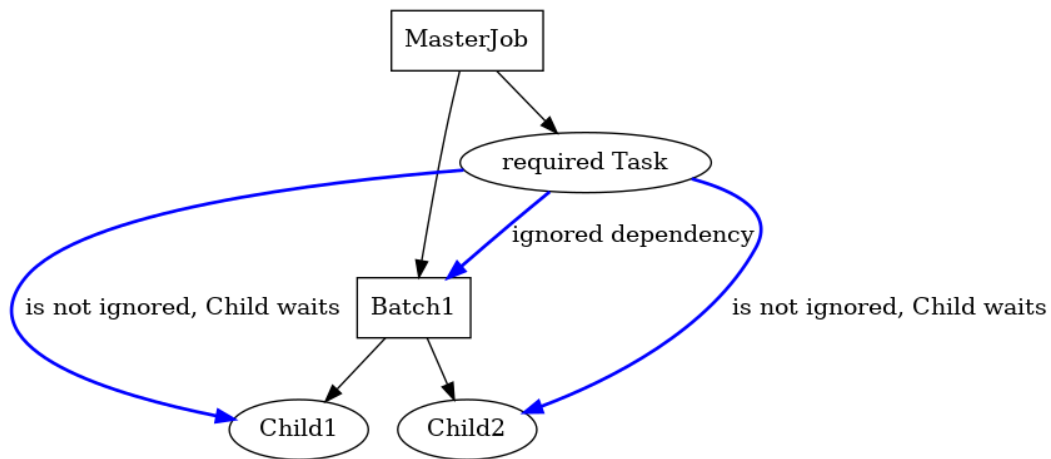


Figure 21.15: Example of a Job Only Ignore

The following fields are shown in the list:

**Icon field** The current state of the dependency is shown in the *Icon* field. The following options are available:



The dependence has not yet been fulfilled or can no longer be fulfilled. Should this icon be visible next to a dependency with the [Dependency Mode ALL](#), the current job cannot be executed by this dependency.

In the [Dependency Mode ANY](#), at least one dependency must not be red for the job to be executed.



The dependency has been fulfilled. In the [Dependency Mode ANY](#), at least one dependency must have the state Green for the job to be able to run. In the [Dependency Mode ALL](#), all the entries must be green for the job to be able to run.

**Required Name** This is the name of the job that has to be a prerequisite for starting the current job run. You can switch to its runtime definition and check its state by clicking the name.

**ID** The runtime Id of the Required Job is shown in this field.

**Final State** If the Required Job has been completed, the job's Final State is shown here. If a Final State is entered here and the dependency is not fulfilled, the reason for this happening must be an incorrect Final State for this dependency. This means that there is a difference between a Final State and a Required State. If the job is to be executed now after all, this can only be done with an Ignore by clicking the *Ignore Dependency* button. If the dependencies in the list are in the [Dependency Mode ANY](#), at least one another dependency has to be fulfilled.

**Required** The required Final State for the job is shown in this field. For the dependency to be fulfilled, the Final State must be the same as the Required State or the Required State must be NONE.

**Dependent Name** The job demanding this dependency is named in this field. This is normally the currently selected job, but it can also be one of the job's parents.

**Check On** The *Check On* field describes the relation of the completion of the Required Job. The following variants are used:

1. All Final

The Required Job and all its children must be final.

2. Job Only

Only the Required Job has to be final; the children are ignored.

More details about the Check On function can be found in [Chapter 13.5.6.1](#).

**ID** This is the Id of the dependent job.

**Mode** The *Mode* defines how the list of jobs is to be handled. The following variants are used:

1. ALL  
All the list's dependencies must be fulfilled before the job can start.
2. ANY  
All least one of the list's dependencies must be fulfilled before the job can start.

More details about Mode can be found in Chapter [13.5.6](#).

**State** This is the current state of the dependency relationship. The following variants are used:

1. OPEN  
The Required Job has still not attained a Final State (it has not yet run, is still running, or an error has occurred). The dependency has not yet been fulfilled, although it can be fulfilled by attaining the correct Final State.
2. FULFILLED  
The Required Job has finished and attained the required Final State. The dependency has been fulfilled.
3. FAILED  
The Required Job has finished and attained an incorrect Final State. The dependency can no longer be fulfilled. In the [Dependency Mode ALL](#), the current job can only be made to start by ignoring the dependency. In the [Dependency Mode ANY](#), at least one of the other dependencies is being fulfilled.

### 21.5.6 Dependents tab

All the other jobs' dependencies on the current job are displayed in the "Dependents" tab. The tab looks like this:

This dialog gives an overview of all the existing dependencies, their current state and the possibility for deliberately ignoring dependencies.

The mask contains the following buttons:

#### Ignore Dependencies

The *Ignore Dependencies* button is used to deliberately ignore one or more dependencies. All the dependencies that are to be ignored must have been previously selected in the mask. When you now click the button, a new window opens

## Details mask for jobs

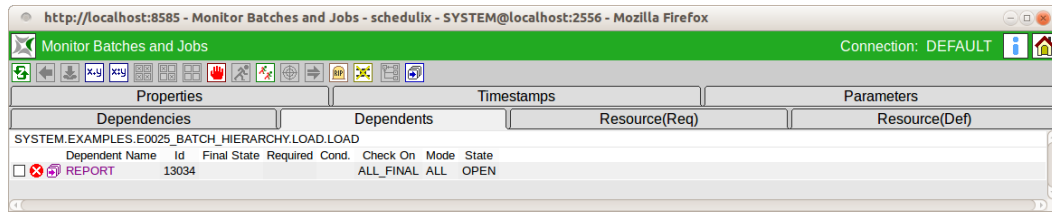


Figure 21.16: Batch and Job Dependents tab

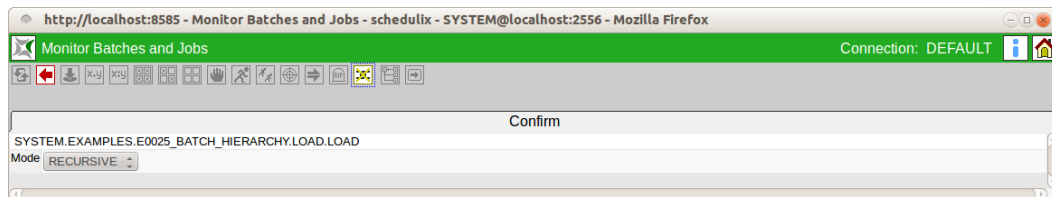


Figure 21.17: Batch and Job Ignore Dependencies

where you can select the Ignore mode and enter a remark. The window looks like in Figure 21.17.

The ignore mode has to be entered in this mask. By default, all the children of a Submitted Entity inherit its dependencies: if B is required by A, each of A's children (CA) is likewise dependent on B. The Ignore mode can now be used to determine the behaviour of the inherited dependency at the children.

The following options are available:

1. Recursive

**This is the default behaviour.**

The Ignore refers not only to the dependent job, but also to the inherited dependencies on the current job held by the children of the dependent job. This means that the parent job ignores this inherited dependency and all its children ignore it as well. If a child has an explicitly stated dependency on the required job which was not inherited, this is not ignored but is observed.

2. Job Only

Only the dependent job itself ignores the dependency. All the children observe the inherited dependency and wait for the current job. This allows the parent job to be executed while forcing its children to wait.

Having entered the Ignore mode and then a comment in the Comment field, you can perform the action by clicking the *Ignore Dependency* button again. The tab list shows all the dependencies that have to be fulfilled before other jobs are able to

start together with their current states. In this list, you can make a selection that can be used to ignore dependencies by clicking the *Ignore Dependencies* button. The following fields are shown in the list:

**Icon field** The current state of the dependency is shown in the *Icon* field. The following options are available:



The dependence has not yet been fulfilled or can no longer be fulfilled. Should this icon be visible next to a dependency with the [Dependency Mode ALL](#), the dependent job cannot be executed by this dependency.

In the [Dependency Mode ANY](#), at least one other dependency (which is not visible here) of the dependent job must not be red for the dependent job to be able to run.



The dependency has been fulfilled. In the [Dependency Mode ANY](#), at least one dependency must have the state Green for the dependent job to be able to run. In the [Dependency Mode ALL](#), it may be necessary for other dependencies (which are not visible here) of the dependent job to be fulfilled for the dependent job to be able to run.

**Dependent Name** This is the name of the job that is dependent upon the fact that the current job has been executed. You can switch to its runtime definition and check its state by clicking the name.

**ID** The runtime Id of the dependent job is shown in this field.

**Final State** If the dependent job has already been completed, the job's Final State is shown here.

**Required** The required Final State for the current job is shown in this field.

**Cond** This field shows whether an additional condition has been defined for the dependency.

**Check On** The *Check On* field describes the relation of the completion of the current job. The following variants are used:

1. All Final  
The current job and all its children must be final.
2. Job Only  
Only the current job itself has to be final; the children are ignored.

More details about the Check On function can be found in Chapter [13.5.6.1](#).

**Mode** The *Mode* defines how the list of jobs is to be handled. The following variants are used:

1. ALL

All the current job's dependencies must be fulfilled before the job can start.

2. ANY

At least one of the current job's dependencies must be fulfilled before the job can start.

More details about Mode can be found in Chapter [13.5.6](#).

**State** This is the current state of the dependency relationship. The following variants are used:

1. OPEN

The current job has still not attained a Final State (it has not yet run, is still running, or an error has occurred). The dependency has not yet been fulfilled, although it can be fulfilled by attaining the correct Final State.

2. FULFILLED

The current job has finished and attained the required Final State. The dependency has been fulfilled.

3. FAILED

The current job has finished and attained an incorrect Final State. The dependency can no longer be fulfilled. In the [Dependency Mode ALL](#), the dependent job can only be made to start by ignoring the dependency. In the [Dependency Mode ANY](#), at least one other dependency (which is not visible here) of the dependent job must be fulfilled.

### 21.5.7 Resource(Req) tab

All the information about the current state of the requested resource is displayed in the "Resource(Req)" tab.

All the resource instances and their allocations to scopes can be viewed in this tab. It is also possible to ignore requests for resources.

