independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix Server

Command Reference

Dieter Stubler Ronald Jeninga

June 6, 2017

Copyright © 2017 independIT GmbH

Legal notice

This work is copyright protected

Copyright © 2017 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

Table of Contents iii

List of Tables ix

I. General 1

1. Introduction 3
Introduction . . . 3

2. Utilities 11
Starting and stopping the server . . . 11
server-start . . . 11
server-stop . . . 12
sdmsh . . . 12
sdms-auto_restart . . . 20
sdms-get_variable . . . 22
sdms-rerun . . . 24
sdms-set_state . . . 25
sdms-set_variable . . . 26
sdms-set_warning . . . 28
sdms-submit . . . 29

II. User Commands 31

3. alter commands 33
alter comment . . . 34
alter environment . . . 36
alter event . . . 37
alter exit state mapping . . . 38
alter exit state profile . . . 39
alter folder . . . 40
alter footprint . . . 42
alter group . . . 43
alter interval . . . 44
alter job . . . 45

alter job definition 50
alter named resource 55
alter resource 57
alter resource state mapping 58
alter resource state profile 59
alter schedule 60
alter scheduled event 62
alter scope 63
alter server 65
alter session 66
alter trigger 68
alter user 70

4. connect commands **71**
connect 72

5. copy commands **75**
copy folder 76
copy named resource 77
copy scope 78

6. create commands **79**
create comment 80
create environment 82
create event 83
create exit state definition 84
create exit state mapping 85
create exit state profile 86
create folder 89
create footprint 91
create group 93
create interval 94
create job definition 98
create named resource 116
create resource 119
create resource state definition 122
create resource state mapping 123
create resource state profile 124
create schedule 125
create scheduled event 127
create scope 129
create trigger 132
create user 140

7. deregister commands **143**
deregister 144

8. disconnect commands **145**
disconnect 146

9. drop commands **147**
drop comment 148
drop environment 150
drop event 151
drop exit state definition 152
drop exit state mapping 153
drop exit state profile 154
drop folder 155
drop footprint 156
drop group 157
drop interval 158
drop job definition 159
drop named resource 160
drop resource 161
drop resource state definition 162
drop resource state mapping 163
drop resource state profile 164
drop schedule 165
drop scheduled event 166
drop scope 167
drop trigger 168
drop user 169

10. finish commands **171**
finish job 172

11. get commands **173**
get parameter 174
get submittag 175

12. kill commands **177**
kill session 178

13. link commands **179**
link resource 180

14. list commands **181**
list calendar 182

list dependency definition . . . 183
list dependency hierarchy . . . 185
list environment . . . 189
list event . . . 190
list exit state definition . . . 191
list exit state mapping . . . 192
list exit state profile . . . 193
list folder . . . 194
list footprint . . . 197
list group . . . 198
list interval . . . 199
list job . . . 201
list job definition hierarchy . . . 208
list named resource . . . 211
list resource state definition . . . 213
list resource state mapping . . . 214
list resource state profile . . . 215
list schedule . . . 216
list scheduled event . . . 218
list scope . . . 220
list session . . . 222
list trigger . . . 224
list user . . . 227

15. move commands **229**
move folder . . . 230
move job definition . . . 231
move named resource . . . 232
move schedule . . . 233
move scope . . . 234

16. multicommand commands **235**
multicommand . . . 236

17. register commands **237**
register . . . 238

18. rename commands **239**
rename environment . . . 240
rename event . . . 241
rename exit state definition . . . 242
rename exit state mapping . . . 243
rename exit state profile . . . 244
rename folder . . . 245

rename footprint . . . 246
rename group . . . 247
rename interval . . . 248
rename job definition . . . 249
rename named resource . . . 250
rename resource state definition . . . 251
rename resource state mapping . . . 252
rename resource state profile . . . 253
rename schedule . . . 254
rename scope . . . 255
rename trigger . . . 256
rename user . . . 257

19. resume commands **259**
resume . . . 260

20. select commands **261**
select . . . 262

21. set commands **263**
set parameter . . . 264

22. show commands **265**
show comment . . . 266
show environment . . . 268
show event . . . 271
show exit state definition . . . 273
show exit state mapping . . . 274
show exit state profile . . . 276
show folder . . . 278
show footprint . . . 282
show group . . . 285
show interval . . . 287
show job . . . 289
show job definition . . . 305
show named resource . . . 314
show resource . . . 318
show resource state definition . . . 323
show resource state mapping . . . 324
show resource state profile . . . 326
show schedule . . . 328
show scheduled event . . . 330
show scope . . . 332
show session . . . 338

show system . . . 340
show trigger . . . 342
show user . . . 346

23. shutdown commands **349**
shutdown . . . 350

24. stop commands **351**
stop server . . . 352

25. submit commands **353**
submit . . . 354

26. suspend commands **357**
suspend . . . 358

III. Jobserver Commands **359**

27. Jobserver Commands **361**
alter job . . . 362
alter jobserver . . . 367
connect . . . 368
deregister . . . 371
disconnect . . . 372
get next job . . . 373
multicommand . . . 375
reassure . . . 376
register . . . 377

IV. Job Commands **379**

28. Job Commands **381**
alter job . . . 382
connect . . . 387
disconnect . . . 390
get parameter . . . 391
get submittag . . . 392
multicommand . . . 393
set parameter . . . 394
set state . . . 395
submit . . . 396

List of Tables

1.1. Valid date formats: . . . 6
1.2. Keywords that can be used with quotes as identifiers . . . 8
1.3. Keywords and synonyms . . . 9
1.4. Reserved words . . . 9

6.1. job definition parameters . . . 107
6.2. Named Resource parameter types . . . 117
6.3. Named Resource usage . . . 118
6.4. job definition parameters . . . 121
6.5. List of trigger types . . . 139

11.1. get parameter output . . . 174
11.2. get submittag output . . . 175

14.1. list dependency definition output . . . 184
14.2. list dependency hierarchy output . . . 188
14.3. list environment output . . . 189
14.4. list event output . . . 190
14.5. list exit state definition output . . . 191
14.6. list exit state mapping output . . . 192
14.7. list exit state profile output . . . 193
14.8. list folder output . . . 196
14.9. list footprint output . . . 197
14.10. list group output . . . 198
14.11. list interval output . . . 200
14.12. list job output . . . 207
14.13. list job definition hierarchy output . . . 210
14.14. list named resource output . . . 212
14.15. list resource state definition output . . . 213
14.16. list resource state mapping output . . . 214
14.17. list resource state profile output . . . 215
14.18. list schedule output . . . 217
14.19. list scheduled event output . . . 219
14.20. list scope output . . . 221
14.21. list session output . . . 223
14.22. list trigger output . . . 226

14.23. list user output 227

22.1. show comment output 267
22.2. show environment output 269
22.3. show environment RESOURCES subtable structure 269
22.4. show environment JOB_DEFINITIONS subtable structure 270
22.5. show event output 272
22.6. show event PARAMETERS subtable structure 272
22.7. show exit state definition output 273
22.8. show exit state mapping output 274
22.9. show exit state mapping RANGES subtable structure 275
22.10. show exit state profile output 277
22.11. show exit state profile STATES subtable structure 277
22.12. show folder output 279
22.13. show folder PARAMETERS subtable structure 280
22.14. show folder DEFINED_RESOURCES subtable structure 281
22.15. show footprint output 283
22.16. show footprint RESOURCES subtable structure 283
22.17. show footprint JOB_DEFINITIONS subtable structure 284
22.18. show group output 285
22.19. show group MANAGE_PRIVS subtable structure 286
22.20. show group USERS subtable structure 286
22.21. show interval output 288
22.22. show interval SELECTION subtable structure 288
22.23. show interval FILTER subtable structure 288
22.24. show job output 295
22.25. show job CHILDREN subtable structure 296
22.26. show job PARENTS subtable structure 296
22.27. show job PARAMETER subtable structure 297
22.28. show job REQUIRED_JOBS subtable structure 299
22.29. show job DEPENDENT_JOBS subtable structure 301
22.30. show job REQUIRED_RESOURCES subtable structure 303
22.31. show job AUDIT_TRAIL subtable structure 303
22.32. show job DEFINED_RESOURCES subtable structure 304
22.33. show job definition output 307
22.34. show job definition CHILDREN subtable structure 308
22.35. show job definition PARENTS subtable structure 309
22.36. show job definition PARAMETER subtable structure 310
22.37. show job definition REQUIRED_JOBS subtable structure 311
22.38. show job definition DEPENDENT_JOBS subtable structure 312
22.39. show job definition REQUIRED_RESOURCES subtable structure 313
22.40. show named resource output 315
22.41. show named resource RESOURCES subtable structure 316

22.42. show named resource PARAMETERS subtable structure 316
22.43. show named resource JOB_DEFINITIONS subtable structure . . . 317
22.44. show resource output 320
22.45. show resource ALLOCATIONS subtable structure 321
22.46. show resource PARAMETERS subtable structure 322
22.47. show resource state definition output 323
22.48. show resource state mapping output 324
22.49. show resource state mapping MAPPINGS subtable structure . . . 325
22.50. show resource state profile output 327
22.51. show resource state profile STATES subtable structure 327
22.52. show schedule output 329
22.53. show scheduled event output 331
22.54. show scope output 333
22.55. show scope RESOURCES subtable structure 335
22.56. show scope CONFIG subtable structure 335
22.57. show scope CONFIG_ENVMAPPING subtable structure 336
22.58. show scope PARAMETERS subtable structure 337
22.59. show session output 339
22.60. show system output 341
22.61. show system WORKER subtable structure 341
22.62. show trigger output 344
22.63. show trigger STATES subtable structure 345
22.64. show user output 347
22.65. show user MANAGE_PRIVS subtable structure 347
22.66. show user GROUPS subtable structure 347
22.67. show user COMMENT subtable structure 347

25.1. submit output 356

27.1. get next job output 374

28.1. get parameter output 391
28.2. get submittag output 392
28.3. submit output 398

Part I.

General

1. Introduction

Introduction

Essentially, this document is divided into three parts. In the BICsuite Scheduling System, there are three types of users (in the broadest sense of the word):

- Users
- Job servers
- Jobs

Each of these users has his own command set at his disposal. These command sets only overlap to a certain extent. For example, for job servers there is the statement **get next job**, which is not valid for either jobs or users. On the other hand, there are forms of the **submit** statement will only make sense in a job context and which can therefore only be implemented by jobs. Obviously only users are allowed to create objects such as Exit State definitions or job definitions. In contrast, there are also statements such as the **connect**
statement which is valid for all types of users.
The structure of this document is oriented to the three types of users. The largest part of this document deals with the user commands, while the two other parts handle job servers and job commands.
For the sake of completeness, the next chapter briefly explains the utility *sdmsh*. This utility is easy to use and is an excellent choice for processing scripts using BICsuite commands.
Since the syntax described here is the only interface to the BICsuite Scheduling Server, all the utilities (and in particular BICsuite!Web) use this web interface.
To simplify the development of proprietary utilities, the server is capable of returning its reactions to statements in various formats. The utility *sdmsh*, for example, uses the **serial** protocol, with which serialised Java objects are transferred. In contrast BICsuite!Web uses the **python** protocol, with which textual representations of Python structures are transferred that can be easily read in using the eval() function.

Syntax diagrams

Syntax diagrams

The syntax diagrams are comprised of different symbols and metasymbols. The symbols and metasymbols are listed and explained in the table below.

Symbol	Meaning
keyword	A keyword in the language. These have to be entered as shown. One example is the keyword **create**.
name	A parameter. In many cases, the user can choose a name or a number to be entered here.
NONTERM	A non-terminal symbol is represented by small caps. A syntax element that is explained further on in the diagram has to be inserted here.
< **all** | **any** >	This syntax element is an optional choice. One of the syntax elements given in the angle brackets, which can obviously also be non-terminal symbols, has to be selected. In the simplest scenario there are only two choices that can be made here, although frequently there are more.
< **all** | **any** >	This is also an optional choice. Unlike the previous syntax element, the underscore of the first element emphasises that this option is the default choice.
[**or alter**]	Optional syntax elements are placed in square brackets.
{ *statename* }	Syntax elements that are placed in braces are repeated 0 to n times.
JOB_PARAMETER {**,** JOB_PARAMETER}	Cases where elements occur at least once are far more common and are shown as represented here.
|	In lists of possible syntax elements, the single possibilities are separated by a |. Such a list is another way of displaying optional choices. These two different forms of presentation are used for purposes of clarity.

Literals

Literals Literals are only required in the language definition for strings, numbers, and dates/times.
Strings are delimited by single quotes, as in

node = 'puma.independit.de'

Integers are shown as either unsigned *integer* or signed *signed_integer* in the syntax diagrams. A *signed_integer* can be prefixed with a **+** or **-** sign. Valid unsigned integers lie in the range of numbers between 0 and $2^{31}-1$. Signed integers are therefore within the range between $-2^{31}+1$ and $2^{31}-1$. If the syntax diagram contains *id*, an unsigned integer between 0 and $2^{63}-1$ is expected here.

Much more complicated are dates/times, particularly in statements concerning the time scheduling. These literals are principally shown as strings with a special format.
The following syntax is used to comply with the notations based on ISO8601 as given in Table 1.1 :

String	*Meaning*	*Range*	*String*	*Meaning*	*Range*
YYYY	year	1970 .. 9999	hh	hour	00 .. 23
MM	month	01 .. 12	mm	minute	00 .. 59
DD	day (of the month)	01 .. 31	ss	second	00 .. 59
ww	week (of the year)	01 .. 53			

- All other strings stand by themselves.
- No differentiation is made between uppercase and lowercase.
- The earliest permissible *point* in time is 1970–01-01T00:00:00 GMT.

Identifier

Identifier

In the BICsuite Scheduling System, objects are identified by their names. (Strictly speaking, objects can also be identified from their internal ID, which is a number, but this practice is not recommended). Valid names comprise a letter, underscore (_), at sign (@) or hash sign (#) followed by numbers, letters or special characters. Language-specific special characters such as the German umlaut are invalid.
Identifiers are treated as being case-insensitive if they are not enclosed in simple quotes. Identifiers enclosed in quotes are case-sensitive. It is therefore not generally recommended to use quotes unless there is a valid reason for doing so.
Identifiers that are allowed to be enclosed in single quotes can also contain spaces and several special characters. Again, this practice is not recommended as spaces are normally interpreted as delimiters and therefore errors can easily occur. Spaces aren't allowed at the beginning or end of an identifier.
There are a number of keywords in the syntax that cannot be readily used as identifiers. Here it may be practicable to use quotes so that the identifiers are not recognised as keywords. Table 1.2 contains a list of such keywords.

There are also a number of synonyms. These are essentially keywords that can be written in more than one way. Only one spelling variation is shown in Table 1.2. The synonyms can be used together arbitrarily. Table 1.3 gives a list of such synonyms.

Format	Example	Simplified format
YYYY	1990	
YYYY−MM	1990−05	YYYYMM
YYYY−MM−DD	1990−05−02	YYYYMMDD
YYYY−MM−DDThh	1990−05−02T07	YYYYMMDDThh
YYYY−MM−DDThh:mm	1990−05−02T07:55	YYYYMMDDThhmm
YYYY−MM−DDThh:mm:ss	1990−05−02T07:55:12	YYYYMMDDThhmmss
−MM	−05	
−MM−DD	−05−02	−MMDD
−MM−DDThh	−05−02T07	−MMDDThh
−MM−DDThh:mm	−05−02T07:55	−MMDDThhmm
−MM−DDThh:mm:ss	−05−02T07:55:12	−MMDDThhmmss
−−DD	−−02	
−−DDThh	−−02T07	
−−DDThh:mm	−−02T07:55	−−DDThhmm
−−DDThh:mm:ss	−−02T07:55:12	−−DDThhmmss
Thh	T07	
Thh:mm	T07:55	Thhmm
Thh:mm:ss	T07:55:12	Thhmmss
T−mm	T−55	
T−mm:ss	T−55:12	T−mmss
T−−ss	T−−12	
YYYYWww	1990W18	
Www	W18	

Table 1.1.: Valid date formats:

As in any language, there are also some reserved words and word combinations. An overview is shown in Table 1.4. A special characteristic of word pairs is that replacing a space with an underscore likewise produces a reserved word. The word **named_resource** is therefore reserved (but "named#resource" isn't).

Versions

Versions

There are three versions of the BICsuite Scheduling System. Since features from later versions are not always present in the earlier versions, the relevant statements or options within the statements are designated accordingly. A letter in the top cor-

ner of the page indicates for which version of the system this statement is available. Deviations from the general statement are shown in the syntax diagram.
The symbols have the following meanings:

Symbol	Meaning
B	This symbol indicates a feature in the Basic version and all later versions.
P	This symbol indicates a feature in the Professional and Enterprise versions and all later versions.
E	This symbol indicates a feature in the Enterprise version.

activate	delay	group	milestone	rawpassword	submitcount
active	delete	header	minute	read	submittag
action	dependency	history	mode	reassure	submitted
add	deregister	hour	month	recursive	sum
after	dir	identified	move	register	suspend
alter	disable	ignore	multiplier	rename	sx
amount	disconnect	immediate	n	required	synchronizing
and	distribution	import	name	requestable	synctime
avg	drop	in	nicevalue	rerun	tag
base	dump	inactive	node	restartable	test
batch	duration	infinite	noinverse	restrict	time
before	dynamic	interval	nomaster	resume	timeout
broken	edit	inverse	nomerge	revoke	timestamp
by	embedded	is	nonfatal	rollback	to
cancel	enable	isx	nosuspend	run	touch
cancelled	endtime	ix	notrace	runnable	trace
cascade	environment	job	notrunc	running	translation
change	errlog	kill	nowarn	runtime	tree
check	error	killed	of	s	trigger
child	event	level	offline	sc	trunc
children	execute	liberal	on	schedule	type
childsuspend	expand	like	online	scope	update
childtag	expired	limits	only	selection	unreachable
clear	factor	line	or	serial	unresolved
command	failure	list	owner	server	usage
comment	fatal	local	parameters	session	use
condition	filter	lockmode	password	set	user
connect	final	logfile	path	shutdown	view
constant	finish	loops	pending	show	warn
content	finished	map	performance	sort	warning
copy	folder	maps	perl	started	week
count	footprint	mapping	pid	starting	with
create	for	master	pool	starttime	workdir
cycle	force	master_id	priority	static	x
day	free_amount	max	profile	status	xml
default	from	min	protocol	stop	year
definition	get	merge	public	strict	
defer	grant	merged	python	submit	

Table 1.2.: Keywords that can be used with quotes as identifiers

Keyword	Synonym	Keyword	Synonym
definition	definitions	minute	minutes
dependency	dependencies	month	months
environment	environments	node	nodes
errlog	errlogfile	parameter	parameters
event	events	profile	profiles
folder	folders	resource	resources
footprint	footprints	schedule	schedules
grant	grants	scope	scopes
group	groups	server	servers
hour	hours	session	sessions
infinit	infinite	state	states, status
interval	intervals	translation	translations
job	jobs	user	users
mapping	mappings	week	weeks
milestone	milestones	year	years

Table 1.3.: Keywords and synonyms

after final	exit state translation	non fatal
all final	ext pid	requestable amount
backlog handling	finish child	resource state
before final	free amount	resource state definition
begin multicommand	get next job	resource state mapping
broken active	ignore dependency	resource state profile
broken finished	immediate local	resource template
change state	immediate merge	resource wait
default mapping	initial state	run program
dependency definition	job definition	rerun program
dependency hierarchy	job definition hierarchy	scheduled event
dependency mode	job final	state profile
dependency wait	job server	status mapping
end multicommand	job state	suspend limit
error text	keep final	submitting user
exec pid	kill program	synchronize wait
exit code	local constant	to kill
exit state	merge mode	until final
exit state mapping	merge global	until finished
exit state definition	merge local	
exit state profile	named resource	

Table 1.4.: Reserved words

2. Utilities

Starting and stopping the server

server-start

Introduction

The utility *server-start* is used to start the scheduling server. *Introduction*

Call

The following commands are used to call server-start: *Call*

server-start [OPTIONS] *config-file*

OPTIONS:
-admin
| **-protected**

The individual options have the following meanings:

Option	Meaning
-admin	The server starts in Admin mode. This means that user logins are disabled apart from the user **SYSTEM**.
-protected	Protected mode is similar to Admin mode. The difference here is that the internal threads (TimerThread and SchedulingThread) are not started. This allows administrative tasks to be carried out without concurrent transactions being performed.

If the server has already been started, the second server either (depending on the configuration) takes over the operation or repeatedly makes an unsuccessful attempt to start.
The server-start utility can be only be used by the user whose ID was used to install the system.

server-stop

Introduction

Introduction The *server-stop* utility is used to stop the scheduling server.

Call

Call The following command is used to call server-stop:

server-stop

Initially, an attempt is made to stop the server 'gracefully'. First, all the user connections are terminated to stop all the internal threads.
If this approach fails or it takes too long, the server is stopped using the operating system's mechanisms.
If the server has not been started, the server-stop command has no effect.
The server-stop utility can be only be used by the user whose ID was used to install the system.

sdmsh

Introduction

Introduction The *sdmsh* utility is a small program that enables the user to interactively work with the scheduling server. In contrast to the BICsuite!Web front end, for instance, this working method is text-oriented. This means it is possible to write scripts and execute them using
sdmsh. The *sdmsh* executable is a small script or batch file that encapsulates the call of the required Java program. Of course, there is no reason why this Java program should not be called manually. It is only there for convenience's sake.

Call

Call The following commands are used to call sdmsh:

sdmsh [OPTIONS] [*username* [*password* [*host* [*port*]]]]

OPTIONS:
< **--host** | **-h** > *hostname*
| < **--port** | **-p** > *portnumber*
| < **--user** | **-u** > *username*
| < **--pass** | **-w** > *password*
| < **--jid** | **-j** > *jobid*

| < **--key** | **-k** > *jobkey*
| < **--**[**no**]**silent** | **-**[**no**]**s** >
| < **--**[**no**]**verbose** | **-**[**no**]**v** >
| < **--ini** | **-ini** > *inifile*
| < **--**[**no**]**tls** | **-**[**no**]**tls** >
| **--**[**no**]**help**
| **--info** *sessioninfo*
| **-**[**no**]**S**
| **--timeout** *timeout*

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	BICsuite!Server Host
< **--port** | **-p** > *portnumber*	BICsuite!Server port
< **--user** | **-u** > *username*	User name (user or jid has to be specified)
< **--pass** | **-w** > *password*	Password (is used in combination with the option **--user**)
< **--jid** | **-j** > *jobid*	Job ID (user or jid has to be specified)
< **--key** | **-k** > *jobkey*	Job key (is used in combination with the option **--jid**)
< **--**[**no**]**silent** | **-**[**no**]**s** >	(Error) Messages are not returned
< **--**[**no**]**verbose** | **-**[**no**]**v** >	[No] Commands, feedback and additional messages are returned
< **--ini** | **-ini** > *inifile*	Use the specified configuration file to set options
< **--**[**no**]**tls** | **-**[**no**]**tls** >	Use access via TLS/SSL [not]
--[**no**]**help**	Return a help text
--info *sessioninfo*	Set the accompanying information as descriptive information about the session
-[**no**]**S**	Silent option. This option is obsolete and exists for reasons of backward compatibility
--timeout *timeout*	The number of seconds after which the server terminates an idle session. The value 0 means no timeout

sdmsh obviously requires information to connect to the correct BICsuite Scheduling System. The necessary data can be specified in the command line or by using an options file. Missing values for the user name and password are queried by *sdmsh*. If values for the host and port are not given, the defaults values "localhost" and 2506 are used. It is not advisable to specify the password in the command line

because this information can frequently be easily read out by other users.

Options file

Options file

The options file has the same format as a Java property file. Please refer to the official Java documentation for details of the precise syntax specification.
The following option files play a role:

- `$SDMSCONFIG/sdmshrc`
- `$HOME/.sdmshrc`
- Optionally, a file specified in the command line

The files are valuated in the given order. If options are present in several files, the value in the last valuated file "wins". Options that are specified in the command line take precedence over all the other specifications.
The following keywords are recognised:

Keyword	Meaning
User	The user's name
Password	The user's password
Host	Name or IP address of the host
Info	Additional information for identifying a connection is set
Port	Port number of the scheduling server (default: 2506)
Silent	(Error) messages are not returned
Timeout	Timeout value for the session (0 means no timeout)
TLS	Use an SSL/TLS connection
Verbose	Commands, feedback and additional messages are returned

Since the user's password is shown in plain text in this file, careful consideration needs to be taken when assigning the access privileges for this file. It is, of course, possible to not specify the password and to enter it every time *sdmsh* is started.
Only the following keywords can be used in configuration files:

Keyword	Meaning
KeyStore	Keystore for TLS/SSL communication
TrustStore	Truststore for TLS/SSL communication
KeyStorePassword	Keystore password
TrustStorePassword	Truststore password

Internal commands

Internal commands

Apart from the BICsuite commands described in the following chapters, *sdmsh* also knows a few simple commands of its own. These are briefly described below. Internal commands do not have to be closed with a semicolon.

disconnect The disconnect command is used to exit *sdmsh*. Because different commands are commonly used to exit a tool in different work environments, an attempt was made here to incorporate many varying formulations.
The syntax for the disconnect command is:

< **disconnect** | **bye** | **exit** | **quit** >

EXAMPLE Here is an example of the disconnect command.

```
ronald@jaguarundi:~$ sdmsh
Connect

CONNECT_TIME : 23 Aug 2007 07:13:30 GMT

Connected

[system@localhost:2506] SDMS> disconnect
ronald@jaguarundi:~$
```

echo If *sdmsh* is being used interactively, it is visually evident which command has just been entered. This is not the case in batch mode, i.e. when processing a script. The echo command can be used to enable and disable the rendering of the entered statement. This is enabled by default.
The syntax for the echo command is:

echo < **on** | **off** >

EXAMPLE The effect of these two options is shown below. Following the command **echo on**

```
[system@localhost:2506] SDMS> echo on
End of Output

[system@localhost:2506] SDMS> show session;
show session;

Session

     THIS :  *
SESSIONID : 1001
START : Tue Aug 23 11:47:34 GMT+01:00 2007
USER : SYSTEM
```

```
UID : 0
IP : 127.0.0.1
TXID : 136448
IDLE : 0
TIMEOUT : 0
STATEMENT : show session

Session shown

[system@localhost:2506] SDMS> echo off

End of Output

[system@localhost:2506] SDMS> show session;

Session

     THIS :  *
SESSIONID : 1001
START : Tue Aug 23 11:47:34 GMT+01:00 2007
USER : SYSTEM
UID : 0
IP : 127.0.0.1
TXID : 136457
IDLE : 0
TIMEOUT : 0
STATEMENT : show session

Session shown

[system@localhost:2506] SDMS>
```

help The help command opens a condensed help text about the internal *sdmsh* commands.
The syntax for the help command is:

help

EXAMPLE The help command only returns a condensed help text about the syntax for the internal *sdmsh* commands. This is shown in the example below. (The lines have been wrapped for this document and so the actual output may differ to what is written here).

```
[fontsize=\small]
[system@localhost:2506] SDMS> help
Condensed Help Feature
----------------------
Internal sdmsh Commands:
disconnect|bye|exit|quit         -- leaves the tool
echo on|off                      -- controls whether the statement text is
                                    printed or not
```

```
help                                -- gives this output
include '<filespec>'                -- reads sdms(h) commands from the given
                                       file
prompt '<somestring>'               -- sets to prompt to the specified value
                                       %H = hostname, %P = port, %U = user,
                                       %% = %
timing on|off                       -- controls whether the actual time is
                                       printed or not

whenever error
continue|disconnect <integer>       -- specifies the behaviour of the program
                                       in case of an error
!<shellcommand>                     -- executes the specified command. sdmsh
                                       has no intelligence
                                       at all regarding terminal I/O

End of Output
[system@localhost:2506] SDMS>
```

include Files can be integrated into BICsuite statements using the include command.
The syntax for the include command is:

include '*filespec*'

EXAMPLE In the following example, a file only containing the command "**show session;**" is inserted.

```
[system@localhost:2506] SDMS> include '/tmp/show.sdms'
Session

     THIS :  *
SESSIONID : 1001
START : Tue Aug 23 11:47:34 GMT+01:00 2007
USER : SYSTEM
UID : 0
IP : 127.0.0.1
TXID : 136493
IDLE : 0
TIMEOUT : 0
STATEMENT : show session

Session shown

[system@localhost:2506] SDMS>
```

prompt The prompt command can be used to specify an arbitrary prompt. There are a number of variable values that can be inserted automatically by the program. The codes for the individual variables are shown in the table below:

Code	Meaning
%H	Hostname des Scheduling Servers
%P	TCP/IP Port
%U	Username
%%	Percent character (%)

The default prompt has the following definition: `[%U@%H:%P] SDMS>`.
The syntax for the prompt command is:

prompt '*somestring*'

EXAMPLE In the following example, an empty prompt is defined first. A BICsuite statement is then executed to make the effect more clearly visible. A simple string is then selected as a prompt, and finally the variables are used.

```
[system@localhost:2506] SDMS> prompt ''
End of Output

show session;
show session;

Session

     THIS :  *
SESSIONID : 1001
START : Tue Aug 23 11:47:34 GMT+01:00 2007
USER : SYSTEM
UID : 0
IP : 127.0.0.1
TXID : 136532
IDLE : 0
TIMEOUT : 0
STATEMENT : show session

Session shown

prompt 'hello world '

End of Output

hello world prompt '[%U@%H:%P] please enter your wish! > '


End of Output

[system@localhost:2506] please enter your wish! >
```

timing The timing command provides information about the execution time for a statement. Normally, this option is disabled and so no information about the

execution time is given. The time is stated in milliseconds.
The syntax for the timing command is:

timing < **off** | **on** >

EXAMPLE The following example shows the timing information for a simple BIC-suite statement. The execution time for the statements and the time that was required to output the result is shown.

```
[system@localhost:2506] SDMS> timing on
End of Output

[system@localhost:2506] SDMS> show session;
Execution Time: 63
show session;

Session

     THIS :  *
SESSIONID : 1002
START : Tue Aug 23 11:53:15 GMT+01:00 2007
USER : SYSTEM
UID : 0
IP : 127.0.0.1
TXID : 136559
IDLE : 0
TIMEOUT : 0
STATEMENT : show session

Session shown

[system@localhost:2506] SDMS>
Render Time   : 143
```

whenever An error handling routine is absolutely essential particularly when *sdmsh* is being used to execute scripts. The whenever statement tells
sdmsh how to deal with errors. By default errors are ignored, which also corresponds to the desired behaviour for interactive working. The syntax for the whenever command is:

whenever error < **continue** | **disconnect** *integer* >

EXAMPLE The example below shows the behaviour of both the **continue** option and the **disconnect** option. The Exit Code for a process that was started by the Bourne shell (and other Unix shells as well) can be shown by outputting the variable `$?` .

```
[system@localhost:2506] SDMS> whenever error continue
End of Output
```

```
[system@localhost:2506] SDMS> show exit state definition does_not_exist;
show exit state definition does_not_exist;

ERROR:03201292040, DOES_NOT_EXIST not found

[system@localhost:2506] SDMS> whenever error disconnect 17

End of Output

[system@localhost:2506] SDMS> show exit state definition does_not_exist;
show exit state definition does_not_exist;

ERROR:03201292040, DOES_NOT_EXIST not found

[system@localhost:2506] SDMS>
ronald@jaguarundi:~$ echo $?
17
ronald@jaguarundi:~$
```

Shell call It frequently happens that a shell command has to be quickly executed, for instance to see what the file that is to be run (using **include**) is called. If no special capabilities are required of the terminal, such as is the case when calling an editor, a shell command can be executed by prefixing it with an exclamation mark. The syntax for a shell call is:

!*shellcommand*

EXAMPLE In the following example, a short list of all the *sdmsh* scripts in the `/tmp` directory is outputted.

```
[system@localhost:2506] SDMS> !ls -l /tmp/*.sdms
-rw-r--r-- 1 ronald ronald 15 2007-08-23 09:30 /tmp/ls.sdms
End of Output

[system@localhost:2506] SDMS>
```

sdms-auto_restart

Introduction

Introduction

The utility *sdms-auto_restart* is used to automatically restart jobs that have failed. A number of simple conditions have to be met to do this. Probably the most important condition is that the job defines a parameter AUTORESTART with the value TRUE. This parameter can naturally also be set at a higher level.
The following parameters influence the behaviour of the autorestart utilities:

Parameter	Effect
AUTORESTART	The autorestart only functions if this parameter is set to ″TRUE″
AUTORESTART_MAX	Defines the maximum number of automatic restarts if set
AUTORESTART_COUNT	Is set by the aurorestart utility to count the number of restarts
AUTORESTART_DELAY	The time in minutes before a job is restarted

The autorestart utility can be defined as a trigger. The trigger types IMMEDIATE_LOCAL and FINISH_CHILD can be used.

The logic of the option files that applies for the sdmsh utility is also used for sdms-auto_restart.

Call

The following commands are used to call sdms-auto_restart: *Call*

sdms-auto_restart [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--user** | **-u** > *username*
< **--pass** | **-w** > *password* < **--failed** | **-f** > *jobid*

OPTIONS:
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--delay** | **-d** > *seconds*
< **--max** | **-m** > *number*
< **--warn** | **-W** >

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login
< **--failed** | **-f** > *jobid*	Job ID of the job that is to be restarted
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts
< **--help** | **-h** >	Returns a condensed help
< **--delay** | **-d** > *minutes*	Number of minutes for the delay until the job is restarted
< **--max** | **-m** > *number*	Maximum number of automatic restarts
< **--warn** | **-W** >	The warning flag is set when the maximum number of restarts has been reached

sdms-get_variable

Introduction

Introduction

The utility *sdms-get_variable* offers a simple way of reading out job parameters from the scheduling system.
The logic of the option files that applies for the sdmsh utility is also used for sdms-get_variable.

Call

Call

The following commands are used to call sdmsh-get_variable:

sdms-get_variable [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--jid** | **-j** > *jobid*
< **--name** | **-n** > *parametername*

OPTIONS:
< **--user** | **-u** > *username*

< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--mode** | **-m** > *mode*

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--key** | **-k** > *jobkey*	for the login (for a connection as job)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--mode** | **-m** > *mode*	Mode for determining the parameter (liberal, warn, strict)

Example

Example

The example below shows how to get the variable value of the variable RESPONSE of job 5175119.

```
ronald@cheetah:~$ sdms-get_variable -h localhost -p 2506 \
                  -j 5175119 -u donald -w duck -n RESPONSE
```

sdms-rerun

Introduction

Introduction

The utility *sdms-rerun* is used to rerun a job in a restartable state from a script or program.

The logic of the option files that applies for the sdmsh utility is also used for sdms-rerun.

Call

Call

The following commands are used to call sdms-rerun:

sdms-rerun [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--jid** | **-j** > *jobid*

OPTIONS:
< **--user** | **-u** > *username*
< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--suspend** | **-S** >
< **--delay** | **-D** > *delay*
< **--unit** | **-U** > *unit*
< **--at** | **-A** > *at*

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--suspend** | **-S** >	The job is suspended
< **--delay** | **-D** > *delay*	The job is automatically resumed after *delay* units
< **--unit** | **-U** > *unit*	Unit for the delay option (default MINUTE)
< **--at** | **-A** > *at*	Automatic resume at the specified time

sdms-set_state

Introduction

Introduction

The utility *sdms-set_state* offers a simple way of setting the state of a job in the scheduling system.
The logic of the option files that applies for the sdmsh utility is also used for sdms-set_state.

Call

Call

The following commands are used to call sdmsh-set_state:

sdms-set_state [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--jid** | **-j** > *jobid*
< **--state** | **-S** > *statename*

OPTIONS:
< **--user** | **-u** > *username*

< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--force** | **-f** >

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--key** | **-k** > *jobkey*	for the login (for a connection as job)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--force** | **-f** >	Force if job does not define a mapping for the specified state

sdms-set_variable

Introduction

Introduction

The utility *sdms-set_variable* offers a simple way of setting job parameters in the scheduling system.
The logic of the option files that applies for the sdmsh utility is also used for sdms-set_variable.

Call

The following commands are used to call sdms-set_variable: *Call*

sdms-set_variable [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--jid** | **-j** > *jobid*
parametername value { *parametername value*}

OPTIONS:
< **--user** | **-u** > *username*
< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--case** | **-C** >

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--key** | **-k** > *jobkey*	for the login (for a connection as job)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--case** | **-C** >	Names are case-sensitive

sdms-set_warning

Introduction

Introduction

The utility *sdms-set_warning* is used to set the warning flag for a job. A text can be optionally specified. The warning flag can be set for a job by users who have been granted the Operate privilege. A job can set the warning flag for itself.

The logic of the option files that applies for the sdmsh utility is also used for sdms-set_warning.

Call

Call

The following commands are used to call sdms-set_warning:

sdms-set_warning [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--jid** | **-j** > *jobid*

OPTIONS:
< **--user** | **-u** > *username*
< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--warning** | **-m** > *warning*

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--key** | **-k** > *jobkey*	for the login (for a connection as job)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--warning** | **-m** > *warning*	Warning text

sdms-submit

Introduction

Introduction

The utility *sdms-submit* is used to start jobs or batches. These can be started as a standalone workflow or also as a child of an existing job. In the latter case, if it is defined in the parent-child hierarchy an alias can be specified to identify the job or batch that is to be submitted.
The logic of the option files that applies for the sdmsh utility is also used for sdms-submit.

Call

Call

The following commands are used to call sdms-submit:

sdms-submit [OPTIONS] < **--host** | **-h** > *hostname*
< **--port** | **-p** > *portnumber* < **--job** | **-J** > *jobname*

OPTIONS:
< **--user** | **-u** > *username*
< **--pass** | **-w** > *password*
< **--jid** | **-j** > *jobid*
< **--key** | **-k** > *jobkey*

< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--tag** | **-T** > *tag*
< **--master** | **-M** >
< **--suspend** | **-S** >
< **--delay** | **-D** > *delay*
< **--unit** | **-U** > *unit*
< **--at** | **-A** > *at*

The individual options have the following meanings:

Option	Meaning
< **--host** | **-h** > *hostname*	Host name of the scheduling server
< **--port** | **-p** > *portnumber*	Port of the scheduling server
< **--user** | **-u** > *username*	User name for the login
< **--pass** | **-w** > *password*	Password for the login (for a connection as user)
< **--key** | **-k** > *jobkey*	for the login (for a connection as job)
< **--silent** | **-s** >	Reduces the number of messages that are returned
< **--verbose** | **-v** >	Increases the number of messages that are returned
< **--timeout** | **-t** > *minutes*	Number of minutes for attempting to get a server connection
< **--cycle** | **-c** > *minutes*	Number of minutes for the delay between two attempts to set up a server connection
< **--help** | **-h** >	Returns a condensed help text about calling the utility
< **--tag** | **-T** > *tag*	Tag for dynamic submits
< **--master** | **-M** >	Submit for a master, no child
< **--suspend** | **-S** >	The job is suspended
< **--delay** | **-D** > *delay*	The job is automatically resumed after *delay* units
< **--unit** | **-U** > *unit*	Unit for the delay option (default MINUTE)
< **--at** | **-A** > *at*	Automatic resume at the specified time

Part II.

User Commands

3. alter commands

alter comment

Purpose

Purpose The purpose of the alter comment statement is to change the comment for the specified object.

Syntax

Syntax The syntax for the alter comment statement is

alter [**existing**] **comment on** OBJECTURL
with CC_WITHITEM

OBJECTURL:
distribution *distributionname* **for pool** *resourcepath* **in** *serverpath*
| **environment** *environmentname*
| **exit state definition** *statename*
| **exit state mapping** *mappingname*
| **exit state profile** *profilename*
| **event** *eventname*
| **resource** *resourcepath* **in** *folderpath*
| **folder** *folderpath*
| **footprint** *footprintname*
| **group** *groupname*
| **interval** *intervalname*
| **job definition** *folderpath*
| **job** *jobid*
| **named resource** *resourcepath*
| **parameter** *parametername* **of** PARAM_LOC
| **resource state definition** *statename*
| **resource state mapping** *mappingname*
| **resource state profile** *profilename*
| **scheduled event** *schedulepath* **.** *eventname*
| **schedule** *schedulepath*
| **resource** *resourcepath* **in** *serverpath*
| < **scope** *serverpath* | **job server** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

CC_WITHITEM:
CC_TEXTITEM {, CC_TEXTITEM}

| **url =** *string*

PARAM_LOC:

folder *folderpath*
| **job definition** *folderpath*
| < **scope** *serverpath* | **job server** *serverpath* >

TRIGGEROBJECT:

resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

CC_TEXTITEM:

tag = < **none** | *string* > **, text =** *string*
| **text =** *string*

Description

Description

The alter comment command is used to change the condensed description or URL of the description of the object in question. Of course, the type of information can be changed as well. The comment is versioned. This means that comments are not overwritten. When the commented object is displayed, the displayed comment is the one that matches the version of the displayed object.
The optional **existing** keyword is used to prevent error messages from being displayed and the current operation from being terminated. This is particularly useful in conjunction with multicommands.

