independIT Integrative Technologies GmbH
Bergstraße 6
D–86529 Schrobenhausen

schedulix Server

Command Reference
Release 2.10

Dieter Stubler Ronald Jeninga

12. Mai 2022

Inhaltsverzeichnis

Tabellenverzeichnis

Teil I

Allgemein

Kapitel 1

Einleitung

Einleitung

Dieses Dokument ist im Wesentlichen in drei Teile gegliedert. Im BICsuite Scheduling System gibt es drei Arten von Benutzern (im weitesten Sinne des Wortes):

- Users
- Jobservers
- Jobs

Jeder dieser Benutzer hat einen eigenen Befehlssatz zur Verfügung. Diese Befehlssätze sind jeweils nur teilweise überlappend. Es gibt z.B. für Jobserver das **get next job** Statement, das weder für Jobs noch für User gültig ist. Dafür gibt es Formen des **submit** Statements die nur in einem Job-Kontext einen Sinn ergeben und daher auch nur von Jobs ausgeführt werden dürfen. Das Anlegen von Objekten wie Exit State Definitions oder Job Definitions ist natürlich nur den Usern vorbehalten. Im Gegensatz dazu gibt es auch Statements, wie z.B. das **connect** Statement, die für alle Arten von Benutzern gültig sind.
Die Gliederung dieses Dokumentes richtet sich nach den drei Arten von Benutzern. Der größte Teil des Dokuments befasst sich mit den User Commands, die beiden anderen Teile mit den Jobservern und Job Commands.
Zur Vollständigkeit wird im nächsten Kapitel noch kurz auf das Utility *sdmsh* eingegangen. Dieses Utility ist einfach zu handhaben und stellt eine exzellente Wahl für die Verarbeitung von Skripten mit BICsuite-Kommandos dar.
Da die im Folgenden beschriebenen Syntax die einzige Schnittstelle zum BICsuite Scheduling Server ist, benutzen alle Dienstprogramme, also insbesondere auch BICsuite!Web diese Schnittstelle.
Um die Entwicklung eigener Utilities zu vereinfachen kann der Server seine Reaktionen auf Statements in verschiedenen Formaten ausgeben. Das Utility *sdmsh* benutzt zum Beispiel das **serial** Protokoll in dem serialized Java Objects übermittelt werden. Dagegen benutzt BICsuite!Web das **python** Protokoll, in dem textuelle

Darstellungen von Python-Strukturen übermittelt werden, die mittels der `eval`()-Funktion leicht eingelesen werden können.

Syntax Diagramme

Syntax Diagramme

Die Syntax Diagramme sind aus verschiedenen Symbolen und Metasymbolen aufgebaut. Die Symbole und Metasymbole sind in der nachfolgenden Tabelle aufgeführt und erläutert.

Symbol	Bedeutung
keyword	Ein Schlüsselwort der Sprache. Diese müssen wie dargestellt eingegeben werden. Ein Beispiel ist das Schlüsselwort **create**.
name	Ein Parameter. Hier kann in vielen Fällen ein selbst gewählter Name oder eine Zahl eingesetzt werden.
NONTERM	Ein nicht terminales Symbol wird mittels SMALL CAPS dargestellt. An dieser Stelle muss ein im späteren Verlauf des Diagrammes näher erläutertes Syntaxelement eingesetzt werden.
< **all** | **any** >	Dieses Syntaxelement stellt eine Wahlmöglichkeit dar. Eines der in den spitzen Klammern angegebene Syntaxelemente muss gewählt werden, das können natürlich auch nonterminale Symbole sein. Im einfachsten Fall gibt es nur zwei Wahlmöglichkeiten, häufig aber auch mehr.
< **all** | **any** >	Auch in diesem Fall handelt es sich um eine Wahlmöglichkeit. Im Gegensatz zum vorherigen Syntaxelement wird durch das Unterstreichen des ersten Elementes hervorgehoben, dass diese Wahlmöglichkeit der Default ist.
[**or alter**]	Optionale Syntaxelemente werden in eckigen Klammern gestellt.
{ *statename* }	Syntaxelemente, die in geschweiften Klammern gestellt sind, werden 0 bis n mal wiederholt.
JOB_PARAMETER {**,** JOB_PARAMETER}	Der Fall, dass Elemente mindestens einmal vorkommen, ist weitaus häufiger und wird wie gezeigt dargestellt.
|	In Listen von möglichen Syntaxelementen werden die einzelne Möglickeiten durch einen | voneinander getrennt. So eine Liste ist eine andere Darstellung einer Wahlmöglichkeit. Beide verschiedene Darstellungsformen dienen jedoch der Übersichtlichkeit.

Literale

Literale

In der Sprachdefinition werden nur für Zeichenketten, Zahlen und Datum/Uhrzeitangaben Literale benötigt.
Zeichenketten werden durch einfache Hochkommas abgegrenzt wie in

node = 'puma.independit.de'

Ganze Zahlen werden entweder als vorzeichenlose *integer* oder vorzeichenbehaftete *signed_integer* in den Syntaxdiagrammen dargestellt. Einem *signed_integer* darf ein Vorzeichen (**+** oder **-**) vorangestellt werden. Gültige vorzeichenlose Integers liegen im Zahlenbereich zwischen 0 und $2^{31}-1$. Integers mit Vorzeichen liegen damit im Zahlenbereich zwischen $-2^{31}+1$ und $2^{31}-1$. Wenn in den Syntaxdiagrammen die Rede von *id* ist, wird hier eine vorzeichenlose ganze Zahl zwischen 0 und $2^{63}-1$ erwartet.
Weitaus am schwierigsten sind die Datum/Uhrzeitangaben, insbesondere in den Statements die mit dem Time-Scheduling zu tun haben. Grundsätzlich werden diese Literale als Zeichenketten mit einem speziellen Format dargestellt.
Zur Vereinbarung der an ISO 8601 angelehnten Notationen in Tabelle 1.1 wird folgende Schreibweise verwendet:

Zeichen	*Bedeutung*	*zul.Bereich*	*Zeichen*	*Bedeutung*	*zul.Bereich*
YYYY	Jahr	1970 .. 9999	hh	Stunde	00 .. 23
MM	Monat	01 .. 12	mm	Minute	00 .. 59
DD	Tag (des Monats)	01 .. 31	ss	Sekunde	00 .. 59
ww	Woche (des Jahres)	01 .. 53			

- Alle anderen Zeichen stehen für sich selbst.
- Es erfolgt keine Unterscheidung von Groß- und Kleinschreibung.
- Der früheste zulässige *Zeitpunkt* ist 1970–01–01T00:00:00 GMT.