The *Ignore Resources* button has the following meaning:



## Details mask for jobs

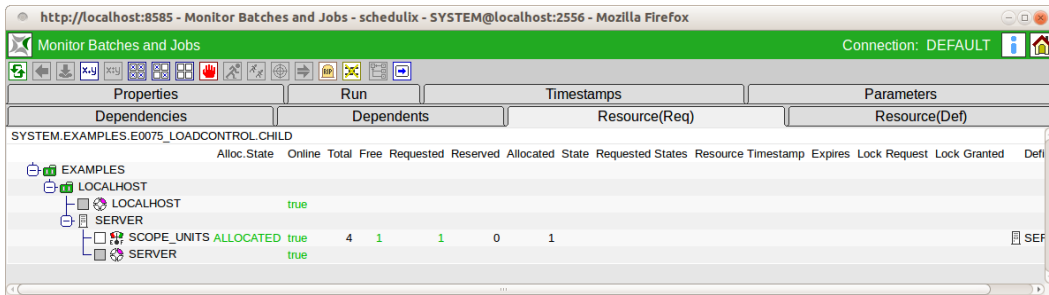


Figure 21.18: Job Resource Requirements



### Ignore Resources

The *Ignore Resources* button is used to deliberately ignore one or more resource requests. All the resource requests that are to be ignored must have been previously selected in the mask. When you now click the button, a new mask opens where you can enter a remark.

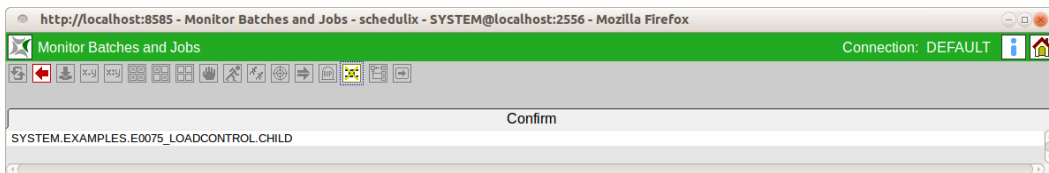


Figure 21.19: Job Ignore Resource Requirement

After you have entered a comment in the Comment field, clicking the *Ignore Resource* button again performs the action.

All the requested resources of the job are displayed in the Resources list. These are displayed hierarchically using the scopes and jobserver where the requested resources are located. The following columns are shown in the list:

All the requested resources together with their names are displayed in the first column of the list in a hierarchy (if the resources are located in scopes and jobserver).

The resources for an Ignore can be selected by clicking the preceding *Ignore Dependencies* button. If the preceding field is grey this is not possible (static resources).

Clicking the resource name opens the [12.4.2.3](#) dialog, where the current allocation of the resource can be examined.

**Allocation State** The Allocation State is the current state of the resource request. The following variants are used:

1. None

## Details mask for jobs

These are static or system resources that are checked at the time of the submit. If these are no longer present or are offline, an error message is returned at the submit time.

### 2. Blocked

The resource is being used by another job and cannot be allocated. The job is now waiting for this resource. There may be different reasons for a block. The criteria preventing an allocation are shown in red in the list.

### 3. Allocated

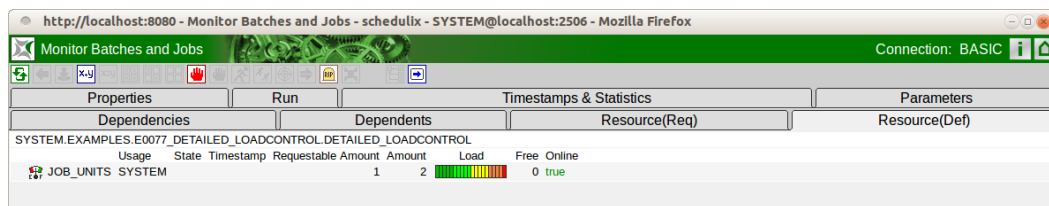
The resource has been allocated to the current job, fulfilling the requirement.

### 4. Available

The resource is available but has not yet been allocated.

## 21.5.8 Resource(Def) tab

All the information about the current state of the resources defined by this Submitted Entity is displayed in the "Resource(Def)" tab. The tab looks like this:




Dependencies	Dependents	Resource(Req)	Resource(Def)				
SYSTEM.EXAMPLES.E0077_DETAILED_LOADCONTROL.DETAILED_LOADCONTROL							
Usage	State	Timestamp	Requestable Amount	Amount	Load	Free	Online
JOB_UNITS SYSTEM			1	2		0	true

Figure 21.20: Batch and Job Defined Resources

All the defined resources of the job are displayed in the Resources list. The following columns are shown in the list:

All the defined resources together with their names are displayed in the first column of the list. Clicking the resource name opens the [12.4.2.3](#) dialog, where the current allocation of the resource can be examined.

**Usage** The *Usage* field specifies the "Resource" type. More details about resource types can be found in the [Named Resource](#) dialog.

**State** The *State* field shows the current state of the resource if the usage has been set to "Synchronizing" and a [Resource State Profile](#) has been assigned.

**Timestamp** The *Timestamp* shows the time of a resource's last status transition if the resource usage is "Synchronizing" and a [Resource State Profile](#) has been assigned.

## Details mask for jobs

**Requestable Amount** If the resource usage is "Synchronizing" or "System", the *Requestable Amount* is the maximum amount that can be requested by a job.

**Amount** If the resource usage is "Synchronizing" or "System", the *Amount* is the total amount that is available for all the jobs.

**Load** If the resource usage is "Synchronizing" or "System", *Load* graphically displays where the resource is allocated.

**Free** If the resource usage is "Synchronizing" or "System", *Free* is the non-allocated amount that is still available for all the jobs.

**Online** This indicates whether the resource is "online", i.e. can be requested at this time.

### 21.5.9 Parameters tab

All the input and output parameters currently defined in this job and their actual values are shown in the "Parameters" tab.

The tab looks like this:

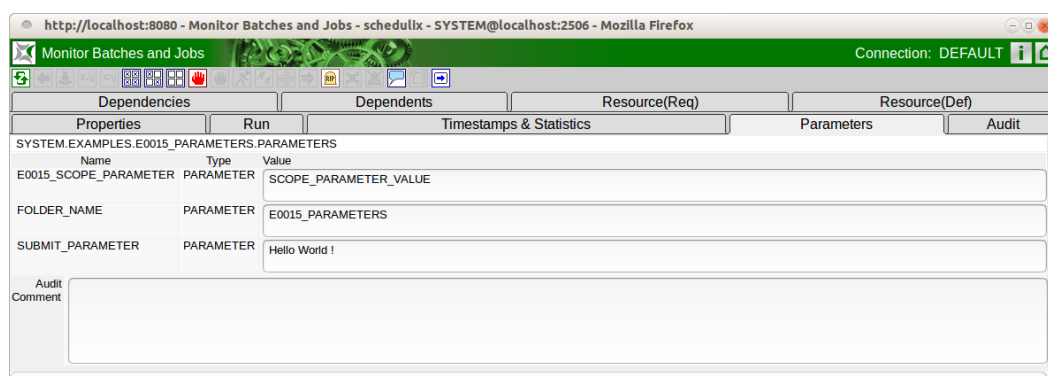


Figure 21.21: Batch and job parameters

This tab is for information purposes only; no changes can be made here.

The "Parameters" list shows all the parameters that are currently defined in the job. If the active user is a member of either the "ADMIN" group or another group with "operate" privileges for the job or batch, parameters of type PARAMETER and RESULT can be edited. In this case the field *Audit Comment* is displayed. The reason for the modification can be entered here and will be recorded in the Audit Trail.

## Details mask for jobs

Information about parameters can be found in [Chapter 13.5.10](#).

# 22 Search Running Jobs

## 22.1 View

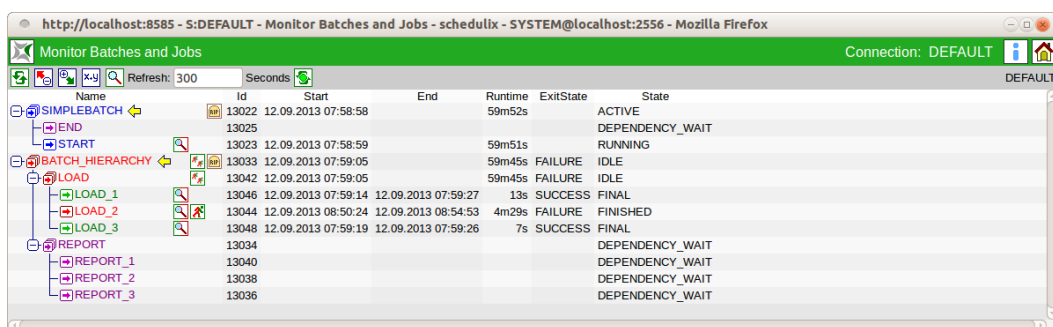


Figure 22.1: Search Running Jobs

## 22.2 Concept

### 22.2.1 In short

In the *Search Running Jobs* dialog you can search for and view all the currently running or completed jobs, batches and milestones.

### 22.2.2 Detailed description

In this console you can check all the jobs, restart failed jobs, ignore resources, etc.

The Search Running Jobs window is also just a "default" view (of the bookmark "Default Search System") and can be customised and modified by the user accordingly. The "default" implementation is described here.

## 22.3 Navigator

The navigation window is the same as the ["Details" mask for the Running Master Job](#). More information can be found there.

## 22.4 Navigator Details query mask

When you call the dialog, in contrast to the [Running Master Jobs](#) dialog the query mask opens first.