Output

Output

This statement returns a confirmation of a successful operation

alter environment

Purpose

Purpose The purpose of the alter environment statement is to alter the properties of the specified environment.

Syntax

Syntax The syntax for the alter environment statement is

alter [**existing**] **environment** *environmentname*
with ENV_WITH_ITEM

alter [**existing**] **environment** *environmentname*
add (ENV_RESOURCE {**,** ENV_RESOURCE} **)**

alter [**existing**] **environment** *environmentname*
delete (*resourcepath* {**,** *resourcepath*} **)**

ENV_WITH_ITEM:
resource = none
| **resource = (** ENV_RESOURCE {**,** ENV_RESOURCE} **)**

ENV_RESOURCE:
resourcepath [< **condition =** *string* | **condition = none** >]

Description

Description The alter environment statement is used to change the resource requests that are defined in this environment. Running jobs are not affected.
The "**with resource =**" form of the statement replaces the existing group of resource requests. The other types either add the specified requests or deletes them. It is considered an error to delete a request that is not part of the environment or to add a request for an already required resource.
Only administrators are authorised to perform this action.

Output

Output This statement returns a confirmation of a successful operation

alter event

Purpose

The purpose of the alter event statement is to change properties of the specified event. *Purpose*

Syntax

The syntax for the alter event statement is *Syntax*

alter [**existing**] **event** *eventname*
with EVENT_WITHITEM {**,** EVENT_WITHITEM}

EVENT_WITHITEM:
action =
submit *folderpath* [**with parameter = (** PARAM {**,** PARAM} **)**]
| **group =** *groupname*

PARAM:
parametername **=** < *string* | *number* >

Description

The alter event statement is used to change the properties of an event. A parameter for a job submit can be specified using the **with parameter** clause. For a detailed description of these options, refer to the create event statement on page 83. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

alter exit state mapping

Purpose

Purpose The purpose of the alter exist state mapping statement is to change properties of the specified mapping.

Syntax

Syntax The syntax for the alter exit state mapping statement is

alter [**existing**] **exit state mapping** *mappingname*
with map = (*statename* { **,** *signed_integer* **,** *statename* } **)**

Description

Description The alter exit state mapping statement defines the mapping of the Exit Codes for logical Exit States. The simplest form of this statement only specifies one Exit State. This means that the job acquires this Exit State when it finishes regardless of its Exit Code. More complex definitions specify more than one Exit State and at least one delimitation.
A statement like

```
alter exit state mapping example1
with map = (   failure,
            0, success,
            1, warning,
            4, failure);
```

defines the following mapping:

Exit code range from	Exit code range until	Resulting exit state
$-\infty$	-1	failure
0	0	success
1	3	warning
4	∞	failure

Output

Output This statement returns a confirmation of a successful operation

alter exit state profile

Purpose

Purpose

The purpose of the alter exit state profile statement is to change properties of the specified profile.

Syntax

Syntax

The syntax for the alter exit state profile statement is

alter [**existing**] **exit state profile** *profilename*
with WITHITEM {**,** WITHITEM}

WITHITEM:
default mapping = < **none** | *mappingname* >
| **force**
| **state = (** ESP_STATE {**,** ESP_STATE} **)**

ESP_STATE:
statename < **final** | **restartable** | **pending** > [OPTION { OPTION}]

OPTION:
< **unreachable** | **broken** | **batch default** | **dependency default** >

Description

Description

The alter exit state profile statement is used to add Exit States to the profile or delete them, as well as to define the default Exit State Mapping. For a detailed description of these options, refer to the create exit state profile statement on page 86.

force The **force** option labels the Exit State Profiles as being invalid, which only means that the integrity still has to be verified. The label is removed after a successful verification. The verification is carried out by submitting a job definition that uses the Exit State Profiles. The purpose of the **force** flag is to be capable of changing several Exit State Profiles (and perhaps some other objects) without the need for a consistent state after each change.

Output

Output

This statement returns a confirmation of a successful operation

alter folder

Purpose

Purpose The purpose of the alter foder statement is to alter the properties of a folder.

Syntax

Syntax The syntax for the alter folder statement is

alter [**existing**] **folder** *folderpath*
with WITHITEM {**,** WITHITEM}

WITHITEM:
environment = < **none** | *environmentname* >
| **group =** *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **parameter = none**
| **parameter = (** *parametername* **=** *string* {**,** *parametername* **=** *string*} **)**

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

Description

Description The alter folder statement changes the properties of a folder. For a detailed description of these options, refer to the create folder statement on page 89.
If the optional **existing** keyword has been specified, no error is created if the specified folder does not exist.
Although the folder SYSTEM cannot be created, dropped or renamed, it can be altered to some extend. It is not possible to change the owning group, but it is possible to specify an environment or to create parameters.

Output

This statement returns a confirmation of a successful operation *Output*

alter footprint

Purpose

Purpose The purpose of the alter footprint statement is to change the properties of the specified footprint.

Syntax

Syntax The syntax for the alter footprint statement is

alter [**existing**] **footprint** *footprintname*
with resource = (REQUIREMENT {**,** REQUIREMENT} **)**

alter [**existing**] **footprint** *footprintname*
add resource = (REQUIREMENT {**,** REQUIREMENT} **)**

alter [**existing**] **footprint** *footprintname*
delete resource = (*resourcepath* {**,** *resourcepath*} **)**

REQUIREMENT:
ITEM { ITEM}

ITEM:
amount = *integer*
| < **nokeep** | **keep** | **keep final** >
| *resourcepath*

Description

Description The alter footprint command changes the list of resource requests. There are three kinds of this statement.

- The first one determines all the resource requests.
- The second one adds resource requests to the request list.
- The third kind removes requests from the list.

For a detailed description of these options, refer to the create footprint statement on page 91.

Output

Output This statement returns a confirmation of a successful operation

alter group

Purpose

The purpose of the alter group statement is to alter the user to group assignments. *Purpose*

Syntax

The syntax for the alter group statement is *Syntax*

alter [**existing**] **group** *groupname*
with WITHITEM

alter [**existing**] **group** *groupname*
ADD_DELITEM {**,** ADD_DELITEM}

WITHITEM:
user = none
| **user = (** *username* {**,** *username*} **)**

ADD_DELITEM:
< **add** | **delete** > **user = (** *username* {**,** *username*} **)**

Description

The alter group command is used to define which users belong to the group. There are two kinds of this statement: *Description*

- The first one defines the list of users who belong to the group.
- The second one adds users to the group or deletes them.

In all cases, deleting users from their default group is considered to be an error.
It is not possible to delete users from the PUBLIC group.
If a user does not belong to a group, any attempt made to delete the user from this group is ignored.
If the **existing** keyword has been specified, it is not considered to be an error if the group does not exist.

Output

This statement returns a confirmation of a successful operation *Output*

alter interval

Purpose

Purpose The purpose of the alter interval statement is to change properties of the specified interval.

Syntax

Syntax The syntax for the alter interval statement is

alter [**existing**] **interval** *intervalname*
with WITHITEM {**,** WITHITEM}

WITHITEM:
base = < **none** | *period* >
| **duration =** < **none** | *period* >
| **embedded =** < **none** | *intervalname* >
| **endtime =** < **none** | *datetime* >
| **filter = none**
| **filter = (** *intervalname* {**,** *intervalname*} **)**
| < **noinverse** | **inverse** >
| **selection = none**
| **selection = (** IVAL_SELITEM {**,** IVAL_SELITEM} **)**
| **starttime =** < **none** | *datetime* >
| **synctime =** *datetime*
| **group =** *groupname*

IVAL_SELITEM:
< *signed_integer* | *datetime* | *datetime* **-** *datetime* >

Description

Description The alter interval command is used to change an interval definition. For a detailed description of these options, refer to the create interval statement on page 94.
If the **existing** keyword has been specified, it is not considered to be an error if the interval does not exist.

Output

Output This statement returns a confirmation of a successful operation

alter job

Purpose

The purpose of the alter job statement is to change properties of the specified job. *Purpose*
This statement is is used by job administrators, jobservers, and by the job itself.

Syntax

The syntax for the alter job statement is *Syntax*

alter job *jobid*
with WITHITEM {**,** WITHITEM}

alter job
with WITHITEM {**,** WITHITEM}

WITHITEM:
< **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **comment =** *string*
| **error text =** *string*
| **exec pid =** *pid*
| **exit code =** *signed_integer*
| **exit state =** *statename* [**force**]
| **ext pid =** *pid*
| **ignore resource = (** *id* {**,** *id*} **)**
| **ignore dependency = (** *jobid* [**recursive**] {**,** *jobid* [**recursive**]} **)**
| **kill**
| **nicevalue =** *signed_integer*
| **priority =** *integer*
| **renice =** *signed_integer*
| **rerun** [**recursive**]
| **resume**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **run =** *integer*
| **state =** JOBSTATE
| **timestamp =** *string*
| **warning =** *string*

JOBSTATE:

broken active
broken finished
dependency wait
error
finished
resource wait
running
started
starting
synchronize wait

Description

Description The alter job command is used for several purposes. Firstly, job servers use this command to document the progress of a job. All the state transitions a job undergoes during the time when the job is the responsibility of a job server are performed using the alter job command.
Secondly, some changes such as ignoring dependencies or resources, as well as changing the priority of a job, are carried out manually by an administrator.
The Exit State of a job in a Pending State can be set by the job itself or by a process that knows the job ID and key of the
job that is to be changed.

cancel The cancel option is used to cancel the addressed job and all non-Final Children. A job can only be cancelled if neither the job itself nor one of its children is active. Cancelling a running job will set the job in a cancelling state. The effective cancel is postponed until the job is finished
If a Scheduling Entity is dependent upon the cancelled job, it can become unreachable. In this case the dependent job does not acquire the Unreachable Exit State defined in the Exit State Profiles, but is set as having the Job State "Unreachable". It is the operator's task to restore this job back to the job state "Dependency Wait" by ignoring dependencies or even to cancel it.
Cancelled jobs are considered to be just like Final Jobs without a Final Exit. This means that the parents of a cancelled job become final without taking into consideration the Exit State of the cancelled job. In this case the dependent jobs of the parents continue running normally.
The cancel option can only be used by users.

comment The comment option is used to document an action or to add a comment to the job. Comments can have a maximum length of 1024 characters. Any number of comments can be saved for a job.
Some comments are saved automatically. For example, if a job attains a Restartable State, a log is written to document this fact.

error text The error text option is used to write error information about a job. This can be done by the responsible job server or a user. The server can write this text itself as well.
This option is normally used if the job server cannot start the corresponding process. Possible cases are where it is not possible to switch to the defined working directory, if the executable program cannot be found, or when opening the error log file triggers an error.

exec pid The exec pid option is used exclusively by the job server to set the process ID of the control process within the server.

exit code The exit code option is used by the job server to tell the repository server with which Exit Code the process has finished. The repository server now calculates the matching Exit State from the Exit State Mapping that was used.

exit state The exit state option is used by jobs in a pending state to set their state to another value. This is usually a Restartable or Final State.
Alternatively, this option can be used by administrators to set the state of a non-final job.
If the Force Flag is not being used, the only states that can be set are those which are theoretically attainable by applying the Exit State Mapping to any Exit Code. The set state must exist in the Exit State Profile.

ext pid The ext pid option is used exclusively by the job server to set the process ID of the started user process.

ignore resource The ignore resource option is used to revoke individual Resource Requests. The ignored resource is then no longer requested.
If the parameters of a resource are being referenced, that resource cannot be ignored.
If invalid IDs have been specified, it is skipped. All other specified resources are ignored. Invalid IDs in this context are the IDs of resources that are not requested by the job.
The ignoring of resources is logged.

ignore dependency The ignore dependency option is used to ignore defined dependencies. If the **recursive** flag is used, not only do the job or batch ignore the dependencies, but its children do so as well.

kill The kill option is used to submit the defined Kill Job. If no Kill Job has been defined, it is not possible to forcibly terminate the job from within BICsuite. The job obviously has to be active, that means it must be **running**, **killed** or **broken_active**.

The last two states are not regular cases. When a Kill Job has been submitted, the Job State is **to_kill**. After the Kill Job has terminated, the Job State of the killed job is set to **killed** unless it has been completed, in which case it is **finished** or **final**. This means that the job with the Job State **killed** is always still running and that at least one attempt has been made to terminate it.

nicevalue The nicevalue option is used to change the priority or the nicevalue of a job or batch and all of its children. If a child has several parents, any changes you make can, but do not necessarily have to, affect the priority of the child in the nicevalue of one of the parents. Where there are several parents, the maximum nicevalue is searched for.

This means that if Job C has three Parents P1, P2 and P3, whereby P1 sets a nice value of 0, P2 sets a nicevalue of 10 and P3 a nicevalue of -10, the effective nicevalue is -10. (The lower the nicevalue the better). If the nicevalue for P2 is changed to -5, nothing happens because the -10 of P3 is better than -5. If the nicevalue of P3 falls to 0, the new effective nicevalue for Job C is -5.

The nicevalues can have values between -100 and 100. Values that exceed this range are tacitly adjusted.

priority The priority option is used to change the (static) priority of a job. Because batches and milestones are not executed, priorities are irrelevant to them.

Changing the priority only affects the changed job. Valid values lie between 0 and 100. In this case, 100 corresponds to the lowest priority and 0 is the highest priority.

When calculating the dynamic priority of a job, the scheduler begins with the static priority and adjusts it according to how long the job has already been waiting. If more than one job has the same dynamic priority, the job with the lowest job ID is scheduled first.

renice The renice option is similar to the nicevalue option with the difference that the renice option functions relatively while the nicevalue option functions absolutely. If some batches have a nicevalue of 10, a renice of -5 causes the nicevalue to rise to 5. (It rises because the lower the number, the higher the priority).

rerun The rerun option is used to restart a job in a Restartable State. If you attempt to restart a job that is not restartable, an error message is displayed. A job is restartable if it is in a Restartable State or it has the Job State **error** or **broken_finished**.

If the **recursive** flag has been specified, the job itself and all its direct and indirect children that are in a Restartable State are restarted. If the job itself is final, this is not considered to be an error. It is therefore possible to recursively restart batches.

resume The resume option is used to reactivate a suspended job or batch. There are two ways to do this. The suspended job or batch can either be reactivated immediately or a delay can be set.
A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
For details about specifying a time, refer to the overview on page 6. The resume option can be used together with the suspend option. Here, the job is suspended and then resumed again after (or at) a specified time.

run The run option is used by the job server to ensure that the modified job matches the current version.
Theoretically, the computer could crash after a job has been started by a job server. To complete the work, the job is manually restarted from another job server. After the first system has been booted, the job server can attempt to change the job state to broken_finished without knowing anything about what happened after the crash. Using the run option then prevents the wrong state from being set.

state The state option is mainly used by job servers, but it can also be used by administrators. It is not recommended to do so unless you know exactly what you are doing.
The usual procedure is that the job server sets the state of a job from **starting** to **started**, from **started** to **running**, and from **running** to
finished. In the event of a crash or any other problems, it is possible for the job server to set the job state to **broken_active** or **broken_finished**. This means that the Exit Code of the process is not available and the Exit State has to be set manually.

suspend The suspend option is used to suspend a batch or job. It always functions recursively. If a parent is suspended, its children are all suspended as well. The resume option is used to reverse the situation.
The effect of the restrict option is that cwa resume can be done by members of the group ADMIN only.

timestamp The timestamp option is used by the job server to set the timestamps of the state transition in keeping with the local time from the perspective of job server.

Output

This statement returns a confirmation of a successful operation *Output*

alter job definition

Purpose

Purpose The purpose of the alter job definition statement is to change properties of the specified job definition.

Syntax

Syntax The syntax for the alter job definition statement is

alter [**existing**] **job definition** *folderpath* **.** *jobname*
with WITHITEM {**,** WITHITEM}

alter [**existing**] **job definition** *folderpath* **.** *jobname*
AJD_ADD_DEL_ITEM {**,** AJD_ADD_DEL_ITEM}

WITHITEM:
children = none
| **children = (** JOB_CHILDDEF {**,** JOB_CHILDDEF} **)**
| **dependency mode =** < **all** | **any** >
| **environment =** *environmentname*
| **errlog =** < **none** | *filespec* [< **notrunc** | **trunc** >] >
| **footprint =** < **none** | *footprintname* >
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **kill program =** < **none** | *string* >
| **logfile =** < **none** | *filespec* [< **notrunc** | **trunc** >] >
| **mapping =** < **none** | *mappingname* >
| < **nomaster** | **master** >
| **nicevalue =** < **none** | *signed_integer* >
| **parameter = none**
| **parameter = (** JOB_PARAMETER {**,** JOB_PARAMETER} **)**
| **priority =** < **none** | *signed_integer* >
| **profile =** *profilename*
| **required = none**
| **required = (** JOB_REQUIRED {**,** JOB_REQUIRED} **)**
| **rerun program =** < **none** | *string* >
| **resource = none**
| **resource = (** REQUIREMENT {**,** REQUIREMENT} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **runtime =** *integer*
| **runtime final =** *integer*

| **run program =** < **none** | *string* >
| < **nosuspend** | **suspend** >
| **timeout = none**
| **timeout =** *period* **state** *statename*
| **type =** < **job** | **milestone** | **batch** >
| **group =** *groupname*
| **workdir =** < **none** | *string* >

AJD_ADD_DEL_ITEM:
add [**or alter**] **children = (** JOB_CHILDDEF {**,** JOB_CHILDDEF} **)**
| **add** [**or alter**] **parameter = (** JOB_PARAMETER {**,** JOB_PARAMETER} **)**
| **add** [**or alter**] **required = (** JOB_REQUIRED {**,** JOB_REQUIRED} **)**
| **add** [**or alter**] **resource = (** REQUIREMENT {**,** REQUIREMENT} **)**
| **alter** [**existing**] **children = (** JOB_CHILDDEF {**,** JOB_CHILDDEF} **)**
| **alter** [**existing**] **parameter = (** JOB_PARAMETER {**,** JOB_PARAMETER} **)**
| **alter** [**existing**] **required = (** JOB_REQUIRED {**,** JOB_REQUIRED} **)**
| **alter** [**existing**] **resource = (** REQUIREMENT {**,** REQUIREMENT} **)**
| **delete** [**existing**] **children = (** *folderpath* {**,** *folderpath*} **)**
| **delete** [**existing**] **parameter = (** *parmlist* **)**
| **delete** [**existing**] **required = (** *folderpath* {**,** *folderpath*} **)**
| **delete** [**existing**] **resource = (** *resourcepath* {**,** *resourcepath*} **)**

JOB_CHILDDEF:
JCD_ITEM { JCD_ITEM}

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

JOB_PARAMETER:
parametername < [JP_WITHITEM] [**default =** *string*] | JP_NONDEFWITH >
[**local**] [< **export =** *parametername* | **export = none** >]

JOB_REQUIRED:
JRQ_ITEM { JRQ_ITEM}

REQUIREMENT:
JRD_ITEM { JRD_ITEM}

JCD_ITEM:
alias = < **none** | *aliasname* >
| < **enable** | **disable** >
| *folderpath* **.** *jobname*
| **ignore dependency = none**
| **ignore dependency = (** *dependencyname* {**,** *dependencyname*} **)**
| < **childsuspend** | **suspend** | **nosuspend** >
| **merge mode =** < **nomerge** | **merge local** | **merge global** | **failure** >
| **nicevalue =** < **none** | *signed_integer* >
| **priority =** < **none** | *signed_integer* >
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| < **static** | **dynamic** >

JP_WITHITEM:
import
| **parameter**
| **reference child** *folderpath* **(** *parametername* **)**
| **reference** *folderpath* **(** *parametername* **)**
| **reference resource** *resourcepath* **(** *parametername* **)**
| **result**

JP_NONDEFWITH:
constant = *string*
| JP_AGGFUNCTION **(** *parametername* **)**

JRQ_ITEM:
dependency *dependencyname*
| *folderpath* **.** *jobname*
| **mode =** < **all final** | **job final** >
| **state = none**
| **state = (** JRQ_REQ_STATE {**,** JRQ_REQ_STATE} **)**
| **state = all reachable**
| **state = default**
| **state = unreachable**

| **unresolved =** < **error** | **ignore** | **suspend** | **defer** >

JRD_ITEM:

amount = *integer*
| **expired =** < **none** | *signed_period* >
| < **nokeep** | **keep** | **keep final** >
| **lockmode =** LOCKMODE
| **nosticky**
| *resourcepath*
| **state = none**
| **state = (** *statename* {**,** *statename*} **)**
| **state mapping =** < **none** | *rsmname* >
| **sticky**
[**(** < *identifier* | *folderpath* | *identifier* **,** *folderpath* | *folderpath* **,** *identifier* > **)**]

JP_AGGFUNCTION:

avg
| **count**
| **max**
| **min**
| **sum**

JRQ_REQ_STATE:

statename [< **condition =** *string* | **condition = none** >]

LOCKMODE:

n
| **s**
| **sc**
| **sx**
| **x**

Description

The alter job definition command has two different variants. *Description*

- The first is similar to the create job definition statement and is used to redefine the job definition. All the affected options are overwritten. All the unaddressed options remain as they are.
- The second variant is used to add, edit or delete entries from the lists of children, resource requests, dependencies or parameters.

The options are described in detail in the create job definition command on page 98. This also applies for the options in the child, resource request, dependency and parameter definitions.
If the **existing** keyword is being used, an error is not triggered if the addressed job definition does not exist. The same applies if the **existing**
keyword is being used while the list entries are being deleted or edited.

Output

Output This statement returns a confirmation of a successful operation

alter named resource

Purpose

The purpose of the alter named resource statement is to change its properties. *Purpose*

Syntax

The syntax for the alter named resource statement is *Syntax*

alter [**existing**] **named resource** *resourcepath*
with WITHITEM {**,** WITHITEM}

WITHITEM:
group = *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **parameter = none**
| **parameter = (** PARAMETER {**,** PARAMETER} **)**
| **state profile =** < **none** | *rspname* >

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

PARAMETER:
parametername **constant =** *string*
| *parametername* **local constant** [**=** *string*]
| *parametername* **parameter** [**=** *string*]

Description

Description The alter named resource statement is used to change the properties of the Named Resource. For a detailed description of the options, refer to the description of the create named resource statement on page 116.
If the **existing** keyword has been specified, attempting to modify a non-existent Named Resource will not trigger an error.

Output

Output This statement returns a confirmation of a successful operation

alter resource

Purpose

The purpose of the alter resource statement is to change properties of resources. *Purpose*

Syntax

The syntax for the alter resource statement is *Syntax*

alter [**existing**] RESOURCE_URL
with WITHITEM {**,** WITHITEM}

RESOURCE_URL:
resource *resourcepath* **in** *folderpath*
| **resource** *resourcepath* **in** *serverpath*

WITHITEM:
amount = < **infinite** | *integer* >
| < **online** | **offline** >
| **parameter = none**
| **parameter = (** PARAMETER {**,** PARAMETER} **)**
| **requestable amount =** < **infinite** | *integer* >
| **state =** *statename*
| **touch** [**=** *datetime*]
| **group =** *groupname*

PARAMETER:
parametername **=** < *string* | **default** >

Description

The alter resource statement is used to change the properties of resources. For a detailed description of the options, refer to the description of the create resource statement on page 119. *Description*
If the **existing** keyword has been specified, attempting to modify a non-existent resource will not trigger an error.

Output

This statement returns a confirmation of a successful operation *Output*

alter resource state mapping

Purpose

Purpose The purpose of the alter resource state mapping statement is to change properties of the mapping.

Syntax

Syntax The syntax for the alter resource state mapping statement is

alter [**existing**] **resource state mapping** *mappingname*
with map = (WITHITEM {**,** WITHITEM} **)**

WITHITEM:
statename **maps** < *statename* | **any** > **to** *statename*

Description

Description The alter resource state mapping statement is used to change the properties of the Resource State Mapping. For a detailed description of the options, refer to the description of the create resource state mapping statement on page 123.
If the **existing** keyword has been specified, attempting to modify a non-existent Resource State Mapping will not trigger an error.

Output

Output This statement returns a confirmation of a successful operation

alter resource state profile

Purpose

The purpose of the alter resource state profile statement is to change properties of the specified resource state profile. *Purpose*

Syntax

The syntax for the alter resource state profile statement is *Syntax*

alter [**existing**] **resource state profile** *profilename*
with WITHITEM {**,** WITHITEM}

WITHITEM:
initial state = *statename*
| **state = (** *statename* {**,** *statename*} **)**

Description

The alter resource state profile statement is used to change the properties of the Resource State Profile. For a detailed description of the options, refer to the description of the Resource State Profile statement on page 124. *Description*
If the **existing** keyword has been specified, attempting to modify a non-existent Resource State Profile does not return an error.

Output

This statement returns a confirmation of a successful operation *Output*

alter schedule

Purpose

Purpose The purpose of the alter schedule statement is to change properties of the specified schedule.

Syntax

Syntax The syntax for the alter schedule statement is

alter [**existing**] **schedule** *schedulepath*
with WITHITEM {**,** WITHITEM}

WITHITEM:
< **active** | **inactive** >
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **interval =** < **none** | *intervalname* >
| **time zone =** *string*
| **group =** *groupname*

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

Description

Description The alter schedule statement is used to change the properties of a schedule. For a detailed description of the options for the create schedule statement, refer to page 125.
If the **existing** keyword has been specified, attempting to modify a non-existent schedule will not trigger an error.

Output

This statement returns a confirmation of a successful operation *Output*

alter scheduled event

Purpose

Purpose The purpose of the alter scheduled event statement is to change properties of the specified scheduled event.

Syntax

Syntax The syntax for the alter scheduled event statement is

alter [**existing**] **scheduled event** *schedulepath* **.** *eventname*
with WITHITEM {**,** WITHITEM}

WITHITEM:
< **active** | **inactive** >
| **backlog handling =** < **last** | **all** | **none** >
| **calendar =** < **active** | **inactive** >
| **horizon =** < **none** | *integer* >
| **suspend limit =** < **default** | *period* >
| **group =** *groupname*

Description

Description The alter scheduled event statement is used to change the properties of a specified Scheduled Event. For a detailed description of the options for the create scheduled event statement, refer to page 127.
If the **existing** keyword has been specified, attempting to modify a non-existent Scheduled Event does not return an error.

Output

Output This statement returns a confirmation of a successful operation

alter scope

Purpose

The purpose of the alter scope statement is to change the properties of the specified scope. *Purpose*

Syntax

The syntax for the alter scope statement is *Syntax*

alter [**existing**] < **scope** *serverpath* | **job server** *serverpath* >
with JS_WITHITEM {**,** JS_WITHITEM}

alter [**existing**] **job server**
with < **fatal** | **nonfatal** > **error text =** *string*

alter [**existing**] **job server**
with dynamic PARAMETERS

JS_WITHITEM:
config = none
| **config = (** CONFIGITEM {**,** CONFIGITEM} **)**
| < **enable** | **disable** >
| **error text =** < **none** | *string* >
| **group =** *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **node =** *nodename*
| **parameter = none**
| **parameter = (** PARAMETERITEM {**,** PARAMETERITEM} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

PARAMETERS:
parameter = none
| **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**

CONFIGITEM:
parametername **= none**
| *parametername* **= (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**

| *parametername* **=** < *string* | *number* >

PRIVILEGE:

create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

PARAMETERITEM:

parametername **= dynamic**
| *parametername* **=** < *string* | *number* >

PARAMETERSPEC:

parametername **=** < *string* | *number* >

Description

Description The alter scope command is a user command. This command is used to modify the configuration or other properties of a scope.

Output

Output This statement returns a confirmation of a successful operation

alter server

Purpose

The purpose of the alter server statement is to enable or disable user connections, or to define the trace level. *Purpose*

Syntax

The syntax for the alter server statement is *Syntax*

alter server with < **enable** | **disable** > **connect**

alter server with schedule

alter server with trace level = *integer*

Description

The alter server command can be used to turn the function for connecting to the server on and off. If this function has been disabled, only the user "System" can connect to the server. *Description*
The alter server command is also used to define the logged types of server messages. The following information types are defined:

Type	Meaning
Fatal	A fatal error has occurred. The server is run down.
Error	An error has occurred.
Info	An important informational message that was not written due to an error.
Warning	A warning.
Message	An informative message.
Debug	Messages that can be used for troubleshooting.

Fatal messages, error messages and info messages are always written to the server log file. Warnings are written if the trace level is 1 or higher, normal messages are written if the trace level is 2 or higher. Debug messages provide a lot of output and are returned for trace level 3.

Output

This statement returns a confirmation of a successful operation *Output*

alter session

Purpose

Purpose The purpose of the alter session statement is to specify the used protocol, the session timeout value or the trace level for the specified session.

Syntax

Syntax The syntax for the alter session statement is

alter session [*sid*]
with WITHITEM {**,** WITHITEM}

WITHITEM:
command = (*sdms-command* **)**
| **protocol =** PROTOCOL
| **session =** *string*
| **timeout =** *integer*
| < **trace** | **notrace** >
| **trace level =** *integer*

PROTOCOL:
json
| **line**
| **perl**
| **python**
| **serial**
| **xml**

Description

Description The alter session command can be used to enable and disable the trace. If the trace is enabled, all the issued commands are logged in the log file. A communication protocol can also be selected. An overview of the currently defined protocols is shown in the table below.

Protokoll	Meaning
Line	Plain ASCII output
Perl	The output is offered as a Perl structure that can be easily evaluated by the Perl script using eval.
Python	Like Perl, but this is a Python structure.
Serial	Serialized Java objects.
Xml	Outputs an xml structure.

The timeout parameter for the session can be set as a last resort. A timeout of 0 means that no timeout is active. Any number greater than 0 indicates the number of seconds after which a session is automatically disconnected.

Output

This statement returns a confirmation of a successful operation *Output*

alter trigger

Purpose

Purpose The purpose of the alter trigger statement is to change properties of the specified trigger.

Syntax

Syntax The syntax for the alter trigger statement is

alter [**existing**] **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
with WITHITEM {**,** WITHITEM}

TRIGGEROBJECT:
resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

WITHITEM:
< **active** | **inactive** >
| < **nowarn** | **warn** >
| **limit state =** < **none** | *statename* >
| < **nomaster** | **master** >
| **rerun**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **state = none**
| **state = (** < *statename* {**,** *statename*} |
CT_RSCSTATUSITEM {**,** CT_RSCSTATUSITEM} > **)**
| **submit after** *folderpath*
| **submit** *folderpath*
| **submitcount =** *integer*
| < **nosuspend** | **suspend** >
| [**type =**] CT_TRIGGERTYPE
| **group =** *groupname*

CT_EVENT:
< **create** | **change** | **delete** >

CT_RSCSTATUSITEM:

< *statename* **any** | *statename statename* | **any** *statename* >

CT_TRIGGERTYPE:
before final
| **immediate local**
| **immediate merge**
| **warning**

Description

Description

The alter trigger statement is used to change the properties of a defined trigger. If the **existing** keyword has been specified, changing an existing trigger will not return an error.
For a detailed description of these options, refer to the create trigger statement on page 132.

Output

Output

This statement returns a confirmation of a successful operation

alter user

Purpose

Purpose The purpose of the alter user statement is to change properties of the specified user.

Syntax

Syntax The syntax for the alter user statement is

alter [**existing**] **user** *username*
with WITHITEM {**,** WITHITEM}

alter [**existing**] **user** *username*
ADD_DELITEM {**,** ADD_DELITEM}

WITHITEM:
default group = *groupname*
| < **enable** | **disable** >
| **group = (** *groupname* {**,** *groupname*} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

ADD_DELITEM:
< **add** | **delete** > **group = (** *groupname* {**,** *groupname*} **)**

Description

Description The alter user statement is used to change the properties of a defined user. If the **existing** keyword has been specified, attempting to modify a non-existent user will not trigger an error.
For a detailed description of these options, refer to the create user statement on page 140.
The second variant of the statement is used to delete or add the user from or to the specified groups.

Output

Output This statement returns a confirmation of a successful operation

4. connect commands

connect

Purpose

Purpose The purpose of the connect statement is to authenticate a user to the server.

Syntax

Syntax The syntax for the connect statement is

connect *username* **identified by** *string* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
command = (*sdms-command* **)**
| **protocol =** PROTOCOL
| **session =** *string*
| **timeout =** *integer*
| < **trace** | **notrace** >
| **trace level =** *integer*

PROTOCOL:
json
| **line**
| **perl**
| **python**
| **serial**
| **xml**

Description

Description The connect command is used to authenticate the connected process on the server to. A communication protocol can be optionally specified. The default protocol is **line**.
The selected protocol defines the output format. All protocols except for **serial** return ASCII output. The protocol **serial**
returns a serialized Java object.
An executable command can also be returned when the connection is established. In this case, the output of the accompanying command is used as the output for the connect command. If the command fails, but the connect was successful, the connection remains active.
An example for all protocols except the **serial** protocol is given below.

The line protocol The line protocol only returns an ASCII text as the result from a command.

```
connect donald identified by 'duck' with protocol = line;

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

The XML protocol The XML protocol returns an XML structure as the result from a command.

```
connect donald identified by 'duck' with protocol = xml;
<OUTPUT>
<DATA>
<TITLE>Connect</TITLE>
<RECORD>
<CONNECT_TIME>19 Jan 2005 11:15:16 GMT</CONNECT_TIME></RECORD>
</DATA>
<FEEDBACK>Connected</FEEDBACK>
</OUTPUT>
```

The python protocol The python protocol returns a Python structure that can be valuated using the Python eval function.

```
connect donald identified by 'duck' with protocol = python;
{
'DATA' :
{
'TITLE' : 'Connect',
'DESC' : [
'CONNECT_TIME'
],
'RECORD' : {
'CONNECT_TIME' : '19 Jan 2005 11:16:08 GMT'}
}
,'FEEDBACK' : 'Connected'
}
```

The perl protocol The perl protocol returns a Perl structure that can be valuated using the Perl eval function.

```
connect donald identified by 'duck' with protocol = perl;
{
'DATA' =>
{
```

```
'TITLE' => 'Connect',
'DESC' => [
'CONNECT_TIME'
],
'RECORD' => {
'CONNECT_TIME' => '19 Jan 2005 11:19:19 GMT'}
}
,'FEEDBACK' => 'Connected'
}
```

Output

Output This statement returns a confirmation of a successful operation

5. copy commands

copy folder

Purpose

Purpose The purpose of the copy folder statement is to copy a folder including all contents to some other place in the folder hierarchy.

Syntax

Syntax The syntax for the copy folder statement is

copy FOLDER_OR_JOB {**,** FOLDER_OR_JOB} **to** *folderpath*

copy FOLDER_OR_JOB {**,** FOLDER_OR_JOB} **to** *foldername*

FOLDER_OR_JOB:
[< **folder** *folderpath* | **job definition** *folderpath* >]

Description

Description If a folder has been copied, every object in the folder is copied as well. If there are any relationships between objects that were copied as the result of a folder copy operation (e.g. dependencies, children, triggers, etc.), these are changed accordingly and mapped to the resulting objects from the copy.
For example, if a folder SYSTEM.X.F containing two jobs A and B, and with SYSTEM.X.F.B dependent upon SYSTEM.X.F.A, is copied to the folder SYSTEM.Y, the newly created job SYSTEM.Y.F.B will be dependent upon the newly created job SYSTEM.Y.F.A.
Note that if the jobs were copied using a copy job definition command, the new job SYSTEM.Y.F.B would still be dependent upon SYSTEM.X.F.A. This may not correspond to the user's view.

Output

Output This statement returns a confirmation of a successful operation

copy named resource

Purpose

The purpose of the copy named resource statement is to copy a named resource into another category. *Purpose*

Syntax

The syntax for the copy named resource statement is *Syntax*

copy named resource *resourcepath* **to** *resourcepath*

copy named resource *resourcepath* **to** *resourcename*

Description

The copy named resource command is used to save a copy of a Named Resource or an entire category. *Description*
If the specified target resourcepath already exists as a category, a Named Resource or category with the same name as the source object is created within this category.
If the specified target resourcepath already exists as a Named Resource, this is regarded as an error.

Output

This statement returns a confirmation of a successful operation *Output*

copy scope

Purpose

Purpose The purpose of the copy scope statement is to copy a scope including all contents to some other place within the scope hierarchy.

Syntax

Syntax The syntax for the copy scope statement is

copy < **scope** *serverpath* | **job server** *serverpath* > **to** *serverpath*

copy < **scope** *serverpath* | **job server** *serverpath* > **to** *scopename*

Description

Description The copy named resource command is used to save a copy of entire scopes. This copy also includes the resource and parameter definitions.
If the specified target servicepath already exists as a scope, a scope with the same name as the source object is created within this category.
If the specified target serverpath already exists as a job server, this is regarded as an error.
Since a job server is only regarded as a special type of scope, it is possible to copy job servers using this command. In this case, this command is identical to the copy job server command.

Output

Output This statement returns a confirmation of a successful operation

6. create commands

create comment

Purpose

Purpose The purpose of the create comment statement is to store a comment for the specified object.

Syntax

Syntax The syntax for the create comment statement is

create [**or alter**] **comment on** OBJECTURL
with CC_WITHITEM

OBJECTURL:
distribution *distributionname* **for pool** *resourcepath* **in** *serverpath*
| **environment** *environmentname*
| **exit state definition** *statename*
| **exit state mapping** *mappingname*
| **exit state profile** *profilename*
| **event** *eventname*
| **resource** *resourcepath* **in** *folderpath*
| **folder** *folderpath*
| **footprint** *footprintname*
| **group** *groupname*
| **interval** *intervalname*
| **job definition** *folderpath*
| **job** *jobid*
| **named resource** *resourcepath*
| **parameter** *parametername* **of** PARAM_LOC
| **resource state definition** *statename*
| **resource state mapping** *mappingname*
| **resource state profile** *profilename*
| **scheduled event** *schedulepath* **.** *eventname*
| **schedule** *schedulepath*
| **resource** *resourcepath* **in** *serverpath*
| < **scope** *serverpath* | **job server** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

CC_WITHITEM:
CC_TEXTITEM {, CC_TEXTITEM}

| **url =** *string*

PARAM_LOC:

folder *folderpath*
| **job definition** *folderpath*
| < **scope** *serverpath* | **job server** *serverpath* >

TRIGGEROBJECT:

resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

CC_TEXTITEM:

tag = < **none** | *string* > **, text =** *string*
| **text =** *string*

Description

Description

The create comment statement is used to create the condensed description or the URL of the description for the object to be commented on.
The optional keyword **or alter** is used to update the comment (if one exists). If it is not specified, the presence of a comment will trigger an error.

Output

Output

This statement returns a confirmation of a successful operation

create environment

Purpose

Purpose The purpose of the create environment statement is to define a set of static named resources which are needed in the scope a job wants to run.

Syntax

Syntax The syntax for the create environment statement is

create [**or alter**] **environment** *environmentname* [**with** ENV_WITH_ITEM]

ENV_WITH_ITEM:
resource = none
| **resource = (** ENV_RESOURCE {**,** ENV_RESOURCE} **)**

ENV_RESOURCE:
resourcepath [< **condition =** *string* | **condition = none** >]

Description

Description The create environment statement is used to define a series of Static Resource Requests which describe the requisite environment that a job needs. Since the environments cannot be created by ordinary users, and jobs have to describe the environment that they require to run, environments can be used to force jobs to use a specific job server.

Resources The Resources clause is used to specify the Required (Static) Resources. Specified resources that are not static will trigger an error. Since only static resources are specified, no further information is required. It is permissible to specify an empty environment (an environment without resource requests). This is not advisable, though, because it means a loss of control.

Output

Output This statement returns a confirmation of a successful operation

create event

Purpose

Purpose

The purpose of the create event statement is to define an action which can be executed by the time scheduling engine.

Syntax

Syntax

The syntax for the create event statement is

create [**or alter**] **event** *eventname*
with EVENT_WITHITEM {**,** EVENT_WITHITEM}

EVENT_WITHITEM:
action =
submit *folderpath* [**with parameter = (** PARAM {**,** PARAM} **)**]
| **group =** *groupname*

PARAM:
parametername **=** < *string* | *number* >

Description

Description

The create event statement is used to define an action that can be scheduled by the Time Scheduling module. The defined action is the submission of a master submittable job or batch.

action The submit part of the statement is a restricted variant of the submit command (see page 354).

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

Output

Output

This statement returns a confirmation of a successful operation

create exit state definition

Purpose

Purpose The purpose of the create exit state definition statement is to create a symbolic name for the state of a job.

Syntax

Syntax The syntax for the create exit state definition statement is

create [**or alter**] **exit state definition** *statename*

Description

Description The create exit state definition statement is used to create a symbolic name for the Exit State of a job, milestone or batch.
The optional keyword **or alter** is used to prevent error messages from being triggered and the current transaction from being aborted if an Exit State Definition already exists. This is particularly useful in conjunction with multicommands. If it is not specified, the existence of an Exit State Definition with the specified name will trigger an error.

Output

Output This statement returns a confirmation of a successful operation

Example

Example In the following examples, symbolic names have been created for Job States.