Format	**Beispiel**	**Vereinfachtes 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

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite		
Format	**Beispiel**	**Vereinfachtes Format**
−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	

Tabelle 1.1: Gültige Datumsformate

Identifier

Identifier

Innerhalb des BICsuite Scheduling Systems werden Objekte anhand ihres Namens identifiziert. (Strikt genommen können die Objekte auch über ihre interne Id, eine Nummer, identifiziert werden, aber diese Praxis wird nicht empfohlen). Gültige Namen bestehen aus einem Buchstaben, Underscore (_), At-Zeichen (@) oder Hash-Zeichen (#), gefolgt von Ziffern, Buchstaben oder genannte Sonderzeichen. Sprachabhängige Sonderzeichen wie z.B. deutsche Umlaute sind ungültig.
Soweit Identifier nicht in einfachen Quotes eingeschlossen werden, werden sie ohne Berücksichtigung der Groß- oder Kleinschreibung behandelt. Werden sie jedoch in Quotes eingeschlossen, werden sie case sensitive behandelt. Es wird daher generell nicht empfohlen Quotes zu benutzen, es sei denn es liegt ein triftiger Grund vor.
Identifier, die in einfachen Quotes eingeschlossen werden, dürfen auch Leerzeichen enthalten. Auch hier gilt, dass diese Praxis nicht empfohlen wird, da Leerzeichen normalerweise als Trennzeichen aufgefasst werden und daher leicht Fehler entstehen können. Insbesondere Leerzeichen am Ende des Namens führen leicht zu schwer auffindbaren Fehlern.
Es gibt eine Anzahl Schlüsselworte in der Syntax die nicht ohne Weiteres als Identifier benutzt werden können. Hier kann das Benutzen von Quotes sinnvoll sein, denn dadurch werden die Identifier nicht als Schlüsselworte erkannt. Die Tabelle 1.2 gibt eine Liste solcher Schlüsselworte.

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	

Tabelle 1.2: Keywords die mit Quotes als Identifier verwendet werden dürfen

Des Weiteren gibt es eine Anzahl Synonyme. Das sind im Wesentlichen Schlüsselworte die mehrere Schreibweisen erlauben. In Tabelle 1.2 wird nur eine dieser Schreibweisen gezeigt. Die Synonyme dürfen beliebig durcheinander benutzt werden. In Tabelle 1.3 gibt es eine Liste solcher Synonyme.

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

Tabelle 1.3: Keywords und Synonyme

Wie in jeder Sprache gibt es auch reservierte Worte, bzw. Wortkombinationen. Eine Übersicht wird in Tabelle 1.4 gezeigt. Bei den Wortpaaren gilt als Besonderheit, dass ein Ersetzen des Leerzeichens durch einen Underscore ebenfalls ein reserviertes Wort ergibt. Das Wort **named_resource** ist damit auch reserviert. (”named#resource” jedoch nicht).

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
Fortsetzung auf der nächsten Seite		

Fortsetzung der vorherigen Seite		
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	

Tabelle 1.4: Reservierte Worte

Editionen

Editionen

Es gibt drei Editionen des BICsuite Scheduling Systems. Da Features der höheren Editionen nicht immer in den niedrigeren Editionen vorhanden sind, werden die dazu gehörigen Statements, bzw. die entsprechenden Optionen innerhalb der Statements entsprechend gekennzeichnet. Oben an der äußeren Ecke der Seite wird mittels eines Buchstaben angegeben in welcher Edition des Systems dieses Statement zur Verfügung steht. Abweichungen vom allgemeinen Statement werden im Syntaxdiagramm dargestellt.

Die Symbole haben folgende Bedeutung:

Symbol	Bedeutung
B	Dieses Symbol kennzeichnet ein Feature der Basic und alle höheren Editionen.
P	Dieses Symbol kennzeichnet ein Feature der Professional und Enterprise Edition.
E	Dieses Symbol kennzeichnet ein Feature der Enterprise Edition.

Kapitel 2

Utilities

Starten und Stoppen des Servers

server-start

Einleitung

Das Utility *server-start* wird benutzt um den Scheduling Server zu starten. *Einleitung*

Aufruf

Der Aufruf von *server-start* sieht folgendermaßen aus: *Aufruf*

server-start [OPTIONS] *config-file*

OPTIONS:
-admin
| **-protected**

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
-admin	Der Server startet im "**admin**"-Modus. Das bedeutet, dass User Logins bis auf den Benutzer **SYSTEM** disabled sind.
-protected	Der "**protected**"-Modus ist vergleichbar mit dem admin-Modus. Der Unterschied ist, dass auch die internen Threads (TimerThread und SchedulingThread) nicht gestartet werden. Damit können administrative Aufgaben erledigt werden, ohne dass nebenläufige Transaktionen durchgeführt werden.

Wenn der Server bereits gestartet ist, wird, je nach Konfiguration, der zweite Server entweder den Betrieb übernehmen, oder aber immer wieder erfolglos versuchen zu starten.
Das Utility *server-start* kann nur von dem Benutzer, unter dessen Kennung das System installiert wurde, benutzt werden.

server-stop

Einleitung

Einleitung Das Utility *server-stop* wird benutzt um den Scheduling Server zu stoppen.

Aufruf

Aufruf Der Aufruf von *server-stop* sieht folgendermaßen aus:

server-stop

Zuerst wird versucht den Server sanft anzuhalten. Dabei werden zuerst alle User Connections beendet um anschließend alle internal Threads zu beenden.
Ist dieser Ansatz nicht erfolgreich, bzw. dauert es zu lange, wird der Server mit Betriebssystemmitteln beendet.
Wenn der Server nicht gestartet ist, hat die Benutzung des *server-stop* Befehls keine Auswirkung.
Das Utility *server-stop* kann nur von dem Benutzer, unter dessen Kennung das System installiert wurde, benutzt werden.

sdmsh

Einleitung

Einleitung

Das Utility *sdmsh* ist ein kleines Programm, welches ein interaktives Arbeiten mit dem Scheduling Server ermöglicht. Im Gegensatz zu etwa dem BICsuite!Web Frontend ist dieses Arbeiten textorientiert. Es ist damit möglich Skripte zu schreiben und diese über *sdmsh* ausführen zu lassen.

Das Executable *sdmsh* ist ein kleines Skript (oder eine Batch-Datei), das den Aufruf des benötigten Java-Programmes kapselt. Es spricht natürlich nichts dagegen dieses Java-Programm manuell aufzurufen.Das Skript dient nur dem Komfort.