The query mask looks like this:

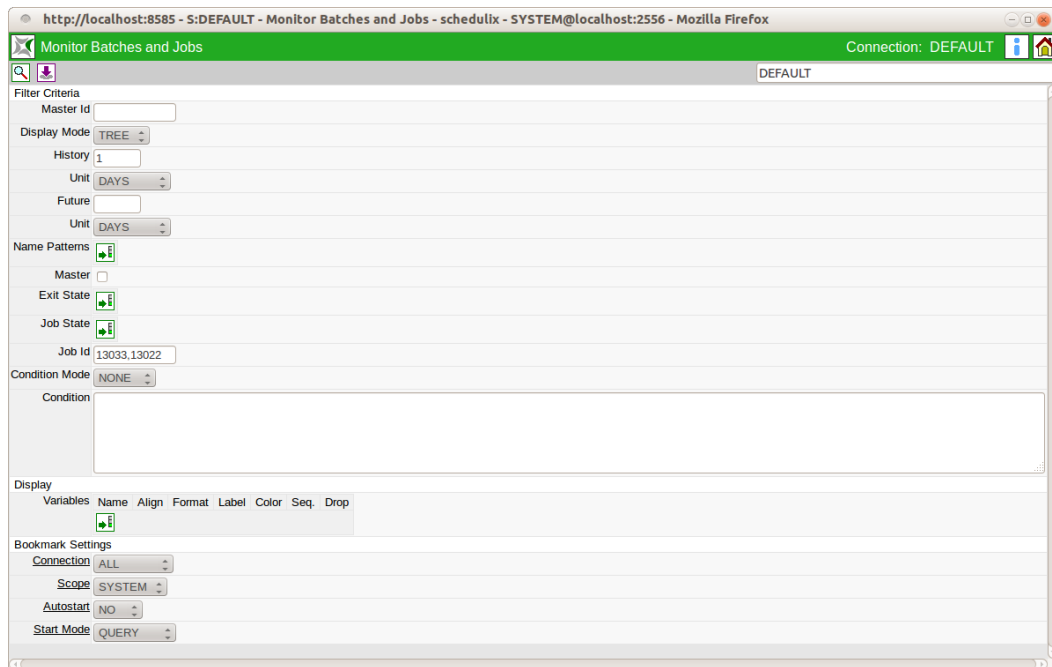


Figure 22.2: Search Running Jobs query mask

The query mask for the Navigation Details corresponds to the [query mask for Running Master Jobs](#).

The following fields are additionally defined here, however:

**Master Id** A runtime Id for a Master Job can be entered in the *Master Id* field. All the found jobs must have been submitted by this Master Id.

## Navigator Details query mask

**Display Mode** The list view can be configured in the *Display Mode* field. The following options are available:

1. LIST

The output is displayed as a list.

2. TREE

The output is shown as a hierarchy based on the parent-child hierarchy. The hierarchy levels can be expand and closed again in a tree.

**Master** The *Master* field is a filter (if the switch has been set) for displaying just Master Submittable jobs.

**Job State list** In the *Job State list*, jobs which have the entered Job State can be filtered. If no Job States have been entered, they are ignored by the search.

**Job Id** In this field you can enter a specific *Job Id* that you want to search for. A maximum of one job can be returned as a result.

**Enable Bulk Operations** This flag whether checkboxes for selection and bulk operations will be available in the navigator view.

**Bulk Select Hits only** If this flag is set, only search hits can be selected for bulk operations.





# 23 Calendar

## 23.1 View

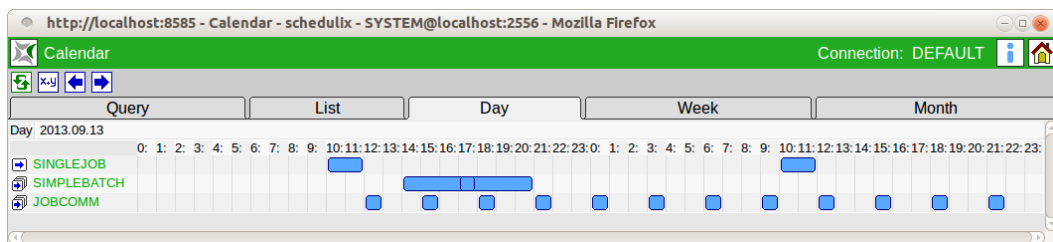


Figure 23.1: Calendar

## 23.2 Concept

### 23.2.1 In short

The calendar function gives an overview of scheduled executions of jobs and batches configured using the Time Scheduling module.

For a scheduled job or batch to be included in this overview, the calendar function must have been activated (Calendar = active) when the schedule was created.

## 23.3 Detailed description

The calendar is displayed in different ways. Firstly, there is the simple, chronologically ordered list in which the pending submits are listed. The evaluation period can be changed to prevent the list from becoming unnecessarily long and thus confusing. By default, only the next 24 hours are initially displayed.

The second view is the day view. Here the day is displayed as a period of 24 hours. Bars indicate when which job is to be started and, provided that this information is maintained by the user, how long it lasts.

The third and fourth views correspond to the day view, but here the period is displayed with a lower granularity. The week view shows seven days split up into 6-hour periods. The month view shows (maximum) 31 days.

## Detailed description

In the week and month views, the jobs or batches that are to be executed are also shown as bars. However, the runtime of the objects is not as clearly displayed as in the day view because the bars are at least as long as the smallest unit in the respective view.

This means that a batch with an expected runtime of eight hours is actually displayed as 8 hours in the day view. In the week view, however, it is represented by two fields (i.e. 12 hours). In the month view it visually takes up a whole day.

### 23.3.1 Query tab

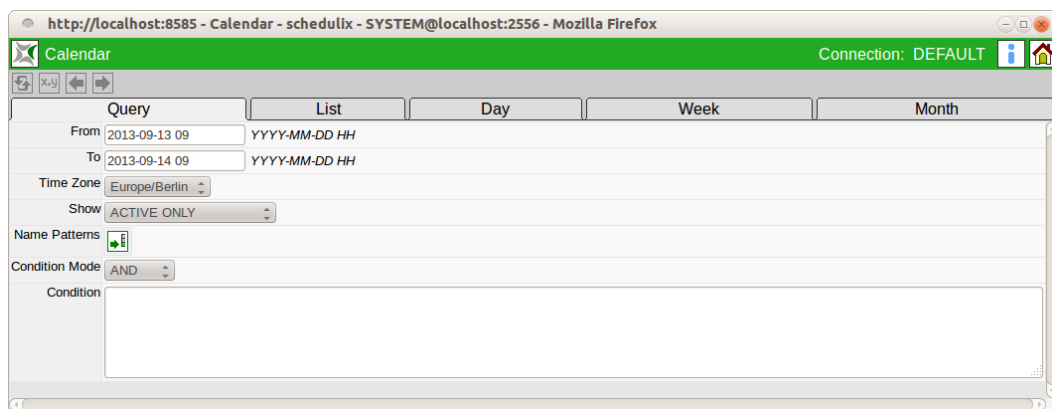


Figure 23.2: Calendar Query tab

In the "Query" tab you can specify search criteria for the Job Definitions that are to be displayed. All the other tabs are affected by these restrictions.

The *From* and *To* fields define the period for the start times of the jobs and batches that are to be displayed.

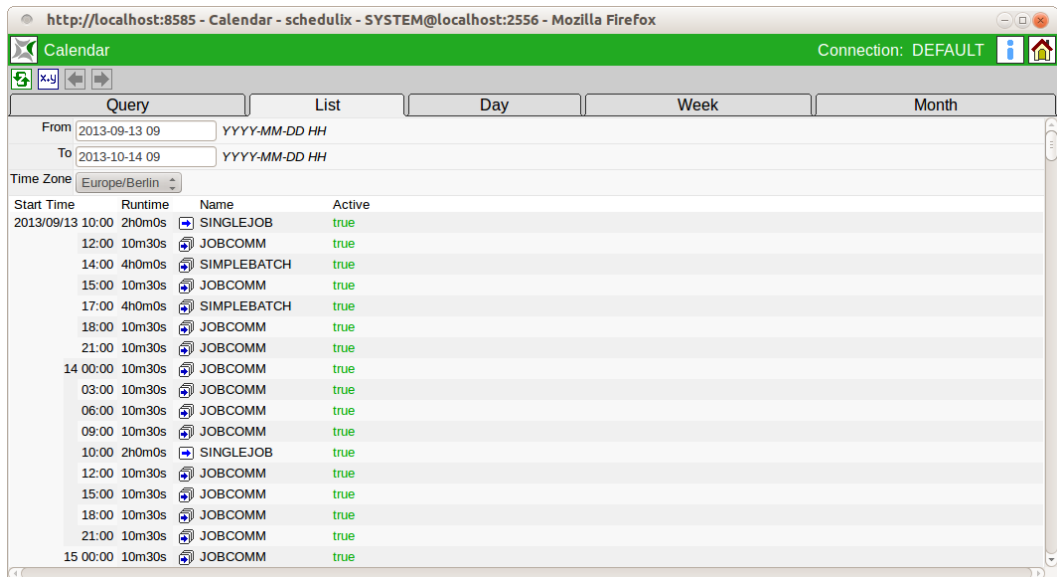
The time zone to be used for the calendar can be selected in the *Time Zone* field.

Simple conditions linked with Boolean operators can be entered in the *Condition* field. In the simple conditions you can test parameters with specific contents. The parameters are addressed with **JOB.parametername**. The following example shows a (syntactically) valid condition:

```
job.developer == 'ronald' or  
job.developer == 'dieter'
```

## Detailed description

### 23.3.2 List tab



Start Time	Runtime	Name	Active
2013/09/13 10:00	2h0m0s	SINGLEJOB	true
12:00	10m30s	JOBCOMM	true
14:00	4h0m0s	SIMPLEBATCH	true
15:00	10m30s	JOBCOMM	true
17:00	4h0m0s	SIMPLEBATCH	true
18:00	10m30s	JOBCOMM	true
21:00	10m30s	JOBCOMM	true
14 00:00	10m30s	JOBCOMM	true
03:00	10m30s	JOBCOMM	true
06:00	10m30s	JOBCOMM	true
09:00	10m30s	JOBCOMM	true
10:00	2h0m0s	SINGLEJOB	true
12:00	10m30s	JOBCOMM	true
15:00	10m30s	JOBCOMM	true
18:00	10m30s	JOBCOMM	true
21:00	10m30s	JOBCOMM	true
15 00:00	10m30s	JOBCOMM	true

Figure 23.3: Calendar List tab

The "List" tab shows all the start times of the scheduled jobs and batches for the period set in the *From* and *To* fields in a simple list.