```
create exit state definition success;
create exit state definition error;
create exit state definition reached;
create exit state definition warning;
create exit state definition wait;
create exit state definition skip;
create exit state definition unreachable;
```

create exit state mapping

Purpose

The purpose of the create exit state mapping statement is to create a mapping between the numerical exit code of a process and a symbolic exit state. *Purpose*

Syntax

The syntax for the create exit state mapping statement is *Syntax*

create [**or alter**] **exit state mapping** *mappingname*
with map = (*statename* { **,** *signed_integer* **,** *statename* } **)**

Description

The create exit state mapping statement defines the mapping of Exit Codes to logical Exit States. The simplest form of this statement only specifies one Exit State. This means that the job automatically reaches this Exit State after it has finished regardless of its Exit Code. More complex definitions specify more than one Exit State and at least one delimitation. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

Example

The example below shows a relatively simple, yet realistic mapping of Exit Codes to logical Exit States. *Example*
The statement

```
create exit state mapping example1
with map = (   error,
    0, success,
    1, warning,
    4, error);
```

defines the following mapping:

Exit Code Range from	Exit Code Range to	Resultant Exit State
$-\infty$	-1	error
0	0	success
1	3	warning
4	∞	error

create exit state profile

Purpose

Purpose The purpose of the create exit state profile statement is to define a set of valid exit states.

Syntax

Syntax The syntax for the create exit state profile statement is

create [**or alter**] **exit state profile** *profilename*
with WITHITEM {**,** WITHITEM}

WITHITEM:
default mapping = < **none** | *mappingname* >
| **force**
| **state = (** ESP_STATE {**,** ESP_STATE} **)**

ESP_STATE:
statename < **final** | **restartable** | **pending** > [OPTION { OPTION}]

OPTION:
< **unreachable** | **broken** | **batch default** | **dependency default** >

Description

Description The create exit state profile statement is used to define a quantity of valid Exit States for a job, milestone or batch.

default mapping With the **default mapping** clause it is possible to define which Exit State Mapping is to be used if no other mapping has been specified. This makes it considerably easier to create jobs.

force While an Exit State Profile is being created, the force option has no effect and is ignored. If **or alter** is specified and the Exit State Profile that you want to create already exists, the force option delays the integrity check until later.

state The **state** clause defines which Exit State Profiles are valid within this definition. Each Exit State Definition must be classified as being **final**, **restartable** or **pending**. If a job has reached the **final** state it can no longer be started, which means that the state can no longer change. If a job has reached the **restartable** state,

it can be started again. This means that the state of such a job can change as well. **pending** means that a job cannot be restarted, but it is not final either. The state must be be set externally.
The order in which the Exit States are defined is relevant. The first specified Exit State has the highest preference, while the most recently specified Exit State has the lowest preference. Normally, **final** states are specified later than **restartable** states. A state's preference is used to decide which state is visible when several different Exit States of children are merged.
Just one Exit State can be declared as being an **unreachable** state. This means that a job, batch or milestone with this profile is mapped to the specified state as soon as it has become unreachable. This Exit State must be **final**.
A maximum of one Exit State within a profile can be designated as being a **broken** state. This means that a job will reach this state as soon as it has switched to the **error** or **broken_finished** state. This can be handled using a trigger. The Exit State that is defined as being a **broken** state must be **restartable**.
A maximum of one state can be declared as being a **batch default** state. An empty batch assumes this status. This allows for an explicit deviation from the standard behaviour. If no status is designated as being **batch default**, an empty batch will automatically assume the final status with the lowest preference that is not designated as being **unreachable**. If such a status does not exist, the **unreachable** state is also considered a candidate.
Any number of Final States can be designated as **dependency default** states. Dependencies that define a default dependency are fulfilled if the required job assumes one of the states designated as **dependency default**.

Output

This statement returns a confirmation of a successful operation *Output*

Example

These examples show how the Exit State Profiles `example_1` and `example_2` are created. *Example*
In the first, very simple example, the Exit State of `success` is to be a Final State.

```
create exit state profile example_1
with
    state = ( success final );
```

In the second example, the Exit State `failure` is defined as being restartable. This state has a higher priority than the (final) state `success` and must therefore be listed as the first state.

```
create exit state profile example_2
with
    state = ( failure restartable,
```

```
            success final
    );
```

create folder

Purpose

The purpose of the create folder statement is to create a container for job definitions and/or other folders. *Purpose*

Syntax

The syntax for the create folder statement is *Syntax*

create [**or alter**] **folder** *folderpath* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:

environment = < **none** | *environmentname* >
| **group =** *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **parameter = none**
| **parameter = (** *parametername* **=** *string* {**,** *parametername* **=** *string*} **)**

PRIVILEGE:

create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

Description

This command creates a folder and has the following options: *Description*

environment option If an environment has been assigned to a folder, every job in the folder and its subfolders will inherit all the Resource Requests from the Environment Definition.

group option The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

parameter option The parameter option can be used to define key/value pairs for the folder. The complete list of parameters must be specified within a command.

inherit grant option The **inherit grants** clause allows you to define which privileges are to be inherited through the hierarchy. If this clause is not specified, all privileges are inherited by default.

Output

Output This statement returns a confirmation of a successful operation

create footprint

Purpose

The purpose of the create footprint statement is to create a set of often used system resource requirements. *Purpose*

Syntax

The syntax for the create footprint statement is *Syntax*

create [**or alter**] **footprint** *footprintname*
with resource = (REQUIREMENT {**,** REQUIREMENT} **)**

REQUIREMENT:
ITEM { ITEM}

ITEM:
amount = *integer*
| < **nokeep** | **keep** | **keep final** >
| *resourcepath*

Description

The create footprint command creates a set of Resource Requests which can be reused. The Required Resources are all System Resources. The Required Resources are described by their names, a set with zero by default, and optionally a keep option. *Description*

keep The keep option in a Resource Request defines the time when the resource is released. The keep option is valid for both System and Synchronizing Resources. There are three possible values. Their meanings are explained in the table below:

Value	Meaning
nokeep	The resource is released at the end of the job. This is the default behaviour.
keep	The resource is released as soon as the job has reached the Final State.
keep final	The resource is released when the job and all its children are final.

amount The amount option is only valid with requests for Named Resources of the type ″System″ or ″Synchronizing″. The amount in a Resource Request expresses how many units of the Required Resource are allocated.

Output

Output This statement returns a confirmation of a successful operation

create group

Purpose

Purpose

The purpose of the create group statement is to create an object to which privileges can be granted.

Syntax

Syntax

The syntax for the create group statement is

create [**or alter**] **group** *groupname* [**with** WITHITEM]

WITHITEM:
user = none
| **user = (** *username* {**,** *username*} **)**

Description

Description

The create group statement is used to create a group. If the **or alter** keyword has been specified, an already existing group is changed. Otherwise, an already existing group is considered an error.

user The **user** clause is used to specify which users are group members.

Output

Output

This statement returns a confirmation of a successful operation

create interval

Purpose

Purpose The purpose of the create interval statement is to define a periodic or aperiodic pattern at which events can, must not, be triggered.

Syntax

Syntax The syntax for the create interval statement is

create [**or alter**] **interval** *intervalname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
base = < **none** | *period* >
| **duration =** < **none** | *period* >
| **embedded =** < **none** | *intervalname* >
| **endtime =** < **none** | *datetime* >
| **filter = none**
| **filter = (** *intervalname* {**,** *intervalname*} **)**
| < **noinverse** | **inverse** >
| **selection = none**
| **selection = (** IVAL_SELITEM {**,** IVAL_SELITEM} **)**
| **starttime =** < **none** | *datetime* >
| **synctime =** *datetime*
| **group =** *groupname*

IVAL_SELITEM:
< *signed_integer* | *datetime* | *datetime* **-** *datetime* >

Description

Description The intervals are the core of the Time Scheduling. They can regarded as block patterns. These patterns can be periodic or non-periodic. Within a **period** (**Base**) which, in the case of a non-periodic interval, has a length infinity (∞), there are blocks of a predetermined length **Duration**. The last block may be incomplete if the period length is not an integer multiple of the duration is. The duration can also have a length ∞
. This means that the blocks have the same length as the periods.
It is not necessary for all of the blocks to be actually present. You can choose which blocks are present. This **choice** can be made by specifying the block number relative to the beginning or end of a period ($1, 2$,3 or $-1, -2, -3$) or by stating "from - to" (all days between 3.4. and 7.6.).

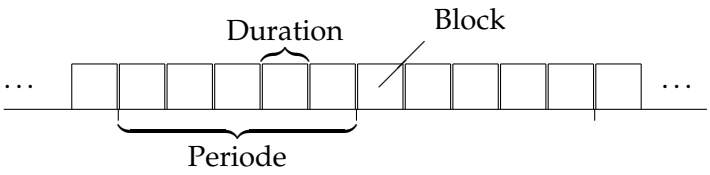

Figure 6.1.: How periods and blocks are displayed

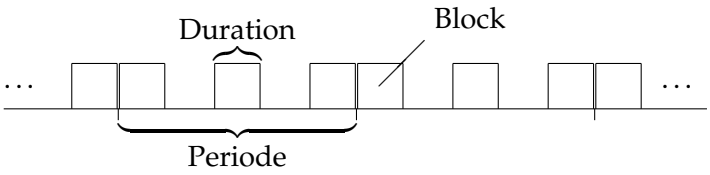

Figure 6.2.: A more complex pattern

This results in complex patterns as shown in Figure 6.2.

The selection is 1-based, i.e. the first block has the number 1. The last block is addressed with the number -1. This means that a block 0 does not exist.

Essentially, an interval can be described using the following parameters: Base frequency (period length), duration and selection. Since an interval does not necessarily always have to be valid, a start and end time can still be specified.

Infinite intervals With a non-periodic interval without a duration (infinity), the start time plays a special role: it then defines the only positive edge of this interval. Similarly, an end time defines the only negative edge.

When a selection is made, this respectively results in blocks being created. The selection "-0315T18:40" creates a block from 18:40 to 18:41 every year on March 15.

Selecting blocks using the position (first, second, etc.) is, of course, nonsense. This is also ignored for infinite intervals.

Inverse If, for example, the time between Christmas and New Year has been positively defined for a particular purpose, at the moment there is no way to easily define the complementary time. In this example this is not a serious problem, but with more complex patterns this incapability will result in complex and error-prone dual definitions.

For this reason, an **Inverse** flag has been implemented which causes the specified selection list to be interpreted complementarily, i.e. only those blocks that would not have been chosen without a set invert flag are selected. In the case of the last working day of the month, the inverse flag is set on all working days except for the last working day of that month.

Filter The selection of blocks can be restricted even further. For example, if you have defined an interval "day of the month" (i.e. the base is one month, the duration is one day) and then selected the second block, such an interval would have a block on the respective second day of a month. If you want to define this only for the odd months (January, March, May, etc.), that would not be possible without a filter function because of the leap years.
The solution to the problem is to define a further interval (month of the year) with the selection $1, 3, 5, 7, 9, 11$. This interval is then specified as a filter for the first interval.
Here, the first interval only shows a block if the second interval also shows a block at that "time".
If several intervals have been specified as a filter, it is sufficient for one of these intervals to have a block at the required time (OR). To map an AND relationship between the filter intervals, the filter intervals are created as a chain (A filters B C filters, etc.). The order of the filters is not important.

Embedded Unfortunately, the world is not always so simple. In particular, it is not inconsequential whether you first perform an operation and then make a selection, or if you have to choose first and then perform the operation. In other words, there is a big difference if you
talking about the last day of the month - if this is a working day - or about the last working day of the month.
We obviously also want to include this possibility for making a differentiation in our model. An **embedding** functionality has been implemented for this purpose.
Here, we begin by taking over all the parameters for the embedded interval. This is followed by an evaluation of the selection list. Although it is allowed, selecting a "from - to" period is obviously senseless since this functionality can also be achieved with simple multiplication. Much more interesting is the possibility of making a relative selection. If the working days in a month are embedded and then the day -1 is selected, for instance, overall we now have an interval that defines the last working day of each month. If, on the other hand, the interval with the working days in a month is multiplied by an interval that returns the last day of a month, we will only get a hit if the last day of the month is a working day.
Embedding can therefore also be understood as follows: When selecting the blocks, not *all* of the embedded blocks are considered (and above all counted), but only the *active* blocks.

Synchronisation What have still not been taken into consideration are those situations involving multiple single periods. A period of 40 days, for example, could have its rising edge at midnight (00:00) on any day. That is why a synchronisation time (**synctime**) has been implemented which selects the earliest edge that is $\geq$ this point in time. If no such time has been explicitly specified, the date when the definition was created (*create*) is used.

Fundamentally, the first block of a period initially starts at its beginning. In cases where this is not possible (period = ∞, duration $>$ period, Period XOR Duration have the unit "week"), the beginning of the period is used as the synchronisation time. If this is not possible either (period = ∞), the normal synchronisation time is used. The result of this approach is that the *first* block of a period may be incomplete as well (and is then *never* active).

Output

This statement returns a confirmation of a successful operation *Output*

create job definition

Purpose

Purpose The purpose of the create job definition statement is to create a scheduling entity object which can be submitted, standalone or as part of a larger hierarchy.

Syntax

Syntax The syntax for the create job definition statement is

create [**or alter**] **job definition** *folderpath* **.** *jobname*
with WITHITEM {**,** WITHITEM}

WITHITEM:

children = none
| **children = (** JOB_CHILDDEF {**,** JOB_CHILDDEF} **)**
| **dependency mode =** < **all** | **any** >
| **environment =** *environmentname*
| **errlog =** < **none** | *filespec* [< **notrunc** | **trunc** >] >
| **footprint =** < **none** | *footprintname* >
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **kill program =** < **none** | *string* >
| **logfile =** < **none** | *filespec* [< **notrunc** | **trunc** >] >
| **mapping =** < **none** | *mappingname* >
| < **nomaster** | **master** >
| **nicevalue =** < **none** | *signed_integer* >
| **parameter = none**
| **parameter = (** JOB_PARAMETER {**,** JOB_PARAMETER} **)**
| **priority =** < **none** | *signed_integer* >
| **profile =** *profilename*
| **required = none**
| **required = (** JOB_REQUIRED {**,** JOB_REQUIRED} **)**
| **rerun program =** < **none** | *string* >
| **resource = none**
| **resource = (** REQUIREMENT {**,** REQUIREMENT} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **runtime =** *integer*
| **runtime final =** *integer*
| **run program =** < **none** | *string* >
| < **nosuspend** | **suspend** >
| **timeout = none**

| **timeout =** *period* **state** *statename*
| **type =** < **job** | **milestone** | **batch** >
| **group =** *groupname*
| **workdir =** < **none** | *string* >

JOB_CHILDDEF:
JCD_ITEM { JCD_ITEM}

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

JOB_PARAMETER:
parametername < [JP_WITHITEM] [**default =** *string*] | JP_NONDEFWITH >
[**local**] [< **export =** *parametername* | **export = none** >]

JOB_REQUIRED:
JRQ_ITEM { JRQ_ITEM}

REQUIREMENT:
JRD_ITEM { JRD_ITEM}

JCD_ITEM:
alias = < **none** | *aliasname* >
| < **enable** | **disable** >
| *folderpath* **.** *jobname*
| **ignore dependency = none**
| **ignore dependency = (** *dependencyname* {**,** *dependencyname*} **)**
| < **childsuspend** | **suspend** | **nosuspend** >
| **merge mode =** < **nomerge** | **merge local** | **merge global** | **failure** >
| **nicevalue =** < **none** | *signed_integer* >
| **priority =** < **none** | *signed_integer* >

| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| < **static** | **dynamic** >

JP_WITHITEM:

import
| **parameter**
| **reference child** *folderpath* **(** *parametername* **)**
| **reference** *folderpath* **(** *parametername* **)**
| **reference resource** *resourcepath* **(** *parametername* **)**
| **result**

JP_NONDEFWITH:

constant = *string*
| JP_AGGFUNCTION **(** *parametername* **)**

JRQ_ITEM:

dependency *dependencyname*
| *folderpath* **.** *jobname*
| **mode =** < **all final** | **job final** >
| **state = none**
| **state = (** JRQ_REQ_STATE {**,** JRQ_REQ_STATE} **)**
| **state = all reachable**
| **state = default**
| **state = unreachable**
| **unresolved =** < **error** | **ignore** | **suspend** | **defer** >

JRD_ITEM:

amount = *integer*
| **expired =** < **none** | *signed_period* >
| < **nokeep** | **keep** | **keep final** >
| **lockmode =** LOCKMODE
| **nosticky**
| *resourcepath*
| **state = none**
| **state = (** *statename* {**,** *statename*} **)**
| **state mapping =** < **none** | *rsmname* >
| **sticky**
[**(** < *identifier* | *folderpath* | *identifier* **,** *folderpath* | *folderpath* **,** *identifier* > **)**]

JP_AGGFUNCTION:

avg
| **count**
| **max**
| **min**
| **sum**

JRQ_REQ_STATE:
statename [< **condition =** *string* | **condition = none** >]

LOCKMODE:
n
| **s**
| **sc**
| **sx**
| **x**

Description

This command creates or optionally modifies job, batch or milestone definitions. Since jobs, batches and milestones have a lot in common, in the following we have mainly used the general technical term ”Scheduling Entity” whenever the behaviour is the same for all three types of job definitions. The expressions ”job”, ”batch” and ”milestone” are used for Scheduling Entities of the corresponding type Job, Milestone and Batch. *Description*
If the ”or alter” modifier is being used, the command (if a Scheduling Entity of the same name already exists) changes it according to the specified options.

aging The aging describes how quickly the priority is upgraded.

children The Children section of a job definition statement defines a list of child objects and is used to build up a hierarchy that enables the modelling of complex job structures.
Whenever a Scheduling Entity is submitted, all the static children are recursively submitted.
In addition, children that are not static can be submitted during the execution be a Running Job or Trigger.
The children are then specified using a comma-separated list of Scheduling Entity path names and additional properties.
The properties of the Child Definitions are described below:

ALIAS This option allows the implementation of the submitted jobs to be kept independent of the folder structure, and it will function regardless of whether objects are moved within the folder structure.

The alias for a Child Definition is only used when jobs submit dynamic children.

IGNORE DEPENDENCY Dependencies of parent jobs are normally inherited by their children. In some rare situations this is undesirable. In this case the **ignore dependency** option can be used to ignore such dependencies.

MERGE MODE A single Scheduling Entity can be used as a child of more than one Parent Scheduling Entity. If two or more such parents are part of a Master Run, the same children are repeatedly instantiated within this Master Run. This is not always a desirable situation. Setting the Merge Mode controls how the system handles this scenario.

The following table gives an overview of the possible Merge Modes and their meanings:

merge mode	Description
nomerge	A duplicate instance of the Scheduling Entity is created. This is the default behaviour.
merge global	A duplicate instance is not created. A link is created between the Parent Submitted Entity and the already existing Child Submitted Entity.
merge local	Like Merge Global, but only Submitted Entities that were created in a single submit are merged.
failure	The submit attempting to create a duplicate Submitted Entity fails.

NICEVALUE The nicevalue defines an offset of the priority used to calculate the priorities of the child and its children. Values between -100 and 100 are permitted.

PRIORITY The specified priority in a Child Definition overwrites the priority of the Child Scheduling Entity Definition. Values between 0 (high priority) and 100 (low priority) are permitted.

TRANSLATION Setting the Exit State Translation for a child results in the Exit State of the child being translated to an Exit State which is merged in the resultant Exit State of the Parent Submitted Entity.

If no translation is specified, a Child State that is not at the same time a valid Parent State is ignored.

If a translation has been specified, all the Child States have to translated to a valid Parent State.

SUSPEND CLAUSE The child suspend clause defines whether a new Submitted Job is suspended in the context of this Child Definition.

The table below shows the possible values and their meaning regarding the suspend clause:

suspend clause	Description
suspend	The child is suspended regardless of the value of the suspend flag specified in the Child Scheduling Entities.
nosuspend	The child is not suspended regardless of the value of the suspend flag specified in the Child Scheduling Entities Definition.
childsuspend	The child is suspended if the suspended flag has been set in the Child Scheduling Entity.

If **suspend** has been specified, a resume clause can optionally be given as well which triggers an automatic resume at the specified time or at the end of the specified interval.
The submit time is taken as the reference for partially qualified points in time. T16:00 means, therefore, that if the submit time 15:00 has been set, the job will start after about an hour. If the submit time is later than 16:00, however, the job will wait until the next day.

DYNAMIC CLAUSE The child dynamic clause defines whether the child is always automatically submitted by the system when the parent is submitted as well.
Dynamic children are used by Running Jobs in the context of Trigger Definitions and programmatic submits. To be able to submit a child, this child must be defined as a dynamic child.
The table below shows the possible values in the dynamic clause and their meanings.

dynamic clause	Description
static	The child is automatically submitted with the parent.
dynamic	The child is not automatically submitted with the parent.

Milestones use different semantics for their children. Whenever a Scheduling Entity is dynamically submitted in a Master Run that is also a child of a milestone in the same Master Run, the Submitted Scheduling Entity is bound to this milestone as a child. This means that a milestone can only be final if its dependencies have been fulfilled and all its children are final. In other words, a Milestone collects child instances that are dynamically submitted by other Submitted Entities and waits until these Submitted Entities have finished. For this to function correctly, a dependency of the Submitted Scheduling Entity should be defined.

dependency mode The dependency mode defines which Required Submitted Entities have to achieve a Final State before the dependent Submitted Entity can exit the 'Dependency Wait' System State.
The table below shows the possible Dependency Modes and their meanings.

dependency mode	Description
all	The Submitted Entity exits the Dependency Wait State after all the dependencies have been fulfilled.
any	The Submitted Entity exits the Dependency Wait State after at least one dependency has been fulfilled.

environment Each job has to define which environment is needed to execute the job.
The job can only be executed by job servers that fulfil all the Static Resource requirements listed in the Environment Definition.
The environment option only applies for jobs.

errlog The errlog option defines the file where error outputs (stderr) from the process to be executed are written.
If the file name is relative, the file is created relative to the working directory of the job.
This option is only valid for jobs.

footprint Footprints are sets of requirements for System Resources. If several jobs are defined with similar requirements, this is made that much easier by using footprints.
The job can only be executed by job servers that fulfil all the Static Resource requirements listed in the Footprint Definition.
The footprint option only applies for jobs.

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

inherit grant The **inherit grants** clause allows you to define which privileges are to be inherited through the hierarchy. If this clause is not specified, all privileges are inherited by default.

kill program This option is used to create the possibility for prematurely terminating running processes from within the Scheduling System.
Usually, the kill program contains the PID of the Running Job as a parameter (e.g. `kill -9 ${PID}`).
For details about command line parsing, variants and parameter substitutions, refer to the ”run program” option on page 114.

logfile The logfile option defines the file where the standard output (stdout) from the process to be executed is written.
If the file name is relative, the file is created relative to the working directory of the job.
This option is only valid for jobs.

mapping The mapping option defines the Exit State Mapping that is used to translate operating system Exit Codes of an executable program to an Exit State. If a job does not have a mapping, the default Exit State Mapping of the job's Exit State Profile is used.
For a detailed description of the Exit State Mapping, refer to the "create exit state mapping" command on page 85.

nicevalue The nicevalue option defines a correction that is used for the calculation of the priorities for the job and its children. Values between -100 and 100 are permitted.

parameter The parameters section defines which parameters and input values are required by a job and how the job exchanges data with other jobs and the scheduling system.
The parameters can be used in the specification of the Run program, Rerun program, Kill program, working directory, log file and error log file, as well as in triggers and Dependency Conditions.
A job can also query or set parameters at runtime. Variables that that have been defined at runtime and not by the job definition are only visible to the job itself and cannot be referenced. The same is also true, of course, for all variables that are defined as **local** as well as for the system variables mentioned below.
Occasionally, however, it is necessary to make one or more of the (e.g.) system variables known externally. This can be easily done by means of a small trick. If the value of a parameter contains a character string in the form `$something` (i.e. the characters $ followed by a name), this is interpreted as being the name of a variable, and an attempt is made to resolve this variable *in the scope of the object that delivered the original value for the parameter*.
This is how, for example, a job `SYSTEM.A` can define a constant called `MYJOBNAME` with `$JOBNAME` as its content. If the constant `MYJOBNAME` is now addressed from outside the system via a reference, the delivered result is the value `SYSTEM.A`.
A number of system variables are always defined for each job. These are set by the system and can be read by the job.

These system variables are:

Name	Description
JOBID	Submitted entity id for the job
MASTERID	Submitted entity id for the Master Job or Batch
KEY	”Password” of the job for connecting to the scheduling system as a job with ”JOBID”
PID	The operating system process id of the job. This parameter is only set for Kill programs.
LOGFILE	Name of the log file (stdout)
ERRORLOG	Name of the error log file (stderr)
SDMSHOST	Host name of the scheduling server
SDMSPORT	Listen port of the scheduling server
JOBNAME	Name of the job
JOBTAG	Child tag for the job is given if the job is being dynamically submitted
TRIGGERNAME	Name of the trigger
TRIGGERTYPE	Type of trigger (JOB_DEFINITION or NAMED_RESOURCE)
TRIGGERBASE	Name of the triggering object that activates the trigger
TRIGGERBASEID	ID of the triggering Object Definition that activates the trigger
TRIGGERBASEJOBID	ID of the triggering object that activates the trigger
TRIGGERORIGIN	Name of the triggering object that defines the trigger
TRIGGERORIGINID	ID of the triggering Object Definition that defines the trigger
TRIGGERORIGINJOBID	ID of the triggering object that defines the trigger
TRIGGERREASON	Name of the triggering object that directly or indirectly activates the trigger
TRIGGERREASONID	ID of the triggering Object Definition that directly or indirectly activates the trigger
TRIGGERREASONJOBID	ID of the triggering object that directly or indirectly activates the trigger
TRIGGERSEQNO	Number of times the trigger was activated
TRIGGEROLDSTATE	The old state of the object caused by the trigger for Resource Trigger
TRIGGERNEWSTATE	(New) status of the object that causes the trigger to be activated
SUBMITTIME	Submit time
STARTTIME	Start time
EXPRUNTIME	Expected runtime
JOBSTATE	Exit State of the job

Continued on next page

Continued from previous page

Name	Description
MERGEDSTATE	Merged Exit State of the job
PARENTID	ID of the Parent Job (submission tree)
STATE	Current state of the job (Running, Finished, etc.)
ISRESTARTABLE	Is the job restartable? 1 = yes, 0 = no
SYNCTIME	Time of the transition to Synchronize Wait
RESOURCETIME	Time of the transition to Resource Wait
RUNNABLETIME	Time of the transition to Runnable
FINISHTIME	Finish time
SYSDATE	Current date
SEID	ID of the job definition
TRIGGERWARNING	Text in the warning that activated this trigger
LAST_WARNING	Text in the last issued warning. If no current warning is present, this parameter is empty.
RERUNSEQ	The number of reruns until now
SCOPENAME	Name of the scope (job server) in which the job is running or last ran

Table 6.1.: List of System Variables

The TRIGGER... system variables are only populated if the job was submitted by a trigger. For a more detailed description of the TRIGGER... system variables, refer to the create trigger statement on page 132.
When a job is executed, the parameters used in commands, workdir and file specifications are resolved conform to the sequence given below:

1. System variable
2. The job's own address space
3. The address space of the job and submitting parents, from bottom to top
4. The address space of the job server executing the job
5. The address space of the parent scopes of the job server executing the job, from bottom to top
6. The job definition's parent folders, from bottom to top
7. The parent folders of the parent jobs, from bottom to top

If the configuration parameter 'ParameterHandling' for the server has been set to 'strict' (default), accessing variables that are not defined in the job definition will trigger an error message unless it is a system variable.
If the contents of a variable includes a reference to a another parameter, this parameter is evaluated and replaced in the context of the defining job.

The different parameter types and their semantics are described below:

IMPORT Import-type parameters are used to hand over the data for a Job Scheduling Environment to another job. This type is almost like the parameter type, although import type parameters cannot be handed over like parameters when a job is submitted. Import-type parameters can have a default value, which is used if no value can be acquired from the scheduling environment.

PARAMETER Parameter-type parameters are used to hand over the data from a Job Scheduling Environment to another job. This type is almost like the import type, but parameter-type parameters can be handed over as parameters when a job is submitted. Parameter-type parameters can have a default value, which is used if no value can be acquired from the scheduling environment.

REFERENCE Reference-type parameters are normally used to hand over results from one job to another.
The fully qualified name of the job definition and the name of the referencing parameter are required to create a reference. The Submitted Entity with the closest match to the job definition of the reference is sought to resolve the reference. If this allocation cannot be made clearly enough, this triggers an error message. If a matching Submitted Entity could not be found, the default value (if defined) is returned.

REFERENCE CHILD Child Reference parameters are used to refer to the parameters of direct or indirect children. This can be useful for reporting purposes, for example. A Child Reference parameter is defined using a fully qualified job definition name together with the name of the parameter to be qualified. When resolving the parameter, the Submission Hierarchy is searched downwards instead of upwards as is the case with Reference Parameters. The behaviour for the resolution is otherwise identical to the resolution of Reference Parameters.

REFERENCE RESOURCE Resource Reference-type parameters are used to refer to parameters of allocated resources.
This parameter type requires the fully qualified name of a Named Resource together with an additional parameter name to specify the default reference. The prerequisite for using a Resource Reference parameter is that the resource is also requested. The value is determined in the context of the allocated resource.

RESULT Result-type parameters can acquire a value from the job (using the API). As long as this value has not been set, the optional default value is returned when the value is queried.

CONSTANT Constant-type parameters are parameters that have a value specified in the definition. This value can therefore not change during runtime.

LOCAL These variables are only visible from the perspective of the defining job.

priority The priority of a job determines the order in which jobs are executed. Values between 0 (high priority) and 100 (low priority) are permitted. The priority

option only applies for jobs.

profile The profile defines the Exit State Profile that describes the valid Exit State of the Scheduling Entity.
For a detailed description of the Exit State Profile, refer to the ”create exit state profile” command on page 86.

required The ’required’ section defines the dependencies of other submitted entities in a Master Run which must be fulfilled until the Submitted Entity is capable of carrying on running.
Whether all the dependencies have to be fulfilled or just one of them is defined by the ’dependency’ mode’.
Dependencies are defined in a comma-separated list of fully qualified names of Scheduling Entities (including folder path names).
Dependencies only apply between the Submitted Entities of the Master Run. Synchronizing Resources have to be used to synchronise the Submitted Entities from different Master Runs.
After the Submitted Entity instances of the Submitted Scheduling Entity hierarchy have been created, the system searches for the dependencies as follows: Beginning with the parent of the dependent Submitted Entity, all the children are searched for an instance of the Required Scheduling Entity whereby the branch with the dependent Submitted Entity is obviously ignored. If no instance is found, the search continues in the Submit Hierarchy Parents until precisely one instance has been found. If an instance can still not be found, the property ’unresolved’ defines how this situation is handled by the system. If more than one Submitted Entity is found, the submit fails with an ’ambiguous dependency resolution’ error.
During the execution of a Master Run, a Scheduling Entity can attain an ’unreachable’ state because the dependencies can no longer be fulfilled. This can happen if a Required Scheduling Entity reaches a Final State that is not entered in the list of required states for dependencies or by cancelling a Submitted Entity that is required by another Submitted Entity. These two cases are handled differently.
If the unreachable situation is caused by a Submitted Entity that finishes with an unsuitable Exit State, the system determines the Exit State Profile of the dependent Submitted Entity and sets the Exit State to the state that is marked as being ’unreachable’ in the profile.
If none of the Profile States is marked as an unreachable state or the unreachable state was caused by a Submitted Entity being cancelled, the dependent Submitted Entity is set to the unreachable state, which can only be resolved by an operator ignoring the dependency or cancelling the dependent entity.
All the direct or indirect children of a job or batch inherit all the parent’s dependencies. This means that no child of a job or batch can exit the dependency wait state as long as the parent itself is in this state. Children of milestones do not inherit the dependencies from their parent.

The properties of the dependency definitions are described below:

CONDITION It is possible to stipulate a condition for a dependency. The dependency is only fulfilled if the evaluation of the condition returns the truth value "true". If no condition is specified, the condition is always deemed to have been fulfilled.

DEPENDENCY NAME A name can be optionally specified for the dependency when defining a function. Children (both direct and indirect) can refer to the name in order to ignore this dependency.

MODE The mode property is only relevant if the required Scheduling Entity is a job with children. In this case, the Dependency Mode defines the time when the dependency is fulfilled.
The table below shows the possible values and their meanings.

dependency mode	Description
all_final	The required job and all its children must have reached a Final State.
job_final	Only the required job itself has to reach a Final State, the state of the children is irrelevant.

STATE The state property of a dependency defines a list of Final States that the required Scheduling Entity can achieve to fulfil the dependency.
Without this option, the dependency is fulfilled if the required Scheduling Entity reaches a Final State.
It is also possible to stipulate a condition for a state. If a condition has been specified, the dependency is only deemed to have been fulfilled if the condition is fulfilled as well. The syntactic rules for specifying conditions are the same as those that apply to triggers. For more details, refer to the create trigger statement on page 132. Several implicit definitions are also available as options:

- **default** — The dependency is fulfilled if the predecessor has reached one of the states that are defined in its profile as being a default dependency.
- **all reachable** — The dependency is fulfilled if the predecessor has reached one of the states that are not defined as being unreachable.
- **reachable** — The dependency is fulfilled if the predecessor has reached the state defined as being unreachable.

UNRESOLVED The unresolved property specifies how the system should handle a situation where no Submitted Entity instance could be found during a Submit Operation for a required Scheduling Entity.
The possible behavioural patterns are described in the table below:

unresolved	Description
error	The submit operation fails with an error message
ignore	The dependency is tacitly ignored
suspend	The dependency is ignored, but the dependent Submitted Entity is placed in a 'suspended' state and requires a user action to continue

rerun program If a rerun program command line has been defined for a job, this is executed instead of the run command line when the job is restarted after a failure.

For details about command line parsing, variants and the substitution parameter, refer to the "run program" option on page 114.

resource The "resource" section of a job definition defines resource requirements in addition to those requirements indirectly defined by the "environment" and "footprint" options.

If the same Named Resource as in the footprint is required here, the requirement in the Resource Section overwrites the requirement in the footprint.

Since environments only require Named Resources with the usage "static" and footprints only require Named Resources with the usage "system", the Resource Section in a job definition is the only place where resource requirements for Named Resources with the usage "synchronizing" can be defined.

Resource requirements are defined by the fully qualified path name to a Named Resource defined with the following additional requirement options:

AMOUNT The amount option is only valid with requests for Named Resources of the type "System" or "Synchronizing". The amount in a Resource Request expresses how many units of the Required Resource are allocated.

EXPIRED The expired option is only valid for Synchronizing Resources with a defined Resource State Profile. If the expired option is specified, the time to which the Resource State of the resource has been set cannot be less recent than the time given by the expire option. A negative Expire value means that a resource must be at least as old as given here. The Resource State can only be set by the old resource command (see page

57) or automatically when defining a Resource State Mapping which converts the Exit State and Resource State into a new Resource State. Even if, in such a case, the new Resource State is the same as the old Resource State, the Resource State is considered to have been set.

LOCKMODE The lockmode option in a resource requirement is only valid for Synchronizing Resources. Five possible lockmodes are defined:

Name	Meaning
X	Exclusive lock
S	Shared lock
SX	Shared exclusive lock
SC	Shared compatible lock
N	Nolock

The important aspect here is the compatibility matrix:

	X	**S**	**SX**	**SC**	**N**
X	N	N	N	N	Y
S	N	Y	N	Y	Y
SX	N	N	Y	Y	Y
SC	N	Y	Y	Y	Y
N	Y	Y	Y	Y	Y

The purpose of the exclusive lock is to have exclusive access to the resource to be able to set the Resource State and possibly parameter values. A common example of where the exclusive lock is used is when reloading a database table.
The purpose of the shared lock is to allow other users to use the resource in the same way while preventing them from making any changes. The most frequent scenario for using shared locks is for a large-scale ongoing reading of a database table. Other read processes can simply be tolerated, but no write transactions are allowed.
The purpose of the shared exclusive lock is to have a second shared lock which is not compatible with the normal shared lock. If we use the normal use shared lock for large read transactions, then we use the shared exclusive lock for small write transactions. Small write transactions can easily run in parallel, but if they create a large read transaction when doing so, they will almost certainly cause a ”snapshot too old” or other similar problems.
The purpose of the shared compatible lock is to have a shared lock that is compatible with both the shared and exclusive locks. This lock type is intended for short read transactions which do not conflict with small write transactions or large read transactions. Small read transactions obviously don’t conflict with other small read transactions. Running small read and large write transactions in parallel may cause problems.
The purpose of the nolock is to ensure that the resource exists and that all the other properties of the resource cover requirements. The resource is not locked and anything can happen, including state changes.

STATE The state option is only valid for Synchronizing Resources with a Resource State Profile. It is used to specify valid Resource States for this job. A resource can only be allocated if it is in one of the required states.

STATE MAPPING The state mapping option is only valid for Synchronizing Resources that specify a Resource State Profile and are requested with an ”exclusive” lockmode. The mapping defines a function that maps the combinations of Exit States and Resource States in a new Resource State. For more detailed information about resource state mappings, refer to the create resource state mapping statement on page 123.

KEEP The keep option in a Resource Request defines the time when the resource is released. The keep option is valid for both System and Synchronizing Resources. There are three possible values. Their meanings are explained in the table below:

Value	Meaning
nokeep	The resource is released at the end of the job. This is the default behaviour.
keep	The resource is released as soon as the job has reached the Final State.
keep final	The resource is released when the job and all its children are final.

STICKY The sticky option is only valid for Synchronizing Resources. If sticky is specified, the resource is allocated by the master batch (this is called a MASTER_RESERVATION) for as long as other jobs in the batch that require the ”sticky” resource. The amount and lockmode for the Master Reservation are derived from all the sticky requirements of all the children. The amount is the maximum needed by any job.
The lockmode is exclusive as long as at least two jobs exist which request the resource with a lockmode other than nolock. An exception is the combination of Shared and Shared Compatible lock requests. This combination results in lockmode Shared.
An attempt is made to fulfil all the requirements from the Master Reservation.
A name can be optionally assigned for the sticky allocation. As a basic principle, only those requests with the same name are taken into account for the previously described method. That’s why a master batch can have several MASTER_RESERVATIONS at the same time. Several separate critical regions can be realised within a sequence with the aid of the names.
A parent job or batch can be specified in addition to, or even instead of, the name. The corresponding instance of the parent is then determined at runtime from the submission hierarchy. The sticky request is only valid from the parent downwards. In principle, this can be interpreted as if the parent’s ID represents a part of the name of the sticky request. This mechanism allows separate critical regions to be easily implemented in dynamically submitted sub-workflows.

runtime The runtime option is used to define the estimated runtime of a job. This time can be valuated when activating triggers.

run program The run program command line is mandatory for jobs because it specifies the command that is to be executed for this job.
The command line is separated by whitespace characters in a command and a list of arguments. The first element in the command line is regarded as the name of the executable program that is to be run, and the rest are the parameters for the program.
Whether the job server uses the PATH environment variable when searching for the executable file is a characteristic of the job server.
System and job parameters can be addressed with $ Notation.
Quoting can be used to forward whitespace characters and $ characters as part of the command line. The quoting complies with Unix Bourne shell rules. This means that double quotes prevent whitespace characters from being interpreted as separators. Single quotes also prevent variables from being resolved. Backticks can be used for quoting. The parts of the command line that have been quoted in backticks are regarded as having been single quoted, but the backticks remain a part of the argument. Other quotes are removed.

Example:

The run command line `'sh -c ''example.sh ${JOBID} \$HOME'' '$SHELL''` will execute the program `'sh'` with the parameters `'-c'`, `'example.sh 4711 $HOME'` and `'$SHELL'` (assuming that the Submitted Entity has the ID 4711).
If the executable program (the first element of the command line) is a valid integer, the command line is not run by the job server. 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 ('./42' instead of '42').

suspend The suspend option defines whether a Submitted Entity is suspended at the submit time.
If the suspend option is specified, the resume clause can be optionally used. This can then trigger an automatic resume at or after the specified time.
If the resume time is specified by the incomplete date format (see also page 6), the resume takes place at the first suitable time after the submit time.
If a submit takes place at 16:00, for example, and T17:30 is entered as the resume time, the resume will take place on the same day at 17:30. But if T15:55 is specified as the resume time, the job will have to wait until the next day at 15:55.

timeout The timeout clause of a job definition defines the maximum time for which the job waits until its resource requirements are fulfilled.

When the timeout condition is reached, the job gets the Exit State specified in the timeout clause. This Exit State must be an element of the Exit State Profile.
If no timeout option is given, the job will wait until all the requirements have been fulfilled.

type The type option specifies the Scheduling Entity type that is being created or modified.

workdir The workdir of a Scheduling Entity-type job defines the directory where the run, rerun or kill program is executed.

master The master option defines whether this Scheduling Entity can be submitted in order to create a Master Run.

Output

This statement returns a confirmation of a successful operation *Output*

create named resource

Purpose

Purpose The purpose of the create named resource statement is to define a class of resources.

Syntax

Syntax The syntax for the create named resource statement is

create [**or alter**] **named resource** *resourcepath*
with WITHITEM {**,** WITHITEM}

WITHITEM:
group = *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **parameter = none**
| **parameter = (** PARAMETER {**,** PARAMETER} **)**
| **state profile =** < **none** | *rspname* >
| **usage =** RESOURCE_USAGE

PRIVILEGE:
create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

PARAMETER:
parametername **constant =** *string*
| *parametername* **local constant** [**=** *string*]
| *parametername* **parameter** [**=** *string*]

RESOURCE_USAGE:

category
| **static**
| **synchronizing**
| **system**

Description

The create named resource statement is used to define classes of resources. These classes define the name, the usage type and optionally the utilised Resource State Profile as well as the parameters. *Description*

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

parameter It may be useful to use its parameters in conjunction with allocating resources. For example, a resource like RESOURCE.TEMP_SPACE could have a parameter called LOCATION. This would allow a job to use a resource and allocate temporary storage space somewhere dependent upon the current instance of the Named Resource.
There are three types of parameters in a resource context:

Typ	Bedeutung
constant	This parameter type defines the value that is constant for all resources.
local constant	This parameter type defines a non-variable parameter whose value can deviate between instances of the same Named Resource.
parameter	The value of such a parameter can be changed by jobs that have exclusively locked this resource.

Table 6.2.: Named Resource parameter types

state profile A State Resource Profile can be specified in the case of Synchronizing Resources. This allows jobs to request the resource in a particular state. Resource State changes can be used to activate triggers.

usage The usage of the Named Resource can be one of the following:

Usage	Meaning
category	Categories behave like folders and can be used to arrange the Named Resources in a clearly organised hierarchy.
static	Static resources are resources which, if requested, must be present in the scope in which the job is running but which cannot be used up. Possible examples of Static Resources are a particular operating system, shared libraries for DBMS access operations or the presence of a C compiler.
system	System Resources are resources that can be counted. Possible examples are the number of processes, the capacity of the temporary memory or the availability of (a number of) tape drives.
synchronizing	Synchronizing Resources are the most complex resources and are used to synchronise multiple access operations. One possible example is a database table. Multiple access operations may be tolerated or not depending on the type of access (large read transactions, large write transactions, multiple small write transactions, etc.).
pool	pool-type Named Resources are used to create so-called Resource Pools. These pools allow the distribution of amounts for System Resources to be regulated centrally and flexibly.

Table 6.3.: Named Resource usage

factor When creating a Named Resource, the factor by which the specified amounts in a resource request are multiplied can be specified. The default factor is 1. This factor can be overwritten for each instance of this Named Resource (i.e. for each resource).

inherit grant The **inherit grants** clause allows you to define which privileges are to be inherited through the hierarchy. If this clause is not specified, all privileges are inherited by default.

Output

Output This statement returns a confirmation of a successful operation

create resource

Purpose

The purpose of the create resource statement is to create an instance of a named resource within a scope, folder or job definition. *Purpose*

Syntax

The syntax for the create resource statement is *Syntax*

create [**or alter**] **resource** *resourcepath* **in** < *serverpath* | *folderpath* >
with WITHITEM {**,** WITHITEM}

WITHITEM:
amount = < **infinite** | *integer* >
| < **online** | **offline** >
| **parameter = none**
| **parameter = (** PARAMETER {**,** PARAMETER} **)**
| **requestable amount =** < **infinite** | *integer* >
| **state =** *statename*
| **touch** [**=** *datetime*]
| **group =** *groupname*

PARAMETER:
parametername **=** < *string* | **default** >

Description

The create resource statement is used to instantiate Named Resources within scopes, folders or job definitions. In the latter case, only a template is created which is materialised as soon as the job is submitted and automatically destroyed as soon as the Master Run is Final or Cancelled. *Description*
If the **or alter** option is specified, an existing resource is changed; otherwise, it is considered to be an error if the resource already exists.

amount The **amount** clause defines the Available Amount for this resource. The amount option is not specified in the case of static resources.

base multiplier The base multiplier is only relevant if the Resource Tracing is being used. The base multiplier determines the multiplication factor for **trace base**. If the trace base is designated as being B and the trace multiplier as being M,

the mean allocation is determined for the periods $B * M^0$, $B * M^1$ and $B * M^2$. The default value is 600 (10 minutes) so that the values for B, $10B$ and $100B$ (in minutes) are determined.

factor A Resource Factor has been implemented to allow resource requirements for jobs to be adjusted externally. This can be set in both the Named Resource and individually in the resource. Whether a job can be allocated a particular resource is determined by comparing the original request with the Requestable Amount. However, the actual allocation is taken from

$$\text{ceil}(\text{Requirement} * \text{Factor})$$

.

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

online The **online** clause defines whether the resource is online or offline.
A resource is not available if it is offline. This means that a job that requires this resource cannot run within this scope. But since the resource can be set to online, the job will wait and will not be set to an error state.
This also applies to static resources.

parameter The **parameter** clause is used to set the values of the parameters that have been defined for the Named Resource.
Parameters that are declared as a constant at Named Resource Level are not permitted here. All the other parameters can be specified, although this is not mandatory.
If a parameter or a default value for this parameter has not been specified at Named Resource Level, the resolution returns an empty string.
If parameter name = default is specified when changing the resource, the parameter takes on the default value analogue to the Named Resource.
If the parameter is changed on the Named Resource level, this is visible on the Resource level for all the parameters that have been set to the default value.
A number of system variables are always defined for each resource. These are set by the system and are available to jobs which allocate the resource for read access using "RESSOURCEREFERENCES".
These system variables are:

Name	Description
STATE	The Resource State of a "synchronizing" resource with a state model

Continued on next page

Continued from previous page

Name	Description
AMOUNT	The total amount of available resources
FREE_AMOUNT	The total amount of available free resources
REQUESTABLE_AMOUNT	The maximum amount that can be allocated by a job
REQUESTED_AMOUNT	The amount requested by the job
TIMESTAMP	The touch timestamp of a ”synchronizing” resource with a state model

Table 6.4.: List of System Variables

requestable amount The **requestable amount** clause defines the amount of this resource that can be requested by a single job. This does not have to be the same as the available amount. If the requested amount is smaller than the amount, it is certain that a job cannot allocate all the available resources. If the Requestable Amount is greater than the amount, jobs can request more than the available amount without triggering a ”cannot run in any scope” error.
If the Requestable Amount is not specified, it is the same as the amount.
The requestable amount option is not specified in the case of static resources.

state The **state** clause defines the resource’s state.
This option is only valid for Synchronizing Resources with a Resource State Profile.

tag To facilitate evaluating the trace table, resources and pools can now be marked with a tag. This tag should be unique within the resources and pools (i.e. the use of a tag for both a resource and a pool is prohibited as well).

touch The **touch** clause defines the last time when the status of the resource (of a job) was changed. This timestamp is not set if a Resource State has been set manually.
This option is only valid for Synchronizing Resources with a Resource State Profile.

trace base Tracing is deactivated if the trace base is **none**. Otherwise it is the basis for the valuation period.

trace interval The trace interval is the minimum time in seconds between when Trace Records are written. Tracing is deactivated if the trace interval is **none**.

Output

This statement returns a confirmation of a successful operation *Output*

create resource state definition

Purpose

Purpose The purpose of the create resource state definition statement is to create a symbolic name for a state of a resource.

Syntax

Syntax The syntax for the create resource state definition statement is

create [**or alter**] **resource state definition** *statename*

Description

Description The create resource state definition statement is used to define a symbolic name for a Resource State.
The optional keyword **or alter** is used to prevent error messages from being triggered and the current transaction from being aborted if a Resource State Definition already exists. If it is not specified, the existence of a Resource State Definition with the specified name will trigger an error.

Output

Output This statement returns a confirmation of a successful operation

Example

Example A number of names for Resource States are defined in these examples.