Aufruf

Aufruf

Der Aufruf von *sdmsh* sieht folgendermaßen aus:

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*

Die einzelne Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	BICsuite Server Host
< **--port** | **-p** > *portnumber*	BICsuite Server Port
< **--user** | **-u** > *username*	Username (user oder jid muss spezifiziert werden)
< **--pass** | **-w** > *password*	Passwort (wird in Kombination mit der **--user** Option verwendet)
< **--jid** | **-j** > *jobid*	Job Id (user oder jid muss spezifiziert werden)
< **--key** | **-k** > *jobkey*	Job Key (wird in Kombination mit der **--jid** Option verwendet)
< **--**[**no**]**silent** | **-**[**no**]**s** >	[Keine] (error) Meldungen werden nicht ausgegeben
< **--**[**no**]**verbose** | **-**[**no**]**v** >	[Keine] Kommandos, Feedbacks und zusätzliche Meldungen werden ausgegeben
< **--ini** | **-ini** > *inifile*	Benutze die genannte Konfigurationsdatei zum Setzen von Optionen.
< **--**[**no**]**tls** | **-**[**no**]**tls** >	Benutze den Zugang über TLS/SSL [nicht].
--[**no**]**help**	Gebe einen Hilfetext aus.
--info *sessioninfo*	Setze die mitgegebene Information als beschreibende Information der Session.

Option	Bedeutung
-[**no**]**S**	Silent Option. Die Option ist obsolet und aus Gründen der Rückwärtskompatibilität vorhanden.
--timeout *timeout*	Die Anzahl Sekunden nach welchen der Server eine idle-Session terminiert. Der Wert 0 bedeutet kein timeout.

Selbstverständlich benötigt *sdmsh* Informationen um sich mit dem richtigen BICsuite Scheduling System zu verbinden. Dazu können die benötigten Daten auf der Kommandozeile oder mittels einer Options-Datei spezifiziert werden. Fehlende Werte für Username und Passwort werden von *sdmsh* erfragt. Falls Werte für den Host und Port fehlen, werden die Defaults "localhost" und 2506 benutzt. Es wird nicht empfohlen das Passwort auf der Kommandozeile zu spezifizieren, da diese Information vielfach sehr einfach von anderen Benutzern gelesen werden kann.

Options-Datei

Die *Options-Datei* hat das Format eines Java-Propertyfiles. (Für die genaue Syntaxspezifikation verweisen wir auf die offizielle Java-Dokumentation.) *Options-Datei*
Es spielen folgende Options-Dateien eine Rolle:

- `$SDMSCONFIG/sdmshrc`
- `$HOME/.sdmshrc`
- Optional eine auf der Befehlszeile spezifizierte Datei

Die Dateien werden in der angegebenen Reihenfolge ausgewertet. Falls Optionen in mehreren Dateien vorkommen "gewinnt" der Wert in der zuletzt ausgewerteten Datei. Options, die auf der Befehlszeile spezifiziert werden, haben Vorrang vor allen anderen Spezifikationen.
Folgende Schlüsselworte werden erkannt:

Keyword	Bedeutung
User	Name des Benutzers
Password	Passwort des Benutzers
Host	Name oder IP-Adresse des Hosts
Info	Zusätzliche Information zur Identifikation einer Connection wird gesetzt
Port	Portnumber des Scheduling Servers (Default: 2506)
Silent	(Fehler)Meldungen werden nicht ausgegeben
Timeout	Timeout-Wert für die Session (0 ist kein Timeout)
TLS	Benutze eine SSL/TLS Connection.
Verbose	Kommandos, Feedbacks und weitere Meldungen werden ausgegeben.

Da das Passwort des Benutzers in Klartext in dieser Datei steht, muss sorgfaltig mit den Zugriffsrechten für diese Datei umgegangen werden. Es ist natürlich möglich das Passwort nicht zu spezifizieren und dieses bei jedem Start von *sdmsh* einzugeben.
Die folgenden Schlüsselworte sind ausschließlich in Konfigurationsdateien erlaubt:

Keyword	Bedeutung
KeyStore	Keystore für TLS/SSL Kommunication
TrustStore	Truststore für TLS/SSL Kommunication
KeyStorePassword	Keystore Passwort
TrustStorePassword	Truststore Passwort

Interne Befehle

Interne Befehle

Abgesehen von den in den nachfolgenden Kapiteln beschriebenen BICsuite-Befehlen, kennt *sdmsh* noch einige einfache eigene Befehle. Diese werden im Folgenden kurz beschrieben. (Interne Befehle müssen nicht mit einem Semikolon abgeschlossen werden.)

disconnect Mit dem *disconnect* Befehl wird *sdmsh* verlassen. Da in den verschiedenen Arbeitsumgebungen verschiedene Befehle für das Verlassen eines Tools gebräuchlich sind, wurden hier viele Alternativen erlaubt.
Die Syntax des *disconnect* Befehls ist:

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

BEISPIEL Hier folgt ein Beispiel für den *disconnect* Befehl:

```
ronald@jaguarundi:~$ sdmsh

Connect

CONNECT_TIME : 23 Aug 2007 07:13:30 GMT

Connected

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

echo Im Falle einer interaktiven Benutzung von *sdmsh* ist sichtbar welcher Befehl gerade eingegeben wurde. Dies ist im Batch-Betrieb, d.h. beim Verarbeiten eines Skriptes, nicht der Fall. Mit dem *echo* Befehl kann das Wiedergeben des eingegebenen Statements ein- und ausgeschaltet werden. Per Default ist es eingeschaltet.
Die Syntax des *echo* Befehls ist:

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

BEISPIEL Untenstehend wird der Effekt von beiden Möglichkeiten gezeigt. Nach dem Befehl **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 Der *help* Befehl gibt eine kurze Hilfe zu den *sdmsh* internen Befehlen. Die Syntax des *help* Befehls ist:

help

BEISPIEL Der *help* Befehl gibt nur eine kurze Hilfe zur Syntax der internen *sdmsh* Befehle aus. Dies ist im untenstehenden Beispiel ersichtlich. (Die Zeilen wurden für dieses Dokument umgebrochen, der tatsächliche Output kann daher abweichen).