The time zone to be used for the calendar can be selected in the *Time Zone* field.

The scheduled start time is shown in the column *Start time*. For reasons of clarity, only the data that has changed in comparison with the previous row is displayed in this column and row. If two successive rows refer to the same day and hour, only the minutes of the time are shown in the second row.

### 23.3.3 Day tab

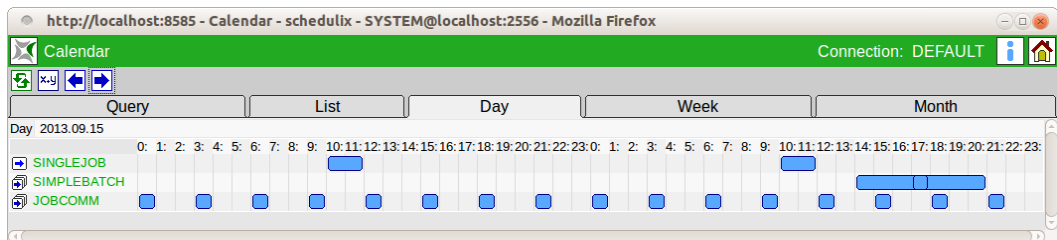


Figure 23.4: Calendar Day tab

The "Day" tab graphically shows all the scheduled jobs and batches for both the selected day and the following day.

## Detailed description

If a job or batch is probably still active at the beginning, it is shown in the row "KEEP\_RESOURCE" as illustrated here. The representation of overlapping runs can also be seen in the row.

### 23.3.4 Week tab

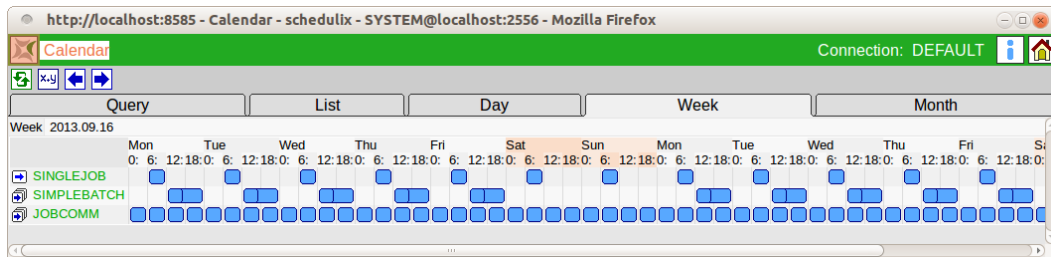


Figure 23.5: Calendar Week tab

The "Week" tab graphically shows all the scheduled jobs and batches for both the selected week and the following week.

### 23.3.5 Month tab

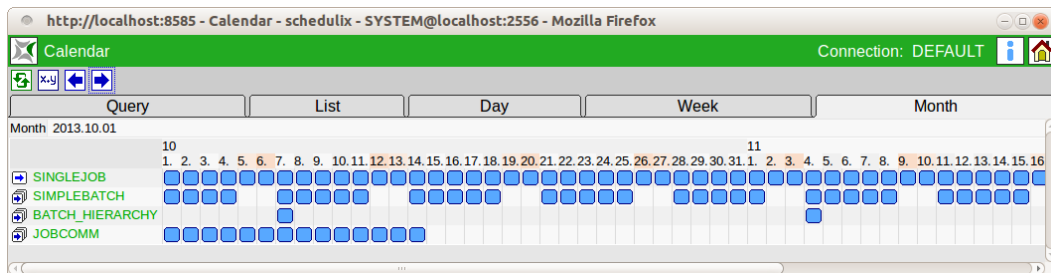


Figure 23.6: Calendar Month tab

The "Month" tab graphically shows all the scheduled jobs and batches for both the selected month and the following month.

# 24 System Information

## 24.1 View

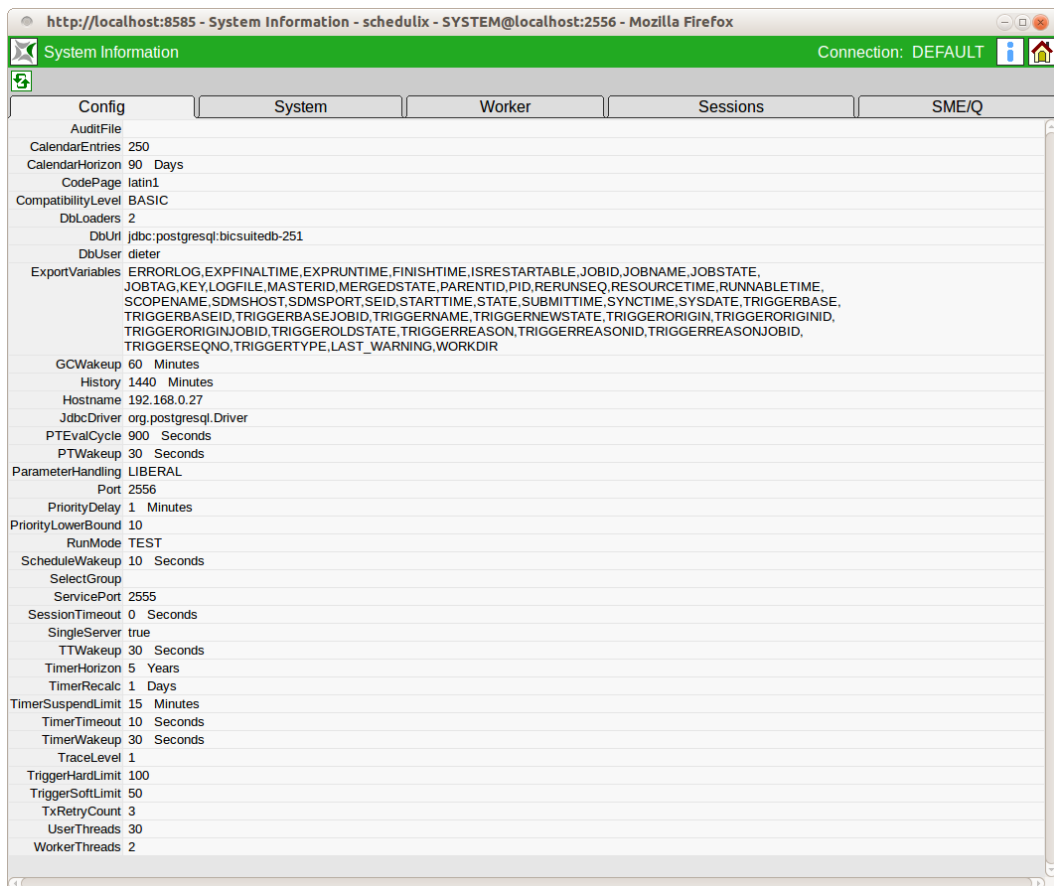


Figure 24.1: System Information

## 24.2 Concept

### 24.2.1 In short

The menu option "System information" in the main menu provides an overview of the server and system parameters, a list of worker activities, as well as a list of the current sessions.

### 24.2.2 Config tab

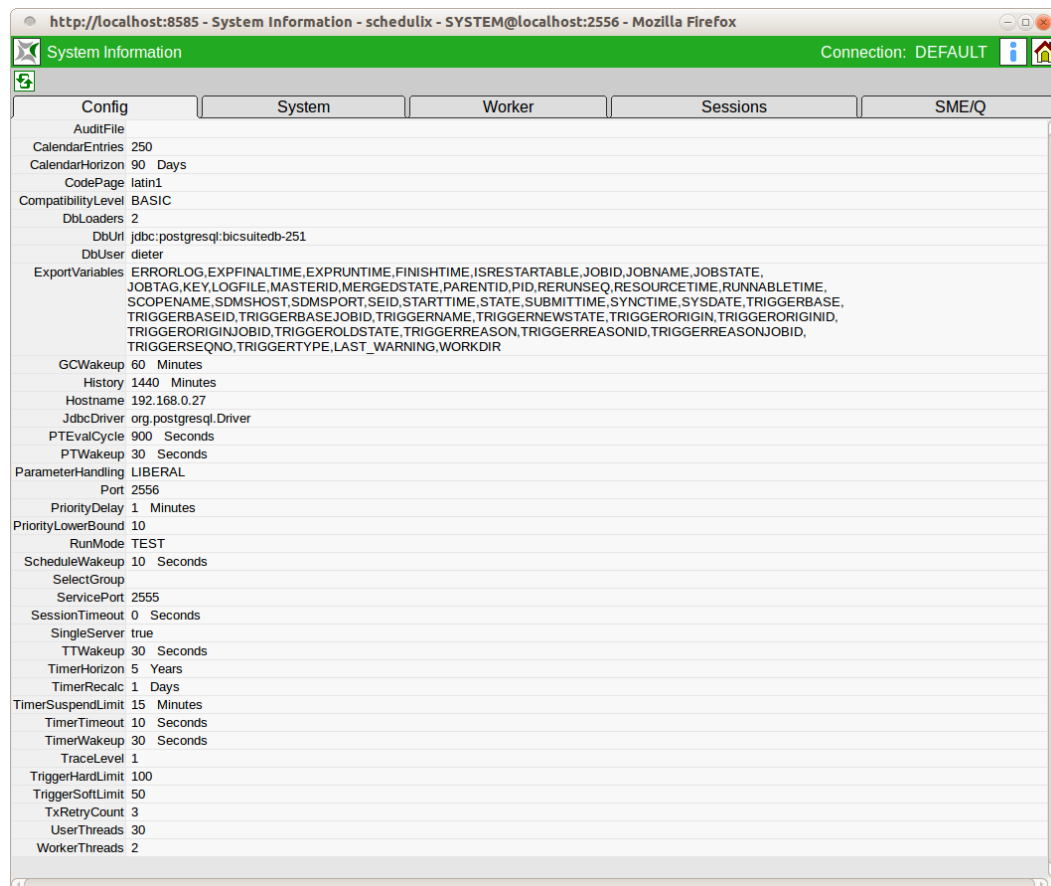


Figure 24.2: System configuration

The "Config" tab shows the current server configuration. The parameters are sorted here in alphabetical order. To change a parameter, you have to modify the `$BICSUITECONFIG/server.conf` file and restart the Scheduling Server.