```
create resource state definition empty;
create resource state definition valid;
create resource state definition invalid;
create resource state definition stage1;
create resource state definition stage2;
create resource state definition stage3;
```

create resource state mapping

Purpose

Purpose

The purpose of the create resource state mapping statement is to define a mapping between the exit states of a job and the resulting resource state of a resource.

Syntax

Syntax

The syntax for the create resource state mapping statement is

create [**or alter**] **resource state mapping** *mappingname*
with map = (WITHITEM {**,** WITHITEM} **)**

WITHITEM:
statename **maps** < *statename* | **any** > **to** *statename*

Description

Description

The create resource state mapping statement defines the mapping of Exit States in combination with Resource States to create new Resource States.
The first state name must be an Exit State. The second and third state have to each be a Resource State. If a job terminates with the given Exit State, the resource state is set to the new state if the current state matches the first named state. If **any** is specified as the initial state, any Resource State is mapped to the new one. If both a specific mapping and a general mapping have been specified, the specific mapping has the highest priority.

Output

Output

This statement returns a confirmation of a successful operation

Example

Example

create resource state profile

Purpose

Purpose The purpose of the create resource state profile statement is to create a set of valid resource states.

Syntax

Syntax The syntax for the create resource state profile statement is

create [**or alter**] **resource state profile** *profilename*
with WITHITEM {**,** WITHITEM}

WITHITEM:
initial state = *statename*
| **state = (** *statename* {**,** *statename*} **)**

Description

Description The create resource state profile statement is used to define a set of valid Resource States for a (Named) Resource.

state The **state** clause defines which Resource State Definitions are valid within this profile.

initial state The initial state clause determines the initial state of a resource with this profile. The initial state does not have to be present in the list of states from the **state** clause. This allows a resource to be created without it immediately playing an active role in the system.

Output

Output This statement returns a confirmation of a successful operation

Example

Example In this example, the Exit State is to become invalid if it is empty.

```
create resource state profile example1
with
   state = (empty);
```

create schedule

Purpose

The purpose of the create schedule statement is to create an active container for scheduled events. *Purpose*

Syntax

The syntax for the create schedule statement is *Syntax*

create [**or alter**] **schedule** *schedulepath* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:

< **active** | **inactive** >
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **interval =** < **none** | *intervalname* >
| **time zone =** *string*
| **group =** *groupname*

PRIVILEGE:

create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**
| **submit**
| **use**
| **view**

Description

With the create schedule statement, complex schedules can be created for jobs and batches using simple definitions. *Description*

active The **active** option causes the schedule to always trigger events in step with the specified interval (assuming that any events have been defined). The **inactive** option, on the other hand, prevents the schedule from triggering events in step with the specified interval. A hierarchical arrangement of schedules thus allows exception periods (such as downtimes) to be defined, for example.

Group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

Interval The given interval acts as a 'clock' for the schedule. If an event is linked to the schedule, this event is triggered in rhythm with the interval.

inherit grant The **inherit grants** clause allows you to define which privileges are to be inherited through the hierarchy. If this clause is not specified, all privileges are inherited by default.

Output

Output This statement returns a confirmation of a successful operation

create scheduled event

Purpose

The purpose of the create scheduled event is to define a connection between a schedule and an event. *Purpose*

Syntax

The syntax for the create scheduled event statement is *Syntax*

create [**or alter**] **scheduled event** *schedulepath* **.** *eventname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
< **active** | **inactive** >
| **backlog handling =** < **last** | **all** | **none** >
| **calendar =** < **active** | **inactive** >
| **horizon =** < **none** | *integer* >
| **suspend limit =** < **default** | *period* >
| **group =** *groupname*

Description

Scheduled Events represent a link between events (what is to be done) and schedules (when should it be done). *Description*

cwbacklog handling The backlog handling function indicates how events that happened during a server downtime are to be handled. The three possible actions are shown in the table below:

Action	Meaning
last	Only the last event is triggered
all	All the events that happened in the meantime are triggered
none	None of the events that happened in the meantime are triggered

Group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

active Scheduled Events can be marked as being **active** or **inactive**. If they are marked as being active, events are triggered. Correspondingly, events are not triggered if the Scheduled Event is marked as being inactive. This option can be used to deactivate Scheduled Events without the definition being lost.

suspend limit The suspend limit defines the length of the delay before a job belonging to an event is automatically submitted with the suspend option. A delay can arise if, for whatever reason, the Scheduling Server goes offline. After the server has booted up again, events that have happened during the downtime are triggered dependent upon the **backlog handling** option. This means that the execution time is later than the scheduled execution time.

Output

Output This statement returns a confirmation of a successful operation

create scope

Purpose

The purpose of the create scope statement is to create a scope within the scope hierarchy. *Purpose*

Syntax

The syntax for the create scope statement is *Syntax*

create [**or alter**] < **scope** *serverpath* | **job server** *serverpath* > [**with** JS_WITHITEM {**,** JS_WITHITEM}]

JS_WITHITEM:

config = none
| **config = (** CONFIGITEM {**,** CONFIGITEM} **)**
| < **enable** | **disable** >
| **error text =** < **none** | *string* >
| **group =** *groupname* [**cascade**]
| **inherit grant = none**
| **inherit grant = (** PRIVILEGE {**,** PRIVILEGE} **)**
| **node =** *nodename*
| **parameter = none**
| **parameter = (** PARAMETERITEM {**,** PARAMETERITEM} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

CONFIGITEM:

parametername **= none**
| *parametername* **= (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**
| *parametername* **=** < *string* | *number* >

PRIVILEGE:

create content
| **drop**
| **edit**
| **execute**
| **monitor**
| **operate**
| **resource**

| **submit**
| **use**
| **view**

PARAMETERITEM:
parametername **= dynamic**
| *parametername* **=** < *string* | *number* >

PARAMETERSPEC:
parametername **=** < *string* | *number* >

Description

Description The create scope command is used to define a scope or job server and its properties.

Config The config option allows a job server to be configured using key/value pairs.
The configuration is inherited downwards so that general configuration parameters can be set at scope level. This means that they are valid for all the job servers created below this level provided that the parameters at the lower level are not overwritten. When the job server logs onto the scheduling server, the server is given the list with the configuration parameters.

Enable The enable option allows the job server to connect to the repository server. This option is not valid for scopes and is tacitly ignored if it is specified.

Disable The disable option forbids the job server from connecting to the repository server. This option is not valid for scopes and is tacitly ignored if it is specified.

Group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

Node The node specifies the computer on which the job server is running. This field has a purely documentary character.

Parameter Parameters can be used for communication and data transfer purposes between jobs. They are available for use with the jobs and programs that are executed within the jobs.
The parameters of scopes and job servers can be used to specify information about a job's runtime environment.

A Dynamic Parameter is fulfilled after the job server has logged on from within its own process environment. If the process environment of a job server is changed, attention has to be paid to this Dynamic Variable because otherwise race conditions can easily arise.

Password The password option is used to set the password for the job server. This option is not valid for scopes and is tacitly ignored if it is specified.

Output

This statement returns a confirmation of a successful operation *Output*

create trigger

Purpose

Purpose The purpose of the create trigger statement is to create an object which submits a job dynamically when a certain condition is met.

Syntax

Syntax The syntax for the create trigger statement is

create [**or alter**] **trigger** *triggername* **on** CT_OBJECT [< **noinverse** | **inverse** >]
with WITHITEM {**,** WITHITEM}

CT_OBJECT:
job definition *folderpath*

WITHITEM:
< **active** | **inactive** >
| < **nowarn** | **warn** >
| **limit state =** < **none** | *statename* >
| < **nomaster** | **master** >
| **rerun**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **state = none**
| **state = (** < *statename* {**,** *statename*} |
CT_RSCSTATUSITEM {**,** CT_RSCSTATUSITEM} > **)**
| **submit after** *folderpath*
| **submit** *folderpath*
| **submitcount =** *integer*
| < **nosuspend** | **suspend** >
| [**type =**] CT_TRIGGERTYPE
| **group =** *groupname*

CT_EVENT:
< **create** | **change** | **delete** >

CT_RSCSTATUSITEM:
< *statename* **any** | *statename statename* | **any** *statename* >

CT_TRIGGERTYPE:
before final
| **immediate local**
| **immediate merge**
| **warning**

Description

Description

The create trigger statement is used to create an object that waits for a certain event to happen following which a job or batch is submitted in response to this event.
If the **or alter** option is specified, an existing trigger is changed; otherwise, it is considered to be an error if the trigger already exists.
Triggers can be defined for Scheduling Entities or Synchronizing (Named) Resources. In the latter case, the trigger is valuated every time the state of the resource or instance of the Named Resource changes. Resource Triggers are always so-called Master Triggers, i.e. they submit a new Master Batch or Master Job. Although triggers in Scheduling Entities can submit Master Batches, by default they submit new children. These children must be defined as (dynamic) children of the triggering Scheduling Entities.

active The active option enables the trigger to be activated or deactivated. This means that the trigger action can be temporarily suppressed without having to delete the trigger.

check The check option is only valid for **until final** and **until finished** triggers. It defines the time intervals between two evaluations of the conditions.
The condition is always evaluated when a job finishes regardless of the defined intervals.

condition The condition option can be specified to define an additional condition which has to be checked before the trigger is activated. This condition is a Boolean expression and the trigger is activated if this condition returns 'true'.

BOOLEAN OPERATORS Since this condition is a Boolean expression, Boolean operators can be used to create multiple complex conditions. This Boolean operators are:

- **not** (unary negation operator)
- **and**
- **or**

The usual priority rules apply. The 'not' operator takes priority over the 'and' operator, which in turn takes priority over the 'or' operator. Parentheses can be used to force a valuation sequence.
It is also permitted to use the Boolean constants **false** and **true**.

COMPARISON OPERATORS Comparisons can be used as part of Boolean expressions. The following comparison operators are defined.

- == (equal to)
- >= (greater than or equal to)
- <= (less than or equal to)
- ! = (not equal to)
- > (greater than)
- < (less than)
- =~ (pattern matches)
- ! ~ (pattern does not match)

All comparison operators can work with strings. With character strings, the 'larger than' and 'less than' operators use the ASCII value of the characters. The matching operators do not work with numbers.
For a full description of the regular expressions that can be used by the match operators, please refer to the original Java documentation for java.util.regexp.

NUMERIC OPERATORS Since it cannot be guaranteed that decisions cannot only be made by comparing two values, the use of (numeric) operators is also permitted. The valid operators are:

- + (unary operator)
- − (unary negation operator)
- ∗ (multiplication operator)
- / (division operator)
- % (modulo Operator)
- + (binary addition operator)
- − (binary subtraction operator)

LITERALS AND VARIABLES Literals are numbers (integers and floating point numbers) or character strings. Strings are delimited using double quotes ("). It is possible to use variables that are resolved within the context of the triggering job or resource. Variables are addressed by prefixing their name with a dollar sign ($).
When a variable is resolved, it is initially assumed that it is a trigger variable. If this is not the case, it is interpreted as a job variable. This kind of resolution is often, but unfortunately not always, correct. The prefix `job.`, `trigger.` or `resource.` can be used to explicitly specify which object will initiate a search for the variable.
Variables are usually created in uppercase. This can be prevented by quoting the name. However, the name is converted back to uppercase when addressing the variables in conditions. To avoid this, the name and prefix (where applicable) have to be written in braces.
The operands are interpreted as character strings or numbers depending upon the operator and the first operand. Multiplication, division, modulo and subtraction operations, as well as unary processes, are only defined for numeric values. The addition operator in a character string context causes the operands to be strung together.

FUNCTIONS Not everything can be simply expressed using (numeric) expressions, and so some additional functions have been added. The following functions are defined at this time:

- **abs(***expression***)** – the absolute value of the expression is returned
- **int(***expression***)** – the integer value of the expression is returned
- **lowercase(***expression***)** – the result of the expression is converted to lowercase and returned
- **round(***expression***)** – the expression is rounded and returned
- **str(***expression***)** – the expression is returned as a character string
- **substr(***source***,** *from* [**,** *until*]**)** – returns part of the character string *source* beginning at the position *from* up to the end of the string or, if *until* is spezcified, up to the position *until*
- **str(***expression***)** – the expression is returned without a space at the end
- **uppercase(***expression***)** – the result of the expression is converted to uppercase and returned

Functions can be nested in one another without any restrictions.

EXAMPLES To clarify this, here are some statements that specify the conditions. Since conditions are not just found in trigger definitions, some other examples are given here as well. However, the syntax is always the same.
The first example shows a trigger that is activated when the job state changes to `WARNING` or `FAILURE` after it has already processed some rows (`$NUM_ROWS > 0$`).

```
CREATE OR ALTER TRIGGER ON_FAILURE
    ON JOB DEFINITION SYSTEM.EXAMPLES.E0100_TRIGGER.TRIGGER
WITH
    STATES = (FAILURE, WARNING),
    SUBMIT SYSTEM.EXAMPLES.E0100_TRIGGER.ON_FAILURE,
    IMMEDIATE MERGE,
    ACTIVE,
    NOMASTER,
    SUBMITCOUNT = 3,
    NOWARN,
    NOSUSPEND,
    CONDITION = '$NUM_ROWS > 0';
```

The second example shows an environment that requires the value of the resource variable `AVAILABLE` to begin with a `T` (such as TRUE, True, true or Tricky).

```
CREATE ENVIRONMENT SERVER@LOCALHOST
WITH RESOURCE = (
    RESOURCE.EXAMPLES.STATIC.NODE.LOCALHOST
        CONDITION = '$RESOURCE.AVAILABLE =~ "[tT].*"',
    RESOURCE.EXAMPLES.STATIC.USER.SERVER
);
```

The third example shows the same as the second one, except that here the parameter name is defined as being mixed case.