```
[system@localhost:2506] SDMS> help
Condensed Help Feature
----------------------
```

```
Internal sdmsh Commands:
disconnect|bye|exit|quit          -- leaves the tool
echo on|off                       -- controls whether the statementtext 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 Mit dem *include* Befehl können Dateien mit BICsuite Statements eingebunden werden.
Die Syntax des *include* Befehls ist:

include '*filespec*'

BEISPIEL Im folgenden Beispiel wird eine Datei, die nur den Befehl "**show session;**" enthält, eingefügt.

```
[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 Mit dem *prompt* Befehl kann ein beliebiger Prompt spezifiziert werden. Dabei gibt es eine Anzahl von variablen Werten, die vom Programm automatisch eingefügt werden können.
In nachfolgender Tabelle stehen die Codes für die einzelnen Variablen.

Code	Bedeutung
%H	Hostname des Scheduling Servers
%P	TCP/IP Port
%U	Username
%%	Prozentzeichen (%)

Der Default *prompt* hat folgende Definition: `[%U@%H:%P] SDMS>`.
Die Syntax des *prompt* Befehls ist:

prompt '*somestring*'

BEISPIEL Im folgenden Beispiel wird zuerst ein leerer Prompt definiert. Daraufhin wird, um den Effekt deutlicher sichtbar zu machen, ein BICsuite Statement ausgeführt. Anschließend wird eine einfache Zeichenkette als Prompt gewählt und zum Schluss werden die Variablen benutzt.

```
[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 Mit dem *timing* Befehl bekommt man Information bezüglich der Ausführungszeit eines Statements. Normalerweise ist diese Option ausgeschaltet und es wird keine Angabe der Ausführungszeit gemacht. Die Angaben erfolgen in Millisekunden.
Die Syntax des *timing* Befehls ist:

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

BEISPIEL Im folgenden Beispiel wird timing Information für ein einfaches BICsuite Statement gezeigt. Sichtbar wird die Ausführungszeit des Statements, sowie die Zeit die zum Ausgeben des Resultats benötigt war.

```
[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 Insbesondere wenn *sdmsh* zur Ausführung von Skripten benutzt wird, ist eine Möglichkeit zur Fehlerbehandlung unerlässlich. Mit dem *whenever* Statement wird *sdmsh* gesagt wie es mit Fehlern umgehen soll. Default-mäßig werden Fehler ignoriert, was für ein interaktives Arbeiten auch dem gewünschten Verhalten entspricht.
Die Syntax des *whenever* Statements ist:

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

BEISPIEL Das untenstehende Beispiel zeigt sowohl das Verhalten von der **continue** Option, als auch das Verhalten der **disconnect** Option. Der Exit Code eines Prozesses der von der Bourne-Shell gestartet wurde (und auch andere Unix-Shells) kann durch Ausgabe der Variablen `$?` sichtbar gemacht werden.

```
[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-Aufruf Es kommt häufig vor, dass schnell ein Shell-Kommando abgesetzt werden muss, etwa um zu sehen wie die Datei, die man (mittels **include**) ausführen möchte, auch heißt. Soweit keine besonderen Fähigkeiten vom Terminal verlangt werden, wie das z.B. bei einem Aufruf eines Editors der Fall wäre, kann ein Shell-Kommando durch das Voranstellen eines Ausrufezeichens ausgeführt werden.
Die Syntax eines *Shell-Aufrufes* ist:

!*shellcommand*

BEISPIEL Im folgenden Beispiel wird eine Liste aller *sdmsh* Skripte im `/tmp` Verzeichnis ausgegeben.

```
[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

Einleitung

Einleitung

Das Utility *sdms-auto_restart* dient dazu Jobs, die fehlgeschlagen sind, automatisch zu restarten. Dazu müssen eine Anzahl einfacher Voraussetzungen erfüllt sein. Die wohl wichtigste Voraussetzung ist, dass der Job einen Parameter AUTORESTART mit dem Wert TRUE definiert. Natürlich kann dieser Parameter auch auf höherer Ebene gesetzt sein.
Folgende Parameter haben einen Einfluss auf das Verhalten des AUTORESTART Utilities:

Parameter	Wirkung
AUTORESTART	Der Autorestart funktioniert nur wenn dieser Parameter den Wert "TRUE" hat.
AUTORESTART_MAX	Definiert, wenn gesetzt, die maximale Anzahl automatischen Restarts
AUTORESTART_COUNT	Wird vom Autorestart Utility gesetzt um die Anzahl Restarts zu zählen
AUTORESTART_DELAY	Die Zeit in Minuten bevor ein Job erneut gestartet wird

Das AUTORESTART Utility kann als Trigger definiert werden. Hierfür bieten sich die Triggertypen IMMEDIATE_LOCAL sowie FINISH_CHILD an.
Die Logik der Options-Datei, die für das Utility *sdmsh* gilt, findet auch für *sdms-auto_restart* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-auto_restart* sieht folgendermaßen aus:

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** >

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--failed** | **-f** > *jobid*	Job Id des zu restartenden Jobs
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--delay** | **-d** > *minutes*	Die Anzahl Minuten die gewartet wird, bis der Job erneut gestartet wird
< **--max** | **-m** > *number*	Die maximum Anzahl automatischer Restarts
< **--warn** | **-W** >	Das Warning Flag wird gesetzt, wenn die maximum Anzahl Restarts erreicht wurde.

sdms-get_variable

Einleitung

Einleitung

Das Utility *sdms-get_variable* bietet eine einfache Möglichkeit Job Parameter aus dem Scheduling System zu lesen.
Die Logik der Options-Dateien, die für das Utility *sdmsh* gilt, findet auch für *sdms-get_variable* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-get_variable* sieht folgendermaßen aus:

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*

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--key** | **-k** > *jobkey*	Passwort für die Anmeldung (für eine Connection als Job)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Option	Bedeutung
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--mode** | **-m** > *mode*	Modus für die Parameterermittlung (liberal, warn, strict)

Beispiel

Beispiel Das nachfolgende Beispiel zeigt wie der Inhalt der variablen ANTWORT von Job 5175119 ermittelt werden kann.

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

sdms-rerun

Einleitung

Einleitung

Das Utility *sdms-rerun* wird genutzt um aus einem Skript oder Programm heraus einen Job in restartable State erneut zu starten.
Die Logik der Options-Dateien die für das Utility *sdmsh* gilt, findet auch für *sdms-rerun* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-rerun* sieht folgendermaßen aus:

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*

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Option	Bedeutung
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--suspend** | **-S** >	Der Job wird suspended
< **--delay** | **-D** > *delay*	Nach delay Units wird der Job automatisch resumed
< **--unit** | **-U** > *unit*	Einheit für die delay-Option (default MINUTE)
< **--at** | **-A** > *at*	Automatischer Resume zum angegebenen Zeitpunkt

sdms-set_state

Einleitung

Einleitung

Das Utility *sdms-set_state* bietet eine einfache Möglichkeit den Status eines Jobs im Scheduling System zu setzen.
Die Logik der Options-Dateien, die für das Utility *sdmsh* gilt, findet auch für *sdms-set_state* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-set_state* sieht folgendermaßen aus:

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

OPTIONS:
< **--user** | **-u** > *username*
< **--pass** | **-w** > *password*
< **--key** | **-k** > *jobkey*
< **--silent** | **-s** >
< **--verbose** | **-v** >
< **--timeout** | **-t** > *minutes*
< **--cycle** | **-c** > *minutes*
< **--help** | **-h** >
< **--case** | **-C** >
< **--**[**no**]**force** | **-**[**no**]**f** >

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--key** | **-k** > *jobkey*	Passwort für die Anmeldung (für eine Connection als Job)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Option	Bedeutung
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten, die versucht wird eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten, die zwischen zwei Versuchen eine Serververbindung aufzubauen, gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--case** | **-C** >	Namen sind case sensitive
< **--state** | **/-S** > *state*	Der zu setzende Status
< **--force** | **-f** >	Erzwingt die Statusänderung, auch wenn kein Mapping für den Status existiert

sdms-set_variable

Einleitung

Einleitung

Das Utility *sdms-set_variable* bietet eine einfache Möglichkeit Job Parameter im Scheduling System zu setzen.
Die Logik der Options-Dateien die für das Utility *sdmsh* gilt, findet auch für *sdms-set_variable* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-set_variable* sieht folgendermaßen aus:

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

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

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--key** | **-k** > *jobkey*	Passwort für die Anmeldung (für eine Connection als Job)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Option	Bedeutung
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--case** | **-C** >	Namen sind case sensitive

sdms-set_warning

Einleitung

Einleitung

Das Utility *sdms-set_warning* wird benutzt um das Warning Flag eines Jobs zu setzen. Optional kann ein Text spezifiziert werden. Man kann als Benutzer für einen Job das Warning Flag setzen wenn man das Operate-Privileg hat. Ein Job kann das Warning Flag für sich selbst setzen.
Die Logik der Options-Dateien die für das Utility *sdmsh* gilt, findet auch für *sdms-set_warning* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-set_warning* sieht folgendermaßen aus:

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*

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	Username für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--key** | **-k** > *jobkey*	Passwort für die Anmeldung (für eine Connection als Job)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Option	Bedeutung
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--warning** | **-m** > *warning*	Text zur Warnung

sdms-submit

Einleitung

Einleitung

Das Utility *sdms-submit* wird benutzt um Jobs oder Batches zu starten. Diese können als eigenständiger Ablauf oder aber als Child eines bereits vorhandenen Jobs gestartet werden. Im letzteren Fall kann, wenn in der Parent-Child-Hierarchie definiert, ein Alias zur Identifizierung des zu submittenden Jobs oder Batches spezifiziert werden.
Die Logik der Options-Dateien die für das Utility *sdmsh* gilt, findet auch für *sdms-submit* Anwendung.

Aufruf

Aufruf

Der Aufruf von *sdms-submit* sieht folgendermaßen aus:

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*

Die einzelnen Options haben folgende Bedeutung:

Option	Bedeutung
< **--host** | **-h** > *hostname*	Hostname des Scheduling Servers
< **--port** | **-p** > *portnumber*	Port des Scheduling Servers