If the environment variable `BICSUITECONFIG` has not been set, `$BICSUITEHOME/etc` is used as the configuration directory.

For security reasons, not all the server parameters are displayed here. In particular this includes the access information for the database system. The currently used server parameters are as follows:

### **CalendarEntries**

The parameter **CalendarEntries** defines the maximum number of calendar entries per Scheduled Event. The parameter **CalendarHorizon** normally takes effect first, but if there is an extremely large number of entries in the horizon, this parameter prevents the creation of nonsensical volume of calendar entries. As the horizon can be defined for each schedule, a sort of Denial of Service attack could be made on it. If this parameter is not defined, the default value of 300 entries applies.

### **CalendarHorizon**

The parameter **CalendarHorizon** defines the default horizon for the calendars. If this parameter is not defined, the default value of 62 days applies (always at least two months).

### **CodePage**

The parameter **CodePage** defines the active character encoding. This parameter is now obsolete.

### **CompatibilityLevel**

The parameter **CompatibilityLevel** defines which options and commands are recognised by the server as being valid. The possible values are: **BASIC**, **PROFESSIONAL** and **ENTERPRISE**. The maximum level is dependent upon the licensed version.

By default the **CompatibilityLevel** is set to the licensed version.

### **DbLoaders**

The parameter **DbLoaders** determines how many threads for loading the object memory are started when the server starts. Depending on the number of available processors, a higher number of threads can make the server start more quickly.

By default the number of available processors (but no more than 5) is taken here.

### **DbUrl**

The parameter **DbUrl** defines which database the server is to use. The valid syntax is dependent upon the database system.

There is obviously no default value for this parameter.

## **DbUser**

The parameter **DbUser** defines the user name that is to be used for working with the database.

There is no default value for this parameter either.

## **ExportVariables**

The displayed parameter **ExportVariables** is a composition of two configuration parameters: **ExportVariables** and **UserExportVariables**. These two variables are handed over to the jobserver as a list of key/value pairs when fetching a job. In turn, the jobserver has the option of setting the variables in the environment of the process that is to be started. Whether it does this, or under what name it sets the variables in the environment, is dependent upon the jobserver's configuration.

This functionality has been split into two parameters because the default entry for the parameter **ExportVariables** is the list of standard variables. To prevent the administrator from having to type the entire list just to export one more parameter, the additional tags can be specified in the parameter **UserExportVariables**.

## **GCWakeup**

"Obsolete" objects are deleted from the object cache at regular intervals. The parameter **GCWakeup** defines the interval between the cleanups. It is not practical to enter a small value for this parameter because the effort is only dependent upon the number of existing objects (essentially the Submitted Entities) and not upon the number of objects to be cleaned up.

The default value for this parameter is 240 (minutes).

## **History**

The parameter **History** defines the time period for keeping information about master instances of job or batch definitions. Older jobs or batches are only kept in the object cache if they are still active in some way (i.e. if they are neither final nor cancelled).

By default, information is kept for the last 10 days (14,400 minutes).

## **History Limit**

The parameter **History Limit** defines the maximum time period for keeping information about master instances of job or batch definitions that are no longer active. This limit applies if the parameter **MinHistoryCount** is set to a figure other than 0. If History is set to 10 days, History Limit to 30 days and MinHistoryCount to 10, only the last 4 to 5 versions are kept in the object cache for a weekly job or batch.

By default, information is kept for the last 10 days (14,400 minutes).



**Hostname**

The parameter **Hostname** is the name of the Scheduling Server. This is also the content of the standard job parameter **SDMSHOST**. The default value is "localhost".

**JdbcDriver**

The parameter **JdbcDriver** defines which class is to be used as the JDBC driver. The name of this class is dependent upon the database system that is being used. This class must be present in the CLASSPATH. The CLASSPATH environment variable is set in the file `$BICSUITECONFIG/java.conf`. There is obviously no default value for this parameter.

**MinHistoryCount**

The parameter **MinHistoryCount** defines the minimum number of master instances of a job or batch definition that are kept in the object cache. This allows a job or batch that is only seldom executed to be kept in the object cache for much longer than is defined in the parameter **History**. However, the maximum time that such a job or batch is kept in the object cache is still restricted by the parameter **History Limit**. By default, this parameter has the value 0 and is therefore not active.

**MaxHistoryCount**

The Parameter **MaxHistoryCount** defines the maximum number of master instances of a job or batch definition that are kept in the object cache. This allows a job or batch that is frequently executed to be kept in the object cache substantially shorter than is defined in the parameter **History**. An excessive memory usage caused by such jobs can be prevented this way.

By default, this parameter has the value 0 and is therefore not active.

**ParameterHandling**

The parameter **ParameterHandling** defines how the server is to respond when parameters are addressed that have not been declared. The possible values are:

Value	Meaning
STRICT	An error message is returned if an undeclared parameter is used.
WARN	A warning is written to the server's log file. The value of the variable is returned (if found).
LIBERAL	The value of the variable is returned (if found).

Table 24.1: ParameterHandling options

## Concept

From the perspective of software engineering, **ParameterHandling** should be defined as STRICT. From the perspective of convenience, WARN or LIBERAL are preferable.

The default setting is LIBERAL.

### Port

The parameter **Port** defines which TCP port the server uses for communicating with the clients. Any port can be chosen as long as it is not a port below 1024, since only highly privileged users are able to use these. Because the Scheduling Server has been designed so that no special privileges are required to operate it, this server shouldn't receive any special privileges either.

The default port is 2506.

### PriorityDelay

The parameter **PriorityDelay** defines the time after which the (effective) priority of a job is raised. Automatically raising the priority prevents jobs from waiting indefinitely for resources.

The default value is 30 (minutes).

### PriorityLowerBound

The parameter **PriorityLowerBound** determines the highest priority that can be attained through the normal ageing of jobs. If the parameter is greater than 0, a number of priority levels will remain free which can then be used by the administrator to guarantee specific jobs a place at the front of the queue.

PriorityLowerBound is set to 10 by default.

### RunMode

The parameter **RunMode** should be set to PRODUCTION in productive environments. In the development environment at independIT this parameter is occasionally set to TEST. Setting RunMode to TEST enables the command "RUN TEST ...". The impact of using this command is not defined as it is used for testing purposes. It is quite possible that using the command will cause the server to crash.

RunMode is set to PRODUCTION unless it has been explicitly set to TEST.

### ScheduleWakeUp

The parameter **ScheduleWakeUp** defines when the Resource Scheduling thread is to wake up. Since the Resource Scheduler uses up practically no time at all if there is nothing to do, the interval between two activations can be kept relatively short.

The default value is 30 (seconds).

### **SelectGroup**

The parameter **SelectGroup** defines which group is allowed to use the Select statement. If this parameter is not set or it contains the name of a non-existent group, only members of the ADMIN group and members of groups with the SELECT privilege may use the Select statement. This parameter exists for backward compatibility reasons.

### **ServicePort**

The parameter **ServicePort** is the port via which (only) an administrator can connect to the server. It can be used to still gain access (usually read access) to the server in the event of a malfunction.

The default value is 2505.

### **SessionTimeout**

The parameter **SessionTimeout** defines how long a session can be idle by default before it is terminated by the server. Normally, this parameter only plays a role with interactive sessions using `sdmsh`.

The default value is 30 (seconds).

### **SingleServer**

The parameter **SingleServer** is a Boolean value. If it is set to "true", the server assumes that no other server is able to access the repository. When it boots up after an "ungraceful shutdown", it ignores any set tickets and sets its own ticket. Should a second server be unexpectedly active, that server will realise that the repository now belongs to another server and will stop working. If both servers have the parameter set to "true", this will result in a kind of ping-pong effect.

The default setting is "false".

### **TTWakeup**

The parameter **TTWakeup** defines how often the tickets in the repository are to be at least renewed. The parameters should be coordinated with one other (i.e. if possible be the same) in an environment with multiple servers.

The default value is 30 (seconds).

### **TimerHorizon**

The parameter **TimerHorizon** defines how far into the future the Timer thread is to look to find the next execution time for a job.

The default value is 5 (years).

### TimerRecalc

The parameter **TimerRecalc** determines after how long another attempt is to be made to find the next execution time for a job after the previous search has been terminated because the TimerHorizon was reached. The default value is 5 (days).

### TimerSuspendLimit

The parameter **TimerSuspendLimit** defines the time after which a job is submitted as being suspended if the server was not active at the scheduled Submit time. This limit can be set individually for each Scheduled Event. If this is not done, however, the value for the TimerSuspendLimit applies. The default time is 15 (minutes).

### TimerWakeup

The parameter **TimerWakeup** determines at what intervals the Timer thread is activated. A higher value will result in inaccuracies concerning the actual Submit time as well as potentially long runtimes for the Time Schedule process. A low value is significantly less harmful. The default value of 30 (seconds) has proved to produce good results.

### TraceLevel

The parameter **TraceLevel** determines how much and what kind of logging information the server writes to the log file. The table below shows what type of messages are logged on which Trace Levels:

Type	Trace Level	Remark
FATAL	All	FATAL messages are logged on the server in the event of a serious malfunction and should be reported immediately to the independIT support team.
ERROR	All	ERROR messages are logged when errors occur. These types of errors are significantly less critical than FATAL class errors. It is advisable to investigate these types of errors and to eliminate their causes where possible.

*Continues on next page*

---

*Continued from previous page*

---

Type	Trace Level	Remark
INFO	All	INFO messages contain important or interesting information. Examples of such messages are those that are written during the server startup phase.
WARNING	> 0	WARNING messages are interesting messages that frequently deal with operating inaccuracies. An example of this are the warnings that occur when the parameter <code>ParameterHandling</code> is set to <code>WARN</code> or <code>STRICT</code> .
MESSAGE	> 1	MESSAGE messages are relatively unimportant but can nonetheless be very interesting. For example, all executed commands and their execution times are logged as MESSAGES.
DEBUG	3	DEBUG messages can be helpful for troubleshooting purposes. However, Trace Level 3 will produce a huge flood of messages. In normal operation, these messages are less meaningful and are, more than anything else, distracting.