```
CREATE ENVIRONMENT SERVER@LOCALHOST
WITH RESOURCE = (
    RESOURCE.EXAMPLES.STATIC.NODE.LOCALHOST
        CONDITION = '${RESOURCE.Available} =~ "[tT].*"',
    RESOURCE.EXAMPLES.STATIC.USER.SERVER
);
```

event The event option is only relevant for Object Monitor Triggers. It specifies for which types of events the trigger should be activated.

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

main The master option is used to define whether the job is submitted as a master or not. This option is only relevant for Job Triggers because Resource Triggers are always submitted as a master.

master The master option is used to define whether the job is submitted as a master or not. This option is only relevant for Job Triggers because Resource Triggers are always submitted as a master.

parent The parent option is only relevant for Object Monitor Triggers. It can also only be specified in combination with the main option.
If it is specified, a search is run for the corresponding job (or batch) within the tree submitted using the main job is sought and the trigger jobs are appended below the parent.

rerun The rerun option can only react to restartable states and initiate an automatic rerun. In many cases, it will be practicable to also specify the suspend/resume options to allow a certain period of time between the resumes.
Either the submit option or the rerun option have to be specified.

resume The resume option can be used together with the suspend option to cause a delayed execution. There are two ways to do this. A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
If an incomplete time is defined, such as `T16:00`,
the time for the trigger activation is taken as the reference time.

state The state option is valid for all triggers apart from **until final** and **until finished** triggers. A list of Exit States can be specified for triggers that act on jobs. When the job in which the trigger is defined reaches an Exit State that is listed in the Trigger Definition, this activates the trigger (unless a condition has been specified that is valuated as false).
A list of state changes can be specified in the case of a trigger that acts on a (Named) resource. This allows each state change to be explicitly addressed. It is possible to activate a trigger when a state is exited by using the keyword **any** on the right. It is always possible to activate a trigger on reaching a certain state by specifying **any** on the left. The state option is omitted to activate a trigger after every state change.

submit The submit option defines which job or batch is submitted when the trigger is activated.
Either the submit option or the rerun option have to be specified.

submitcount The submitcount option is only permitted for triggers that act on jobs. It defines the number of times that a trigger can be activated. If this option is not specified, a submitcount of 1 is used.
If a submitcount of 0 is specified, the submitcount is set to the server parameter TriggerSoftLimit (the default value for this is 50). In the case of a rerun trigger, however, a submitcount of 0 means that there is no limit to the number of restart attempts.
If a submitcount greater than the TriggerSoftLimit is specified, the submitcount is restricted to the server parameter TriggerHardLimit (the default value for this is

100). This is done to avoid endless loops. The TriggerHardLimit can be set in the server configuration to $2^{31} - 1$ in order to virtually eliminate the restriction above.

suspend The suspend option is used to submit the job or batch in a suspend state. This option is valid for all trigger types.

type There are several types of triggers on jobs. The most important difference between them is the time at which they are checked. The table below shows a list of all the types with a brief description of their behaviour.
It must be emphasised that the type option is not valid for (named) resource triggers.

Field	Description
Type	Check time
after final	Only after a final state is reached is a check run to establish whether the defined trigger has to be activated. If the trigger is not a Master Trigger, the newly submitted job will have the same parent as the triggering job. A special situation arises if the triggering job triggers its own submit. In this case, the newly submitted job replaces the triggering job. Since this exchange takes place before the dependency was checked, all the dependent jobs wait until the newly submitted job is final.
before final	Immediately before a final state is reached, a check is run to establish whether the defined trigger is to be activated. This is the last opportunity to submit new children. If this is done, the job or batch will not reach a Final State at this time.
finish child	A finish child trigger checks whether it is to be activated every time when a direct or indirect child finishes.
immediate local	The immediate local trigger local checks whether it has to be activated when a job is terminated. Only the Exit State of the job is taken into consideration.

Continues on next page

Continued from previous page

Field	Description
immediate merge	The immediate merge trigger checks whether it has to be activated as soon as the Merged Exit State changes.
until final	The until final trigger periodically checks whether it has to be activated. This check starts as soon as a job or batch has been submitted and does not stop until it is final. The until final trigger imperatively requires a condition. This condition is checked at least once. This check takes place when the job or batch switches to the finished state.
until finished	The until finished trigger is similar to the final trigger. The only difference is that the until finished trigger stops the check as soon as the job is finished. The until finished trigger imperatively requires a condition. This condition is checked at least once. This check takes place when the job or batch switches to the finished state.

Table 6.5.: Description of the different types of triggers

Output

This statement returns a confirmation of a successful operation *Output*

create user

Purpose

Purpose The purpose of the create user statement is to create a pair of values which can be used to authenticate oneself to the server.

Syntax

Syntax The syntax for the create user statement is

create [**or alter**] **user** *username*
with WITHITEM {**,** WITHITEM}

WITHITEM:
default group = *groupname*
| < **enable** | **disable** >
| **group = (** *groupname* {**,** *groupname*} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

Description

Description The create user statement is used to create a user. If "**or alter**" is specified, an already existing user is changed. Otherwise, an existing user will trigger an error. The **default group** clause is used to specify the Default Group.

default group The default group clause defines the group that is used as the owner for all its objects created by the user if an explicit group was not specified when the object was created.
The default group must be one of the user's groups.

enable The enable option allows the user to connect to the repository server.

disable The disable option forbids the user from connecting to the repository server.

group The group clause is used to specify the groups to which the user belongs. Every user is a member of the Public system group.

password The password option is used to set the password for the user.

rawpassword The rawpassword is used to set the user's password that is required to connect to the repository server. The rawpassword is the already encrypted password.
The rawpassword option has been implemented to be able to dump and restore users.

Output

This statement returns a confirmation of a successful operation *Output*

7. deregister commands

deregister

Purpose

Purpose The purpose of the deregister statement is to notify the server that the jobserver is not to process jobs anymore. See also the register statement on page 238.

Syntax

Syntax The syntax for the deregister statement is

deregister *serverpath* **.** *servername*

Description

Description The deregister statement is used to notify the server about a more or less permanent failure of a job server.
This message prompts different server actions. Firstly, all the running jobs on the job server (i.e. jobs in the state **started**, **running**,
to_kill and **killed**) are set to the state **broken_finished**. Jobs in the state **starting** are reset to **runnable**. The job server is then removed from the list of job servers that are able to process jobs so that this job server is consequently no longer allocated any more jobs. A side effect of this is that jobs that can only run on this server due to their resource requirements are set to the state **error** with the message "Cannot run in any scope because of resource shortage". Finally, a complete reschedule is executed so that jobs are redistributed among the job servers. The job server is added to the list of job-processing job servers again by re-registering it (refer to the register statement on page 238).

Output

Output This statement returns a confirmation of a successful operation

8. disconnect commands

disconnect

Purpose

Purpose The purpose of the disconnect statement is to terminate the server connection.

Syntax

Syntax The syntax for the disconnect statement is

disconnect

Description

Description The connection to the server can be shut down using the disconnect statement.

Output

Output This statement returns a confirmation of a successful operation

9. drop commands

drop comment

Purpose

Purpose The purpose of the drop comment statement is to remove the comment.

Syntax

Syntax The syntax for the drop comment statement is

drop [**existing**] **comment on** OBJECTURL

OBJECTURL:
distribution *distributionname* **for pool** *resourcepath* **in** *serverpath*
| **environment** *environmentname*
| **exit state definition** *statename*
| **exit state mapping** *mappingname*
| **exit state profile** *profilename*
| **event** *eventname*
| **resource** *resourcepath* **in** *folderpath*
| **folder** *folderpath*
| **footprint** *footprintname*
| **group** *groupname*
| **interval** *intervalname*
| **job definition** *folderpath*
| **job** *jobid*
| **named resource** *resourcepath*
| **parameter** *parametername* **of** PARAM_LOC
| **resource state definition** *statename*
| **resource state mapping** *mappingname*
| **resource state profile** *profilename*
| **scheduled event** *schedulepath* **.** *eventname*
| **schedule** *schedulepath*
| **resource** *resourcepath* **in** *serverpath*
| < **scope** *serverpath* | **job server** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

PARAM_LOC:
folder *folderpath*
| **job definition** *folderpath*
| < **scope** *serverpath* | **job server** *serverpath* >

TRIGGEROBJECT:

resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

Description

Description

The drop comment statement deletes the existing comment for the specified object. If the **existing** keyword is not specified, the absence of a comment is considered to be an error.

Output

Output

This statement returns a confirmation of a successful operation

drop environment

Purpose

Purpose The purpose of the drop environment statement is to remove the specified environment.

Syntax

Syntax The syntax for the drop environment statement is

drop [**existing**] **environment** *environmentname*

Description

Description The drop environment statement is used to delete a definition from an environment. An error is triggered if jobs are still using this environment. If the **existing** keyword is being used, it is not considered to be an error if the specified environment does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop event

Purpose

The purpose of the drop event statement is to remove the specified event. *Purpose*

Syntax

The syntax for the drop event statement is *Syntax*

drop [**existing**] **event** *eventname*

Description

The drop environment statement is used to delete a definition of an event. If the **existing** keyword is being used, it is not considered to be an error if the specified event does not exist. *Description*
An event cannot be deleted if Scheduled Events belong to it.

Output

This statement returns a confirmation of a successful operation *Output*

drop exit state definition

Purpose

Purpose The purpose of the drop exit state definition statement is to remove the specified exit state definition.

Syntax

Syntax The syntax for the drop exit state definition statement is

drop [**existing**] **exit state definition** *statename*

Description

Description The drop exit state definition statement is used to delete an Exit State Definition. It is considered to be an error if Exit State Profiles are still using this Exit State Definition. If the **existing** keyword is being used, it is not considered to be an error if the specified Exit State Definition does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop exit state mapping

Purpose

Purpose

The purpose of the drop exist state mapping statement is to remove the specified mapping.

Syntax

Syntax

The syntax for the drop exit state mapping statement is

drop [**existing**] **exit state mapping** *mappingname*

Description

Description

The drop exit state mapping statement is used to delete an Exit State Mapping. It is considered to be an error if jobs or Exit State Profiles are still using this Exit State Mapping. If the
existing keyword is being used, it is not considered to be an error if the specified Exit State Mapping does not exist.

Output

Output

This statement returns a confirmation of a successful operation

drop exit state profile

Purpose

Purpose The purpose of the drop exit state profile statement is to remove the specified profile.

Syntax

Syntax The syntax for the drop exit state profile statement is

drop [**existing**] **exit state profile** *profilename*

Description

Description The drop exit state profile statement is used to delete a definition of an Exit State Profile. It is considered to be an error if jobs are still using this Exit State Profile. If the **existing** keyword is being used, it is not considered to be an error if the specified Exit State Profile does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop folder

Purpose

The purpose of the drop folder statement is to remove a folder and its contents from the system. *Purpose*

Syntax

The syntax for the drop folder statement is *Syntax*

drop [**existing**] FOLDER_OR_JOB {**,** FOLDER_OR_JOB} [**cascade**] [**force**]

FOLDER_OR_JOB:
[< **folder** *folderpath* | **job definition** *folderpath* >]

Description

The drop folder statement removes folders and their contents from the system. There are two options: *Description*

Cascade The cascade option deletes folders, job definitions and subfolders, but only if they are not referenced to the job definitions, for example as required job.

Force With the force option, references to job definitions are removed as well. Force implies cascade.
Folders cannot be deleted if they are not empty unless cascade or force has been specified.

Output

This statement returns a confirmation of a successful operation *Output*

drop footprint

Purpose

Purpose The purpose of the drop footprint statement is to remove the specified footprint.

Syntax

Syntax The syntax for the drop footprint statement is

drop [**existing**] **footprint** *footprintname*

Description

Description The drop footprint statement is used to delete footprints and resource requirements. If the **existing** keyword is being used, it is not considered to be an error if the specified footprint does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop group

Purpose

The purpose of the drop group statement is to remove a group from the system. *Purpose*

Syntax

The syntax for the drop group statement is *Syntax*

drop [**existing**] **group** *groupname*

Description

The drop group statement is used to delete a group. If there are still any group members in this group, their membership is automatically terminated. *Description*
It is considered to be an error if the group is still the owner of an object.
It is not possible to delete a group that is defined as the Default Group for a user.
If the **existing** keyword is being used, it is not considered to be an error if the specified group does not exist.

Output

This statement returns a confirmation of a successful operation *Output*

drop interval

Purpose

Purpose The purpose of the drop interval statement is to remove the specified interval.

Syntax

Syntax The syntax for the drop interval statement is

drop [**existing**] **interval** *intervalname*

Description

Description The drop interval statement is used to delete intervals. If the **existing** keyword is being used, it is not considered to be an error if the specified interval does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop job definition

Purpose

The purpose of the drop job definition statement is to remove the specified scheduling entity object. *Purpose*

Syntax

The syntax for the drop job definition statement is *Syntax*

drop [**existing**] **job definition** *folderpath* **.** *jobname* [**force**]

Description

The drop job definition statement deletes the given job definition. *Description*
If a job definition is referenced (for instance as Required Job), it cannot be deleted unless the force option is specified. If the force option is being used, all references to a job definition are also deleted.

Output

This statement returns a confirmation of a successful operation *Output*

drop named resource

Purpose

Purpose The purpose of the drop named resource statement is to delete a class of resources.

Syntax

Syntax The syntax for the drop named resource statement is

drop [**existing**] **named resource** *resourcepath* [**cascade**]

Description

Description The drop named resource statement is used to delete Named Resources. It is considered to be an error if the Named Resource is still instantiated in scopes, job definitions and/or folders and the **cascade** option is not specified.
On the other hand, Scope Resources as well as folders and Job Definition Resources are deleted if the **cascade** option is specified.
If any requirements exist for the Named Resources that are to be deleted, the statement will fail.
If the **existing** keyword is being used, it is not considered to be an error if the specified Name Resource does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop resource

Purpose

The purpose of the drop resource statement is to remove an instance of a named resource from a scope, folder or job definition. *Purpose*

Syntax

The syntax for the drop resource statement is *Syntax*

drop [**existing**] RESOURCE_URL [**force**]

RESOURCE_URL:
resource *resourcepath* **in** *folderpath*
| **resource** *resourcepath* **in** *serverpath*

Description

The drop resource statement is used to delete a resource. It is considered to be an error if the resource is still being allocated by Running Jobs. *Description*
If the **existing** keyword is being used, it is not considered to be an error if the specified resource does not exist.

Output

This statement returns a confirmation of a successful operation *Output*

drop resource state definition

Purpose

Purpose The purpose of the drop resource state definition statement is to remove the definition.

Syntax

Syntax The syntax for the drop resource state definition statement is

drop [**existing**] **resource state definition** *statename*

Description

Description The drop resource state definition statement is used to delete Resource State Definitions. It is considered to be an error if Resource State Profiles are still using this Resource State Definition. If the **existing** keyword is being used, it is not considered to be an error if the specified Resource State Definition does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop resource state mapping

Purpose

The purpose of the drop resource state mapping statement is to delete the mapping. *Purpose*

Syntax

The syntax for the drop resource state mapping statement is *Syntax*

drop [**existing**] **resource state mapping** *mappingname*

Description

The drop resource state mapping statement is used to delete a Resource State Mapping. It is considered to be an error if job definitions are using this Resource State Mapping. If the **existing** keyword is being used, it is not considered to be an error if the specified Resource State Mapping does not exist. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

drop resource state profile

Purpose

Purpose The purpose of the drop resource state profile statement is to remove a resource state profile.

Syntax

Syntax The syntax for the drop resource state profile statement is

drop [**existing**] **resource state profile** *profilename*

Description

Description The drop resource state profile statement is used to delete the definition of a Resource State Profile. It is considered to be an error if Named Resources are still using this Resource State Profile. If the **existing** keyword is being used, it is not considered to be an error if the specified Resource State Profile does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop schedule

Purpose

The purpose of the drop schedule statement is to remove the specified schedule. *Purpose*

Syntax

The syntax for the drop schedule statement is *Syntax*

drop [**existing**] **schedule** *schedulepath*

Description

The drop schedule statement is used to delete schedules. If the **existing** keyword is being used, it is not considered to be an error if the specified schedule does not exist. *Description*
A schedule cannot be deleted if it has a Scheduled Event that belongs to it. It cannot be deleted either if child objects exist.

Output

This statement returns a confirmation of a successful operation *Output*

drop scheduled event

Purpose

Purpose The purpose of the drop scheduled event is to remove the specified scheduled event.

Syntax

Syntax The syntax for the drop scheduled event statement is

drop [**existing**] **scheduled event** *schedulepath* **.** *eventname*

Description

Description The drop interval statement is used to delete Scheduled Events. If the **existing** keyword is being used, it is not considered to be an error if the specified Schedule Event does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop scope

Purpose

The purpose of the drop scope statement is to remove a scope and its contents from the scope hierarchy. *Purpose*

Syntax

The syntax for the drop scope statement is *Syntax*

drop [**existing**] < **scope** *serverpath* | **job server** *serverpath* > [**cascade**]

Description

This statement is synonymous to the drop job server statement. The cascade option deletes the scope together with its contents. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

drop trigger

Purpose

Purpose The purpose of the drop trigger statement is to remove the specified trigger.

Syntax

Syntax The syntax for the drop trigger statement is

drop [**existing**] **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]

TRIGGEROBJECT:

resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

Description

Description The drop trigger statement is used to delete a trigger.
If the **existing** keyword is being used, it is not considered to be an error if the specified trigger does not exist.

Output

Output This statement returns a confirmation of a successful operation

drop user

Purpose

The purpose of the drop user statement is to remove the user from the system. *Purpose*

Syntax

The syntax for the drop user statement is *Syntax*

drop [**existing**] **user** *username*

Description

The drop user statement is used to logically delete a user. If the **existing** keyword is being used, it is not considered to be an error if the specified user does not exist. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

10. finish commands

finish job

Purpose

Purpose The purpose of the finish job command is to inform the server about the termination of a job.

Syntax

Syntax The syntax for the finish job statement is

finish job *jobid*
with exit code = *signed_integer*

finish job
with exit code = *signed_integer*

Description

Description The finish job command is used by the job server to report the Exit Code for a process to the server. During the course of repair work, it may be necessary for an administrator to tell the server in this way that a job has terminated. Jobs can themselves report that they have finished. To do this, they connect to the server and use the second form of the statement.

Output

Output This statement returns a confirmation of a successful operation

11. get commands

get parameter

Purpose

Purpose The purpose of the get parameter statement is to get the value of the specified parameter within the context of the requesting job, respectively the specified job.

Syntax

Syntax The syntax for the get parameter statement is

get parameter *parametername* [< **strict** | **warn** | **liberal** >]

get parameter of *jobid parametername* [< **strict** | **warn** | **liberal** >]

Description

Description The get parameter statement is used to get the value of the specified parameter within the context of a job.
The additional option has the following meaning:

Option	Meaning
strict	The server returns an error if the requested parameter is not explicitly declared in the job definition.
warn	A message is written to the server's log file when an attempt is made to determine the value of an undeclared parameter.
liberal	An attempt to query an undeclared parameter is tacitly allowed.

The default behaviour depends on the configuration of the server.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
VALUE	Value of the requested parameter

Table 11.1.: Description of the output structure of the get parameter statement

get submittag

Purpose

The purpose of the get submittag statement is to get a (server local) unique identifier from the server. This identifier can be used to avoid race conditions between frontend and backend when submitting jobs. *Purpose*

Syntax

The syntax for the get submittag statement is *Syntax*

get submittag

Description

The get submittag statement is used to acquire an identification from the server. This prevents race conditions between the front end and back end when jobs are submitted. *Description*
Such a situation arises when feedback about the submit does not reach the front end due to an error. By using a submittag, the front end can safely start a second attempt. The server recognises whether the job in question has already been submitted and responds accordingly. This reliably prevents the job from being submitted twice.

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
VALUE	The requested Submit Tag

Table 11.2.: Description of the output structure of the get submittag statement

12. kill commands

kill session

Purpose

Purpose The purpose of the kill session is to terminate the specified session.

Syntax

Syntax The syntax for the kill session statement is

kill session *sid*

Description

Description The list session command can be used to display a list of active sessions. The displayed session ID can be used to terminate the session in question with the kill session command. Only administrators (i.e. members of the Admin group) are allowed to use this statement. It is not possible to terminate your own session.

Output

Output This statement returns a confirmation of a successful operation

13. link commands

link resource

Purpose

Purpose The purpose of the link resource statement is to create a reference to a resource os another scope.

Syntax

Syntax The syntax for the link resource statement is

link resource *resourcepath* **in** *serverpath* **to** < **scope** *serverpath* | **job server** *serverpath* > [**force**]

Description

Description With the link resource statement it is possible to make the resource of another scope visible and usable in a scope. This is necessary if a logical process requires resources from more than one scope. This is very well the case, for example, with processes that communicate with a database system.
From the system's perspective, it can scarcely differentiate between a Resource Link and the referenced resource. All operations such as allocating, locking, reading or setting variables take place on the base resource. This means that the link behaves as if it were the base resource. The only difference lies in the view of the allocations. With the base resource, all the allocations are shown. With a link, only those allocations that take place via the link are shown.
It is also possible to set links to links.
The **force** option can be used to overwrite an existing link. An already existing resource is deleted and the link is created. These operations are obviously only possible if the resource or link is not being used, i.e. if there are no allocations or reservations present.

Output

Output This statement returns a confirmation of a successful operation

14. list commands

list calendar

Purpose

Purpose The purpose of the list calendar statement is to get an overview of scheduled jobs.

Syntax

Syntax The syntax for the list calendar statement is

list calendar [**with** LC_WITHITEM {**,** LC_WITHITEM}]

LC_WITHITEM:
endtime = *datetime*
| **filter =** LC_FILTERTERM {**or** LC_FILTERTERM}
| **starttime =** *datetime*
| **time zone =** *string*

LC_FILTERTERM:
LC_FILTERITEM {**and** LC_FILTERITEM}

LC_FILTERITEM:
(LC_FILTERTERM {**or** LC_FILTERTERM} **)**
| **job .** *identifier* < **cmpop** | **like** | **not like** > RVALUE
| **name like** *string*
| **not (** LC_FILTERTERM {**or** LC_FILTERTERM} **)**

RVALUE:
expr (*string* **)**
| *number*
| *string*

Description

Description The list calendar statement gives you a list of all the calendar entries sorted by the start dates of the executable objects.
If a period is specified, those objects whose start time plus the Expected Final Time lies in the selected period are also displayed.

Output

Output This statement returns a confirmation of a successful operation

list dependency definition

Purpose

The purpose of the list dependency definition statement is to get a list of all dependencies of a job definition. *Purpose*

Syntax

The syntax for the list dependency definition statement is *Syntax*

list dependency definition *folderpath*

Description

The list dependency definition statement gives you a list of all the dependencies of a job definition. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
SE_DEPENDENT_PATH	The folder containing the dependent Scheduling Entity.
DEPENDENT_NAME	The name of the dependent Scheduling Entity.
SE_REQUIRED_PATH	The folder containing the required Scheduling Entity.
REQUIRED_NAME	The name of the required Scheduling Entity.
NAME	The object name
UNRESOLVED_HANDLING	The Unresolved Handling field describes what to do if a dependent object instance is not present in the current Master Batch. The following options are available: Ignore, Error and Suspend

Continued on next page

Continued from previous page

Field	Description
MODE	The Dependency Mode states the context in which the list of dependencies has to be viewed. The following options are available: All and Any
STATE_SELECTION	The State Selection defines how the required Exit States are determined. The options here are FINAL, ALL_REACHABLE, UNREACHABLE and DEFAULT. In the case of FINAL, the required Exit States can be explicitly listed.
ALL_FINALS	This field defines whether the dependency is already fulfilled when a Final State is reached (True) or if the required states are explicitly listed (False).
CONDITION	The condition that has to be fulfilled is entered in the Condition field.
STATES	This is the list of all the valid Exit States which the required object must have for the dependency to be fulfilled and so that the dependent job can start.

Table 14.1.: Description of the output structure of the list dependency definition statement

list dependency hierarchy

Purpose

The purpose of the list dependency hierarchy statement is to get a list of all dependencies of a submitted entity. *Purpose*

Syntax

The syntax for the list dependency hierarchy statement is *Syntax*

list dependency hierarchy *jobid* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

The list dependency hierarchy statement gives you a list of all the dependencies of a Submitted Dependency. *Description*

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The ID of the Dependency Instance.
DD_ID	The ID of the Dependency Definition.
DEPENDENT_ID	This is the ID of the dependent job.
DEPENDENT_NAME	This is the fully qualified name of the dependent job.

Continued on next page

Continued from previous page

Field	Description
REQUIRED_ID	This is the ID of the required job.
REQUIRED_NAME	This is the fully qualified name of the required job.
DEP_STATE	This is the current state of the dependency relationship. The following variants are used: Open, Fulfilled and Filed
DEPENDENCY_PATH	This is a ';'-separated list of job hierarchies (parent-child relationships). Each job hierarchy is a list of path names separated by a colon (':').
SE_DEPENDENT_ID	The ID of the dependent Scheduling Entity.
SE_DEPENDENT_NAME	The fully qualified name of the dependent Scheduling Entity.
SE_REQUIRED_ID	The ID of the required Scheduling Entity.
SE_REQUIRED_NAME	The fully qualified name of the required Scheduling Entity.
DD_NAME	This is the name of the Dependency Definition.
UNRESOLVED_HANDLING	This field shows the setting of the On Unresolved Error field from the submit mask. The following options are available: Ignore, Suspend and None
MODE	States the currently used Dependency Mode (ALL_FINAL or JOB_FINAL).
STATE_SELECTION	The State Selection defines how the required Exit States are determined. The options here are FINAL, ALL_REACHABLE, UNREACHABLE and DEFAULT. In the case of FINAL, the required Exit States can be explicitly listed.
MASTER_ID	This is the ID of the Master Job that was submitted in order to create this runtime object.
SE_TYPE	This is the Scheduling Entity type (job, batch or milestone).
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.
PARENT_NAME	This is the fully qualified name of the parent runtime object that submitted the current job.
OWNER	The group owning the object

Continued on next page

Continued from previous page

Field	Description
SCOPE	This is the fully qualified name of the job server on which the job was started. If the job has not yet been started, 'null' is displayed here.
EXIT_CODE	The Exit Code is the exit value that the Run program had when the process finished.
PID	This is the process ID of the Job Executor.
EXTPID	This is the ID of the process that is being executed.
JOB_STATE	The current Job State.
JOB_ESD	This is the job's Exit State. If the job has not yet finished, 'null' is displayed here.
FINAL_ESD	This is the Merged Exit State.
JOB_IS_FINAL	Specifies whether the job is Final (True) or not (False).
CNT_REQUIRED	The number of jobs that are dependent on the current job if its status is **dependency_wait**.
CNT_RESTARTABLE	The number of Child Jobs in the state **restartable.**
CNT_SUBMITTED	The number of Child Jobs in the state **submitted.**
CNT_DEPENDENCY_WAIT	The number of Child Jobs in the state **dependency_wait.**
CNT_RESOURCE_WAIT	The number of Child Jobs in the state **resource_wait.**
CNT_RUNNABLE	The number of Child Jobs in the state **runnable.**
CNT_STARTING	The number of Child Jobs in the state **starting.**
CNT_STARTED	The number of Child Jobs in the state **started.**
CNT_RUNNING	The number of Child Jobs in the state **running.**
CNT_TO_KILL	The number of Child Jobs in the state **to_kill.**
CNT_KILLED	The number of Child Jobs in the state **killed.**
CNT_CANCELLED	The number of Child Jobs in the state **cancelled.**
CNT_FINAL	The number of Child Jobs in the state **final.**
CNT_BROKEN_ACTIVE	The number of Child Jobs in the state **broken_active.**
CNT_BROKEN_FINISHED	The number of Child Jobs in the state **broken_finished.**
CNT_ERROR	The number of Child Jobs in the state **error.**
CNT_SYNCHRONIZE_WAIT	The number of Child Jobs in the state **synchronize_wait.**

Continued on next page

Continued from previous page

Field	Description
CNT_FINISHED	The number of Child Jobs in the state **finished.**
SUBMIT_TS	The time when the job was submitted.
SYNC_TS	The time when the job reached the state Synchronize_Wait.
RESOURCE_TS	
RUNNABLE_TS	The time when the job reached the state Runnable.
START_TS	The time when the job was started.
FINSH_TS	
FINAL_TS	
ERROR_MSG	The error message that was displayed on reaching the state Error.
DEPENDENT_ID_ORIG	The ID of the object that defined the dependency.
DEPENDENCY_OPERATION	The Dependency Operation defines whether all the dependencies (All) or just one single dependency have to be fulfilled.
CHILD_TAG	Marker for differentiating between multiple dynamically submitted children.
CHILDREN	The number of the children of the job.
REQUIRED	The number of dependent jobs.
DD_STATES	A comma-separated list of the required Exit States.
IS_SUSPENDED	This field defines whether the job is suspended (True) or not (False).
PARENT_SUSPENDED	This field defines whether the job is suspended (True) or not (False) through one of its parents.
CNT_UNREACHABLE	The number of children whose dependencies cannot be fulfilled.
DEPENDENT_PATH_ORIG	The fully qualified name of the object that defined the dependency.
IGNORE	Ignore indicates whether this dependency is ignored (True) or not (False).

Table 14.2.: Description of the output structure of the list dependency hierarchy statement

list environment

Purpose

The purpose of the list environment statement is to get a list of defined environments. *Purpose*

Syntax

The syntax for the list environment statement is *Syntax*

list environment

Description

The list environment statement is used to get a list of defined environments that are visible to the user. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The name of the environment.
PRIVS	String containing the users privileges on the object

Table 14.3.: Description of the output structure of the list environment statement

list event

Purpose

Purpose The purpose of the list event statement is to get a list of all defined events.

Syntax

Syntax The syntax for the list event statement is

list event

Description

Description The list event statement creates a list of all the defined events.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
SCHEDULING_ENTITY	Batch or job that is submitted when this event occurs.
PRIVS	String containing the users privileges on the object

Table 14.4.: Description of the output structure of the list event statement

list exit state definition

Purpose

The purpose of the list exit state definition statement is to get a list of all defined exit states. *Purpose*

Syntax

The syntax for the list exit state definition statement is *Syntax*

list exit state definition

Description

The list exit state definition statement gives you a list of all the Exit States. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.5.: Description of the output structure of the list exit state definition statement

list exit state mapping

Purpose

Purpose The purpose of the list exit state mapping statement is to get a list of all defined mappings.

Syntax

Syntax The syntax for the list exit state mapping statement is

list exit state mapping

Description

Description The list exit state mapping statement gives you a list of all the defined mappings.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.6.: Description of the output structure of the list exit state mapping statement

list exit state profile

Purpose

The purpose of the list exit state profile statement is to get a list of all defined exit state profiles. *Purpose*

Syntax

The syntax for the list exit state profile statement is *Syntax*

list exit state profile

Description

The list exit state profile statement gives you a list of all the defined Exit State Profiles. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
DEFAULT_ESM_NAME	The default Exit State Mapping is active if the job itself does not define something else.
IS_VALID	Flag displayed showing the validity of this Exit State Profile.
PRIVS	String containing the users privileges on the object

Table 14.7.: Description of the output structure of the list exit state profile statement

list folder

Purpose

Purpose The purpose of the list folder statement is to get a (partial) list of all folders defined in the system.

Syntax

Syntax The syntax for the list folder statement is

list folder *folderpath* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >
| FILTERTERM {**or** FILTERTERM}

FILTERTERM:
FILTERITEM {**and** FILTERITEM}

FILTERITEM:
(FILTERTERM {**or** FILTERTERM} **)**
| **name like** *string*
| **not (** FILTERTERM {**or** FILTERTERM} **)**
| **owner in (** *groupname* {**,** *groupname*} **)**

Description

Description The list folder statement gives you a list for the specified folder with all the direct child folders.

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

filter The child folders can be selected by name. Refer to the official Java documentation for the exact syntax used for regular expressions. The various conditions can be combined with one another using **and** and **or**. The usual valuation order of the operators applies (**and** before **or**).

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
TYPE	This states the type of object. The following options are available: Batch, Milestone, Job and Folder.
RUN_PROGRAM	A command line that starts the script or program can be specified in the Run Program field.
RERUN_PROGRAM	The Rerun Program field specifies the command that is to be executed when repeating the job following an error (rerun).
KILL_PROGRAM	The Kill Program field determines which program is to be run to terminate a currently running job.
WORKDIR	This is the working directory of the current job.
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).
TRUNC_LOG	Defines whether the log file is to be renewed or not.
ERRLOGFILE	The Error Logfile field specifies the file in which all the error outputs from the Run program are to be returned.
TRUNC_ERRLOG	Defines whether the Error log file is to be renewed or not.
EXPECTED_RUNTIME	The Expected Runtime describes the anticipated time that will be required to execute a job.
EXPECTED_FINALTIME	The Expected Finaltime describes the anticipated time that will be required to execute a job or batch together with its children.

Continued on next page

Continued from previous page

Field	Description
GET_EXPECTED_RUNTIME	This is a reserved field for future extended functions.
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.
MIN_PRIORITY	This is the minimum effective priority that can be achieved through natural aging.
AGING_AMOUNT	The number of time units after which the effective priority is incremented by 1.
AGING_BASE	The time unit that is used for the aging interval.
SUBMIT_SUSPENDED	Flag that indicates whether the object is to be suspended after the submit.
MASTER_SUBMITTABLE	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.
SAME_NODE	Obsolete
GANG_SCHEDULE	Obsolete
DEPENDENCY_MODE	The Dependency Mode states the context in which the list of dependencies has to be viewed. The following options are available: All and Any
ESP_NAME	This is the name of the Exit State Profile.
ESM_NAME	This is the name of the Exit State Mapping.
ENV_NAME	This is the name of the environment.
FP_NAME	This is the name of the footprint.
SUBFOLDERS	This is the number of folders below the folder.
ENTITIES	This is the number of jobs and batches below the folder.
HAS_MSE	The folder contains at least one job that can be executed as a Master Submittable job.
PRIVS	String containing the users privileges on the object
IDPATH	ID of the path to the object.
HIT	Line is a search hit Y/N

Table 14.8.: Description of the output structure of the list folder statement

list footprint

Purpose

The purpose of the list footprint statement is to get a list of all defined footprints. *Purpose*

Syntax

The syntax for the list footprint statement is *Syntax*

list footprint

Description

The list exit footprint statement gives you a list of all the defined footprints. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.9.: Description of the output structure of the list footprint statement

list group

Purpose

Purpose The purpose of the list group statement is to get a list of all defined groups.

Syntax

Syntax The syntax for the list group statement is

list group

Description

Description The list group statement gives you a list of all the defined groups.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.10.: Description of the output structure of the list group statement

list interval

Purpose

The purpose of the list interval statement is to get a list of all defined intervals. *Purpose*

Syntax

The syntax for the list interval statement is *Syntax*

list interval

Description

The list interval statement gives you a list of all the defined intervals. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
STARTTIME	The beginning of the interval. No edges are generated before this time.
ENDTIME	The end of the interval. No edges are generated after this time.
BASE	The period of the interval.
DURATION	The duration of a block.
SYNCTIME	The time with which the interval is synchronised. The first period of the interval starts at this time.
INVERSE	The definition whether the selection list should be regarded as being positive or negative.
EMBEDDED	The interval from which a selection is subsequently made.

Continued on next page

Continued from previous page

Field	Description
PRIVS	String containing the users privileges on the object

Table 14.11.: Description of the output structure of the list interval statement

list job

Purpose

The purpose of the list job statement is to get a list of submitted entities based on the selectioncriteria specified. *Purpose*

Syntax

The syntax for the list job statement is *Syntax*

list job [*jobid* {**,** *jobid*}] [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >
| FILTERTERM {**or** FILTERTERM}
| **mode =** < **list** | **tree** >
| **parameter = (** *parametername* {**,** *parametername*} **)**

FILTERTERM:
FILTERITEM {**and** FILTERITEM}

FILTERITEM:
(FILTERTERM {**or** FILTERTERM} **)**
| < **final** | **restartable** | **pending** >
| **exit state in (** *statename* {**,** *statename*} **)**
| < **history** | **future** > **=** *period*
| **history between** *period* **and** *period*
| **job .** *identifier* < **cmpop** | **like** | **not like** > RVALUE
| **job in (** *jobid* {**,** *jobid*} **)**
| **job server in (** *serverpath* {**,** *serverpath*} **)**
| **job status in (** JOBSTATE {**,** JOBSTATE} **)**
| **master**
| **master_id in (** *jobid* {**,** *jobid*} **)**
| **merged exit state in (** *statename* {**,** *statename*} **)**
| **name in (** *folderpath* {**,** *folderpath*} **)**
| **name like** *string*
| **node in (** *nodename* {**,** *nodename*} **)**
| **not (** FILTERTERM {**or** FILTERTERM} **)**
| **owner in (** *groupname* {**,** *groupname*} **)**
| **submitting user in (** *groupname* {**,** *groupname*} **)**
| **warning**

RVALUE:

expr (*string* **)**
| *number*
| *string*

JOBSTATE:

broken active
| **broken finished**
| **cancelled**
| **dependency wait**
| **error**
| **final**
| **finished**
| **killed**
| **resource wait**
| **runnable**
| **running**
| **started**
| **starting**
| **submitted**
| **synchronize wait**
| **to kill**
| **unreachable**

Description

Description The list job statement gives you a list of Submitted Entities. The selection of the jobs can be finely specified as required that by defining a filter. Job parameter names can also be specified that are then visible in the output.
The statement `list job` without any further information is equivalent to the statement `list job with master` and therefore outputs the list of all the Master Jobs and Batches.

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

mode **list** mode just outputs a list of selected jobs. If the **tree** mode is defined, however, all the parents for each selected job are outputted as well.

parameter Additional information about the selected jobs can be outputted by specifying parameter names. The parameters are valuated in the context of each job

and the value of the parameter is displayed in the output. If this fails, the output is an empty string. This means that specifying non-existent parameter names does not have any adverse consequences.
This allows state or progress details for jobs to be easily and clearly are displayed.

filter A large number of filters are available for filtering all the jobs present in the system. The individual filters can be combined with one another using Boolean operators. The usual order of priority operator applies here.
The individual filter functions are briefly described here.

FINAL, RESTARTABLE, PENDING This filter selects all the jobs in the state **final** respectively **restartable** or **pending**
.

EXIT STATE All jobs that are in an Exit State defined in the specified list are selected. This is the job's own Exit State, and not the Merged Exit State which also takes the Exit States of the children into consideration.

HISTORY By defining a history, only those jobs that have become final at the earliest before the given time are selected. All non-final jobs are selected.

FUTURE Scheduled future jobs are also outputted by specifying a future. These events are determined based on Scheduled Events and calendar entries. 'SCHEDULED" is outputted as the state of such jobs.

JOB.IDENTIFIER This filter is used to select all those jobs whose defined parameters fulfil the condition. This allows all the jobs of a developer to be easily selected, for example. (This obviously assumes that each job has a parameter with the developer's name).
The **expr** Function can be used to perform calculations The expression