< **--user** | **-u** > *username*	User Name für die Anmeldung
< **--pass** | **-w** > *password*	Passwort für die Anmeldung (für eine Connection als User)
< **--key** | **-k** > *jobkey*	Passwort für die Anmeldung (für eine Connection als Job)
< **--silent** | **-s** >	Reduzierte Ausgabe von Meldungen
< **--verbose** | **-v** >	Erhöhte Anzahl Meldungen
< **--timeout** | **-t** > *minutes*	Anzahl Minuten die versucht wird um eine Verbindung zum Server zu bekommen
< **--cycle** | **-c** > *minutes*	Anzahl Minuten die zwischen zwei Versuchen eine Serververbindung aufzubauen gewartet wird
< **--help** | **-h** >	Gibt eine kurze Hilfe zum Aufruf des Utilities aus
< **--tag** | **-T** > *tag*	Tag für dynamic Submits
< **--master** | **-M** >	Submit eines Masters, kein Child
< **--suspend** | **-S** >	Der Job wird suspended
< **--delay** | **-D** > *delay*	Nach delay Units wird der Job automatisch resumed
< **--unit** | **-U** > *unit*	Einheit für die delay Option (default MINUTE)
< **--at** | **-A** > *at*	Automatischer Resume zum angegebenen Zeitpunkt

Teil II

User Commands

Kapitel 3

alter commands

alter comment

Zweck

Zweck Das *alter comment* Statement wird eingesetzt um den Kommentar zu einem Objekt zu ändern.

Syntax

Syntax Die Syntax des *alter comment* Statements ist

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*
| **exit state translation** *transname*
| **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* | **jobserver** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

CC_WITHITEM:
    CC_TEXTITEM {**,** CC_TEXTITEM}
  | **url =** *string*

PARAM_LOC:
    folder *folderpath*
  | **job definition** *folderpath*
  | **named resource** *resourcepath*
  | < **scope** *serverpath* | **jobserver** *serverpath* >

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

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

Beschreibung

Beschreibung

Der *alter comment* Befehl wird verwendet um die Kurzbeschreibung, bwz. die URL der Beschreibung vom beschriebenen Objekt zu ändern. Natürlich kann der Typ der Information ebenso verändert werden. Der Kommentar ist versioniert. Das bedeutet, dass Kommentare nicht überschrieben werden. Wenn das kommentierte Objekt angezeigt wird, ist der angezeigte Kommentar der Kommentar, der zur Version des angezeigten Objektes passt.
Das optionale Schlüsselwort **existing** wird verwendet um das Auftreten der Fehlermeldungen und das Abbrechen der aktuellen Durchführung zu verhindern. Das ist im Zusammenhang mit *multicommands* besonders nützlich.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter environment

Zweck

Zweck Das *alter environment* Statement wird eingesetzt um die Eigenschaften des spezifizierten Environments zu ändern.

Syntax

Syntax Die Syntax des *alter environment* Statements ist

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** >]

Beschreibung

Beschreibung Das *alter environment* Statement wird benutzt um die Resource-Anforderungen, die in diesem Envrionment definiert sind, zu ändern. Laufende Jobs sind nicht davon betroffen.
Die "**with resource =**" Form des Statements ersetzt die aktuelle Gruppe von Resource-Anforderungen. Die anderen Arten fügen die spezifizierten Anforderungen zu oder löschen sie. Es wird als Fehler angesehen eine Anforderung zu löschen welche kein Teil des Environments ist oder eine Anforderung für eine bereits benötigte Resource zuzufügen.
Nur Administratoren sind befugt diese Handlung durchzuführen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter event

Zweck

Das *alter event* Statement wird eingesetzt um Eigenschaften des spezifizierten Events zu ändern. *Zweck*

Syntax

Die Syntax des *alter event* Statements ist *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* >

Beschreibung

Das *alter event* Statement wird benutzt um die Eigenschaften eines Events zu ändern. Mit der **with parameter** Klausel kann ein Parameter für den Submit eines Jobs spezifiziert werden. (Für eine ausführliche Beschreibung der Optionen, siehe das *create event* Statement auf Seite 109.) *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter exit state mapping

Zweck

Zweck Das *alter exist state mapping* Statement wird eingesetzt um Eigenschaften des spezifizierten Mappings zu ändern.

Syntax

Syntax Die Syntax des *alter exit state mapping* Statements ist

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

Beschreibung

Beschreibung Das *alter exit state mapping* Statement definiert das Mapping der Exit Codes zu logischen Exit States. Die einfachste Form des Statements spezifiziert nur einen Exit State. Das bedeutet, dass der Job, ohne Rücksicht auf seinen Exit Code zu nehmen, diesen Exit State bei Beendigung bekommt. Komplexere Definitionen spezifizieren mehr als ein Exit State und mindestens eine Abgrenzung.
Ein Statement wie

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

definiert das folgende Mapping:

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

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter exit state profile

Zweck

Zweck

Das *alter exit state profile* Statement wird eingesetzt um Eigenschaften des spezifizierten Profiles zu ändern.

Syntax

Syntax

Die Syntax des *alter exit state profile* Statements ist

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:
batch default
| **broken**
| **dependency default**
| **disable**
| **unreachable**

Beschreibung

Beschreibung

Das *alter exit state profile* Statement wird benutzt um Exit States am Profile zuzufügen oder zu löschen, sowie das Default Exit State Mapping zu definieren. (Für eine ausführliche Beschreibung der Optionen siehe das *create exit state profile* Statement auf Seite 112.)

force Die **force** Option kennzeichnet die Exit State Profiles als invalid. Das bedeutet nur, dass die Integrität noch geprüft werden muss. Nach einer erfolgreichen Überprüfung wird die Kennzeichnung gelöscht. Die Überprüfung wird beim Submitten einer Job Definition, welche die Exit State Profiles verwendet, durchgeführt. Das Ziel vom **force** Flag ist es imstande zu sein mehrere Exit State Profiles, und vielleicht andere Objekte, zu ändern, ohne die Notwendigkeit eines konsistenten Zustands nach jeder Änderung.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter exit state translation

Zweck

Zweck

Das *alter exit state translation* Statement wird eingesetzt um Eigenschaften der spezifizierten Exit State Translation zu ändern.

Syntax

Syntax

Die Syntax des *alter exit state translation* Statements ist

alter [**existing**] **exit state translation** *transname*
with translation = (*statename* **to** *statename* {**,** *statename* **to** *statename*}
)

Beschreibung

Beschreibung

Das *alter exit state translation* Statement ändert eine vorher definierte Exit State Translation. Laufende Jobs sind davon nicht betroffen.
Wenn das optionale Schlüsselwort **existing** spezifiziert ist, wird kein Fehler erzeugt, wenn die spezifizierte Exit State Translation nicht gefunden wurde.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter folder

Zweck

Zweck Das *alter folder* Statement wird eingesetzt um die Eigenschaften eines Folders zu ändern.

Syntax

Syntax Die Syntax des *alter folder* Statements ist

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**

Beschreibung

Beschreibung Das *alter folder* Statement ändert die Eigenschaften eines Folders. (Für eine ausführliche Beschreibung der Optionen, siehe das *create folder* Statement auf Seite 116.)