---

Table 24.2: Overview of the Trace Levels

### TriggerHardLimit

The parameter **TriggerHardLimit** defines the maximum number of times that one and the same trigger can be initiated.

The default setting is 100.

### TriggerSoftLimit

If triggers are created without a `FireLimit` being specified, the parameter **TriggerSoftLimit** sets the maximum number of times that an instance of this trigger can be initiated.

The default setting is 50.

### TxRetryCount

In some cases the performance of a transaction may fail even though the transaction is semantically and syntactically correct. In this case, the server will try to perform the transaction again. How often this takes place is defined by the parameter **TxRetryCount**.

By default, a maximum of three attempts are made.

### **UserThreads**

An important parameter in the server configuration is the parameter **UserThreads**. This parameter defines the maximum number of sessions that can be connected simultaneously. Users who are using the web front end do not have to be included in the full number because the Zope server sets up a new connection to the Scheduling Server for each communication step and then breaks it again, respectively reuses cached connections.

The jobserver usually remain connected with the Scheduling Server, however, and they each occupy one thread.

The default value for this parameter is 10.

### **WorkerThreads**

The parameter **WorkerThreads** defines how many threads are to be started to respond to read queries. In contrast to write transactions, many read transactions are extremely complex. In particular this includes monitoring-related read transactions.

By default, two such threads are started. In larger environments it makes sense to raise this number.

### 24.2.3 System tab

Config	System	Worker	Sessions	SME/Q
VERSION	2.5.1			
MAX_LEVEL	OPEN			
NUM_CPU	4			
MEM_USED	155451392 Bytes			
MEM_FREE	149877608 Bytes			
MEM_MAX	310640640 Bytes			
STARTTIME	Thu Sep 12 10:28:00 CEST 2013			
UPTIME	7 sec(s)			
HITRATE	0 %			
JdbcDriver	org.postgresql.Driver			
JAVA.VERSION	1.6.0_31			
JAVA.VENDOR	Sun Microsystems Inc.			
JAVA.VENDOR.URL	http://java.sun.com/			
JAVA.HOME	/home/dieter/Software/Java/jdk1.6.0_31/jre			
JAVA.VM.SPECIFICATION.VERSION	1.0			
JAVA.VM.SPECIFICATION.VENDOR	Sun Microsystems Inc.			
JAVA.VM.SPECIFICATION.NAME	Java Virtual Machine Specification			
JAVA.VM.VERSION	20.6-b01			
JAVA.VM.VENDOR	Sun Microsystems Inc.			
JAVA.VM.NAME	Java HotSpot(TM) Server VM			
JAVA.SPECIFICATION.VERSION	1.6			
JAVA.SPECIFICATION.VENDOR	Sun Microsystems Inc.			
JAVA.SPECIFICATION.NAME	Java Platform API Specification			
JAVA.CLASS.VERSION	50.0			
JAVA.CLASS.PATH	/home/dieter/BICsuite/Versions/2.5.1/SDMS/sandbox/OPENSOURCE-linux/lib/BICsuite.jar: /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/ojdbc14.jar:/usr/share/java/mysql-connector-java.jar: /home/dieter/BICsuite/sqljdbc_4.0/enu/sqljdbc4.jar:/usr/share/java/postgresql-jdbc4.jar			
JAVA.LIBRARY.PATH	/home/dieter/Software/Java/jdk1.6.0_31/jre/lib/i386/server:/home/dieter/Software/Java/jdk1.6.0_31/jre/lib/i386: /home/dieter/Software/Java/jdk1.6.0_31/jre/.lib/i386:/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/lib: /usr/java/packages/lib/i386/lib:/usr/lib			
JAVA.IO.TMPDIR	/tmp			
JAVA.COMPILER	NONE			
JAVA.EXT.DIRS	/home/dieter/Software/Java/jdk1.6.0_31/jre/lib/ext:/usr/java/packages/lib/ext			
OS.NAME	Linux			
OS.ARCH	i386			
OS.VERSION	3.2.0-52-generic			

Figure 24.3: System runtime environment

This tab shows some runtime information together with details of the Java Virtual Machine. The information may differ to the information shown here depending upon the hardware and operating system or Java Virtual Machine that you are using. You should therefore refer to the relevant documentation for a correct interpretation of this information.

#### VERSION

The *VERSION* field shows the software version of the implemented Scheduling Server.

#### MAX\_LEVEL

The *MAX\_LEVEL* field shows the licensed version of the Scheduling Server.

### **NUM\_CPU**

The *NUM\_CPU* field shows the number of processors reported by the Java Virtual Machine.

### **MEM\_USED**

The *MEM\_USED* field shows how much memory is currently being used by the Java Virtual Machine.

### **MEM\_FREE**

The *MEM\_FREE* field shows how much of the occupied memory is still available for the server.

### **MEM\_MAX**

The *MEM\_MAX* field shows the maximum amount of memory that the Java Virtual Machine is allowed to occupy. If *MEM\_MAX* and *MEM\_FREE* have the same value, this means that no more memory is available apart from the free disk space reported under *MEM\_FREE*. In itself this is not a problem. However, if the capacity *MEM\_FREE* is almost used up, the Virtual Machine will be increasingly frequently occupied with garbage collection, which can have a serious negative impact on the system performance. In this case, restarting the server (with a possible modification of the memory configuration in `$BICSUITECONFIG/java.conf`) is the lesser evil.

### **STARTTIME**

The *STARTTIME* field shows the time when the server was started.

### **UPTIME**

The *UPTIME* field is for convenience only and shows how long the server is already running.

### **HITRATE**

The *HITRATE* field shows the effectiveness of a caching algorithm in the Resource Scheduling thread. Since this cache only demonstrates how effective it is in case of a complete reschedule, even relatively low values will give an indication of the effectiveness.



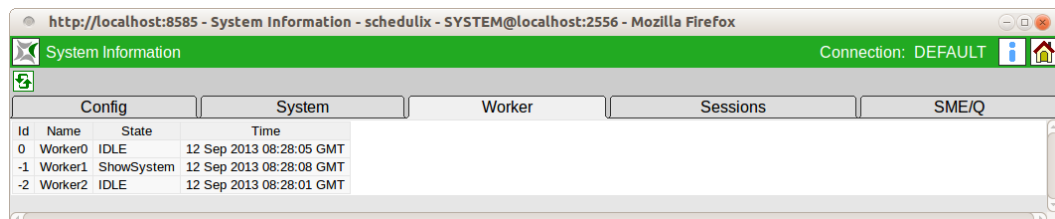
## JdbcDriver

The entry under "JdbcDriver" was only included in this tab for the sake of completeness because the other Java-related configuration parameters are given here as well. This is the same value as the one in the "Config" tab.

## Other entries

All other entries are dependent upon the hardware, operating system and Java Virtual Machine that you are using. Although in principle the values are relatively easy to interpret, you should refer to the Java documentation for a more detailed description of their meanings.

### 24.2.4 Worker tab



Id	Name	State	Time
0	Worker0	IDLE	12 Sep 2013 08:28:05 GMT
-1	Worker1	ShowSystem	12 Sep 2013 08:28:08 GMT
-2	Worker2	IDLE	12 Sep 2013 08:28:01 GMT

Figure 24.4: Worker thread information

The activity of the Worker threads is shown in a table in the "Worker" tab. The worker with the ID 0 is always the worker responsible for processing the write transactions. All other workers are responsible for processing the read transactions. If all the Worker threads are regularly active, this indicates that there are too few of them and their number should be increased.

#### Id

The worker's internal number is shown here.

#### Name

The worker's name is shown here.

#### State

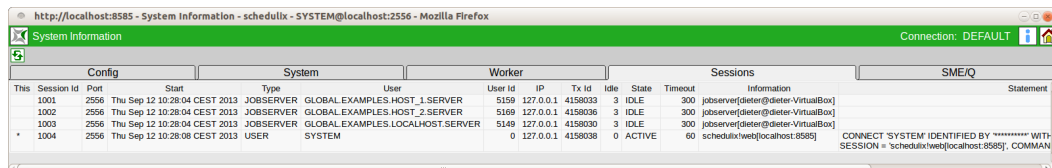
The *State* column shows what the respective worker is doing at the moment. The entry here is either IDLE, if the worker has nothing to do, or the class name of the object which he is currently processing. Normally, the type of statement can be easily derived from the name of the class.

## Concept

### Time

This column shows the start time of the most recently executed statement or the statement that is being executed.

### 24.2.5 Sessions tab



Session Id	Port	Start	Type	User	User Id	IP	Tx Id	Idle	State	Timeout	Information	SME/Q	Statement
1001	2556	Thu Sep 12 10:28:04 CEST 2013	JOBSERVER	GLOBALEXAMPLES.HOST_1.SERVER	5159	127.0.0.1	4158033	3	IDLE	300	jobsver(dieter@dieter-VirtualBox)		
1002	2556	Thu Sep 12 10:28:04 CEST 2013	JOBSERVER	GLOBALEXAMPLES.HOST_2.SERVER	5169	127.0.0.1	4158036	3	IDLE	300	jobsver(dieter@dieter-VirtualBox)		
1003	2556	Thu Sep 12 10:28:04 CEST 2013	JOBSERVER	GLOBALEXAMPLES.LOCALHOST.SERVER	5149	127.0.0.1	4158030	3	IDLE	300	jobsver(dieter@dieter-VirtualBox)		
* 1004	2556	Thu Sep 12 10:28:08 CEST 2013	USER	SYSTEM	0	127.0.0.1	4158038	0	ACTIVE	60	CONNECT SYSTEM IDENTIFIED BY ***** WITH SESSION = 'schedulxweb@localhost:8585', COMMAND		

Figure 24.5: Session information

A table of all the currently connected sessions and their activities is shown in the "Sessions" tab.