```
job.starttime < expr('job.sysdate - job.expruntime * 1.5')
```

determines those jobs, that exceeded their expected runtime by more than 50%.

JOB IN (ID, . . .) This filter option is equivalent to specifying Jobids after "**list job**". Only those jobs with one of the specified IDs are selected.

JOB SERVER Only those jobs running on the specified job server are selected.

JOB STATE This filter selects only those jobs that have one of the specified job states. For example, it is then easy to find all the jobs in the state **broken_finished**.

MASTER Only the Master Jobs and Batches are selected.

MASTER_ID Only jobs that belong to the specified Master Jobs and Batches are selected.

MERGED EXIT STATE All jobs that are in a Merged Exit State defined in the specified list are selected. This is the Exit State that results from a job's own Exit State in combination with the Exit States of the children.

NAME IN (FOLDERPATH, . . .) The jobs whose associated Scheduling Entity is included in the specified list are selected.

NAME LIKE STRING The jobs whose associated Scheduling Entity has the matching name are selected. Refer to the official Java documentation for more details about the syntax used for regular expressions.

NODE Jobs running on one of the specified nodes are selected. In this context, the node designates the entry for the **node** of the job server.

OWNER Only the jobs of the defined owners (groups) are selected.

SUBMITTING USER Only jobs that have been submitted by the specified user are selected.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
MASTER_ID	This is the ID of the Master Job.
HIERARCHY_PATH	The Hierarchy Path is the full path of the current entry. The single hierarchy levels are separated by a period.
SE_TYPE	This is the Scheduled Entity type.
PARENT_ID	This is the ID of the parent.
OWNER	The group owning the object
SCOPE	The scope or job server to which the job is allocated.
HTTPHOST	The host name of the scope for accessing log files via HTTP.
HTTPPORT	The HTTP port number of the job server for accessing log files via HTTP.
EXIT_CODE	The Exit Code of the executed process.
PID	The Pid is the process identification number of the monitoring job server process on the respective host system.
EXTPID	The Extpid is the process identification number of the utility process.
STATE	The State is the current state of the job.
IS_DISABLED	Indicates whether the submitted entitity was submitted disabled.

Continued on next page

Continued from previous page

Field	Description
IS_CANCELLED	Indicates whether a Cancel operation was performed on the job.
JOB_ESD	The job_esd is the Exit State of the job.
FINAL_ESD	The final_esd is the Merged Exit State of the job or batch job with all the Child Exit States.
JOB_IS_FINAL	This field defines whether the job itself is final.
CNT_RESTARTABLE	These are the children of the job that are in a restartable state.
CNT_SUBMITTED	The number of Child Jobs in a submitted state.
CNT_DEPENDENCY_WAIT	The number of Child Jobs in a Dependency Wait state.
CNT_SYNCHRONIZE_WAIT	The number of Child Jobs in a Synchronize Wait state.
CNT_RESOURCE_WAIT	The number of Child Jobs in a Resource Wait state.
CNT_RUNNABLE	The number of Child Jobs in the state **runnable.**
CNT_STARTING	The number of Child Jobs in the state **starting.**
CNT_STARTED	The number of Child Jobs in the state **started.**
CNT_RUNNING	The number of Child Jobs in the state **running.**
CNT_TO_KILL	The number of Child Jobs in the state **to_kill.**
CNT_KILLED	The number of Child Jobs in the state **killed.**
CNT_CANCELLED	The number of Child Jobs in the state **cancelled.**
CNT_FINISHED	The number of Child Jobs in the state **finished.**
CNT_FINAL	The number of Child Jobs in the state **final.**
CNT_BROKEN_ACTIVE	The number of Child Jobs in the state **broken_active.**
CNT_BROKEN_FINISHED	The number of Child Jobs in the state **broken_finished.**
CNT_ERROR	The number of Child Jobs in the state **error.**
CNT_UNREACHABLE	The number of Child Jobs that have become unreachable.
CNT_WARN	The number of Child Jobs with a warning.
SUBMIT_TS	This is the time when the job is submitted.
RESUME_TS	The time when the job is automatically resumed.
SYNC_TS	The time when the job switched to the state **synchronize_wait**.

Continued on next page

Continued from previous page

Field	Description
RESOURCE_TS	The time when the job switched to the state **resource_wait**.
RUNNABLE_TS	The time when the job switched to the state **runnable**.
START_TS	The time when the job was reported by the job server as having been started.
FINISH_TS	This is the time when the job is finished.
FINAL_TS	The time when the job reached a final state.
PRIORITY	The static priority of a job. This is derived from the defined priority and the nice values of the parent(s).
DYNAMIC_PRIORITY	The dynamic priority of the job. This is the static priority that was corrected dependent on the delay time.
NICEVALUE	The nice value is the correction of the children's priority.
MIN_PRIORITY	This is the minimum value for the dynamic priority.
AGING_AMOUNT	The Aging Amount defines after how many time units the dynamic priority of a job is incremented by one point.
AGING_BASE	The Aging Base defines the time unit for the Aging Amount.
ERROR_MSG	The error message describing why the job switched to the **error** state.
CHILDREN	The number of children of the job or batch.
HIT	This field indicates whether the job was selected based on filter criteria or not.
HITPATH	This field indicates that the job is a direct or indirect parent of a selected job.
SUBMITPATH	This is the list of submitting parents. In contrast to the general parent-child hierarchy, this is always unequivocal.
IS_SUSPENDED	This field defines whether the job or batch itself is suspended.
IS_RESTARTABLE	This field defines whether the job is restartable.
PARENT_SUSPENDED	This field defines whether the job or batch is suspended through one of its parents.

Continued on next page

Continued from previous page

Field	Description
CHILDTAG	The tag that enables a differentiation to be made between multiple children.
IS_REPLACED	This field defines whether the job or batch has been replaced by another one.
WARN_COUNT	This is the number of unattended warnings.
CHILD_SUSPENDED	The number of children that have been suspended.
CNT_PENDING	The number of children in a pending state.
PRIVS	String containing the users privileges on the object
WORKDIR	Name of the working directory of the utility process.
LOGFILE	Name of the utility process log file. The output to `stdout` is written in this log.
ERRLOGFILE	Name of the utility process error log file. The output to `stderr` is written in this log.

Table 14.12.: Description of the output structure of the list job statement

list job definition hierarchy

Purpose

Purpose The purpose of the list job definition hierarchy statement is to get the complete jobtree of the specified job.

Syntax

Syntax The syntax for the list job definition hierarchy statement is

list job definition hierarchy *folderpath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The list job definition statement hierarchy gives you the complete tree structure of the specified job.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
TYPE	This states the type of object. The following options are available: Batch, Milestone, Job and Folder.
RUN_PROGRAM	A command line that starts the script or program can be specified in the Run Program field.
RERUN_PROGRAM	The Rerun Program field specifies the command that is to be executed when repeating the job following an error (rerun).

Continued on next page

Continued from previous page

Field	Description
KILL_PROGRAM	The Kill Program field determines which program is to be run to terminate a currently running job.
WORKDIR	This is the working directory of the current job.
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).
TRUNC_LOG	Defines whether the log file is to be renewed or not.
ERRLOGFILE	The Error Logfile field specifies the file in which all the error outputs from the Run program are to be returned.
TRUNC_ERRLOG	Defines whether the Error log file is to be renewed or not.
EXPECTED_RUNTIME	The Expected Runtime describes the anticipated time that will be required to execute a job.
GET_EXPECTED_RUNTIME	This is a reserved field for future extended functions.
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.
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: Yes, No and Childsuspend
MASTER_SUBMITTABLE	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.
SAME_NODE	Obsolete
GANG_SCHEDULE	Obsolete
DEPENDENCY_MODE	The Dependency Mode states the context in which the list of dependencies has to be viewed. The following options are available: All and Any

Continued on next page

Continued from previous page

Field	Description
ESP_NAME	This is the name of the Exit State Profile.
ESM_NAME	This is the name of the Exit State Mapping.
ENV_NAME	This is the name of the environment.
FP_NAME	This is the name of the footprint.
CHILDREN	This is the number of direct children.
SH_ID	The ID of the Hierarchy Definition.
IS_STATIC	Flag indicating the static or dynamic submits of this job.
IS_DISABLED	Flag indicating the the child should be executed or skipped
SH_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.
SH_SUSPEND	The Submit Suspended switch can be used to delay the actual start of a job run.
SH_ALIAS_NAME	A child can be assigned a new logical name by entering it in the Alias field.
MERGE_MODE	The Merge Mode indicates whether a Child Object is started multiple times within a Master Job run or not. The following options are available: No Merge, Failure, Merge Local and Merge Global
EST_NAME	This is the Exit State Translation.
IGNORED_DEPENDENCIES	Here you can add a list of dependencies which are to be ignored by the child in this parent-child relationship.
HIERARCHY_PATH	The Path describes the parent folder hierarchy of an object. All the parent folders are displayed separated by periods.
STATES	The State is the current state of the job.
PRIVS	String containing the users privileges on the object

Table 14.13.: Description of the output structure of the list job definition hierarchy statement

list named resource

Purpose

The purpose of the list named resource statement is to get a (partial) list of all defined named resources. *Purpose*

Syntax

The syntax for the list named resource statement is *Syntax*

list named resource [*resourcepath*] [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >
| FILTERTERM {**or** FILTERTERM}

FILTERTERM:
FILTERITEM {**and** FILTERITEM}

FILTERITEM:
(FILTERTERM {**or** FILTERTERM} **)**
| **name like** *string*
| **not (** FILTERTERM {**or** FILTERTERM} **)**
| **usage in (** RESOURCE_USAGE {**,** RESOURCE_USAGE} **)**

RESOURCE_USAGE:
category
| **static**
| **synchronizing**
| **system**

Description

The list named resource statement gives you a list of all the defined Named Resources. If a resource is specified, this Named Resource and, if the Named Resource is a category, all the children are listed. The list of Named Resources can be shortened accordingly by specifying a filter. *Description*

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

filter Named Resources can be filtered by name and/or usage by specifying filters. Refer to the official Java documentation for the syntax used for regular expressions.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
USAGE	The Usage field specifies the Resource type.
RESOURCE_STATE_PROFILE	This is the Resource State Profile assigned to the resource.
FACTOR	This is the default factor by which Resource Requirement Amounts are multiplied if nothing else has been specified for the resource.
SUBCATEGORIES	This is the number of categories that are present as children below the displayed Named Resources.
RESOURCES	These are the instances of the Named Resource.
PRIVS	String containing the users privileges on the object

Table 14.14.: Description of the output structure of the list named resource statement

list resource state definition

Purpose

The purpose of the list resource state definition is to get a list of all defined resource states. *Purpose*

Syntax

The syntax for the list resource state definition statement is *Syntax*

list resource state definition

Description

The list resource state definition statement gives you a list of all the defined Resource States. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.15.: Description of the output structure of the list resource state definition statement

list resource state mapping

Purpose

Purpose The purpose of the list resource state mapping statement is to get a list of all defined resource state mappings.

Syntax

Syntax The syntax for the list resource state mapping statement is

list resource state mapping

Description

Description

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 14.16.: Description of the output structure of the list resource state mapping statement

list resource state profile

Purpose

The purpose of the list resource state profile statement is to get a list of all currently defined resource state profiles. *Purpose*

Syntax

The syntax for the list resource state profile statement is *Syntax*

list resource state profile

Description

The list resource state profile statement gives you a list of all the defined Resource State Profiles. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
INITIAL_STATE	This field defines the initial state of the resource. This Resource State does not have to be present in the list of valid Resource States.
PRIVS	String containing the users privileges on the object

Table 14.17.: Description of the output structure of the list resource state profile statement

list schedule

Purpose

Purpose The purpose of the list schedule statement is to get a (partial) list of all defined schedules.

Syntax

Syntax The syntax for the list schedule statement is

list schedule *schedulepath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The list schedule statement delivers a list with the specified schedule and all its children.

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
INTERVAL	The name of the interval belonging to the schedule.

Continued on next page

Continued from previous page

Field	Description
TIME_ZONE	The time zone in which the schedule is to be calculated.
ACTIVE	This field defines whether the schedule is marked as being active.
EFF_ACTIVE	This field defines whether the schedule is actually active. This can deviate from ”active” due to the hierarchical organisation.
PRIVS	String containing the users privileges on the object

Table 14.18.: Description of the output structure of the list schedule statement

list scheduled event

Purpose

Purpose The purpose of the list scheduled event is to get a list of all defined scheduled events.

Syntax

Syntax The syntax for the list scheduled event statement is

list scheduled event

Description

Description The list scheduled event statement gives you a list of all the defined Scheduled Events.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
OWNER	The group owning the object
SCHEDULE	The Schedule that determines when the Scheduled Event is to take place.
EVENT	The event that is triggered.
ACTIVE	This flag indicates whether the Scheduled Event is labelled as being active.
EFF_ACTIVE	This flag indicates whether the Scheduled Event is actually active.
BROKEN	The Broken field can be used to check whether an error occurred when the job was submitted.
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.

Continued on next page

Continued from previous page

Field	Description
ERROR_MSG	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.
LAST_START	The last time the job is to be executed by the Scheduling System is shown here.
NEXT_START	The next scheduled time when the task is to be executed by the Scheduling System is shown here.
NEXT_CALC	The next time when a recalculation is to take place.
PRIVS	String containing the users privileges on the object
BACKLOG_HANDLING	The Backlog Handling describes how events that should have been triggered following a downtime are to be handled.
SUSPEND_LIMIT	The Suspend Limit defines the delay after which a job is submitted in a suspended state.
EFFECTIVE_SUSPEND_LIMIT	The Suspend Limit defines the delay after which a job is submitted in a suspended state.
CALENDAR	This flag indicates whether calendar entries are created.
CALENDAR_HORIZON	The defined length of the period in days for which a calendar is created.
EFFECTIVE_CALENDAR_HORIZON	The effective length of the period in days for which a calendar is created.

Table 14.19.: Description of the output structure of the list scheduled event statement

list scope

Purpose

Purpose The purpose of the list scope statement is to get a (partial) list of all defined scopes.

Syntax

Syntax The syntax for the list scope statement is

list < **scope** *serverpath* | **job server** *serverpath* > [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The list scope statement displays a list with the requested scope together with its children.

expand The **expand** option can be used to make the hierarchy visible at children level. This is done by specifying in the list the IDs of the nodes whose children are to be made visible. If **none** is specified as an expand option, only the level below the requested node is made visible.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
TYPE	The type of scope.
IS_TERMINATE	This flag indicates whether a termination order exists.

Continued on next page

Continued from previous page

Field	Description
HAS_ALTERED_CONFIG	The configuration on the server does not match the current configuration on the server.
IS_SUSPENDED	Indicates whether the scope is suspended.
IS_ENABLED	The job server can only log on to the server if the Enable flag is set to Yes.
IS_REGISTERED	Defines whether the job server has sent a register command.
IS_CONNECTED	Indicates whether the job server is connected.
STATE	This is the current state of the resource in this scope.
PID	The PID is the process identification number of the job server process on the respective host system.
NODE	The node specifies the computer on which the job server is running. This field has a purely documentary character.
IDLE	The time that has elapsed since the last command. This only applies for job servers.
NOPDELAY	The time that a job server waits for "NOP".
ERRMSG	This is the most recently outputted error message.
SUBSCOPES	The number of scopes and job servers that are present under this scope.
RESOURCES	The resources present in this scope are displayed here.
PRIVS	String containing the users privileges on the object

Table 14.20.: Description of the output structure of the list scope statement

list session

Purpose

Purpose The purpose of the list session statement is to get a list of connected sessions.

Syntax

Syntax The syntax for the list session statement is

list session

Description

Description The list session statement gives you a list of the connected sessions.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
THIS	The current session is indicated in this field by an asterisk (*).
SESSIONID	The internal server ID for the session.
PORT	The TCP/IP port number at which the session is connected.
START	Time when the connection was set up.
TYPE	Type of connection: user, job server or job
USER	Name of the connecting user, job server or job (Jobid)
UID	ID of the user, job server or job.
IP	IP address of the connecting sessions.
TXID	Number of the last transaction that was executed by the session.
IDLE	The number of seconds since the last statement from a session.
STATE	The state of the session. This is one of the following: IDLE (no activity), QUEUED (statement is waiting to be executed), ACTIVE (statement is being executed), COMMITTING (changes to a write transaction are being written), CONNECTED (not yet authenticated).
TIMEOUT	The idle time after which the session is automatically disconnected.
INFORMATION	Additional information about the session (optional).
STATEMENT	The statement that is currently being executed.
WAIT	This field is not yet documented

Table 14.21.: Description of the output structure of the list session statement

list trigger

Purpose

Purpose The purpose of the list trigger statement is to get a list of defined trigger.

Syntax

Syntax The syntax for the list trigger statement is

list trigger

list trigger for *folderpath*

list trigger of *folderpath*

list trigger for CT_OBJECT

CT_OBJECT:
job definition *folderpath*

Description

Description The list trigger statement gives you a list of all the defined triggers.

Output

Output This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OBJECT_TYPE	The type of object in which the trigger is defined.
OBJECT_SUBTYPE	The subtype of the object in which the trigger is defined.

Continued on next page

Continued from previous page

Field	Description
OBJECT_NAME	Full path name of the object in which the trigger is defined.
ACTIVE	The flag indicates whether the trigger is currently active.
ACTION	Type of triggered action: SUBMIT or RERUN.
STATES	A list of states that cause the trigger to be activated.
SUBMIT_TYPE	The object type that is submitted when the trigger is activated.
SUBMIT_NAME	Name of the job definition that is submitted.
SUBMIT_SE_OWNER	The owner of the object that is submitted.
SUBMIT_PRIVS	The privileges for the object that is to be submitted.
MAIN_TYPE	Type of main job (job/batch).
MAIN_NAME	Name of the main job.
MAIN_SE_OWNER	Owner of the main job.
MAIN_PRIVS	Privileges for the main job.
PARENT_TYPE	Type of parent job (job/batch).
PARENT_NAME	Name of the parent job.
PARENT_SE_OWNER	Owner of the parent job.
PARENT_PRIVS	Privileges for the parent job.
TRIGGER_TYPE	The trigger type that describes when it is activated.
MASTER	Indicates whether the trigger submitted a master or a child.
IS_INVERSE	Indicates whether the trigger submitted a master or a child.
SUBMIT_OWNER	In case of an inverse trigger, the trigger is regarded to belong to the triggered job. The trigger can be regarded as some kind of callback function. This flag has no effects on the trigger's behaviour.
IS_CREATE	Indicates whether the trigger reacts to create events.
IS_CHANGE	Indicates whether the trigger reacts to change events.
IS_DELETE	Indicates whether the trigger reacts to delete events.

Continued on next page

Continued from previous page

Field	Description
IS_GROUP	Indicates whether the trigger handles the events as a group.
MAX_RETRY	The maximum number of trigger activations in a single Submitted Entity.
SUSPEND	Specifies whether the submitted object is suspended.
RESUME_AT	Time of the automatic resume.
RESUME_IN	Number of time units until the automatic resume.
RESUME_BASE	Time unit for resume_in.
WARN	Specifies whether a warning has to be given when the activation limit is reached.
LIMIT_STATE	This field specifies which state the triggering job aqcuires if the fire limit is reached. If the triggering job has a final state already, this specification is ignored. If the value is **NONE**, no state change takes place.
CONDITION	Conditional expression to define the trigger condition.
CHECK_AMOUNT	The amount of CHECK_Base units for checking the condition in the case of non-synchronised triggers.
CHECK_BASE	Units for the CHECK_AMOUNT.
PRIVS	String containing the users privileges on the object
TAG	This field is not yet documented
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 14.22.: Description of the output structure of the list trigger statement

list user

Purpose

The purpose of the list user statement is to get a list of all defined users. *Purpose*

Syntax

The syntax for the list user statement is *Syntax*

list user

Description

The list user statement gives you a list of all the defined users. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
IS_ENABLED	Flag that shows whether the user is allowed to log on.
DEFAULT_GROUP	The default group of the users who are being used by the owners of the object.
PRIVS	String containing the users privileges on the object

Table 14.23.: Description of the output structure of the list user statement

15. move commands

move folder

Purpose

Purpose The purpose of the move folder statement is to rename the folder and/or to move it to some other place in the folder hierarchy.

Syntax

Syntax The syntax for the move folder statement is

move folder *folderpath* **to** *folderpath*

Description

Description The move folder command either moves the specified folder to somewhere else or renames it.

Output

Output This statement returns a confirmation of a successful operation

move job definition

Purpose

Purpose

The purpose of the move job definition statement is to rename a scheduling entity object, and/or move it into some other folder.

Syntax

Syntax

The syntax for the move job definition statement is

move job definition *folderpath* **to** *folderpath*

Description

Description

The move job definition command moves the specified job definition to the specified folder. If the destination folder does not exist, the last part of the fully qualified name is interpreted as being the new name for the job definition. The relationships to other objects are not changed.

Output

Output

This statement returns a confirmation of a successful operation

move named resource

Purpose

Purpose The purpose of the move named resource statement is to rename the named resource and/or to move the resource into another category.

Syntax

Syntax The syntax for the move named resource statement is

move named resource *resourcepath* **to** *resourcepath*

Description

Description The move named resource statement is used to rename a Named Resource or to reorganise categories.
If a Named Resource is moved, the specified destination has to be a category or it must not exist and its parent must be a category.

Output

Output This statement returns a confirmation of a successful operation

move schedule

Purpose

The purpose of the move schedule statement is to rename and/or to move the schedule to some other place in the hierarchy. *Purpose*

Syntax

The syntax for the move schedule statement is *Syntax*

move schedule *schedulepath* **.** *schedulename* **to** *schedulepath*

Description

The move schedule command either moves the specified schedule to somewhere else and/or renames it. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

move scope

Purpose

Purpose The purpose of the move scope statement is to rename a scope and/or to move it to some other place within the scope hierarchy.

Syntax

Syntax The syntax for the move scope statement is

move < **scope** *serverpath* | **job server** *serverpath* > **to** *serverpath*

Description

Description The move scope command either moves the specified scope to somewhere else and/or renames it.

Output

Output This statement returns a confirmation of a successful operation

16. multicommand commands

multicommand

Purpose

Purpose This statement is used to control the behaviour of the SDMS Server.

Syntax

Syntax The syntax for the multicommand statement is

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Description

Description The multicommands allow multiple SDMS commands to be executed together, i.e. in one transaction. This ensures that either all the statements are executed without any errors or nothing happens at all. Not only that, but the transaction is not interrupted by other write transactions.
If the **rollback** keyword is specified, the transaction is undone at the end of the processing. This means that you can test whether the statements can be correctly processed (technically speaking).

Output

Output This statement returns a confirmation of a successful operation

17. register commands

register

Purpose

Purpose The purpose of the register statement is to notify the server that the jobserver is ready to process jobs.

Syntax

Syntax The syntax for the register statement is

register *serverpath* **.** *servername*
with pid = *pid* [**suspend**]

register with pid = *pid*

Description

Description The first form is used by the operator to enable jobs to be executed by the specified job server.
The second form is used by the job server itself to notify the server that it is ready to execute jobs.
Jobs are scheduled for this job server (unless it is suspended) regardless of whether the server is connected or not.
Refer to the deregister statement on page 144.

pid The pid option provides the server with information about the job server's process ID at operating level.

suspend The suspend option causes the job server to be transferred to a suspended state.

Output

Output This statement returns a confirmation of a successful operation

18. rename commands

rename environment

Purpose

Purpose The purpose of the rename environment statement is to give the specified environment another name.

Syntax

Syntax The syntax for the rename environment statement is

rename environment *environmentname* **to** *environmentname*

Description

Description The rename environment statement is used to rename environments. Renaming an environment does not have any effect on the functionality and is only for purposes of clarity.

Output

Output This statement returns a confirmation of a successful operation

rename event

Purpose

The purpose of the rename event is to give the specified event another name. *Purpose*

Syntax

The syntax for the rename event statement is *Syntax*

rename event *eventname* **to** *eventname*

Description

The rename event statement is used to give a specified event a different name. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename exit state definition

Purpose

Purpose The purpose of the rename exist state definition statement is to give the specified exit state definition another name.

Syntax

Syntax The syntax for the rename exit state definition statement is

rename exit state definition *statename* **to** *statename*

Description

Description The rename exit state definition statement is used to rename Exit State Definitions. Renaming an Exit State Definition does not have any effect on the functionality and is only for purposes of clarity.

Output

Output This statement returns a confirmation of a successful operation

rename exit state mapping

Purpose

The purpose of the rename exit state mapping statement is to give the specified mapping another name. *Purpose*

Syntax

The syntax for the rename exit state mapping statement is *Syntax*

rename exit state mapping *mappingname* **to** *profilename*

Description

The rename exit state mapping statement is used to rename Exit State Mappings. Renaming an Exit State Mapping does not have any effect on the functionality and is only for purposes of clarity. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename exit state profile

Purpose

Purpose The purpose of the rename exit state profile statement is to give the specified profile another name.

Syntax

Syntax The syntax for the rename exit state profile statement is

rename exit state profile *profilename* **to** *profilename*

Description

Description The rename exit state profile statement is used to rename Exit State Profiles. Renaming the Exit State Profiles does not have any effect on the functionality and is only for purposes of clarity.

Output

Output This statement returns a confirmation of a successful operation

rename folder

Purpose

The purpose of the rename folder statement is to give a folder another name. *Purpose*

Syntax

The syntax for the rename folder statement is *Syntax*

rename folder *folderpath* **to** *foldername*

Description

The rename folder command renames the specified folder. This is done within the same parent folder. If an object with the new name already exists, this triggers an error message. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename footprint

Purpose

Purpose The purpose of the rename footprint statement is to give the specified footprint another name.

Syntax

Syntax The syntax for the rename footprint statement is

rename footprint *footprintname* **to** *footprintname*

Description

Description The rename footprint statement is used to give a specified footprint a different name.

Output

Output This statement returns a confirmation of a successful operation

rename group

Purpose

The purpose of the rename group statement is to change the name of a group without affecting any other properties. *Purpose*

Syntax

The syntax for the rename group statement is *Syntax*

rename group *groupname* **to** *groupname*

Description

The rename group statement is used to rename groups. Renaming a group does not have any effect on the functionality and is only for purposes of clarity. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename interval

Purpose

Purpose The purpose of the rename interval statement is to give the specified interval another name.

Syntax

Syntax The syntax for the rename interval statement is

rename interval *intervalname* **to** *intervalname*

Description

Description The rename interval statement is used to give a specified interval a different name.

Output

Output This statement returns a confirmation of a successful operation

rename job definition

Purpose

The purpose of the rename job definition statement is to give the job definition another name. *Purpose*

Syntax

The syntax for the rename job definition statement is *Syntax*

rename job definition *folderpath* **to** *jobname*

Description

The job definition rename command renames the specified job definition. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename named resource

Purpose

Purpose The purpose of the rename named resource statement is to give a named resource another name.

Syntax

Syntax The syntax for the rename named resource statement is

rename named resource *resourcepath* **to** *resourcename*

Description

Description The rename named resource statement is used to rename a Named Resource.

Output

Output This statement returns a confirmation of a successful operation

rename resource state definition

Purpose

The purpose of the rename resource state definition statement is to rename the resource state. *Purpose*

Syntax

The syntax for the rename resource state definition statement is *Syntax*

rename resource state definition *statename* **to** *statename*

Description

The rename resource state definition statement is used to rename Resource State Definitions. Renaming a Resource State Definition does not have any effect on the functionality and is only for purposes of clarity. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename resource state mapping

Purpose

Purpose The purpose of the rename resource state mapping statement is to give the specified mapping a new name.

Syntax

Syntax The syntax for the rename resource state mapping statement is

rename resource state mapping *mappingname* **to** *profilename*

Description

Description The rename resource state mapping statement is used to rename Resource State Mappings. Renaming a Resource State Mapping does not have any effect on the functionality and is only for purposes of clarity.

Output

Output This statement returns a confirmation of a successful operation

rename resource state profile

Purpose

Purpose

The purpose of the rename resource state profile is to give the specified resource state profile a new name.

Syntax

Syntax

The syntax for the rename resource state profile statement is

rename resource state profile *profilename* **to** *profilename*

Description

Description

The rename resource state profile statement is used to rename Resource State Profiles. Renaming a Resource State Profile does not have any effect on the functionality and is only for purposes of clarity.

Output

Output

This statement returns a confirmation of a successful operation

rename schedule

Purpose

Purpose The purpose of the rename schedule statement is to give a schedule another name.

Syntax

Syntax The syntax for the rename schedule statement is

rename schedule *schedulepath* **.** *schedulename* **to** *schedulename*

Description

Description The rename schedule command renames the specified schedule.

Output

Output This statement returns a confirmation of a successful operation

rename scope

Purpose

The purpose of the rename scope statement is to give a scope another name. *Purpose*

Syntax

The syntax for the rename scope statement is *Syntax*

rename < **scope** *serverpath* | **job server** *serverpath* > **to** *scopename*

Description

The rename scope command renames the specified scope. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

rename trigger

Purpose

Purpose The purpose of the rename trigger statement is to give the specified trigger another name.

Syntax

Syntax The syntax for the rename trigger statement is

rename trigger *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >] **to** *triggername*

TRIGGEROBJECT:

resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

Description

Description The rename trigger statement is used to rename the trigger. Renaming a trigger does not have any effect on the functionality and is only for purposes of clarity.

Output

Output This statement returns a confirmation of a successful operation

rename user

Purpose

The purpose of the rename user statement is to change the name of a user without altering any other of its properties. *Purpose*

Syntax

The syntax for the rename user statement is *Syntax*

rename user *username* **to** *username*

Description

The rename user statement is used to rename users. Renaming a user does not have any effect on the functionality and is only for purposes of clarity. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

19. resume commands

resume

Purpose

Purpose The purpose of the resume statement is to reactivate the jobserver. See also the suspend statement on page 358.

Syntax

Syntax The syntax for the resume statement is

resume *serverpath*

Description

Description The resume statement is used to reactivate a job server.

Output

Output This statement returns a confirmation of a successful operation

20. select commands

select

Purpose

Purpose The purpose of the select statement is to enable the user to issue (almost) arbitrary queries to the underlying RDBMS.

Syntax

Syntax The syntax for the select statement is

select-statement [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
identifier **category**
| *identifier* **folder**
| *identifier* **job**
| *identifier* **scope**
| **sort (** *signed_integer* {**,** *signed_integer*} **)**

Description

Description The select statement allows practically any number of database select statements to be executed by the Scheduling Server. Refer to the documentation of the database system you are using for information about the syntax that is used for the select statement.
Since executing arbitrary select statements generally represents a vulnerability, administrator privileges are required for this statement. This means that only users belonging to the **admin** group are allowed to use this statement.
Using the with items causes IDs to be translated into names. This function is available for all hierarchically structured object types since this operation is not always easy to perform using SQL means. It is also possible to sort the set of results after replacing the IDs. The columns that are to be used for sorting are addressed according to their position in the set of results (zero-based, i.e. the first column has the number 0).

Output

Output This statement returns a confirmation of a successful operation

21. set commands

set parameter

Purpose

Purpose The purpose of the set parameter statement is to set the value of the specified parameters within the context of the requesting job, respectively the specified job.

Syntax

Syntax The syntax for the set parameter statement is

set parameter *parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,** *parametername* **=** *string*} **identified by** *string*

Description

Description The set parameter statements can be used to set jobs or user parameter values in the context of the job.
If the **identified by** option is specified, the parameter is only set if the pair *jobid* and *string* would allow a logon.

Output

Output This statement returns a confirmation of a successful operation

22. show commands

show comment

Purpose

Purpose The purpose of the show comment statement is to show the comment for the specified object.

Syntax

Syntax The syntax for the show comment statement is

show comment on OBJECTURL

OBJECTURL:
distribution *distributionname* **for pool** *resourcepath* **in** *serverpath*
| **environment** *environmentname*
| **exit state definition** *statename*
| **exit state mapping** *mappingname*
| **exit state profile** *profilename*
| **event** *eventname*
| **resource** *resourcepath* **in** *folderpath*
| **folder** *folderpath*
| **footprint** *footprintname*
| **group** *groupname*
| **interval** *intervalname*
| **job definition** *folderpath*
| **job** *jobid*
| **named resource** *resourcepath*
| **parameter** *parametername* **of** PARAM_LOC
| **resource state definition** *statename*
| **resource state mapping** *mappingname*
| **resource state profile** *profilename*
| **scheduled event** *schedulepath* **.** *eventname*
| **schedule** *schedulepath*
| **resource** *resourcepath* **in** *serverpath*
| < **scope** *serverpath* | **job server** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

PARAM_LOC:
folder *folderpath*
| **job definition** *folderpath*

| < **scope** *serverpath* | **job server** *serverpath* >

TRIGGEROBJECT:
resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

Description

Description

The show comment statement is used to display the saved comment for the specified object. If no comment on the object exists, this is not regarded as being an error; instead, an empty output structure is created and returned. This empty output structure naturally corresponds to the output structure described below, so that it can be easily evaluated by programs without any exception handling.

Output

Output

This statement returns an output structure of type table.

Main output description The data items of the output are described in the table below.

Field	Description
ID	System-wide unique object number.
TAG	The comment tag is a headline of the comment block. It is an optional field.
COMMENT	The comment on the specified object.
COMMENTTYPE	Type of comment, text or URL.
CREATOR	Name of the user who created this pool.
CREATE_TIME	The creation time.
CHANGER	Name of the last user who modified this pool.
CHANGE_TIME	Time of the last modification.
PRIVS	Abbreviation for the privileges for this object held by the requesting user.

Table 22.1.: Description of the output structure of the show comment statement

show environment

Purpose

Purpose The purpose of the show environment statement is to get detailed informatoion about the specified environment.

Syntax

Syntax The syntax for the show environment statement is

show environment *environmentname* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The show environment statement gives you detailed information about the specified environment.

expand Since the number of job definitions in the table JOB_DEFINITIONS can become very large, by default they are not all displayed. If the option **expand = all** is used, all the job definitions as well as their parent folder and the folder hierarchy are outputted. Individual paths in the hierarchy can be selected by specifying individual (folder) IDs.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the environment.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Continued on next page

Continued from previous page

Field	Description
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
RESOURCES	Table of static resources that form this environment. See also table 22.3 on page 269
JOB_DEFINITIONS	Table of jobs and folders that use this environment See also table 22.4 on page 270

Table 22.2.: Description of the output structure of the show environment statement

RESOURCES The layout of the RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
NR_NAME	Full path name of static Named Resources.
CONDITION	The condition that has to be fulfilled for the allocation.
PRIVS	String containing the users privileges on the object

Table 22.3.: Description of the output structure of the show environment subtable

JOB_DEFINITIONS The layout of the JOB_DEFINITIONS table is shown in the table below.

Field	Description
ID	The repository object id
SE_PATH	Full folder path name of job definitions or folders.
TYPE	The object type. The possible values are FOLDER and JOB_DEFINITION.
ENV	An asterisk indicates that the current environment was specified here.

Continued on next page

Continued from previous page

Field	Description
HAS_CHILDREN	true means that there are more environment users further down the tree.
PRIVS	String containing the users privileges on the object

Table 22.4.: Description of the output structure of the show environment subtable

show event

Purpose

The purpose of the show event statement is to get detailed information about the specified event. *Purpose*

Syntax

The syntax for the show event statement is *Syntax*

show event *eventname*

Description

The show event statement gives you detailed information about the specified event. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the show event
OWNER	The group owning the object
SCHEDULING_ENTITY	Batch or job that is submitted when this event occurs.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PARAMETERS	Parameters that are used when submitting the job or batch. See also table 22.6 on page 272
PRIVS	String containing the users privileges on the object

Continued on next page

Continued from previous page

Field	Description
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.5.: Description of the output structure of the show event statement

PARAMETERS The layout of the PARAMETERS table is shown in the table below.

Field	Description
ID	The repository object id
KEY	Name of the parameter.
VALUE	Value of the parameter.

Table 22.6.: Description of the output structure of the show event subtable

show exit state definition

Purpose

The purpose of the show exit state definition statement is to get detailed information about the specified exit state definition. *Purpose*

Syntax

The syntax for the show exit state definition statement is *Syntax*

show exit state definition *statename*

Description

The show exit state definition statement gives you detailed information about the specified Exit State Definition. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the Exit State Definition
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object

Table 22.7.: Description of the output structure of the show exit state definition statement

show exit state mapping

Purpose

Purpose The purpose of the show exist state mapping statement is to get detailed information about the specified mapping.

Syntax

Syntax The syntax for the show exit state mapping statement is

show exit state mapping *mappingname*

Description

Description The show exit state mapping statement gives you detailed information about the specified mapping.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
COMMENT	A comment that can be freely selected by the user.
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
RANGES	The assignment of the respective value ranges shown in a table. See also table 22.9 on page 275

Table 22.8.: Description of the output structure of the show exit state mapping statement

RANGES The layout of the RANGES table is shown in the table below.

Field	Description
ECR_START	Minimum limit of the range (inclusive)
ECR_END	Maximum limit of the range (inclusive)
ESD_NAME	Name of the Exit State to which this area is mapped.

Table 22.9.: Description of the output structure of the show exit state mapping subtable

show exit state profile

Purpose

Purpose The purpose of the show exist state profile statement is to get detailed information about the specified profile.

Syntax

Syntax The syntax for the show exit state profile statement is

show exit state profile *profilename*

Description

Description The show exit state profile statement gives you detailed information about the specified profile.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
DEFAULT_ESM_NAME	The default Exit State Mapping is active if the job itself does not define something else.
IS_VALID	Flag displayed showing the validity of this Exit State Profile.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object

Continued on next page

Continued from previous page	
Field	**Description**
STATES	Table contains Exit States that are valid for this profile. See also table 22.11 on page 277

Table 22.10.: Description of the output structure of the show exit state profile statement

STATES The layout of the STATES table is shown in the table below.

Field	Description
ID	The repository object id
PREFERENCE	The preference for controlling the connection of the Child Exit States.
TYPE	Indicates whether the state is FINAL, PENDING or RESTARTABLE.
ESD_NAME	Name of the Exit State Definition.
IS_UNREACHABLE	Indicates that this Exit State is used when a job is unreachable.
IS_BROKEN	Indicates that this Exit State is used when a job is broken.
IS_BATCH_DEFAULT	Indicates that this Exit State is used when a batch or milestone does not have any children.
IS_DEPENDENCY_DEFAULT	Indicates that this Exit State is used if the state selection DEFAULT was selected in the Dependency Definition.

Table 22.11.: Description of the output structure of the show exit state profile subtable

show folder

Purpose

Purpose The purpose of the show folder statement is to get detailed information about the specified folder.

Syntax

Syntax The syntax for the show folder statement is

show folder *folderpath*

Description

Description The show folder statement gives you detailed information about the specified folder.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the folder.
OWNER	The group owning the object
TYPE	The object type that is always contained in the folder.
ENVIRONMENT	The name of the optional environment.
INHERIT_PRIVS	Privileges that are inherited from the parent folder.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object

Continued on next page

Continued from previous page	
Field	**Description**
PARAMETERS	The parameters table shows all the defined constants for this folder. See also table 22.13 on page 280
DEFINED_RESOURCES	The Defined Resources table shows all the resource instances that are defined for this folder. See also table 22.14 on page 281

Table 22.12.: Description of the output structure of the show folder statement

PARAMETERS The layout of the PARAMETERS table is shown in the table below.

Field	**Description**
ID	The repository object id
NAME	Name of the parameter.
EXPORT_NAME	The export name defines the name under which the value of the parameter is exported to the process's environment.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
EXPRESSION	Expression for the parameter type EXPRESSION.
DEFAULT_VALUE	The default value of the parameter.
REFERENCE_TYPE	Type of object that is being referenced.
REFERENCE_PATH	The path to the object that is being referenced.
REFERENCE_PRIVS	The user's privileges for the object that is being referenced.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
ID	The repository object id
NAME	Name of the parameter.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.

Continued on next page

Continued from previous page

Field	Description
REFERENCE_TYPE	Type of object that is referencing the parameter.
REFERENCE_PATH	The path to the object that is referencing the parameter.
REFERENCE_PRIVS	The user's privileges for the object that is referencing the parameter.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.13.: Description of the output structure of the show folder subtable

DEFINED_RESOURCES The layout of the DEFINED_RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
NR_ID	ID of the Named Resource.
NAME	Name of the Named Resource.
USAGE	It is the usage of the Named Resource (STATIC, SYSTEM or SYNCHRONISING)
NR_PRIVS	String containing the abreviations for the user privileges for this Named Resource.
TAG	The tag is an optional short name for the resource.
OWNER	The group owning the object
LINK_ID	ID of the referenced resource.
LINK_SCOPE	Scope of the referenced resource.
STATE	The Resource State of the resource.
REQUESTABLE_AMOUNT	The maximum amount of resources that can be requested by a job.
AMOUNT	The actual amount that is available.
FREE_AMOUNT	The free amount that can be allocated.
TOTAL_FREE_AMOUNT	Free amount available for allocations including the free amount of pooled resources if it is a pool.

Continued on next page

Continued from previous page

Field	Description
IS_ONLINE	This is the availability status of the resource.
FACTOR	This is the correction factor by which the requested amount is multiplied.
TIMESTAMP	The timestamp indicates the last time the Resource State changed.
SCOPE	The scope in which the resource was created.
MANAGER_ID	ID of the Managing Pool.
MANAGER_NAME	Name of the Managing Pool.
MANAGER_SCOPENAME	Name of the scope in which the Managing Pool was created.
HAS_CHILDREN	Flag indicating whether a Pool Child has managed resources/pools. If it is not a pool, this is always FALSE.
POOL_CHILD	This flag indicates whether the displayed resource is a child of the pool.
TRACE_INTERVAL	The trace interval is the minimum time in seconds between when Trace Records are written.
TRACE_BASE	The trace base is the basis for the valuation period (B).
TRACE_BASE_MULTIPLIER	The base multiplier determines the multiplication factor (M) of the trace base.
TD0_AVG	The average resource allocation of the last $B * M^0$ seconds.
TD1_AVG	The average resource allocation of the last $B * M^1$ seconds.
TD2_AVG	The average resource allocation of the last $B * M^2$ seconds.
LW_AVG	The average allocation since the last time a Trace Record was written.
LAST_WRITE	The time the last Trace Record was written.
PRIVS	String containing the users privileges on the object

Table 22.14.: Description of the output structure of the show folder subtable

show footprint

Purpose

Purpose The purpose of the show footprint statement is to get detailed information about the specified footprint.

Syntax

Syntax The syntax for the show footprint statement is

show footprint *footprintname* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The show footprint statement gives you detailed information about the specified footprint.

expand Since the number of job definitions in the table JOB_DEFINITIONS can become very large, by default they are not all displayed. If the option **expand = all** is used, all the job definitions as well as their parent folder and the folder hierarchy are outputted. Individual paths in the hierarchy can be selected by specifying individual (folder) IDs.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the footprint.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Continued on next page

Continued from previous page

Field	Description
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
RESOURCES	Table of system resources that form this footprint. See also table 22.16 on page 283
JOB_DEFINITIONS	Table of job definitions that use this footprint. See also table 22.17 on page 284

Table 22.15.: Description of the output structure of the show footprint statement

RESOURCES The layout of the RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
RESOURCE_NAME	Fully qualified path name of System Named Resources.
AMOUNT	Amount of resource units that are allocated.
KEEP_MODE	Keep Mode specifies when the resource is released (FINISH, JOB_FINAL or FINAL).

Table 22.16.: Description of the output structure of the show footprint subtable

JOB_DEFINITIONS The layout of the JOB_DEFINITIONS table is shown in the table below.

Field	Description
ID	The repository object id
SE_PATH	Folder path name of the object.
TYPE	Type of object.
HAS_CHILDREN	true means that there are more environment users further down the tree.
PRIVS	String containing the users privileges on the object

Continued on next page

Continued from previous page

Field	Description

Table 22.17.: Description of the output structure of the show footprint subtable

show group

Purpose

The purpose of the show group statement is to get detailed information about the specified group. *Purpose*

Syntax

The syntax for the show group statement is *Syntax*

show group *groupname*

Description

The show group statement gives you detailed information about the specified group. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the group.
COMMENTTYPE	Type of comment if a comment is defined
COMMENT	Comment if defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
MANAGE_PRIVS	Table of the manage privileges. See also table 22.19 on page 286
USERS	Table of the user groups. See also table 22.20 on page 286

Table 22.18.: Description of the output structure of the show group statement

MANAGE_PRIVS The layout of the MANAGE_PRIVS table is shown in the table below.

Field	Description
PRIVS	String containing the users privileges on the object

Table 22.19.: Description of the output structure of the show group subtable

USERS The layout of the USERS table is shown in the table below.

Field	Description
ID	The repository object id
UID	ID of the user.
NAME	The object name
IS_ENABLED	This flag tells the user whether he can be connected.
DEFAULT_GROUP	The default group of this user.
PRIVS	String containing the users privileges on the object

Table 22.20.: Description of the output structure of the show group subtable

show interval

Purpose

The purpose of the show interval statement is to get detailed information about the interval. *Purpose*

Syntax

The syntax for the show interval statement is *Syntax*

show interval *intervalname*
[**with expand** [< **=** *datetime* | **=** *datetime* **-** *datetime* >]]

Description

The show interval statement displays detailed information about a interval. No rising edges are displayed in the absence of an expand clause. The expand clause can be used to specify a period for which the edges are to be shown. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
STARTTIME	The start time of the interval.
ENDTIME	The end time of the interval.
BASE	The base is a period containing time blocks of defined durations.
DURATION	These are time blocks of defined durations.
SYNCTIME	The time from which the interval begins. If the sync time is not given, the start time is used.
INVERSE	The selection is reversed using Inverse.
EMBEDDED	The name of the embedded interval.
SELECTION	Single blocks are selected using Selection.

Continued on next page

Continued from previous page	
Field	**Description**
	See also table 22.22 on page 288
FILTER	Name(s) of the intervals that filter (multiplication) the output of this interval more finely.
	See also table 22.23 on page 288
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.21.: Description of the output structure of the show interval statement

SELECTION The layout of the SELECTION table is shown in the table below.

Field	Description
ID	The repository object id
VALUE	Number of the selected edge.
PERIOD_FROM	Beginning of the period in which all the occurring edges are considered to have been selected.
PERIOD_TO	End of the period in which all the occurring edges are considered to have been selected.

Table 22.22.: Description of the output structure of the show interval subtable

FILTER The layout of the FILTER table is shown in the table below.

Field	Description
ID	The repository object id
CHILD	Name of the filtering interval.

Table 22.23.: Description of the output structure of the show interval subtable

show job

Purpose

The purpose of the show job statement is to get detailed information about the specified job. *Purpose*

Syntax

The syntax for the show job statement is *Syntax*

show job *jobid* [**with** WITHITEM {**,** WITHITEM}]

show job submittag = *string* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
filter = (FILTERITEM {**,** FILTERITEM} **)**
| **recursive audit**

FILTERITEM:
cancel
| **change priority**
| **clear warning**
| **comment**
| **ignore named resource**
| **ignore resource**
| **ignore dependency** [**recursive**]
| **job in error**
| **kill**
| **renice**
| **rerun** [**recursive**]
| **restartable**
| **resume**
| **set exit state**
| **set resource state**
| **set state**
| **set warning**
| **submit** [**suspend**]
| **suspend**
| **timeout**
| **trigger failure**
| **trigger submit**
| **unreachable**

Description

Description The show job statement gives you detailed information about the specified job. The job can be specified using either its ID or, if a submit tag was specified during the submit, the submit tag.
The filter option is used for selecting audit entries. If the filter option is not specified, all the audit entries are shown. Otherwise, only entries of the type specified in the filter are outputted.
The **recursive audit** option collects all the audit messages for the displayed job and its direct or indirect children.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
SE_NAME	The full path name of the object.
SE_OWNER	Owner of the object.
SE_TYPE	The se_type is the object type (JOB, BATCH or MILESTONE).
SE_RUN_PROGRAM	The Run Program line in the job definition.
SE_RERUN_PROGRAM	The Rerun Program line in the job definition.
SE_KILL_PROGRAM	The Kill Program line in the job definition.
SE_WORKDIR	The Workdir of the job definition.
SE_LOGFILE	The log file of the job definition.
SE_TRUNC_LOG	Defines whether the log file is to be truncated before the process starts or if the log information is to be appended.
SE_ERRLOGFILE	The error log file of the job definition.
SE_TRUNC_ERRLOG	Defines whether the log file is to be truncated before the process starts or if the log information is to be appended.
SE_EXPECTED_RUNTIME	The anticipated runtime of the job definition.
SE_PRIORITY	Priority/nice value of the job definition.
SE_SUBMIT_SUSPENDED	The Suspend Flag of the object.
SE_MASTER_SUBMITTABLE	The Master Submittable Flag of the object.

Continued on next page

Continued from previous page

Field	Description
SE_DEPENDENCY_MODE	The Dependency Mode of the object.
SE_ESP_NAME	The Exit State Profile of the object.
SE_ESM_NAME	The Exit State Mapping the job definition.
SE_ENV_NAME	The environment of the job definition.
SE_FP_NAME	The footprint of the job definition.
MASTER_ID	The object ID of the object at the top level in the hierarchy of this object.
CHILD_TAG	Tag for exclusive identifying jobs that have been submitted several times as children of the same job.
SE_VERSION	The version of definitions that are valid for this Submitted Entity.
OWNER	The group owning the object
PARENT_ID	The parent object in the submission hierarchy.
SCOPE_ID	The ID of the scope.
HTTPHOST	The host name of the scope for accessing log files via HTTP.
HTTPPORT	The HTTP port number of the job server for accessing log files via HTTP.
IS_STATIC	Flag indicating the static or dynamic submits of this job.
MERGE_MODE	Indicates how multiple submits of the same defined object are handled in the current Master Run.
STATE	The state of the job (not to be confused with the Exit State).
IS_DISABLED	Indicates whether the submitted entitity was submitted disabled.
IS_CANCELLED	Shows a Cancel operation that was performed on the job.
JOB_ESD_ID	The Exit State of the job after it has been executed.
JOB_ESD_PREF	The preference for merging the Job Exit States with the Child States.
JOB_IS_FINAL	Flag indicating the final state of the job.
JOB_IS_RESTARTABLE	A flag indicating that this job is restartable.
FINAL_ESD_ID	The final (merged) Exit State of the object.

Continued on next page

Continued from previous page

Field	Description
EXIT_CODE	The Exit Code of the last command execution.
COMMANDLINE	The created command line that is used for the first execution.
RR_COMMANDLINE	Created rerun command line that is used for the last executed rerun.
WORKDIR	The created Workdir.
LOGFILE	The created log file.
ERRLOGFILE	The created error log file.
PID	The process ID of the controller process.
EXT_PID	The process ID of the user process.
ERROR_MSG	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.
KILL_ID	The Submitted Entity ID of the submitted Kill Job.
KILL_EXIT_CODE	The Exit Code of the last executed Kill Program.
IS_SUSPENDED	Flag indicating whether the object is suspended.
IS_SUSPENDED_LOCAL	Flag indicating whether the object is locally suspended (for restart trigger with suspend).
PRIORITY	The current priority of the job.
RAW_PRIORITY	The raw priority value of the job. Unlike the priority, this value is practically unbounded. This is required in order to be able to restore the correct priority after Nice Profile manipulations.
NICEVALUE	The current nice value of the job.
NP_NICEVALUE	The *np_nicevalue* is the nice value which is the effect of activating (and deactivating) nice profiles.
MIN_PRIORITY	Minimum effective priority that can be achieved through natural aging.
AGING_AMOUNT	The number of time units after which the effective priority is incremented by 1.
AGING_BASE	The time unit that is used for the aging interval.
DYNAMIC_PRIORITY	The calculated priority that is currently being used by the system.
PARENT_SUSPENDED	The number of parents that have been suspended.

Continued on next page

Continued from previous page

Field	Description
SUBMIT_TS	The submit time.
RESUME_TS	The time when the job is automatically resumed.
SYNC_TS	The time from which all the dependencies are fulfilled.
RESOURCE_TS	The time from which all the Synchronizing Resources are allocated.
RUNNABLE_TS	The time from which the job switches to the Runnable State.
START_TS	The time when the job was last executed.
FINISH_TS	The end time of the last job execution.
FINAL_TS	The time when the object becomes final.
CNT_SUBMITTED	The number of children in a Submitted state.
CNT_DEPENDENCY_WAIT	The number of children in a Dependency_Wait state.
CNT_SYNCHRONIZE_WAIT	The number of children in a Synchronize_Wait state.
CNT_RESOURCE_WAIT	The number of children in a Resource_Wait state.
CNT_RUNNABLE	The number of children in a Runnable state.
CNT_STARTING	The number of children in a Starting state.
CNT_STARTED	The number of children in a Started state.
CNT_RUNNING	The number of children in a Running state.
CNT_TO_KILL	The number of children in a To_Kill state.
CNT_KILLED	The number of children in a Killed state.
CNT_CANCELLED	The number of children in a Cancelled state.
CNT_FINISHED	The number of children in a Finished state.
CNT_FINAL	The number of children in a Final state.
CNT_BROKEN_ACTIVE	The number of children in a Broken_Active state.
CNT_BROKEN_FINISHED	The number of children in a Broken_Finished state.
CNT_ERROR	The number of children in an Error state.
CNT_RESTARTABLE	The number of children in a Restartable state.
CNT_UNREACHABLE	The number of children in an Unreachable state.
CNT_WARN	Number of children that have warning messages.

Continued on next page

Continued from previous page

Field	Description
WARN_COUNT	Number of warning messages for the current object.
IDLE_TIME	
DEPENDENCY_WAIT_TIME	
SUSPEND_TIME	
SYNC_TIME	
RESOURCE_TIME	
JOBSERVER_TIME	
RESTARTABLE_TIME	
CHILD_WAIT_TIME	
PROCESS_TIME	
ACTIVE_TIME	
IDLE_PCT	
CHILDREN	Table of the children. See also table 22.25 on page 296
PARENTS	Table of the parents. See also table 22.26 on page 296
PARAMETER	Table of the parameters. See also table 22.27 on page 297
REQUIRED_JOBS	Table of the required jobs. See also table 22.28 on page 299
DEPENDENT_JOBS	Table of the dependent jobs. See also table 22.29 on page 301
REQUIRED_RESOURCES	Table of the required resources. See also table 22.30 on page 303
SUBMIT_PATH	The path from the job to the master via the submit hierarchy.
IS_REPLACED	Flag indicating whether the job has been replaced with another one.
TIMEOUT_AMOUNT	The maximum time that the job will wait for its resource.
TIMEOUT_BASE	The maximum time that the job will wait for its resource.
TIMEOUT_STATE	The timeout of the Scheduling Entity.
RERUN_SEQ	The rerun order.
AUDIT_TRAIL	Table of the log entries. See also table 22.31 on page 303
CHILD_SUSPENDED	The number of children that have been suspended.

Continued on next page

Continued from previous page

Field	Description
CNT_PENDING	The number of children in a pending state.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
SE_PRIVS	The privileges for the Scheduling Entity.
SUBMITTAG	Unique marker that is given at the submit time.
UNRESOLVED_HANDLING	Defines what to do if the required object cannot be found.
DEFINED_RESOURCES	Table of the Defined Resources of the object. See also table 22.32 on page 304

Table 22.24.: Description of the output structure of the show job statement

CHILDREN The layout of the CHILDREN table is shown in the table below.

Field	Description
CHILDID	The Submitted Entity ID of the child.
CHILDPRIVS	The privileges for the child object.
CHILDSENAME	The name of the child object.
CHILDSETYPE	The type of child object.
CHILDSEPRIVS	The privileges for the child object.
PARENTID	The ID of the parent.
PARENTPRIVS	The privileges for the parent object.
PARENTSENAME	The name of the parent object.
PARENTSETYPE	The type of parent object.
PARENTSEPRIVS	The privileges for the job definition that belong to the parent.
IS_STATIC	Static flag of the hierarchy definition.
PRIORITY	The priority of the hierarchy definition.
SUSPEND	The suspend mode of the hierarchy definition.
MERGE_MODE	The merge mode of the hierarchy definition.
EST_NAME	The name of the Exit State Translation of the hierarchy definition.

Continued on next page

Continued from previous page

Field	Description
IGNORED_DEPENDENCIES	Ignored Dependencies Flag of the hierarchy definition.

Table 22.25.: Description of the output structure of the show job subtable

PARENTS The layout of the PARENTS table is shown in the table below.

Field	Description
CHILDID	The Submitted Entity ID of the child.
CHILDPRIVS	The privileges for the child object.
CHILDSENAME	The name of the child object.
CHILDSETYPE	The type of child object.
CHILDSEPRIVS	The privileges for the child object.
PARENTID	The ID of the parent.
PARENTPRIVS	The privileges for the parent object.
PARENTSENAME	The name of the parent object.
PARENTSETYPE	The type of parent object.
PARENTSEPRIVS	The privileges for the job definition that belong to the parent.
IS_STATIC	Static flag of the hierarchy definition.
PRIORITY	The priority of the hierarchy definition.
SUSPEND	The suspend mode of the hierarchy definition.
MERGE_MODE	The merge mode of the hierarchy definition.
EST_NAME	The name of the Exit State Translation of the hierarchy definition.
IGNORED_DEPENDENCIES	Ignored Dependencies Flag of the hierarchy definition.

Table 22.26.: Description of the output structure of the show job subtable

PARAMETER The layout of the PARAMETER table is shown in the table below.

Field	Description
ID	The repository object id
NAME	The name of the parameter, variable or expression.

Continued on next page

Continued from previous page

Field	Description
TYPE	The type of parameter, variable or expression.
VALUE	The value of the parameter, variable or expression.

Table 22.27.: Description of the output structure of the show job subtable

REQUIRED_JOBS The layout of the REQUIRED_JOBS table is shown in the table below.

Field	Description
ID	The repository object id
DEPENDENT_ID	ID of the dependent Submitted Entity.
DEPENDENT_PATH	The path from the job to the master via the submit hierarchy.
DEPENDENT_PRIVS	The privileges for the dependent object.
DEPENDENT_ID_ORIG	ID of the original dependent Submitted Entity on which the dependency is defined for dependencies that have been inherited from the parents.
DEPENDENT_PATH_ORIG	The path from the dependent object to the master via the submit hierarchy.
DEPENDENT_PRIVS_ORIG	The privileges for the original dependent object.
DEPENDENCY_OPERATION	Defines whether all or only some dependencies of the original object have to be fulfilled.
REQUIRED_ID	ID of the required Submitted Entity.
REQUIRED_PATH	The path from the required object to the master via the submit hierarchy.
REQUIRED_PRIVS	The privileges for the required object.
STATE	The state of the dependency (OPEN, FULFILLED or FAILED).
DD_ID	ID of the Dependency Definition object.
DD_NAME	Name of the Dependency Definition (deprecated).
DD_DEPENDENTNAME	The full path name of the object.
DD_DEPENDENTTYPE	The type of dependent object.
DD_DEPENDENTPRIVS	Privileges for the dependent object.
DD_REQUIREDNAME	Path name of the definition of the dependent object.

Continued on next page

Continued from previous page

Field	Description
DD_REQUIREDTYPE	The type of required object.
DD_REQUIREDPRIVS	The privileges for the required object definition.
DD_UNRESOLVED_HANDLING	Specifies how to handle unresolvable dependencies during a submit.
DD_MODE	Defines whether just the required job itself or the required job together with its children have to be final.
DD_STATES	List of Exit States that the required object have to achieve to fulfil the dependency.