Wenn das optionale Schlüsselwort **existing** spezifiziert ist, wird keine Fehlermeldung erzeugt wenn der spezifizierte Folder nicht existiert. Obwohl der Folder SYSTEM weder angelegt, noch gelöscht oder umbenannt werden kann, ist es in beschränkten Maßen möglich ihn zu ändern. Es ist nicht möglich die Eigentümer-

gruppe zu ändern, aber es ist möglich ein Environment zu spezifizieren oder Parameter anzulegen.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter footprint

Zweck

Zweck Das *alter footprint* Statement wird eingesetzt um die Eigenschaften des spezifizierten Footprints zu ändern.

Syntax

Syntax Die Syntax des *alter footprint* Statements ist

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*

Beschreibung

Beschreibung Das *alter footprint* Kommando ändert die Liste der Resource-Anforderungen. Es gibt drei Formen des Statements.

- Die erste Form bestimmt alle Resource-Anforderungen.
- Die zweite fügt Resource-Anforderungen zu der Anforderungsliste zu.
- Die dritte Form entfernt Anforderungen aus der Liste.

(Für eine ausführliche Beschreibung der Optionen, siehe das *create footprint* Statement auf Seite 118.)

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter group

Zweck

Das *alter group* Statement wird eingesetzt um die Zuordnung von Benutzern zu Gruppen zu ändern. *Zweck*

Syntax

Die Syntax des *alter group* Statements ist *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*} **)**

Beschreibung

Das *alter group* Kommando wird verwendet um festzulegen welche Benutzer zu der Gruppe gehören. Es gibt zwei Formen des Statements: *Beschreibung*

- Die erste legt die Liste der Benutzer die zu der Gruppe gehören fest.
- Die zweite fügt Benutzer in der Gruppe zu oder löscht sie.

In allen Fällen wird es als Fehler betrachtet, Benutzer aus ihrer Default-Gruppe zu löschen.
Es ist nicht möglich Benutzer aus der Gruppe PUBLIC zu löschen.
Wenn ein Benutzer nicht zu einer Gruppe gehört, wird der Versuch den Benutzer aus dieser Gruppe zu löschen ignoriert.
Ist das Schlüsselwort **existing** spezifiziert, wird es *nicht* als Fehler betrachtet wenn die Gruppe nicht existiert.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter interval

Zweck

Zweck Das *alter interval* Statement wird eingesetzt um Eigenschaften des spezifizierten Intervalls zu ändern.

Syntax

Syntax Die Syntax des *alter interval* Statements ist

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

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

IVAL_DISPATCHITEM:
dispatchname < **active** | **inactive** > IVAL_DISPATCHDEF

CINTERVALNAME:
(*intervalname*
with WITHITEM {**,** WITHITEM} **)**
| *intervalname*

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

IVAL_DISPATCHDEF:
none CINTERVALNAME < **enable** | **disable** >
| CINTERVALNAME CINTERVALNAME < **enable** | **disable** >
| CINTERVALNAME < **enable** | **disable** >

Beschreibung

Beschreibung

Das *alter interval* Kommando wird benutzt um eine Intervalldefinition zu ändern. (Für eine ausführliche Beschreibung der Optionen, siehe den *create interval* Statement auf Seite 121.
Ist das Schlüsselwort **existing** spezifiziert, wird es *nicht* als Fehler betrachtet, wenn der Intervall nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter job

Zweck

Zweck Das *alter job* Statement wird benutzt um Eigenschaften des spezifizierten Jobs zu ändern. Es wird von den Job-Administratoren, Jobservern und vom Job selbst benutzt.

Syntax

Syntax Die Syntax des *alter job* Statements ist

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

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

WITHITEM:
< **disable** | **enable** >
| < **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **clone resume**
| **clone suspend**
| **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 = (** *id* [**recursive**] {**,** *id* [**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**

Beschreibung

Beschreibung

Das *alter job* Kommando wird für mehrere Zwecke genutzt. Als erstes verwenden Jobserver dieses Kommando um den Ablauf eines Jobs zu dokumentieren. Alle Statuswechsel eines Jobs während der Zeit in der der Job innerhalb der Zuständigkeit eines Jobservers fällt, werden mittels des *alter job* Kommandos ausgeführt.
Zweitens werden einige Änderungen, wie z. B. das Ignorieren von Abhängigkeiten oder Resourcen, sowie das Ändern der Priorität eines Jobs, manuell von einem Administrator ausgeführt.
Der Exit State eines Jobs in einem pending State kann vom Job selbst gesetzt werden, bzw. von einem Prozess welcher die Job Id und den Key des zu ändernden Jobs kennt.

cancel Die cancel Option wird benutzt um den adressierten Job und alle nicht final Children zu canceln. Ein Job kann nur gecancelt werden wenn weder der Job selbst noch einer seiner Children aktiv ist.
Wenn ein Scheduling Entity von dem gecancelten Job abhängig ist, kann er unreachable werden. In diesem Fall erhält der abhängige Job nicht den im Exit State Profile definierten unreachable Exit State, sondern wird in den Job Status ”unreachable” versetzt. Es ist Aufgabe des Operators diese Jobs nun mittels des Ignorierens von Abhängigkeiten wieder in den Job Status ”dependency wait” zu versetzen, oder aber diese Jobs auch zu canceln.
Gecancelte Jobs werden wie final Jobs ohne Exit State betrachtet. Das bedeutet, die Parents eines gecancelten Jobs werden final, ohne den Exit State des gecancelten Jobs zu berücksichtigen. Die abhängigen Jobs der Parents laufen in diesem Fall normal weiter.
Die cancel Option kann nur von Benutzern genutzt werden.

clone Die **clone** option wird benutzt um bereits beendete Jobs noch einmal im selben Kontext aus zu führen. Dies kann in seltene Fällen notwendig sein. Wenn etwa in der Fehlerdiagnose eines Nachfolgers festgestellt wird, dass die Ursache einige Jobs zurück liegt, wird es notwendig sein, nach der Ursachenbeseitigung, die ganze Kette noch einmal aus zu führen.
Um zu gewährleisten, dass die Ausführung kontrolliert anfängt wird spezifiziert ob der Clone suspended werden soll, oder nicht.

comment Die comment Option wird benutzt um eine Aktion zu dokumentieren oder um dem Job einen Kommentar zuzufügen. Comments können maximal 1024 Zeichen lang sein. Es kann eine beliebige Anzahl Comments für einen Job gespeichert werden.
Einige Comments werden automatisch gespeichert. Wenn z. B. ein Job einen restartable State erreicht, wird ein Protokoll geschrieben, um diesen Fakt zu dokumentieren.

error text Die error text Option wird benutzt um Fehlerinformation zu einem Job zu schreiben. Dieses kann von dem verantwortlichen Jobserver oder einem Benutzer gemacht werden. Der Server kann diesen Text auch selbst schreiben.
Diese Option wird normalerweise benutzt, wenn der Jobserver den entsprechenden Prozess nicht starten kann. Mögliche Fälle sind die Unmöglichkeit zum definierten Working Directory zu wechseln, die Unauffindbarkeit des ausführbaren Programmes oder Fehler beim Öffnen des Error Logfiles.

exec pid Die exec pid Option wird ausschließlich vom Jobserver benutzt um die Prozess Id des Kontrollprozesses innerhalb des Servers zu setzen.

exit code Die exit code Option wird vom Jobserver benutzt um dem Repository Server mitzuteilen mit welchem Exit Code sich ein Prozess beendet hat. Der Repository Server berechnet jetzt den zugehörigen Exit State aus dem verwendeten Exit State Mapping.