### This

In this column, an asterisk (\*) indicates which of the displayed sessions is the user's own session.

### Session Id

This column shows the number of the session. This is required if you want to terminate the session using the "KILL SESSION" command.

### Port

This column shows the port via which the session is connected with the server.

### Start

This column shows the time when the session was started.

### Type

This column shows what type of client the connection is. The possible values are USER, JOBSERVER or JOB.

### User

The user's name is shown here. In the case of a user, this is the username. If it is a jobserver, this is the server's fully qualified name. The JobId is shown for a job.

## Concept

### UserId

This column shows the object Id of the user, jobserver or job.

### IP

This column shows the IP address of the computer from which the connection has been set up.

### TxId

This column shows the number of the most recently performed transaction. If a statement is just being executed, it is the number of active transaction.

### Idle

This column shows the number of seconds that have elapsed since the last activity. Ideally, for jobserver this value should not be higher than their NOP\_DELAY.

### State

This column shows the state of the session.

Wert	Meaning
IDLE	The session is waiting for user input.
QUEUED	The session has a statement in the queue that is ready to be executed
ACTIVE	A statement in the session is currently being executed.
COMMITTING	A statement in the session is in the final phase of being executed.
CONNECTED	A session has been initiated but the user has not yet provided authentication.

Table 24.3: Session Status

### Timeout

This column shows the number of seconds of idle time that can elapse before the server connection is terminated. A value of 0 means that the server connection will not be terminated.

### Statement

This column shows the statement that is currently being executed by the connection provided, of course, the connection is not IDLE. It goes without saying that

## Concept

passwords in Connect statements are made unrecognisable. On the other hand, the statement is only visible to users with administrator privileges.

### **24.2.6 SME/Q tab**

The "SME/Q" tab shows a table of the Submitted Entities created in each quarter. This tab is only visible if the user belongs to the "ADMIN" group.

#### **Year**

This column shows the year.

#### **Quarter**

This column shows the quarter.

#### **Total**

This column shows the total number of Submitted Entities which were created in this quarter.

#### **Expected Total**

This column contains for the current and not yet completed quarter (1st row in the table) the extrapolated expected number of Submitted Entities by the end of the quarter. This number is not binding and is intended solely for orientation purposes. For all previous quarters, the expected total is the same as the total.

#### **Avg. SME / Day**

This column the average number of Submitted Entities created each day.

# 25 Shell

## 25.1 Image

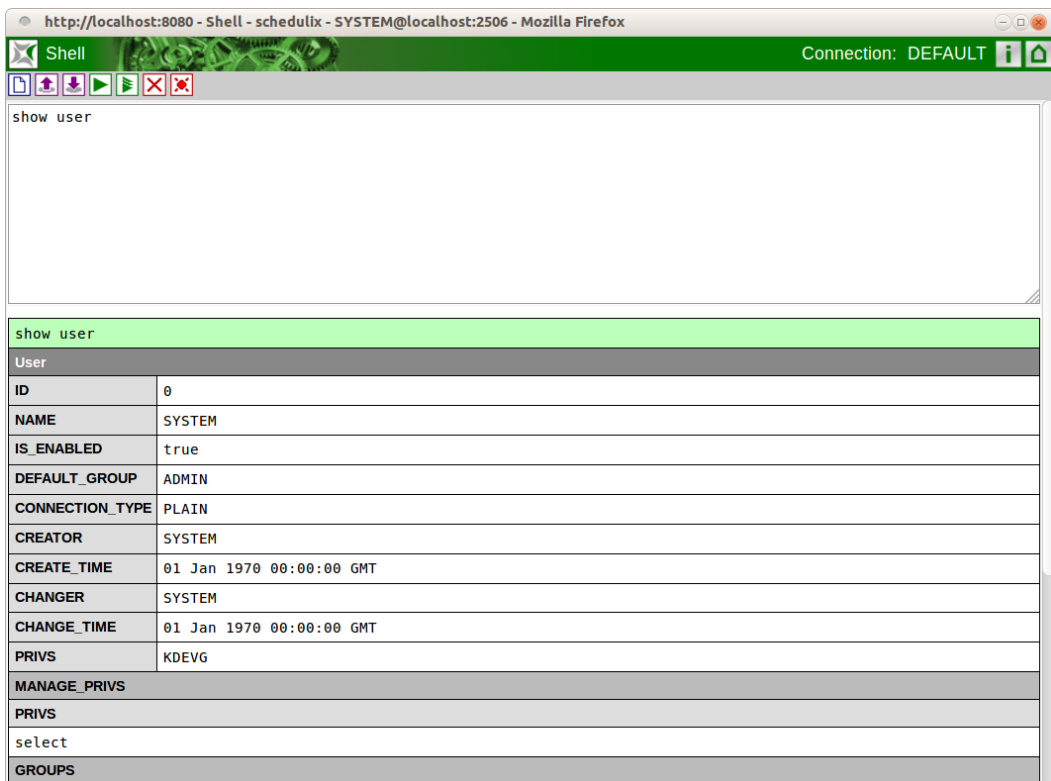


Figure 25.1: Shell

## 25.2 Concept

### 25.2.1 Brief description

API commands can be executed from the web front end with the "Shell" menu item in the main menu.

### 25.2.2 Detailed description

The shell window comprises a button bar, an input field for API commands, and an area below the command input field where the executed command is outputted.

### 25.2.3 Buttons

The following list describes the buttons in the window and their functions:



#### **Delete Entry**

The command input field is cleared.



#### **Load**

Read in commands from a file.



#### **Save**

Write the commands to a file.



#### **Execute**

Executes the command over which the cursor is hovering. If a text has been selected, it is executed.



#### **Execute as script**

Executes all commands in the input field.



#### **Delete all outputs**

All results of commands outputted below the input field are deleted.



#### **Auto clear off**

Deactivates the automatic deletion of outputs before commands are executed.



#### **Auto clear on**

Activates the automatic deletion of outputs before commands are executed.

# Index

- Access control, [159](#)
- Access mode, [133](#)
- Active Schedule, [183](#)
- Adding a line, [18](#)
- After Final, [146](#)
- Alias, [114](#)
- Allocated, [80](#)
- Amount, [59](#), [68](#), [81](#), [102](#), [130](#), [134](#)
- Ausführungsort, [63](#)
- Auto-Refresh, [196](#)
- Automatic Restart, [112](#)
- Autoresume, [213](#)
- Autostart, [207](#)
- Available, [81](#)
  
- Batch, [93](#), [95](#), [107](#), [187](#), [239](#)
- Batch convention, [95](#)
- Batch Default, [39](#)
- Batch in Folder Convention, [152](#)
- Before Final, [146](#)
- Blocked, [80](#)
- Bookmark, [97](#), [191](#), [195](#), [206](#), [235](#)
- Bourne-Shell, [108](#)
- Broken Active, [201](#)
- Broken Finished, [201](#)
- Broken Flag, [39](#)
- Broken State, [39](#)
- Browser, [1](#), [2](#)
  
- Calendar, [109](#), [172](#), [178](#), [185](#), [204](#), [240](#),  
[245](#)
- Calendar funktion, [239](#)
- Calendar horizon, [172](#), [204](#), [245](#)
- Calendar week, [185](#)
- Cancel, [214](#)
  
- Cancel Button, [10](#)
- Cancelled, [200](#)
- Category, [54](#)
- Chain, [151](#)
- Child Process, [107](#)
- Childreference, [137](#)
- Children, [113](#)
- Clipboard, [15](#), [21](#)
- Clone, [99](#)
- Clone Button, [11](#)
- Collapse All, [7](#)
- Comment, [29](#), [210](#), [215](#)
- Compatibility matrix, [82](#)
- Condition, [65](#), [123](#), [129](#), [205](#)
- Configuration, [244](#)
- Configuration parameter, [87](#)
- Connection, [257](#)
- Constant, [57](#), [79](#)
- Convention, [95](#)
- Copy Button, [15](#)
- Current Amount, [78](#)
- Cut Button, [15](#)
  
- Data transfer, [94](#)
- Database system, [244](#)
- Day of Month Interval, [176](#)
- Day of Month interval, [185](#)
- Day of Week Interval, [175](#)
- Day of Week interval, [184](#)
- Day view, [240](#)
- Default Directory, [110](#)
- Default environment, [158](#)
- Defined Amount, [78](#)
- Defined Resource, [133](#)
- Definition error, [200](#)

Deleting a line, 19  
 Dependency, 38, 39, 113, 121, 151  
 Dependency Default, 39  
 Dependency Mode, 121  
 Dependency Wait, 200  
 Dependent Job, 151  
 Deselect All Button, 15  
 Desktop, 6  
 Detail Bookmark, 192  
 Disable, 214  
 Disabled, 39  
 Dispatcher, 182  
 Downtime, 168  
 Drop Button, 11  
 Drop line, 19  
 Dynamic children, 114  
 Dynamic priority, 219  
  
 Edit Button, 15  
 Effective horizon, 172  
 Effective priority, 83  
 Enabled, 75  
 Environment, 61, 63, 96, 100, 109, 127, 128, 158, 220  
 Environment variable, 84, 89  
 Environment variables, 108  
 Error, 39, 197, 200  
 Error Logfile, 111  
 Error logfile, 220  
 Error state, 40  
 Exclusive, 81  
 Exit Code, 33, 218  
 Exit Code Ranges, 34  
 Exit Find, 9  
 Exit State, 31, 33, 37, 41, 49, 94, 99, 199, 204, 216  
 Exit State Definition, 31  
 Exit State Mapping, 33, 38, 109, 219  
 Exit State Profile, 37, 39, 41, 99, 109, 217  
 Exit State Translation, 39, 42, 119  
 Expand All, 8  
 Expected finaltime, 109  
 Expected runtime, 109, 220  
 Expiration, 60, 133  
 Expire, 60, 133  
 Expression, 136, 137  
  
 Filter criteria, 195, 203  
 Final, 201  
 Final State, 38  
 Finish Child, 147  
 Finished, 201  
 Fire Limit, 148  
 Fire limit, 251  
 Folder, 7, 53, 63, 96  
 Folder convention, 99  
 Folder Environment, 63, 64, 100  
 Footprint, 61, 67, 109, 127, 128, 220  
 Free Amount, 78  
 Free amount, 59  
  