JOB_STATE	In the Job State list, you can filter for jobs that have the entered Job State.
IS_SUSPENDED	Flag indicating whether the object is suspended.
PARENT_SUSPENDED	Job is suspended because its parent is suspended.
CNT_SUBMITTED	The number of children in a Submitted state.
CNT_DEPENDENCY_WAIT	The number of children in a Dependency_Wait state.
CNT_SYNCHRONIZE_WAIT	The number of children in a Synchronize_Wait state.
CNT_RESOURCE_WAIT	The number of children in a Resource_Wait state.
CNT_RUNNABLE	The number of children in a Runnable state.
CNT_STARTING	The number of children in a Starting state.
CNT_STARTED	The number of children in a Started state.
CNT_RUNNING	The number of children in a Running state.
CNT_TO_KILL	The number of children in a To_Kill state.
CNT_KILLED	The number of children in a Killed state.
CNT_CANCELLED	The number of children in a Cancelled state.
CNT_FINISHED	The number of children in a Finished state.
CNT_FINAL	The number of children in a Final state.
CNT_BROKEN_ACTIVE	The number of children in a Broken_Active state.
CNT_BROKEN_FINISHED	The number of children in a Broken_Finished state.
CNT_ERROR	The number of children in an Error state.
CNT_RESTARTABLE	The number of children in a Restartable state.
CNT_UNREACHABLE	The number of children in an Unreachable state.

Continued on next page

Continued from previous page

Field	Description
JOB_IS_FINAL	The number of children in an Is_Final state.
CHILD_TAG	Tag for exclusive identifying jobs that have been submitted several times as children of the same job.
FINAL_STATE	The final state of a job.
CHILDREN	The number of children the job has.
IGNORE	Flag indicating whether the Resource Allocation is bring ignored.
CHILD_SUSPENDED	The number of children that have been suspended.
CNT_PENDING	The number of children in a Pending state.
DD_CONDITION	The condition that has to be additionally fulfilled for the dependency to be fulfilled.

Table 22.28.: Description of the output structure of the show job subtable

DEPENDENT_JOBS The layout of the DEPENDENT_JOBS table is shown in the table below.

Field	Description
ID	The repository object id
DEPENDENT_ID	ID of the dependent Submitted Entity.
DEPENDENT_PATH	The path from the job to the master via the submit hierarchy.
DEPENDENT_PRIVS	The privileges for the dependent object.
DEPENDENT_ID_ORIG	ID of the original dependent Submitted Entity on which the dependency is defined for dependencies that have been inherited from the parents.
DEPENDENT_PATH_ORIG	The path from the dependent object to the master via the submit hierarchy.
DEPENDENT_PRIVS_ORIG	The privileges for the original dependent object.
DEPENDENCY_OPERATION	Defines whether all or only some dependencies of the original object have to be fulfilled.
REQUIRED_ID	ID of the required Submitted Entity.
REQUIRED_PATH	The path from the required object to the master via the submit hierarchy.

Continued on next page

Continued from previous page

Field	Description
REQUIRED_PRIVS	The privileges for the required object.
STATE	The state of the dependency (OPEN, FULFILLED or FAILED).
DD_ID	ID of the Dependency Definition object.
DD_NAME	Name of the Dependency Definition (deprecated).
DD_DEPENDENTNAME	The full path name of the object.
DD_DEPENDENTTYPE	The type of dependent object.
DD_DEPENDENTPRIVS	Privileges for the dependent object.
DD_REQUIREDNAME	Path name of the definition of the dependent object.
DD_REQUIREDTYPE	The type of required object.
DD_REQUIREDPRIVS	The privileges for the required object definition.
DD_UNRESOLVED_HANDLING	Specifies how to handle unresolvable dependencies during a submit.
DD_MODE	Defines whether just the required job itself or the required job together with its children have to be final.
DD_STATES	List of Exit States that the required object have to achieve to fulfil the dependency.
JOB_STATE	In the Job State list, you can filter for jobs that have the entered Job State.
IS_SUSPENDED	Flag indicating whether the object is suspended.
PARENT_SUSPENDED	Job is suspended because its parent is suspended.
CNT_SUBMITTED	The number of children in a Submitted state.
CNT_DEPENDENCY_WAIT	The number of children in a Dependency_Wait state.
CNT_SYNCHRONIZE_WAIT	The number of children in a Synchronize_Wait state.
CNT_RESOURCE_WAIT	The number of children in a Resource_Wait state.
CNT_RUNNABLE	The number of children in a Runnable state.
CNT_STARTING	The number of children in a Starting state.
CNT_STARTED	The number of children in a Started state.
CNT_RUNNING	The number of children in a Running state.
CNT_TO_KILL	The number of children in a To_Kill state.

Continued on next page

Continued from previous page

Field	Description
CNT_KILLED	The number of children in a Killed state.
CNT_CANCELLED	The number of children in a Cancelled state.
CNT_FINISHED	The number of children in a Finished state.
CNT_FINAL	The number of children in a Final state.
CNT_BROKEN_ACTIVE	The number of children in a Broken_Active state.
CNT_BROKEN_FINISHED	The number of children in a Broken_Finished state.
CNT_ERROR	The number of children in an Error state.
CNT_RESTARTABLE	The number of children in a Restartable state.
CNT_UNREACHABLE	The number of children in an Unreachable state.
JOB_IS_FINAL	The number of children in an Is_Final state.
CHILD_TAG	Tag for exclusive identifying jobs that have been submitted several times as children of the same job.
FINAL_STATE	The final state of a job.
CHILDREN	The number of children the job has.
IGNORE	Flag indicating whether the Resource Allocation is bring ignored.
CHILD_SUSPENDED	The number of children that have been suspended.
CNT_PENDING	The number of children in a Pending state.
DD_CONDITION	The condition that has to be additionally fulfilled for the dependency to be fulfilled.

Table 22.29.: Description of the output structure of the show job subtable

REQUIRED_RESOURCES The layout of the REQUIRED_RESOURCES table is shown in the table below.

Field	Description
SCOPE_ID	ID of the scope that allocated the resource.
SCOPE_NAME	The fully qualified name of the scope.
SCOPE_TYPE	The type of scope (SCOPE or SERVER, FOLDER, BATCH or JOB).
SCOPE_PRIVS	The privileges for the scope.
RESOURCE_ID	ID of the Required Resource.

Continued on next page

Continued from previous page

Field	Description
RESOURCE_NAME	Categorical path name of the Requested Resource.
RESOURCE_USAGE	The usage of the Required Resource (STATIC, SYSTEM or SYNCHRONIZING).
RESOURCE_OWNER	Name of the owner of the Requested Resource.
RESOURCE_PRIVS	The privileges for the Requested Resource.
RESOURCE_STATE	The state of the Requested Resource.
RESOURCE_TIMESTAMP	Date time of last time state was set for the requested resource.
REQUESTABLE_AMOUNT	The maximum amount of resources that can be requested by a job.
TOTAL_AMOUNT	The amount that can be allocated.
FREE_AMOUNT	The free amount that can be requested.
REQUESTED_AMOUNT	This is the requested amount.
REQUESTED_LOCKMODE	The requested lockmode.
REQUESTED_STATES	The requested Resource State.
RESERVED_AMOUNT	The amount that is reserved by the Requested Resource.
ALLOCATED_AMOUNT	The amount that was allocated by the Requested Resource.
ALLOCATED_LOCKMODE	The lockmode currently allocated by the Requested Resource.
IGNORE	Flag indicating whether the Resource Allocation is bring ignored.
STICKY	Flag indicating whether it is a Sticky Resource Allocation.
STICKY_NAME	Optional name of the sticky resource request
STICKY_PARENT	Parent job within which the sticky request is evaluated
STICKY_PARENT_TYPE	Type of the parent within which the sticky requirement is evaluated
ONLINE	Flag indicating whether the resource is available for an allocation.
ALLOCATE_STATE	The state of the allocation (RESERVED, ALLOCATED, AVAILABLE or BLOCKED).
EXPIRE	Time defining the maximum or minimum age of a resource depending on whether the expire is positive or negative.

Continued on next page

Continued from previous page

Field	Description
EXPIRE_SIGN	Defines the expiration condition, +/- indicating younger/older than.
DEFINITION	Where the Resource Definition is saved.

Table 22.30.: Description of the output structure of the show job subtable

AUDIT_TRAIL The layout of the AUDIT_TRAIL table is shown in the table below.

Field	Description
ID	The repository object id
USERNAME	User name that causes this audit record.
TIME	The time when this audit record was created.
TXID	Transaction number of the change.
ACTION	Action that causes this audit record.
ORIGINID	The original object ID that causes this audit record.
JOBID	This field is not yet documented
JOBNAME	This field is not yet documented
COMMENT	Comment if defined
INFO	Additional system information about the Action Event that caused the audit record.

Table 22.31.: Description of the output structure of the show job subtable

DEFINED_RESOURCES The layout of the DEFINED_RESOURCES table is shown in the table below.

Field	Description
ID	ID of the Defined Resource.
RESOURCE_NAME	Full path name of the Defined Object.
RESOURCE_USAGE	The usage of the resource (STATIC, SYSTEM or SYNCHRONIZING).
RESOURCE_OWNER	The owner of the resource.
RESOURCE_PRIVS	The privileges for the resource.
RESOURCE_STATE	The current state of the resource.
RESOURCE_TIMESTAMP	The last time the Resource State was set for this resource.

Continued on next page

Continued from previous page

Field	Description
REQUESTABLE_AMOUNT	The maximum amount of resources that can be requested by a job.
TOTAL_AMOUNT	The complete amount that can be allocated.
FREE_AMOUNT	The free amount that can be allocated.
ONLINE	Indicates whether the resource can be allocated or not.

Table 22.32.: Description of the output structure of the show job subtable

show job definition

Purpose

The purpose of the show job definition statement is to get detailed information about the specified job definition. *Purpose*

Syntax

The syntax for the show job definition statement is *Syntax*

show job definition *folderpath*

Description

The show job definition statement gives you detailed information about the specified job definition. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The full path name of the job definition.
OWNER	The group owning the object
TYPE	The object type (**Batch**, **Job** or **Milestone**).
INHERIT_PRIVS	Privileges that are inherited from the parent folder.
RUN_PROGRAM	The command line for executing a job for the first time.
RERUN_PROGRAM	The command line for restarting a submitted job.
KILL_PROGRAM	The command line for executing the deletion of a running job.
WORKDIR	The working directory from where the command is to be executed.

Continued on next page

Continued from previous page

Field	Description
LOGFILE	The file to which the stdout of the command is written.
TRUNC_LOG	Indicates whether the log is overwritten or appended.
ERRLOGFILE	The file to which the stderr of the command is written.
TRUNC_ERRLOG	Indicates whether the error log is overwritten or appended.
EXPECTED_RUNTIME	The estimated runtime in seconds. This can be used in triggers for monitoring the runtime.
EXPECTED_FINALTIME	The estimated time in seconds until the Final State is reached. This can be used in triggers for monitoring the runtime.
PRIORITY	Priority/nice value of the job or batch.
MIN_PRIORITY	This is the minimum effective priority that can be achieved through natural aging.
AGING_AMOUNT	The number of time units after which the effective priority is incremented by 1.
AGING_BASE	The time unit that is used for the aging interval.
SUBMIT_SUSPENDED	Flag that indicates whether the object is to be suspended after the submit.
RESUME_AT	If the job is to be submitted as being suspended, an automatic resume takes place at the given time.
RESUME_IN	If the job is to be submitted as being suspended, an automatic resume takes place after the given number of time units.
RESUME_BASE	Specified time unit for RESUME_IN.
MASTER_SUBMITTABLE	Flag indicating whether the job or batch can be submitted as a standalone master.
TIMEOUT_AMOUNT	The number of time units for the delay until the timeout occurs.
TIMEOUT_BASE	The unit that is used to specify the timeout in seconds, minutes, hours or days.
TIMEOUT_STATE	Exit State that is set for a job when the timeout occurs.
DEPENDENCY_MODE	Indicates whether all or just one dependency has to be fulfilled to start the job.

Continued on next page

Continued from previous page

Field	Description
ESP_NAME	Name of the Exit State Profile of the object.
ESM_NAME	Name of the Exit State Mapping that is being used for this job.
ENV_NAME	Name of the environment requesting the job.
FP_NAME	Name of the footprint using the job.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
PRIVS	String containing the users privileges on the object
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
CHILDREN	Table of the children. See also table 22.34 on page 308
PARENTS	Table of the parents. See also table 22.35 on page 309
PARAMETER	Table of the parameters and variables that are defined for this object. See also table 22.36 on page 310
REFERENCES	Table of parameter references to this object.
REQUIRED_JOBS	Table of objects upon which the following objects are dependent. See also table 22.37 on page 311
DEPENDENT_JOBS	Table of objects that are dependent upon the following objects. See also table 22.38 on page 312
REQUIRED_RESOURCES	Table of resource requirements that are not included in the environment and footprint. See also table 22.39 on page 313
DEFINED_RESOURCES	Table of resources to be instantiated at the submit time, visible for submitting children.

Table 22.33.: Description of the output structure of the show job definition statement

CHILDREN The layout of the CHILDREN table is shown in the table below.

Field	Description
ID	The repository object id
CHILDNAME	Full path name of the child object.
CHILDTYPE	The child type (JOB, BATCH or MILESTONE).
CHILDPRIVS	A string containing the user privileges of the child object.
PARENTNAME	Full path name of the parent object.
PARENTTYPE	The parent type (JOB, BATCH or MILESTONE).
PARENTPRIVS	A string containing the user privileges of the parent object.
ALIAS_NAME	Name for referencing to child definitions with dynamic submits.
IS_STATIC	The is_static flag defines whether the job is to be statically or dynamically submitted.
IS_DISABLED	Flag indicating the the child should be executed or skipped
PRIORITY	The nice value that has been added to the children.
SUSPEND	Determines whether the child is to be suspended for the submit.
RESUME_AT	If the job is to be submitted as being suspended, an automatic resume takes place at the given time.
RESUME_IN	If the job is to be submitted as being suspended, an automatic resume takes place after the given number of time units.
RESUME_BASE	Specified time unit for RESUME_IN.
MERGE_MODE	Determines how the condition handles the same object that occurs more than once in the submission hierarchy.
EST_NAME	An Exit State Translation that is used to translate the Exit States of the children to the Exit States of the parents.
IGNORED_DEPENDENCIES	List with the names of the dependencies for ignoring the dependencies of the parents.

Table 22.34.: Description of the output structure of the show job definition subtable

PARENTS The layout of the PARENTS table is shown in the table below.

Field	Description
ID	The repository object id
CHILDNAME	Full path name of the child object.
CHILDTYPE	The child type (JOB, BATCH or MILESTONE).
CHILDPRIVS	A string containing the user privileges of the child object.
PARENTNAME	Full path name of the parent object.
PARENTTYPE	The parent type (JOB, BATCH or MILESTONE).
PARENTPRIVS	A string containing the user privileges of the parent object.
ALIAS_NAME	Name for referencing to child definitions with dynamic submits.
IS_STATIC	The is_static flag defines whether the job is to be statically or dynamically submitted.
IS_DISABLED	Flag indicating the the child should be executed or skipped
PRIORITY	The nice value that has been added to the children.
SUSPEND	Determines whether the child is to be suspended for the submit.
RESUME_AT	If the job is to be submitted as being suspended, an automatic resume takes place at the given time.
RESUME_IN	If the job is to be submitted as being suspended, an automatic resume takes place after the given number of time units.
RESUME_BASE	Specified time unit for RESUME_IN.
MERGE_MODE	Determines how the condition handles the same object that occurs more than once in the submission hierarchy.
EST_NAME	An Exit State Translation that is used to translate the Exit States of the children to the Exit States of the parents.
IGNORED_DEPENDENCIES	List with the names of the dependencies for ignoring the dependencies of the parents.

Table 22.35.: Description of the output structure of the show job definition subtable

PARAMETER The layout of the PARAMETER table is shown in the table below.

Field	Description
ID	The repository object id
NAME	Name of the parameter.
EXPORT_NAME	The export name defines the name under which the value of the parameter is exported to the process's environment.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
EXPRESSION	Expression for the parameter type EXPRESSION.
DEFAULT_VALUE	The default value of the parameter.
REFERENCE_TYPE	Type of object that is being referenced.
REFERENCE_PATH	The path to the object that is being referenced.
REFERENCE_PRIVS	The user's privileges for the object that is being referenced.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
ID	The repository object id
NAME	Name of the parameter.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
REFERENCE_TYPE	Type of object that is referencing the parameter.
REFERENCE_PATH	The path to the object that is referencing the parameter.
REFERENCE_PRIVS	The user's privileges for the object that is referencing the parameter.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.36.: Description of the output structure of the show job definition subtable

REQUIRED_JOBS The layout of the REQUIRED_JOBS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	The object name
DEPENDENTNAME	The full path name of the dependent object.
DEPENDENTTYPE	The type of dependent object (JOB, BATCH or MILESTONE).
DEPENDENTPRIVS	String containing the user privileges of the dependent object.
REQUIREDNAME	The full path name of the required object.
REQUIREDTYPE	The type of required object (JOB, BATCH or MILESTONE).
REQUIREDPRIVS	String containing the user privileges of the required object.
UNRESOLVED_HANDLING	Defines what to do if the required object cannot be found.
MODE	Defines whether dependencies are to be handled as being fulfilled in the case of jobs in a final state (JOB_FINAL) or final states of objects contain the child objects (ALL_FINAL).
STATE_SELECTION	The State Selection defines how the required Exit States are determined. The options here are FINAL, ALL_REACHABLE, UNREACHABLE and DEFAULT. In the case of FINAL, the required Exit States can be explicitly listed.
CONDITION	The additional conditions must be fulfilled.
STATES	Comma-separated list of permitted Exit States that the required object has to achieve to fulfil the dependencies. If an additional condition has been specified, it is shown after the colon.

Table 22.37.: Description of the output structure of the show job definition subtable

DEPENDENT_JOBS The layout of the DEPENDENT_JOBS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	The object name
DEPENDENTNAME	The full path name of the dependent object.

Continued on next page

Continued from previous page

Field	Description
DEPENDENTTYPE	The type of dependent object (JOB, BATCH or MILESTONE).
DEPENDENTPRIVS	String containing the user privileges of the dependent object.
REQUIREDNAME	The full path name of the required object.
REQUIREDTYPE	The type of required object (JOB, BATCH or MILESTONE).
REQUIREDPRIVS	String containing the user privileges of the required object.
UNRESOLVED_HANDLING	Defines what to do if the required object cannot be found.
MODE	Defines whether dependencies are to be handled as being fulfilled in the case of jobs in a final state (JOB_FINAL) or final states of objects contain the child objects (ALL_FINAL).
STATE_SELECTION	The State Selection defines how the required Exit States are determined. The options here are FINAL, ALL_REACHABLE, UNREACHABLE and DEFAULT. In the case of FINAL, the required Exit States can be explicitly listed.
CONDITION	The additional conditions must be fulfilled.
STATES	Comma-separated list of permitted Exit States that the required object has to achieve to fulfil the dependencies. If an additional condition has been specified, it is shown after the colon.

Table 22.38.: Description of the output structure of the show job definition subtable

REQUIRED_RESOURCES The layout of the REQUIRED_RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
RESOURCE_NAME	Full path name of the required Named Resource.
RESOURCE_USAGE	The usage of the required resource (STATIC, SYSTEM or SYNCHRONIZING).

Continued on next page

Continued from previous page

Field	Description
RESOURCE_PRIVS	String containing the user privileges of the Named Resource.
AMOUNT	The required amount with System or Synchronizing Resources.
KEEP_MODE	Indicates whether the resource is to be released at the end of the job (NOKEEP) or in a final job state (KEEP) including all the children (KEEP_FINAL).
IS_STICKY	Indicates whether the resource allocation for subsequent jobs is retained.
STICKY_NAME	Optional name of the sticky resource request
STICKY_PARENT	Parent Job Definition within which the sticky requirement is handled
RESOURCE_STATE_MAPPING	The Resource State Mapping defines how and whether the state of the resource is to be changed after the job has finished.
EXPIRED_AMOUNT	The maximum number of units (expressed by EXPIRED_BASE) that can affect this resource before it enables an allocation of this Synchronizing Resource.
EXPIRED_BASE	The unit for specifying the operation in MINUTES, HOURS, DAYS, WEEKS, MONTHS and YEARS.
LOCKMODE	The lockmode for allocating Synchronizing Resources (N, S, SX, X).
STATES	Comma-separated list of permitted Exit States that the required object has to achieve to fulfil the dependencies.
DEFINITION	The origin of the resource request (REQUIREMENT, FOOTPRINT, FOLDER or ENVIRONMENT).
ORIGIN	Name of the Resource Request Definition, invalid in the case of a complete request.
CONDITION	The optional condition that can be defined for requests for Static Resources.

Table 22.39.: Description of the output structure of the show job definition subtable

show named resource

Purpose

Purpose The purpose of the show named resource statement is to get detailed information about the named resource.

Syntax

Syntax The syntax for the show named resource statement is

show named resource *resourcepath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The show named resource statement gives you detailed information about the Named Resource.

expand Since the number of job definitions in the table JOB_DEFINITIONS can become very large, by default they are not all displayed. If the option **expand = all** is used, all the job definitions as well as their parent folder and the folder hierarchy are outputted. Individual paths in the hierarchy can be selected by specifying individual (folder) IDs.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the Named Resource.
OWNER	Owner of the Named Resource.
USAGE	The Usage field specifies the Resource type.

Continued on next page

Continued from previous page

Field	Description
INHERIT_PRIVS	Privileges that are inherited from the parent folder.
RESOURCE_STATE_PROFILE	This is the Resource State Profile assigned to the resource.
FACTOR	The factor by which allocations are multiplied.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
RESOURCES	These are the instances of the Named Resource. See also table 22.41 on page 316
PARAMETERS	These are the defined parameters of the Named Resource. See also table 22.42 on page 316
JOB_DEFINITIONS	These are the job definitions that request the Named Resource. See also table 22.43 on page 317

Table 22.40.: Description of the output structure of the show named resource statement

RESOURCES The layout of the RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
SCOPE	The names of the scopes, Submitted Entities, Scheduling Entities or folders that offer the respective Named Resource are shown here.
TYPE	This is the resource type.
OWNER	The group owning the object
STATE	Indicates the state of the resource.
REQUESTABLE_AMOUNT	The maximum amount of resources that can be requested by a job.

Continued on next page

Continued from previous page

Field	Description
AMOUNT	The amount states the current number of instances of the Named Resource for this scope or job server.
FREE_AMOUNT	The Free Amount designates the total number of instances of a resource in the selected scope or job server that have not yet been allocated to jobs.
IS_ONLINE	Indicates whether the resource is online or not.
PRIVS	String containing the users privileges on the object

Table 22.41.: Description of the output structure of the show named resource subtable

PARAMETERS The layout of the PARAMETERS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	The name of the parameter.
TYPE	This is the parameter type. Local or Local Constant.
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.
TAG	This field is not yet documented
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.42.: Description of the output structure of the show named resource subtable

JOB_DEFINITIONS The layout of the JOB_DEFINITIONS table is shown in the table below.

Field	Description
ID	The repository object id

Continued on next page

Continued from previous page

Field	Description
NAME	Name of the job definition.
AMOUNT	The amount of the resource that is required by the job.
KEEP_MODE	The value of the Keep parameter for the resource request from the job.
IS_STICKY	Indicates whether it is a Sticky Request or not.
STICKY_NAME	Optional name of the sticky resource request
STICKY_PARENT	Parent Job Definition within which the sticky requirement is handled
RESOURCE_STATE_MAPPING	If a Resource State Mapping was specified in the resource request, it is displayed here.
EXPIRED_AMOUNT	The number of units. If the Expired Amount is positive, this means that the state change cannot have taken place longer ago than the given maximum time. If the amount is negative, it must have taken place at least as long ago as the given minimum time.
EXPIRED_BASE	The unit in minutes, hours, days, weeks, months and years.
LOCKMODE	The lockmode describes the mode for accessing this resource (exclusive, shared, etc.).
STATES	Multiple states that are acceptable for this job are separated by commas.
CONDITION	The condition that can be defined for requests for Static Resources.
PRIVS	String containing the users privileges on the object

Table 22.43.: Description of the output structure of the show named resource subtable

show resource

Purpose

Purpose The purpose of the show resource statement is to get detailed information about the resource.

Syntax

Syntax The syntax for the show resource statement is

show RESOURCE_URL

RESOURCE_URL:
resource *resourcepath* **in** *folderpath*
| **resource** *resourcepath* **in** *serverpath*

Description

Description The show resource statement gives you detailed information about the resource.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the resource.
SCOPENAME	Name of the scope in which the pool was created.
OWNER	The group owning the object
LINK_ID	ID of the referenced resource.
LINK_SCOPE	Scope name of the referenced resource.
BASE_ID	ID of the ultimately referenced resource.
BASE_SCOPE	Scope name of the ultimately referenced resource.
MANAGER_ID	ID of the Managing Pool.
MANAGER_NAME	Name of the Managing Pool.

Continued on next page

Continued from previous page

Field	Description
MANAGER_SCOPENAME	Name of the scope in which the Managing Pool was created.
USAGE	The Usage field specifies the Resource type.
RESOURCE_STATE_PROFILE	This is the Resource State Profile allocated to the resource.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
TAG	The tag is an optional short name for the resource.
STATE	The state is the current state of the resource in this scope or job server.
TIMESTAMP	The timestamp indicates the last time the Resource State changed.
REQUESTABLE_AMOUNT	The Requestable Amount is the maximum amount that can be requested.
DEFINED_AMOUNT	The amount that is available if the resource is not pooled.
AMOUNT	The actual available amount.
FREE_AMOUNT	The Free Amount designates the total number of instances of a resource that have not yet been allocated to jobs.
IS_ONLINE	Is Online is an indicator that states whether the resource is online or not.
FACTOR	This is the correction factor by which the requested amount is multiplied.
TRACE_INTERVAL	The trace interval is the minimum time in seconds between when Trace Records are written.
TRACE_BASE	The trace base is the basis for the valuation period.
TRACE_BASE_MULTIPLIER	The base multiplier determines the multiplication factor of the trace base.
TD0_AVG	The average resource allocation of the last $B * M^0$ seconds.
TD1_AVG	The average resource allocation of the last $B * M^1$ seconds.
TD2_AVG	The average resource allocation of the last $B * M^2$ seconds.

Continued on next page

Continued from previous page

Field	Description
LW_AVG	The average allocation since the last time a Trace Record was written.
LAST_WRITE	The time the last Trace Record was written.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
ALLOCATIONS	This is a table of resource allocations. See also table 22.45 on page 321
PARAMETERS	Additional information about a resource can be saved in the Parameters tab. See also table 22.46 on page 322

Table 22.44.: Description of the output structure of the show resource statement

ALLOCATIONS The layout of the ALLOCATIONS table is shown in the table below.

Field	Description
ID	The repository object id
JOBID	This is the ID of the job instance that was started with either a direct submit of the job or by a submit of the Master Batch or Master Job.
MASTERID	This is the ID of the job or batch instance that was started as a Master Job and contains the current job as a child.
JOBTYPE	This is the type of the job.
JOBNAME	This is the name of the job.
AMOUNT	This is the available amount.
KEEP_MODE	The Keep parameter defines whether the job is given the current "Sticky" resource or not. The following variants are used: KEEP, NO KEEP and KEEP FINAL.

Continued on next page

Continued from previous page

Field	Description
IS_STICKY	The resource is only released if there are no other Sticky Requests for this Named Resource in the same batch.
STICKY_NAME	Optional name of the sticky resource request
STICKY_PARENT	Parent job within which the sticky request is evaluated
STICKY_PARENT_TYPE	Type of the parent within which the sticky requirement is evaluated
LOCKMODE	The lockmode defines which access mode is used to allocate the resource to the current job.
RSM_NAME	The name of the Resource State Mapping.
TYPE	The type of allocation: Available, Blocked, Allocations, Master_Reservation, Reservation
TYPESORT	Aid for sorting the allocations.
P	The priority of the job.
EP	The effective priority of the job.
PRIVS	String containing the users privileges on the object

Table 22.45.: Description of the output structure of the show resource subtable

PARAMETERS The layout of the PARAMETERS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	Name of the parameter.
EXPORT_NAME	The export name defines the name under which the value of the parameter is exported to the process's environment.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
EXPRESSION	Expression for the parameter type EXPRESSION.
DEFAULT_VALUE	The default value of the parameter.
REFERENCE_TYPE	Type of object that is being referenced.

Continued on next page

Continued from previous page

Field	Description
REFERENCE_PATH	The path to the object that is being referenced.
REFERENCE_PRIVS	The user's privileges for the object that is being referenced.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
ID	The repository object id
NAME	Name of the parameter.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
REFERENCE_TYPE	Type of object that is referencing the parameter.
REFERENCE_PATH	The path to the object that is referencing the parameter.
REFERENCE_PRIVS	The user's privileges for the object that is referencing the parameter.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.46.: Description of the output structure of the show resource subtable

show resource state definition

Purpose

The purpose of the show resource state definition is to get detailed information about the specified resource state definition. *Purpose*

Syntax

The syntax for the show resource state definition statement is *Syntax*

show resource state definition *statename*

Description

The show resource state definition statement gives you detailed information about the Resource State Definition. *Description*

Output

This statement returns an output structure of type record. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object

Table 22.47.: Description of the output structure of the show resource state definition statement

show resource state mapping

Purpose

Purpose The purpose of the show resource state mapping statement is to get detailed information about the specified mapping.

Syntax

Syntax The syntax for the show resource state mapping statement is

show resource state mapping *profilename*

Description

Description The show resource state mapping statement gives you detailed information about the specified mapping.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	Name of the Resource State Mapping.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
MAPPINGS	A table with translations from the Exit State to the Resource State. See also table 22.49 on page 325

Table 22.48.: Description of the output structure of the show resource state mapping statement

MAPPINGS The layout of the MAPPINGS table is shown in the table below.

Field	Description
ESD_NAME	Name of the Exit State Definition.
RSD_FROM	The original state of the resource.
RSD_TO	The current state of the resource.

Table 22.49.: Description of the output structure of the show resource state mapping subtable

show resource state profile

Purpose

Purpose The purpose of the show resource state profile is to get detailed information about the specified resorce state profile.

Syntax

Syntax The syntax for the show resource state profile statement is

show resource state profile *profilename*

Description

Description The show resource state profile statement gives you detailed information about the specified Resource State Profile.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
INITIAL_STATE	This field defines the initial state of the resource. This Resource State does not have to be present in the list of valid Resource States.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
STATES	The valid Resource States are shown in the Resource State column in the States table.

Continued on next page

Continued from previous page	
Field	**Description**
	See also table 22.51 on page 327

Table 22.50.: Description of the output structure of the show resource state profile statement

STATES The layout of the STATES table is shown in the table below.

Field	Description
ID	The repository object id
RSD_NAME	Name of the Resource State Definition.
PRIVS	String containing the users privileges on the object

Table 22.51.: Description of the output structure of the show resource state profile subtable

show schedule

Purpose

Purpose The purpose of the show schedule statement is to get detailed information about the specified schedule.

Syntax

Syntax The syntax for the show schedule statement is

show schedule *schedulepath*

Description

Description The show schedule statement gives you detailed information about the specified schedule.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
INHERIT_PRIVS	Privileges that are inherited from the parent folder.
INTERVAL	These are blocks of defined durations in the Time Scheduling.
TIME_ZONE	The time zone in which the schedule is to be calculated.
ACTIVE	Jobs are only submitted if this flag is set.
EFF_ACTIVE	States whether a parent has been set to Inactive by the scheduler.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object

Continued on next page

Continued from previous page

Field	Description
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.52.: Description of the output structure of the show schedule statement

show scheduled event

Purpose

Purpose The purpose of the show scheduled event is to get detailed information about the specified event.

Syntax

Syntax The syntax for the show scheduled event statement is

show scheduled event *schedulepath* **.** *eventname*

Description

Description The show scheduled event statement gives you detailed information about the specified event.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
OWNER	The group owning the object
SCHEDULE	The schedule defines the scheduling times.
EVENT	The event describes what is to be done.
ACTIVE	Jobs are only submitted if this flag is set.
EFF_ACTIVE	States whether a parent has been set to Inactive by the scheduler.
BROKEN	The Broken field can be used to check whether an error occurred when the job was submitted.
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.

Continued on next page

Continued from previous page

Field	Description
ERROR_MSG	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.
LAST_START	The last time the job is to be executed by the Scheduling System is shown here.
NEXT_START	The next scheduled time when the task is to be executed by the Scheduling System is shown here.
NEXT_CALC	If the Next Start field is empty, the Next Calc gives the time when a search is run for a next start time. Otherwise, the new calculation takes place at the time of the Next Start.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
BACKLOG_HANDLING	The backlog determines how the Time Scheduling System behaves after a server downtime.
SUSPEND_LIMIT	The time that defines when jobs are to be submitted in a suspended state, for example after a downtime.
EFFECTIVE_SUSPEND_LIMIT	The time that defines when jobs are to be submitted in a suspended state, for example after a downtime.
CALENDAR	This flag indicates whether calendar entries are created.
CALENDAR_HORIZON	The defined length of the period in days for which a calendar is created.
EFFECTIVE_CALENDAR_HORIZON	The effective length of the period in days for which a calendar is created.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CALENDAR_TABLE	The table with the next start times.

Table 22.53.: Description of the output structure of the show scheduled event statement

show scope

Purpose

Purpose The purpose of the show scope statement is to get detailed information about a scope.

Syntax

Syntax The syntax for the show scope statement is

show < **scope** *serverpath* | **job server** *serverpath* > [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Description

Description The show scope statement gives you detailed information about the scope.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OWNER	The group owning the object
TYPE	The type of scope.
INHERIT_PRIVS	Privileges that are inherited from the parent folder.
IS_TERMINATE	This flag indicates whether a termination order exists.
IS_SUSPENDED	Indicates whether the scope is suspended.
IS_ENABLED	The job server can only log on to the server if the Enable flag is set to Yes.

Continued on next page

Continued from previous page

Field	Description
IS_REGISTERED	Defines whether the job server has sent a register command.
IS_CONNECTED	Indicates whether the job server is connected.
HAS_ALTERED_CONFIG	The configuration on the server does not match the current configuration on the job server.
STATE	This is the current state of the resource in this scope.
PID	The PID is the process identification number of the job server process on the respective host system.
NODE	The node specifies the computer on which the job server is running. This field has a purely documentary character.
IDLE	The time that has elapsed since the last command. This only applies for job servers.
ERRMSG	This is an error message.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
RESOURCES	The resources present in this scope are displayed here. See also table 22.55 on page 335
CONFIG	The configuration of the job server is described in the Config tab. See also table 22.56 on page 335
CONFIG_ENVMAPPING	Whether and under which name the environment variables are visible is configured in this tab. See also table 22.57 on page 336
PARAMETERS	Additional information about a resource can be saved in the Parameters tab. See also table 22.58 on page 337

Table 22.54.: Description of the output structure of the show scope statement

RESOURCES The layout of the RESOURCES table is shown in the table below.

Field	Description
ID	The repository object id
NR_ID	ID of the Named Resource.
NAME	Name of the Named Resource.
USAGE	It is the usage of the Named Resource (STATIC, SYSTEM or SYNCHRONISING)
NR_PRIVS	String containing the abreviations for the user privileges for this Named Resource.
TAG	The tag is an optional short name for the resource.
OWNER	The group owning the object
LINK_ID	ID of the referenced resource.
LINK_SCOPE	Scope of the referenced resource.
STATE	The Resource State of the resource.
REQUESTABLE_AMOUNT	The maximum amount of resources that can be requested by a job.
AMOUNT	The actual amount that is available.
FREE_AMOUNT	The free amount that can be allocated.
TOTAL_FREE_AMOUNT	Free amount available for allocations including the free amount of pooled resources if it is a pool.
IS_ONLINE	This is the availability status of the resource.
FACTOR	This is the correction factor by which the requested amount is multiplied.
TIMESTAMP	The timestamp indicates the last time the Resource State changed.
SCOPE	The scope in which the resource was created.
MANAGER_ID	ID of the Managing Pool.
MANAGER_NAME	Name of the Managing Pool.
MANAGER_SCOPENAME	Name of the scope in which the Managing Pool was created.
HAS_CHILDREN	Flag indicating whether a Pool Child has managed resources/pools. If it is not a pool, this is always FALSE.

Continued on next page

Continued from previous page

Field	Description
POOL_CHILD	This flag indicates whether the displayed resource is a child of the pool.
TRACE_INTERVAL	The trace interval is the minimum time in seconds between when Trace Records are written.
TRACE_BASE	The trace base is the basis for the valuation period (B).
TRACE_BASE_MULTIPLIER	The base multiplier determines the multiplication factor (M) of the trace base.
TD0_AVG	The average resource allocation of the last $B * M^0$ seconds.
TD1_AVG	The average resource allocation of the last $B * M^1$ seconds.
TD2_AVG	The average resource allocation of the last $B * M^2$ seconds.
LW_AVG	The average allocation since the last time a Trace Record was written.
LAST_WRITE	The time the last Trace Record was written.
PRIVS	String containing the users privileges on the object

Table 22.55.: Description of the output structure of the show scope subtable

CONFIG The layout of the CONFIG table is shown in the table below.

Field	Description
KEY	The name of the configuration variable.
VALUE	The value of the configuration variable.
LOCAL	Indicates whether the Key Value Pair is defined at local or parent level.
ANCESTOR_SCOPE	This is the scope in which the Key Value Pair is defined.
ANCESTOR_VALUE	This is the value that is defined at parent level.

Table 22.56.: Description of the output structure of the show scope subtable

CONFIG_ENVMAPPING The layout of the CONFIG_ENVMAPPING table is shown in the table below.

Field	Description
KEY	Name of the environment variable.
VALUE	Name of the environment variable that is to be set.
LOCAL	Indicates whether the Key Value Pair is defined at local or parent level.
ANCESTOR_SCOPE	This is the scope in which the Key Value Pair is defined.
ANCESTOR_VALUE	This is the value that is defined at parent level.

Table 22.57.: Description of the output structure of the show scope subtable

PARAMETERS The layout of the PARAMETERS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	Name of the parameter.
EXPORT_NAME	The export name defines the name under which the value of the parameter is exported to the process's environment.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.
EXPRESSION	Expression for the parameter type EXPRESSION.
DEFAULT_VALUE	The default value of the parameter.
REFERENCE_TYPE	Type of object that is being referenced.
REFERENCE_PATH	The path to the object that is being referenced.
REFERENCE_PRIVS	The user's privileges for the object that is being referenced.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
ID	The repository object id
NAME	Name of the parameter.
TYPE	This is the parameter type.
IS_LOCAL	True for local parameters that are only visible for the job itself.

Continued on next page

Continued from previous page

Field	Description
REFERENCE_TYPE	Type of object that is referencing the parameter.
REFERENCE_PATH	The path to the object that is referencing the parameter.
REFERENCE_PRIVS	The user's privileges for the object that is referencing the parameter.
REFERENCE_PARAMETER	Name of the parameter that is being referenced.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined

Table 22.58.: Description of the output structure of the show scope subtable

show session

Purpose

Purpose The purpose of the show session statement is to get more detailed information about the specified or the current session.

Syntax

Syntax The syntax for the show session statement is

show session [*sid*]

Description

Description The show session statement gives you detailed information about the specified or current session.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
THIS	With This, the proprietary session is shown with an asterisk.
SESSIONID	This is the ID of the session.
START	The time from which the session was started.
USER	Name of the user name used for the session login.
UID	The ID of the user.
IP	The IP address of the computer from which the connection was set up.
TXID	The ID of the last or current transaction.
IDLE	The time in seconds that has elapsed since the last transaction.
TIMEOUT	The time in seconds after which an automatic logout takes place (0 means unlimited).

Continued on next page

Continued from previous page

Field	Description
STATEMENT	The statement that is being executed at this time.

Table 22.59.: Description of the output structure of the show session statement

show system

Purpose

Purpose The purpose of the show system statement is to get information about the actual configuration of the running server.

Syntax

Syntax The syntax for the show system statement is

show system

show system with lock

Description

Description The show system statement gives you detailed information about the current configuration of the running server.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
VERSION	The current version of the software.
MAX_LEVEL	The maximum compatibility level of the software.
NUM_CPU	The number of processors present in the system.
MEM_USED	The amount of used memory.
MEM_FREE	The amount of free memory.
MEM_MAX	The maximum amount of memory that the server can use.
STARTTIME	The time when the server was started.
UPTIME	The time when the server started running.
HITRATE	The hit rate in the environment cache of the Scheduling Thread.

Continued on next page

Continued from previous page	
Field	**Description**
LOCK_HWM	The *lock_hwm* shows the high water mark of active locks in the system. This field is only relevant if multiple writer threads are active.
LOCKS_REQUESTED	The *locks_requested* field shows the total number of locks requested since server startup. This field is only relevant in case of multiple writer threads.
LOCKS_USED	This field shows the number of locks currently in use. It is only relevant in case of multiple writer threads.
LOCKS_DISCARDED	The field *locks_discarded* shows the number of locks removed from the system.
CNT_RW_TX	The number of R/W Transactions since server startup.
CNT_DL	The number of deadlocks since server startup.
CNT_WL	The number of single threaded write worker transactions since server startup.
WORKER	A table with a list of the Worker Threads. See also table 22.61 on page 341
LOCKING STATUS	This field is not yet documented

Table 22.60.: Description of the output structure of the show system statement

WORKER The layout of the WORKER table is shown in the table below.

Field	**Description**
ID	The repository object id
TYPE	This field is not yet documented
NAME	The object name
STATE	The state of the worker.
TIME	The time from which the worker is in a state.

Table 22.61.: Description of the output structure of the show system subtable

show trigger

Purpose

Purpose The purpose of the show trigger statement is to get detailed information about the specified trigger.

Syntax

Syntax The syntax for the show trigger statement is

show trigger *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]

TRIGGEROBJECT:
resource *resourcepath* **in** *folderpath*
| **job definition** *folderpath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** *serverpath*

Description

Description The show trigger statement gives you detailed information about the specified trigger.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
OBJECTTYPE	The type of object in which the trigger is defined.
OBJECTNAME	Full path name of the object in which the trigger is defined.
ACTIVE	States whether the trigger is active or not.
ACTION	The Trigger Action that takes place is either the submit of a job or a rerun of the triggering job.

Continued on next page

Continued from previous page

Field	Description
SUBMIT_TYPE	The object type that is submitted when the trigger is activated.
SUBMIT_NAME	The object type that is submitted when the trigger is activated.
SUBMIT_SE_OWNER	The owner of the object that is submitted.
SUBMIT_PRIVS	The privileges for the object that is to be submitted.
MAIN_TYPE	Type of main job (job/batch).
MAIN_NAME	Name of the main job.
MAIN_SE_OWNER	Owner of the main job.
MAIN_PRIVS	Privileges for the main job.
PARENT_TYPE	Type of parent job (job/batch).
PARENT_NAME	Name of the parent job.
PARENT_SE_OWNER	Owner of the parent job.
PARENT_PRIVS	Privileges for the parent job.
TRIGGER_TYPE	The trigger type that describes when it is activated.
MASTER	Indicates whether the trigger submitted a master or a child.
IS_INVERSE	In case of an inverse trigger, the trigger is regarded to belong to the triggered job. The trigger can be regarded as some kind of callback function. This flag has no effects on the trigger's behaviour.
SUBMIT_OWNER	The owner group that has to be used with the Submitted Entity.
IS_CREATE	Indicates whether the trigger reacts to create events.
IS_CHANGE	Indicates whether the trigger reacts to change events.
IS_DELETE	Indicates whether the trigger reacts to delete events.
IS_GROUP	Indicates whether the trigger handles the events as a group.
MAX_RETRY	The maximum number of trigger activations in a single Submitted Entity.
SUSPEND	Specifies whether the submitted object is suspended.

Continued on next page

Continued from previous page	
Field	**Description**
RESUME_AT	Time of the automatic resume.
RESUME_IN	Number of time units until the automatic resume.
RESUME_BASE	Time unit for resume_in.
WARN	Specifies whether a warning has to be given when the activation limit is reached.
LIMIT_STATE	This field specifies which state the triggering job aqcuires if the fire limit is reached. If the triggering job has a final state already, this specification is ignored. If the value is **NONE**, no state change takes place.
CONDITION	Conditional expression to define the trigger condition.
CHECK_AMOUNT	The amount of CHECK_Base units for checking the condition in the case of non-synchronised triggers.
CHECK_BASE	Units for the CHECK_AMOUNT.
COMMENT	Comment if defined
COMMENTTYPE	Type of comment if a comment is defined
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
STATES	Table of states that cause the trigger to be activated.
	See also table 22.63 on page 345

Table 22.62.: Description of the output structure of the show trigger statement

STATES The layout of the STATES table is shown in the table below.

Field	**Description**
ID	The repository object id
FROM_STATE	The trigger is activated if this is the old Resource State.
TO_STATE	The trigger is activated if this is the new Resource State or the Exit State of the object.

Continued on next page

Continued from previous page

Field	Description

Table 22.63.: Description of the output structure of the show trigger subtable

show user

Purpose

Purpose The purpose of the show user statement is to show detailed information about the user.

Syntax

Syntax The syntax for the show user statement is

show user [*username*]

Description

Description The show user statement gives you detailed information about the user.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	The repository object id
NAME	The object name
IS_ENABLED	Flag that shows whether the user is allowed to log on.
DEFAULT_GROUP	The default group of the users who are being used by the owners of the object.
CREATOR	Name of the user created the object
CREATE_TIME	Date and time of object creation
CHANGER	Name of the last user changed the object
CHANGE_TIME	Date and time of object change
PRIVS	String containing the users privileges on the object
MANAGE_PRIVS	Table of the manage privileges. See also table 22.65 on page 347
GROUPS	Table of groups to which the user belongs. See also table 22.66 on page 347

Continued on next page

Continued from previous page	
Field	**Description**
COMMENTTYPE	Type of comment if a comment is defined
COMMENT	Comment if defined
	See also table 22.67 on page 347

Table 22.64.: Description of the output structure of the show user statement

MANAGE_PRIVS The layout of the MANAGE_PRIVS table is shown in the table below.

Field	Description
PRIVS	String containing the users privileges on the object

Table 22.65.: Description of the output structure of the show user subtable

GROUPS The layout of the GROUPS table is shown in the table below.

Field	Description
ID	The repository object id
NAME	The object name
PRIVS	String containing the users privileges on the object

Table 22.66.: Description of the output structure of the show user subtable

COMMENT The layout of the COMMENT table is shown in the table below.

Field	Description
TAG	This field is not yet documented
COMMENT	Comment if defined

Table 22.67.: Description of the output structure of the show user subtable

23. shutdown commands

shutdown

Purpose

Purpose The purpose of the shutdown statement is to instruct the addressed jobservers to terminate.

Syntax

Syntax The syntax for the shutdown statement is

shutdown *serverpath*

Description

Description The shutdown statement is used to shut down the addressed job server.

Output

Output This statement returns a confirmation of a successful operation

24. stop commands

stop server

Purpose

Purpose The purpose of the stop server statement is to instruct the server to terminate.

Syntax

Syntax The syntax for the stop server statement is

stop server

stop server kill

Description

Description The stop server statement is used to shut down the server. If this should not function correctly for any reason, the server can also be forced to shut down using **kill**.

Output

Output This statement returns a confirmation of a successful operation

25. submit commands

submit

Purpose

Purpose The purpose of the submit statement is to execute a master batch or job as well as all defined children.

Syntax

Syntax The syntax for the submit statement is

submit *folderpath* [**with** WITHITEM {**,** WITHITEM}]

submit *aliasname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
check only
| **childtag =** *string*
| **nicevalue =** *signed_integer*
| **parameter = none**
| **parameter = (** PARAM {**,** PARAM} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **submittag =** *string*
| < **nosuspend** | **suspend** >
| **unresolved =** < **error** | **ignore** | **suspend** | **defer** >
| **group =** *groupname*

PARAM:
parametername **=** < *string* | *number* >

Description

Description The submit statement is used to submit a job or batch. There are two kinds of submit command:

- The first kind is used by users, who can also be programs, and the Time Scheduling Module. This form submits Master Jobs and Batches.
- The second form of the statement is used by jobs to submit dynamic children.

check only The check only option is used to verify whether a Master Submittable Batch or Job can be submitted. This means that a check is run to ascertain whether all the dependencies can be fulfilled and all the referenced parameters are defined. Whether the jobs can be executed in any scope or not is not verified. This is a situation that can arise at any point during the runtime.
Positive feedback means that, from the system's perspective, the job or batch can be submitted.

childtag The childtag option is used by jobs to submit several instances of the same Scheduling Entity and to be able to differentiate between them.
An error is triggered if the same Scheduling Entity is submitted twice using the same childtag. The content of the childtag has no further significance for the Scheduling System.
The maximum length for a childtag is 70 characters. The childtag option is ignored in the case of a Master Submit.

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

nicevalue The nicevalue option defines a correction that is used for the calculation of the priorities for the job and its children. Values between -100 and 100 are permitted.

parameter The parameter option is used to specify the value of Job Parameters for the submit. The parameters are set in the scope of the Master Batch or Job. This means that if parameters are specified that are not defined in the Master Batch or Job, these parameters are invisible to any children.

submittag If the submittag is specified, it must have a unique name for the Submitted Entity. This tag was introduced to be able to programmatically submit jobs and batches and to resubmit the job or batch with the same tag following a crash of one of the components. If the job submit was successful the first time, the second submit will report an error. If not, the second submit will succeed.

unresolved The unresolved option defines how the server is to react to unresolved dependencies. This option is mainly used if parts of a batch are submitted following repair work. The faulty part is normally cancelled and then resubmitted as a Master Run. In this case the previous dependencies have to be ignored otherwise the submit will fail.

suspend The suspend option is used to submit jobs or batches and to suspend them at the same time. If nothing is defined, they are not suspended. This can be explicitly specified at the submit time.
If a job or batch was suspended, neither it nor its children are started. If a job is already running, it will not reach a Final State if it is suspended.

resume The resume option can be used together with the suspend option to cause a delayed execution. There are two ways to do this. A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
This option can be used to reproduce the `at` functionality without creating a schedule.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	ID of the Submitted Entity.

Table 25.1.: Description of the output structure of the submit statement

26. suspend commands

suspend

Purpose

Purpose The purpose of the suspend statement is to prevent further jobs to be executed by this jobserver. See also the resume statement on page 260.

Syntax

Syntax The syntax for the suspend statement is

suspend *serverpath*

Description

Description The suspend statement prevents further jobs from being executed by this job server.

Output

Output This statement returns a confirmation of a successful operation

Part III.

Jobserver Commands

27. Jobserver Commands

alter job

Purpose

Purpose The purpose of the alter job statement is to change properties of the specified job. This statement is is used by job administrators, jobservers, and by the job itself.

Syntax

Syntax The syntax for the alter job statement is

alter job *jobid*
with WITHITEM {**,** WITHITEM}

alter job
with WITHITEM {**,** WITHITEM}

WITHITEM:
< **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **comment =** *string*
| **error text =** *string*
| **exec pid =** *pid*
| **exit code =** *signed_integer*
| **exit state =** *statename* [**force**]
| **ext pid =** *pid*
| **ignore resource = (** *id* {**,** *id*} **)**
| **ignore dependency = (** *jobid* [**recursive**] {**,** *jobid* [**recursive**]} **)**
| **kill**
| **nicevalue =** *signed_integer*
| **priority =** *integer*
| **renice =** *signed_integer*
| **rerun** [**recursive**]
| **resume**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **run =** *integer*
| **state =** JOBSTATE
| **timestamp =** *string*
| **warning =** *string*

JOBSTATE:

broken active
broken finished
dependency wait
error
finished
resource wait
running
started
starting
synchronize wait

Description

Description

The alter job command is used for several purposes. Firstly, job servers use this command to document the progress of a job. All the state transitions a job undergoes during the time when the job is the responsibility of a job server are performed using the alter job command.
Secondly, some changes such as ignoring dependencies or resources, as well as changing the priority of a job, are carried out manually by an administrator.
The Exit State of a job in a Pending State can be set by the job itself or by a process that knows the job ID and key of the
job that is to be changed.

cancel The cancel option is used to cancel the addressed job and all non-Final Children. A job can only be cancelled if neither the job itself nor one of its children is active. Cancelling a running job will set the job in a cancelling state. The effective cancel is postponed until the job is finished
If a Scheduling Entity is dependent upon the cancelled job, it can become unreachable. In this case the dependent job does not acquire the Unreachable Exit State defined in the Exit State Profiles, but is set as having the Job State "Unreachable". It is the operator's task to restore this job back to the job state "Dependency Wait" by ignoring dependencies or even to cancel it.
Cancelled jobs are considered to be just like Final Jobs without a Final Exit. This means that the parents of a cancelled job become final without taking into consideration the Exit State of the cancelled job. In this case the dependent jobs of the parents continue running normally.
The cancel option can only be used by users.

comment The comment option is used to document an action or to add a comment to the job. Comments can have a maximum length of 1024 characters. Any number of comments can be saved for a job.
Some comments are saved automatically. For example, if a job attains a Restartable State, a log is written to document this fact.

error text The error text option is used to write error information about a job. This can be done by the responsible job server or a user. The server can write this text itself as well.
This option is normally used if the job server cannot start the corresponding process. Possible cases are where it is not possible to switch to the defined working directory, if the executable program cannot be found, or when opening the error log file triggers an error.

exec pid The exec pid option is used exclusively by the job server to set the process ID of the control process within the server.

exit code The exit code option is used by the job server to tell the repository server with which Exit Code the process has finished. The repository server now calculates the matching Exit State from the Exit State Mapping that was used.

exit state The exit state option is used by jobs in a pending state to set their state to another value. This is usually a Restartable or Final State.
Alternatively, this option can be used by administrators to set the state of a non-final job.
If the Force Flag is not being used, the only states that can be set are those which are theoretically attainable by applying the Exit State Mapping to any Exit Code. The set state must exist in the Exit State Profile.

ext pid The ext pid option is used exclusively by the job server to set the process ID of the started user process.

ignore resource The ignore resource option is used to revoke individual Resource Requests. The ignored resource is then no longer requested.
If the parameters of a resource are being referenced, that resource cannot be ignored.
If invalid IDs have been specified, it is skipped. All other specified resources are ignored. Invalid IDs in this context are the IDs of resources that are not requested by the job.
The ignoring of resources is logged.

ignore dependency The ignore dependency option is used to ignore defined dependencies. If the **recursive** flag is used, not only do the job or batch ignore the dependencies, but its children do so as well.

kill The kill option is used to submit the defined Kill Job. If no Kill Job has been defined, it is not possible to forcibly terminate the job from within BICsuite. The job obviously has to be active, that means it must be **running**, **killed** or **broken_active**.

The last two states are not regular cases. When a Kill Job has been submitted, the Job State is **to_kill**. After the Kill Job has terminated, the Job State of the killed job is set to **killed** unless it has been completed, in which case it is **finished** or **final**. This means that the job with the Job State **killed** is always still running and that at least one attempt has been made to terminate it.

nicevalue The nicevalue option is used to change the priority or the nicevalue of a job or batch and all of its children. If a child has several parents, any changes you make can, but do not necessarily have to, affect the priority of the child in the nicevalue of one of the parents. Where there are several parents, the maximum nicevalue is searched for.

This means that if Job C has three Parents P1, P2 and P3, whereby P1 sets a nice value of 0, P2 sets a nicevalue of 10 and P3 a nicevalue of -10, the effective nicevalue is -10. (The lower the nicevalue the better). If the nicevalue for P2 is changed to -5, nothing happens because the -10 of P3 is better than -5. If the nicevalue of P3 falls to 0, the new effective nicevalue for Job C is -5.

The nicevalues can have values between -100 and 100. Values that exceed this range are tacitly adjusted.

priority The priority option is used to change the (static) priority of a job. Because batches and milestones are not executed, priorities are irrelevant to them.

Changing the priority only affects the changed job. Valid values lie between 0 and 100. In this case, 100 corresponds to the lowest priority and 0 is the highest priority.

When calculating the dynamic priority of a job, the scheduler begins with the static priority and adjusts it according to how long the job has already been waiting. If more than one job has the same dynamic priority, the job with the lowest job ID is scheduled first.

renice The renice option is similar to the nicevalue option with the difference that the renice option functions relatively while the nicevalue option functions absolutely. If some batches have a nicevalue of 10, a renice of -5 causes the nicevalue to rise to 5. (It rises because the lower the number, the higher the priority).

rerun The rerun option is used to restart a job in a Restartable State. If you attempt to restart a job that is not restartable, an error message is displayed. A job is restartable if it is in a Restartable State or it has the Job State **error** or **broken_finished**.

If the **recursive** flag has been specified, the job itself and all its direct and indirect children that are in a Restartable State are restarted. If the job itself is final, this is not considered to be an error. It is therefore possible to recursively restart batches.

resume The resume option is used to reactivate a suspended job or batch. There are two ways to do this. The suspended job or batch can either be reactivated immediately or a delay can be set.
A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
For details about specifying a time, refer to the overview on page 6. The resume option can be used together with the suspend option. Here, the job is suspended and then resumed again after (or at) a specified time.

run The run option is used by the job server to ensure that the modified job matches the current version.
Theoretically, the computer could crash after a job has been started by a job server. To complete the work, the job is manually restarted from another job server. After the first system has been booted, the job server can attempt to change the job state to broken_finished without knowing anything about what happened after the crash. Using the run option then prevents the wrong state from being set.

state The state option is mainly used by job servers, but it can also be used by administrators. It is not recommended to do so unless you know exactly what you are doing.
The usual procedure is that the job server sets the state of a job from **starting** to **started**, from **started** to **running**, and from **running** to
finished. In the event of a crash or any other problems, it is possible for the job server to set the job state to **broken_active** or **broken_finished**. This means that the Exit Code of the process is not available and the Exit State has to be set manually.

suspend The suspend option is used to suspend a batch or job. It always functions recursively. If a parent is suspended, its children are all suspended as well. The resume option is used to reverse the situation.
The effect of the restrict option is that cwa resume can be done by members of the group ADMIN only.

timestamp The timestamp option is used by the job server to set the timestamps of the state transition in keeping with the local time from the perspective of job server.

Output

Output This statement returns a confirmation of a successful operation

alter jobserver

Purpose

The purpose of the alter jobserver statement is to alter properties of a jobserver. *Purpose*

Syntax

The syntax for the alter jobserver statement is *Syntax*

alter [**existing**] **job server**
with < **fatal** | **nonfatal** > **error text =** *string*

alter [**existing**] **job server**
with dynamic PARAMETERS

PARAMETERS:
parameter = none
| **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**

PARAMETERSPEC:
parametername **=** < *string* | *number* >

Description

The alter scope command is both a user command and a job server command. *Description*
It is used as a user command to change the configuration or other properties of a scope or job server. Further details are described in the create scope command on page 129.
The syntax of the user command corresponds to the first form of the alter scope command. As a job server command, it is used to notify the server about any errors. If the Fatal Flag is used, this means that the job server is shutting down. In the other case, the job server continues running.
The third form of the alter jobserver command is also used by the job server. The job server publishes the values of its dynamic parameter. The server uses published values to resolve parameters in the command line and log file information when retrieving a job.

Output

This statement returns a confirmation of a successful operation *Output*

connect

Purpose

Purpose The purpose of the connect statement is to authenticate a jobserver to the server.

Syntax

Syntax The syntax for the connect statement is

connect job server *serverpath* **.** *servername* **identified by** *string* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:

command = (*sdms-command* **)**
| **protocol =** PROTOCOL
| **session =** *string*
| **timeout =** *integer*
| < **trace** | **notrace** >
| **trace level =** *integer*

PROTOCOL:

json
| **line**
| **perl**
| **python**
| **serial**
| **xml**

Description

Description The connect command is used to authenticate the connected process on the server to. A communication protocol can be optionally specified. The default protocol is **line**.
The selected protocol defines the output format. All protocols except for **serial** return ASCII output. The protocol **serial**
returns a serialized Java object.
An executable command can also be returned when the connection is established. In this case, the output of the accompanying command is used as the output for the connect command. If the command fails, but the connect was successful, the connection remains active.
An example for all protocols except the **serial** protocol is given below.

The line protocol The line protocol only returns an ASCII text as the result from a command.