 Grant, 165  
 Grant Button, 15  
 Group, 31, 34, 38, 42, 46, 50, 53, 54, 64, 65, 68, 74, 99, 106, 160, 163  
  
 Header, 3  
 Help Icon, 6  
 Hide Dependencies, 151  
 Hide Folderpath Button, 11  
 Hide Hierarchypath Button, 12  
 Hide Leaves, 8  
 Hide Locked, 8  
 Hierarchy, 7  
 Historie, 247  
 History, 204  
 Home Icon, 6  
 Horizon, 172  
  
 Idle time, 257  
 Ignored, 81  
 Ignored Dependency, 119  
 Immediate Local, 145  
 Immediate Merge, 145  
 Inheritance, 86  
 Interval, 173, 184



Intervals, [182](#)  
 IP address, [257](#)  
 ISO calendar week, [185](#)  
 ISO Week, [185](#)  
 ISO week, [177](#)  
 ISO Week of Month Interval, [177](#)  
 ISO Week of Month interval, [185](#)  
 ISO week of year, [177](#)  
 ISO Week of Year Interval, [177](#)

Java virtual machine, [253](#)  
 JDBC Driver, [247](#)  
 JdbcDriver, [254](#)  
 Job Definition, [38](#), [60](#), [63](#), [64](#), [67](#), [93](#)  
 Job Environment, [64](#)  
 Job State, [112](#), [197](#), [204](#), [209](#)  
 Job state, [199](#), [216](#)  
 Jobserver, [63](#), [71](#), [200](#), [201](#), [219](#)  
 Jobserver process, [75](#)

Keep, [60](#), [68](#), [81](#), [131](#)  
 Kill, [213](#)  
 Kill Program, [110](#), [201](#)  
 Kill program, [220](#)  
 Killed, [110](#), [201](#)

Load balancing, [94](#), [132](#)  
 Local Constant, [57](#), [79](#)  
 Lockmode, [60](#), [81](#), [82](#), [133](#)  
 Logfile, [110](#), [220](#)  
 Logfile Pattern, [90](#)  
 Logfile Write, [110](#)  
 Logging, [250](#)  
 Login window, [2](#)

Main Desktop, [3](#), [6](#)  
 Main menu, [2](#)  
 Main Schedule, [182](#)  
 Main schedule, [168](#)  
 Master Bookmark, [192](#)  
 Master Job, [93](#), [107](#), [119](#), [131](#), [132](#), [148](#),  
     [187](#), [192](#), [203](#), [217](#)  
 Master job, [195](#)  
 Master Reservation, [81](#)

Memory, [254](#)  
 Merge Global, [119](#)  
 Merge Local, [118](#)  
 Merge Mode, [117](#), [118](#), [217](#)  
 Milestone, [93](#)  
 Mode Search, [192](#)  
 Month of Year Interval, [178](#)  
 Month of Year interval, [185](#)  
 Month view, [239](#)

Name pattern, [204](#)  
 Named Resource, [53](#), [54](#), [65](#), [68](#)  
 New Button, [11](#)  
 Nice Value, [107](#), [115](#), [216](#)  
 NoLock, [82](#)  
 Number of sessions, [252](#)

Operating system, [253](#)  
 Option box, [3](#)  
 Owner, [53](#), [74](#), [99](#), [106](#), [145](#), [216](#)

Parameter, [57](#), [79](#), [84](#), [94](#), [96](#), [101](#), [108](#),  
     [134](#), [135](#), [187](#), [205](#), [244](#)  
 Parameter Type, [135](#)  
 Parent, [39](#), [41](#)  
 Password prompt, [2](#)  
 Paste Button, [15](#)  
 PENDING State, [38](#)  
 PID, [75](#)  
 Pinning, [96](#)  
 Predecessor, [39](#)  
 Priority, [40](#), [82](#), [107](#), [108](#), [115](#), [219](#), [248](#)  
 Privilege, [31](#), [34](#), [38](#), [42](#), [46](#), [50](#), [64](#), [65](#),  
     [68](#), [160](#), [163](#)  
 Privileges, [15](#), [53](#)  
 Process Id, [110](#)  
 Program, [63](#)

Query Mask, [203](#)  
 Query mask, [209](#), [236](#)  
 Quoting, [108](#)

Range of Day Interval, [175](#)  
 Range of Day interval, [184](#)  
 Reference, [137](#)

Refresh, [7](#)  
 Regular Expression, [90](#)  
 Regular expression, [204](#)  
 Rename, [99](#)  
 Repeat Interval, [174](#), [184](#)  
 Requestable Amount, [78](#), [102](#), [134](#)  
 Requestable amount, [59](#)  
 Requested, [80](#)  
 Required Resource, [130](#)  
 Requirement, [130](#)  
 Rerun, [198](#), [213](#)  
 Rerun children, [199](#), [213](#)  
 Rerun Program, [110](#), [219](#)  
 Reservation, [81](#)  
 Reserved, [80](#)  
 Resource, [59](#), [63](#), [67](#), [71](#), [94](#), [127](#)  
 Resource Definition, [54](#)  
 Resource Link, [91](#)  
 Resource requirement, [63](#)  
 Resource scheduler, [248](#)  
 Resource standard parameter, [58](#)  
 Resource State, [43](#), [45](#), [49](#), [56](#), [60](#), [78](#)  
 Resource state, [101](#)  
 Resource State Definition, [43](#)  
 Resource State Mapping, [49](#), [60](#), [82](#),  
     [133](#)  
 Resource State Profile, [45](#), [56](#), [77](#), [78](#),  
     [101](#), [132](#), [133](#)  
 Resource Status, [133](#)  
 Resource Trigger, [57](#)  
 Resource Wait, [200](#)  
 Resource-Status, [82](#)  
 Resourcereference, [137](#)  
 Ressource, [76](#), [101](#)  
 Restart, [112](#)  
 Restartable, [39](#)  
 Resume, [117](#)  
 Resume Time, [149](#)  
 Resume time, [107](#), [190](#), [212](#)  
 Run Program, [110](#), [200](#), [219](#)  
 RunMode, [248](#)  
 Runnable, [200](#)  
 Running, [201](#)  
 Running Jobs, [235](#)  
 Running master jobs, [195](#)  
 Runtime, [199](#)  
 Runtime environment, [63](#), [100](#), [108](#)  
 Save Button, [11](#)  
 Save View, [8](#)  
 Schedules, [168](#)  
 Scheduling Entity, [58](#), [93](#)  
 Scope, [53](#), [59](#), [71](#)  
 Search, [8](#)  
 Search Bookmark, [192](#)  
 Search pattern, [98](#)  
 Select All Button, [13](#)  
 Select Button, [15](#)  
 Select statement, [249](#)  
 Server configuration, [244](#)  
 Server connection, [3](#)  
 Server parameter, [244](#)  
 Server User, [159](#)  
 Server user, [155](#)  
 Session, [256](#)  
 Session timeout, [249](#)  
 Sessions, [252](#)  
 Set State, [213](#)  
 Shared, [81](#)  
 Shared Compatible, [82](#)  
 Shared Exclusive, [81](#)  
 Shell, [259](#)  
 Show Folderpath Button, [11](#)  
 Show Hierarchypath Button, [12](#)  
 Show Leaves, [8](#)  
 Show Locked, [8](#)  
 SQL LIKE Syntax, [98](#)  
 Standard Parameter, [109](#), [139](#)  
 Standard parameter, [109](#)  
 Start Find, [9](#)  
 Start Mode, [207](#)  
 Start time, [169](#), [240](#)  
 Started, [201](#)  
 Starting, [201](#)  
 State transition, [201](#)  
 Static Ressource, [56](#)

Statistics, [221](#)  
 Sticky, [60](#), [81](#), [131](#)  
 Sticky name, [132](#)  
 Sticky parent, [132](#)  
 Sub Schedule, [170](#)  
 Submit, [187](#)  
 Submit Suspended, [116](#)  
 Submitted, [200](#)  
 Submitted Entity, [37](#), [53](#)  
 Suspend, [187](#)  
 Suspend Local, [212](#)  
 Suspend Restricted, [212](#)  
 Suspended, [75](#), [198](#)  
 Synchronize Wait, [200](#)  
 Synchronizing Resource, [43](#), [56](#), [94](#), [131](#)  
 System Bookmark, [97](#)  
 System Bookmarks, [192](#)  
 System Resource, [56](#), [67](#)  
 System variable, [110](#)

TCP port, [248](#)  
 Threads, [245](#), [252](#)  
 Tickets, [249](#)  
 Time of Day Interval, [174](#)  
 Time of Day interval, [184](#)  
 Time Scheduling, [15](#), [239](#)  
 Time scheduling, [168](#)  
 Timeout, [127](#), [257](#)  
 Timeout State, [128](#)  
 Timer thread, [249](#)  
 Timestamp, [221](#)  
 To Kill, [201](#)  
 Toggle Selection Button, [13](#)  
 Trace Level, [250](#)  
 Trigger, [39](#), [57](#), [94](#), [141](#), [251](#)  
 Trigger parameter, [141](#)  
 Trigger type, [145](#)

Unreachable, [39](#), [197](#), [200](#)  
 Unreachable State, [39](#)  
 Unresolved dependency, [216](#)  
 Unresolved Handling, [123](#)  
 Until Final, [148](#)  
 Until Finished, [148](#)

Up Button, [10](#)  
 Usage, [55](#)  
 User, [159](#), [163](#)  
 User Bookmark, [192](#)  
 User group, [63](#)  
 User management, [155](#)  
 User name, [2](#)

Variable, [201](#)  
 View Privilege, [163](#)  
 Virtual machine, [253](#)

Warning, [40](#)  
 Week of Month Interval, [176](#)  
 Week of Month interval, [185](#)  
 Week view, [239](#)  
 Weekday, [177](#), [185](#)  
 Windows Clipboard, [21](#)  
 Worker Threads, [255](#)  
 Working Directory, [110](#)