```
connect donald identified by 'duck' with protocol = line;

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

The XML protocol The XML protocol returns an XML structure as the result from a command.

```
connect donald identified by 'duck' with protocol = xml;
<OUTPUT>
<DATA>
<TITLE>Connect</TITLE>
<RECORD>
<CONNECT_TIME>19 Jan 2005 11:15:16 GMT</CONNECT_TIME></RECORD>
</DATA>
<FEEDBACK>Connected</FEEDBACK>
</OUTPUT>
```

The python protocol The python protocol returns a Python structure that can be valuated using the Python eval function.

```
connect donald identified by 'duck' with protocol = python;
{
'DATA' :
{
'TITLE' : 'Connect',
'DESC' : [
'CONNECT_TIME'
],
'RECORD' : {
'CONNECT_TIME' : '19 Jan 2005 11:16:08 GMT'}
}
,'FEEDBACK' : 'Connected'
}
```

The perl protocol The perl protocol returns a Perl structure that can be valuated using the Perl eval function.

```
connect donald identified by 'duck' with protocol = perl;
{
'DATA' =>
{
```

```
'TITLE' => 'Connect',
'DESC' => [
'CONNECT_TIME'
],
'RECORD' => {
'CONNECT_TIME' => '19 Jan 2005 11:19:19 GMT'}
}
,'FEEDBACK' => 'Connected'
}
```

Output

Output This statement returns a confirmation of a successful operation

deregister

Purpose

Purpose

The purpose of the deregister statement is to notify the server that the jobserver is not to process jobs anymore. See also the register statement on page 238.

Syntax

Syntax

The syntax for the deregister statement is

deregister *serverpath* **.** *servername*

Description

Description

The deregister statement is used to notify the server about a more or less permanent failure of a job server.
This message prompts different server actions. Firstly, all the running jobs on the job server (i.e. jobs in the state **started**, **running**, **to_kill** and **killed**) are set to the state **broken_finished**. Jobs in the state **starting** are reset to **runnable**. The job server is then removed from the list of job servers that are able to process jobs so that this job server is consequently no longer allocated any more jobs. A side effect of this is that jobs that can only run on this server due to their resource requirements are set to the state **error** with the message "Cannot run in any scope because of resource shortage". Finally, a complete reschedule is executed so that jobs are redistributed among the job servers. The job server is added to the list of job-processing job servers again by re-registering it (refer to the register statement on page 238).

Output

Output

This statement returns a confirmation of a successful operation

disconnect

Purpose

Purpose The purpose of the disconnect statement is to terminate the server connection.

Syntax

Syntax The syntax for the disconnect statement is

disconnect

Description

Description The connection to the server can be shut down using the disconnect statement.

Output

Output This statement returns a confirmation of a successful operation

get next job

Purpose

The purpose of the get next job command is to fetch the next assignment from the server. *Purpose*

Syntax

The syntax for the get next job statement is *Syntax*

get next job

Description

The job server uses the get next job statement to fetch the next command to be executed from the server. *Description*

Output

This statement returns an output structure of type table. *Output*

Main output description The data items of the output are described in the table below.

Field	Description
COMMAND	The command to be executed by the job server. (NOP, ALTER, SHUTDOWN, STARTJOB)
CONFIG	Changed configuration. This value is only present in the case of an ALTER command.
ID	The ID of the job to be started; only present for the STARTJOB command.
DIR	The working directory of the job to be started; only present for the STARTJOB command.
LOG	The log file of the job to be started; only present for the STARTJOB command.
LOGAPP	Indicator showing whether the log file is to be opened with Append; only present for the STARTJOB command.

Continued on next page

Continued from previous page

Field	Description
ERR	The error log file of the job to be started; only present for the STARTJOB command.
ERRAPP	Indicator showing whether the error log file is to be opened with Append; only present for the STARTJOB command.
CMD	File name of the executable to be started; only present for the STARTJOB command.
ARGS	The command line parameter of the executable to be started; only present for the STARTJOB command.
ENV	Additional entries for the environment of the executable to be started; only present for the STARTJOB command.
RUN	Number of the run. Refer also to the alter job statement on page 45; only present for the STARTJOB command.
JOBENV	Vector of key value pairs defining the job defined environment variables to set before job execution.

Table 27.1.: Description of the output structure of the get next job statement

multicommand

Purpose

This statement is used to control the behaviour of the SDMS Server. *Purpose*

Syntax

The syntax for the multicommand statement is *Syntax*

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Description

The multicommands allow multiple SDMS commands to be executed together, i.e. in one transaction. This ensures that either all the statements are executed without any errors or nothing happens at all. Not only that, but the transaction is not interrupted by other write transactions. *Description*
If the **rollback** keyword is specified, the transaction is undone at the end of the processing. This means that you can test whether the statements can be correctly processed (technically speaking).

Output

This statement returns a confirmation of a successful operation *Output*

reassure

Purpose

Purpose The purpose of the reassure job statement is to get a confirmation from the server about the necessity of starting a job after a jobserver was started.

Syntax

Syntax The syntax for the reassure statement is

reassure *jobid* [**with run =** *integer*]

Description

Description With the reassure statement a job server gets a confirmation from the server as to whether a job should be started. This statement is used when a job server boots up and there is a job in the **starting** state.

Output

Output This statement returns a confirmation of a successful operation

register

Purpose

Purpose

The purpose of the register statement is to notify the server that the jobserver is ready to process jobs.

Syntax

Syntax

The syntax for the register statement is

register *serverpath* **.** *servername*
with pid = *pid* [**suspend**]

register with pid = *pid*

Description

Description

The first form is used by the operator to enable jobs to be executed by the specified job server.
The second form is used by the job server itself to notify the server that it is ready to execute jobs.
Jobs are scheduled for this job server (unless it is suspended) regardless of whether the server is connected or not.
Refer to the deregister statement on page 144.

pid The pid option provides the server with information about the job server's process ID at operating level.

suspend The suspend option causes the job server to be transferred to a suspended state.

Output

Output

This statement returns a confirmation of a successful operation

Part IV.

Job Commands

28. Job Commands

alter job

Purpose

Purpose The purpose of the alter job statement is to change properties of the specified job. This statement is is used by job administrators, jobservers, and by the job itself.

Syntax

Syntax The syntax for the alter job statement is

alter job *jobid*
with WITHITEM {**,** WITHITEM}

alter job
with WITHITEM {**,** WITHITEM}

WITHITEM:
< **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **comment =** *string*
| **error text =** *string*
| **exec pid =** *pid*
| **exit code =** *signed_integer*
| **exit state =** *statename* [**force**]
| **ext pid =** *pid*
| **ignore resource = (** *id* {**,** *id*} **)**
| **ignore dependency = (** *jobid* [**recursive**] {**,** *jobid* [**recursive**]} **)**
| **kill**
| **nicevalue =** *signed_integer*
| **priority =** *integer*
| **renice =** *signed_integer*
| **rerun** [**recursive**]
| **resume**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **run =** *integer*
| **state =** JOBSTATE
| **timestamp =** *string*
| **warning =** *string*

JOBSTATE:

broken active
broken finished
dependency wait
error
finished
resource wait
running
started
starting
synchronize wait

Description

Description

The alter job command is used for several purposes. Firstly, job servers use this command to document the progress of a job. All the state transitions a job undergoes during the time when the job is the responsibility of a job server are performed using the alter job command.
Secondly, some changes such as ignoring dependencies or resources, as well as changing the priority of a job, are carried out manually by an administrator.
The Exit State of a job in a Pending State can be set by the job itself or by a process that knows the job ID and key of the
job that is to be changed.

cancel The cancel option is used to cancel the addressed job and all non-Final Children. A job can only be cancelled if neither the job itself nor one of its children is active. Cancelling a running job will set the job in a cancelling state. The effective cancel is postponed until the job is finished
If a Scheduling Entity is dependent upon the cancelled job, it can become unreachable. In this case the dependent job does not acquire the Unreachable Exit State defined in the Exit State Profiles, but is set as having the Job State "Unreachable". It is the operator's task to restore this job back to the job state "Dependency Wait" by ignoring dependencies or even to cancel it.
Cancelled jobs are considered to be just like Final Jobs without a Final Exit. This means that the parents of a cancelled job become final without taking into consideration the Exit State of the cancelled job. In this case the dependent jobs of the parents continue running normally.
The cancel option can only be used by users.

comment The comment option is used to document an action or to add a comment to the job. Comments can have a maximum length of 1024 characters. Any number of comments can be saved for a job.
Some comments are saved automatically. For example, if a job attains a Restartable State, a log is written to document this fact.

error text The error text option is used to write error information about a job. This can be done by the responsible job server or a user. The server can write this text itself as well.
This option is normally used if the job server cannot start the corresponding process. Possible cases are where it is not possible to switch to the defined working directory, if the executable program cannot be found, or when opening the error log file triggers an error.

exec pid The exec pid option is used exclusively by the job server to set the process ID of the control process within the server.

exit code The exit code option is used by the job server to tell the repository server with which Exit Code the process has finished. The repository server now calculates the matching Exit State from the Exit State Mapping that was used.

exit state The exit state option is used by jobs in a pending state to set their state to another value. This is usually a Restartable or Final State.
Alternatively, this option can be used by administrators to set the state of a non-final job.
If the Force Flag is not being used, the only states that can be set are those which are theoretically attainable by applying the Exit State Mapping to any Exit Code. The set state must exist in the Exit State Profile.

ext pid The ext pid option is used exclusively by the job server to set the process ID of the started user process.

ignore resource The ignore resource option is used to revoke individual Resource Requests. The ignored resource is then no longer requested.
If the parameters of a resource are being referenced, that resource cannot be ignored.
If invalid IDs have been specified, it is skipped. All other specified resources are ignored. Invalid IDs in this context are the IDs of resources that are not requested by the job.
The ignoring of resources is logged.

ignore dependency The ignore dependency option is used to ignore defined dependencies. If the **recursive** flag is used, not only do the job or batch ignore the dependencies, but its children do so as well.

kill The kill option is used to submit the defined Kill Job. If no Kill Job has been defined, it is not possible to forcibly terminate the job from within BICsuite. The job obviously has to be active, that means it must be **running**, **killed** or **broken_active**.

The last two states are not regular cases. When a Kill Job has been submitted, the Job State is **to_kill**. After the Kill Job has terminated, the Job State of the killed job is set to **killed** unless it has been completed, in which case it is **finished** or **final**. This means that the job with the Job State **killed** is always still running and that at least one attempt has been made to terminate it.

nicevalue The nicevalue option is used to change the priority or the nicevalue of a job or batch and all of its children. If a child has several parents, any changes you make can, but do not necessarily have to, affect the priority of the child in the nicevalue of one of the parents. Where there are several parents, the maximum nicevalue is searched for.

This means that if Job C has three Parents P1, P2 and P3, whereby P1 sets a nice value of 0, P2 sets a nicevalue of 10 and P3 a nicevalue of -10, the effective nicevalue is -10. (The lower the nicevalue the better). If the nicevalue for P2 is changed to -5, nothing happens because the -10 of P3 is better than -5. If the nicevalue of P3 falls to 0, the new effective nicevalue for Job C is -5.

The nicevalues can have values between -100 and 100. Values that exceed this range are tacitly adjusted.

priority The priority option is used to change the (static) priority of a job. Because batches and milestones are not executed, priorities are irrelevant to them.

Changing the priority only affects the changed job. Valid values lie between 0 and 100. In this case, 100 corresponds to the lowest priority and 0 is the highest priority.

When calculating the dynamic priority of a job, the scheduler begins with the static priority and adjusts it according to how long the job has already been waiting. If more than one job has the same dynamic priority, the job with the lowest job ID is scheduled first.

renice The renice option is similar to the nicevalue option with the difference that the renice option functions relatively while the nicevalue option functions absolutely. If some batches have a nicevalue of 10, a renice of -5 causes the nicevalue to rise to 5. (It rises because the lower the number, the higher the priority).

rerun The rerun option is used to restart a job in a Restartable State. If you attempt to restart a job that is not restartable, an error message is displayed. A job is restartable if it is in a Restartable State or it has the Job State **error** or **broken_finished**.

If the **recursive** flag has been specified, the job itself and all its direct and indirect children that are in a Restartable State are restarted. If the job itself is final, this is not considered to be an error. It is therefore possible to recursively restart batches.

resume The resume option is used to reactivate a suspended job or batch. There are two ways to do this. The suspended job or batch can either be reactivated immediately or a delay can be set.
A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
For details about specifying a time, refer to the overview on page 6. The resume option can be used together with the suspend option. Here, the job is suspended and then resumed again after (or at) a specified time.

run The run option is used by the job server to ensure that the modified job matches the current version.
Theoretically, the computer could crash after a job has been started by a job server. To complete the work, the job is manually restarted from another job server. After the first system has been booted, the job server can attempt to change the job state to broken_finished without knowing anything about what happened after the crash. Using the run option then prevents the wrong state from being set.

state The state option is mainly used by job servers, but it can also be used by administrators. It is not recommended to do so unless you know exactly what you are doing.
The usual procedure is that the job server sets the state of a job from **starting** to **started**, from **started** to **running**, and from **running** to
finished. In the event of a crash or any other problems, it is possible for the job server to set the job state to **broken_active** or **broken_finished**. This means that the Exit Code of the process is not available and the Exit State has to be set manually.

suspend The suspend option is used to suspend a batch or job. It always functions recursively. If a parent is suspended, its children are all suspended as well. The resume option is used to reverse the situation.
The effect of the restrict option is that cwa resume can be done by members of the group ADMIN only.

timestamp The timestamp option is used by the job server to set the timestamps of the state transition in keeping with the local time from the perspective of job server.

Output

Output This statement returns a confirmation of a successful operation

connect

Purpose

The purpose of the connect statement is to authenticate a job to the server. *Purpose*

Syntax

The syntax for the connect statement is *Syntax*

connect job *jobid* **identified by** *string* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
command = (*sdms-command* **)**
| **protocol =** PROTOCOL
| **session =** *string*
| **timeout =** *integer*
| < **trace** | **notrace** >
| **trace level =** *integer*

PROTOCOL:
json
| **line**
| **perl**
| **python**
| **serial**
| **xml**

Description

The connect command is used to authenticate the connected process on the server to. A communication protocol can be optionally specified. The default protocol is **line**. *Description*
The selected protocol defines the output format. All protocols except for **serial** return ASCII output. The protocol **serial**
returns a serialized Java object.
An executable command can also be returned when the connection is established. In this case, the output of the accompanying command is used as the output for the connect command. If the command fails, but the connect was successful, the connection remains active.
An example for all protocols except the **serial** protocol is given below.

The line protocol The line protocol only returns an ASCII text as the result from a command.

```
connect donald identified by 'duck' with protocol = line;

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

The XML protocol The XML protocol returns an XML structure as the result from a command.

```
connect donald identified by 'duck' with protocol = xml;
<OUTPUT>
<DATA>
<TITLE>Connect</TITLE>
<RECORD>
<CONNECT_TIME>19 Jan 2005 11:15:16 GMT</CONNECT_TIME></RECORD>
</DATA>
<FEEDBACK>Connected</FEEDBACK>
</OUTPUT>
```

The python protocol The python protocol returns a Python structure that can be valuated using the Python eval function.

```
connect donald identified by 'duck' with protocol = python;
{
'DATA' :
{
'TITLE' : 'Connect',
'DESC' : [
'CONNECT_TIME'
],
'RECORD' : {
'CONNECT_TIME' : '19 Jan 2005 11:16:08 GMT'}
}
,'FEEDBACK' : 'Connected'
}
```

The perl protocol The perl protocol returns a Perl structure that can be valuated using the Perl eval function.

```
connect donald identified by 'duck' with protocol = perl;
{
'DATA' =>
{
```

```
'TITLE' => 'Connect',
'DESC' => [
'CONNECT_TIME'
],
'RECORD' => {
'CONNECT_TIME' => '19 Jan 2005 11:19:19 GMT'}
}
,'FEEDBACK' => 'Connected'
}
```

Output

This statement returns a confirmation of a successful operation *Output*

disconnect

Purpose

Purpose The purpose of the disconnect statement is to terminate the server connection.

Syntax

Syntax The syntax for the disconnect statement is

disconnect

Description

Description The connection to the server can be shut down using the disconnect statement.

Output

Output This statement returns a confirmation of a successful operation

get parameter

Purpose

Purpose

The purpose of the get parameter statement is to get the value of the specified parameter within the context of the requesting job, respectively the specified job.

Syntax

Syntax

The syntax for the get parameter statement is

get parameter *parametername* [< **strict** | **warn** | **liberal** >]

get parameter of *jobid parametername* [< **strict** | **warn** | **liberal** >]

Description

Description

The get parameter statement is used to get the value of the specified parameter within the context of a job.
The additional option has the following meaning:

Option	Meaning
strict	The server returns an error if the requested parameter is not explicitly declared in the job definition.
warn	A message is written to the server's log file when an attempt is made to determine the value of an undeclared parameter.
liberal	An attempt to query an undeclared parameter is tacitly allowed.

The default behaviour depends on the configuration of the server.

Output

Output

Main output description The data items of the output are described in the table below.

Field	Description
VALUE	Value of the requested parameter

Table 28.1.: Description of the output structure of the get parameter statement

get submittag

Purpose

Purpose The purpose of the get submittag statement is to get a (server local) unique identifier from the server. This identifier can be used to avoid race conditions between frontend and backend when submitting jobs.

Syntax

Syntax The syntax for the get submittag statement is

get submittag

Description

Description The get submittag statement is used to acquire an identification from the server. This prevents race conditions between the front end and back end when jobs are submitted.
Such a situation arises when feedback about the submit does not reach the front end due to an error. By using a submittag, the front end can safely start a second attempt. The server recognises whether the job in question has already been submitted and responds accordingly. This reliably prevents the job from being submitted twice.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
VALUE	The requested Submit Tag

Table 28.2.: Description of the output structure of the get submittag statement

multicommand

Purpose

This statement is used to control the behaviour of the SDMS Server. *Purpose*

Syntax

The syntax for the multicommand statement is *Syntax*

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Description

The multicommands allow multiple SDMS commands to be executed together, i.e. in one transaction. This ensures that either all the statements are executed without any errors or nothing happens at all. Not only that, but the transaction is not interrupted by other write transactions. *Description*
If the **rollback** keyword is specified, the transaction is undone at the end of the processing. This means that you can test whether the statements can be correctly processed (technically speaking).

Output

This statement returns a confirmation of a successful operation *Output*

set parameter

Purpose

Purpose The purpose of the set parameter statement is to set the value of the specified parameters within the context of the requesting job, respectively the specified job.

Syntax

Syntax The syntax for the set parameter statement is

set parameter *parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,** *parametername* **=** *string*} **identified by** *string*

Description

Description The set parameter statements can be used to set jobs or user parameter values in the context of the job.
If the **identified by** option is specified, the parameter is only set if the pair *jobid* and *string* would allow a logon.

Output

Output This statement returns a confirmation of a successful operation

set state

Purpose

The purpose of the set state statement is to set the exit state of a job in a pending exit state. *Purpose*

Syntax

The syntax for the set state statement is *Syntax*

set state = *statename*

Description

The set state statement is used to set the Exit State of a job to a Pending Exit State. *Description*

Output

This statement returns a confirmation of a successful operation *Output*

submit

Purpose

Purpose The purpose of the submit statement is to execute a master batch or job as well as all defined children.

Syntax

Syntax The syntax for the submit statement is

submit *folderpath* [**with** WITHITEM {**,** WITHITEM}]

submit *aliasname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
check only
| **childtag =** *string*
| **nicevalue =** *signed_integer*
| **parameter = none**
| **parameter = (** PARAM {**,** PARAM} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **submittag =** *string*
| < **nosuspend** | **suspend** >
| **unresolved =** < **error** | **ignore** | **suspend** | **defer** >
| **group =** *groupname*

PARAM:
parametername **=** < *string* | *number* >

Description

Description The submit statement is used to submit a job or batch. There are two kinds of submit command:

- The first kind is used by users, who can also be programs, and the Time Scheduling Module. This form submits Master Jobs and Batches.
- The second form of the statement is used by jobs to submit dynamic children.

check only The check only option is used to verify whether a Master Submittable Batch or Job can be submitted. This means that a check is run to ascertain whether all the dependencies can be fulfilled and all the referenced parameters are defined. Whether the jobs can be executed in any scope or not is not verified. This is a situation that can arise at any point during the runtime.
Positive feedback means that, from the system's perspective, the job or batch can be submitted.

childtag The childtag option is used by jobs to submit several instances of the same Scheduling Entity and to be able to differentiate between them.
An error is triggered if the same Scheduling Entity is submitted twice using the same childtag. The content of the childtag has no further significance for the Scheduling System.
The maximum length for a childtag is 70 characters. The childtag option is ignored in the case of a Master Submit.

group The group option is used to set the owner group to the specified value. The user must belong to this group unless he belongs to the ADMIN privileged group. In this case, any group can be specified.

nicevalue The nicevalue option defines a correction that is used for the calculation of the priorities for the job and its children. Values between -100 and 100 are permitted.

parameter The parameter option is used to specify the value of Job Parameters for the submit. The parameters are set in the scope of the Master Batch or Job. This means that if parameters are specified that are not defined in the Master Batch or Job, these parameters are invisible to any children.

submittag If the submittag is specified, it must have a unique name for the Submitted Entity. This tag was introduced to be able to programmatically submit jobs and batches and to resubmit the job or batch with the same tag following a crash of one of the components. If the job submit was successful the first time, the second submit will report an error. If not, the second submit will succeed.

unresolved The unresolved option defines how the server is to react to unresolved dependencies. This option is mainly used if parts of a batch are submitted following repair work. The faulty part is normally cancelled and then resubmitted as a Master Run. In this case the previous dependencies have to be ignored otherwise the submit will fail.

suspend The suspend option is used to submit jobs or batches and to suspend them at the same time. If nothing is defined, they are not suspended. This can be explicitly specified at the submit time.
If a job or batch was suspended, neither it nor its children are started. If a job is already running, it will not reach a Final State if it is suspended.

resume The resume option can be used together with the suspend option to cause a delayed execution. There are two ways to do this. A delay can be achieved by specifying either the number of time units for the delay the time when the job or batch is to be activated.
This option can be used to reproduce the `at` functionality without creating a schedule.

Output

Output This statement returns an output structure of type record.

Main output description The data items of the output are described in the table below.

Field	Description
ID	ID of the Submitted Entity.

Table 28.3.: Description of the output structure of the submit statement