exit state Die exit state Option wird von Jobs in einem pending State benutzt, um ihren State auf einen anderen Wert zu setzen. Dies wird normalerweise ein restartable oder final State sein. Alternativ dazu kann diese Option von Administratoren benutzt werden, um den State von einem nonfinal Job zu setzen. Sofern das Force Flag nicht benutzt wird, sind die einzigen States die gesetzt werden können, die States, welche, durch die Anwendung des Exit State Mappings auf irgendeinem Exit Code, theoretisch erreichbar sind. Der gesetzte State muss im Exit State Profile vorhanden sein.

ext pid Die ext pid Option wird ausschließlich vom Jobserver genutzt, um die Prozess Id des gestarteten Benutzerprozesses zu setzen.

ignore resource Die ignore resource Option wird benutzt um einzelne Resource Requests aufzuheben. Die ignorierte Resource wird nicht mehr beantragt.
Wenn Parameter einer Resource referenziert werden, kann diese Resource nicht ignoriert werden.
Wenn ungültige Id's spezifiziert wurden, wird dies übergangen. Alle anderen spezifizierten Resources werden ignoriert. Ungültige Id's in diesem Kontext sind Id's von Resources die von dem Job nicht beantragt werden.
Das Ignorieren von Resources wird protokolliert.

ignore dependency Die ignore dependency Option wird benutzt um definierte Dependencies zu ignorieren. Wenn das **recursive** Flag benutzt wird, ignorieren nicht nur der Job oder Batch selbst, sondern auch seine Children die Dependencies.

kill Die kill Option wird benutzt um den definierten Kill Job zu submitten. Wenn kein Kill Job definiert ist, ist es nicht möglich den Job vom BICsuite aus erzwungenermaßen zu terminieren. Natürlich muss der Job aktiv sein, das bedeutet, der Job State muss **running**, **killed** oder **broken_active** sein. Die letzten beiden States sind keine regulären Fälle.
Wenn ein Kill Job submitted wurde, ist der Job State **to_kill**. Nachdem der Kill Job beendet wurde, wird der Job State des killed Jobs in den State **killed** gesetzt, es sei denn er ist beendet, dann wird der Job State **finished** oder **final** sein. Das bedeutet, dass der Job mit dem Job State **killed** immer noch running ist und dass mindestens ein Versuch gemacht wurde, den Job zu terminieren.

nicevalue Die nicevalue Option wird benutzt um die Priorität oder den nicevalue eines Jobs oder Batches und allen seinen Children zu ändern. Hat ein Child mehrere Parents, kann eine Änderung, muss aber nicht, in dem nicevalue von einem der Parents Auswirkungen auf die Priorität des Childs haben. In dem Fall, dass es mehrere Parents gibt wird das maximale nicevalue gesucht.
Also, wenn Job C drei Parents P1, P2 und P3 hat und P1 setzt einen Nicevalue von 0, P2 einen von 10 und P3 einen von -10, ist der effektive nicevalue -10. (Umso niederiger der nicevalue, umso besser). Wenn der nicevalue von P2 auf -5 geändert wird, passiert nichts, weil die -10 von P3 besser als -5 ist. Wenn jetzt der nicevalue von P3 auf 0 sinkt, wird die neue effektive nicevalue für Job C -5.
Die nicevalues können Werte zwischen -100 und 100 haben. Werte die diese Spanne übersteigen, werden stillschweigend angepasst.

priority Die priority Option wird benutzt, um die (statische) Priorität eines Jobs zu ändern. Weil Batches und Milestones nicht ausgeführt werden, haben Prioritäten keine Bedeutung für sie.

Ein Wechsel der Priorität betrifft nur den geänderten Job. Gültige Werte liegen zwischen 0 und 100. Dabei korrespondiert 100 mit der niedrigsten Priorität und 0 mit der höchsten Priorität.
Bei der Berechnung der dynamischen Priorität eines Jobs startet der Scheduler mit der statischen Priorität und passt dies, entsprechend der Zeit in der der Job schon wartet, an. Wenn mehr als ein Job die gleiche dynamische Priorität hat, wird der Job mit der niedrigsten Job Id als erster gescheduled.

renice Die renice Option gleicht der nicevalue Option mit dem Unterschied, dass die renice Option relativ arbeitet, während die nicevalue Option absolut arbeitet. Wenn einige Batches einen nicevalue von 10 haben bewirkt eine renice von -5, dass die nicevalue auf 5 zunimmt. (Zunahme, weil je niedriger die Nummer, desto höher die Piorität).

rerun Die rerun Option wird benutzt um einen Job in einem restartable State neu zu starten. Der Versuch einen Job, der nicht restartable ist, neu zu starten, führt zu einer Fehlermeldung. Ein Job ist restartable, wenn er in einem restartable State oder in einem **error** oder **broken_finished** Job State ist.
Wenn das **recursive** Flag spezifiziert ist, wird der Job selbst und alle direkten und indirekten Children, die in einem restartable State sind, neu gestartet. Wenn der Job selbst final ist, wird das in dem Fall *nicht* als Fehler betrachtet. Es ist also möglich Batches rekursiv neu zu starten.

resume Die resume Option wird benutzt um einen suspended Job oder Batch zu reaktivieren. Es gibt dabei zwei Möglichkeiten. Erstens kann der suspended Job oder Batch sofort reaktiviert werden, und zweitens kann eine Verzögerung eingestellt werden.
Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
(Für die Spezifikation einer Zeit siehe auch die Übersicht auf Seite 21.)
Die resume Option kann zusammen mit der suspend Option verwendet werden. Dabei wird der Job suspended und nach der (bzw. zur) spezifizierten Zeit wieder resumed.

run Die run Option wird vom Jobserver benutzt zwecks der Sicherstellung, dass der geänderte Job mit der aktuellen Version übereinstimmt.
Theoretisch ist es möglich, dass nachdem ein Job von einem Jobserver gestartet wurde, der Computer abstürzt. Um die Arbeit zu erledigen wird der Job mittels eines manuellen Eingriffs, von einem anderen Jobserver, neu gestartet. Nach dem Hochfahren des ersten Systems kann der Jobserver versuchen den Job State nach **broken_finished** zu ändern, ohne über das Geschehen nach dem Absturz Bescheid

zu wissen. Das Benutzen der run Option verhindert nun das fälschliche Setzen des Status.

state Die state Option wird hauptsächlich von Jobservern benutzt, kann aber auch von Administratoren benutzt werden. Es wird nicht empfohlen dies so zu machen, es sei denn Sie wissen genau was Sie tun.
Die übliche Prozedur ist, dass der Jobserver den State eines Jobs von **starting** nach **started**, von **started** nach **running** und von **running** nach **finished** setzt. Im Falle eines Absturzes oder anderen Problemen ist es möglich dass der Jobserver einen Job in einen **broken_active** oder **broken_finished** State setzt. Das bedeutet, der Exit Code von dem Prozess steht nicht zur Verfügung und der Exit State muss manuell gesetzt werden.

suspend Die suspend Option wird benutzt um einen Batch oder Job zu suspendieren. Sie arbeitet nur dann rekursiv wenn **local** nicht spezifiziert ist. Wenn ein Parent suspended ist, sind auch alle Children suspended. Die resume Option wird benutzt um die Situation umzukehren. Die **restrict** Angabe bewirkt, dass nur Benutzer der ADMIN Gruppe die Suspendierung wieder aufheben können.

timestamp Die timestamp Option wird vom Jobserver benutzt um die Timestamps der State-Wechsel zu setzen, gemäß der lokalen Zeit aus Sicht des Jobservers.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter job definition

Zweck

Zweck Das *alter job definition* Statement wird eingesetzt um die Eigenschaften der spezifizierten Job Definition zu ändern.

Syntax

Syntax Die Syntax des *alter job definition* Statements ist

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* >
| **condition =** < **none** | *string* >
| < **enable** | **disable** >
| *folderpath* **.** *jobname*
| **ignore dependency = none**
| **ignore dependency = (** *dependencyname* {**,** *dependencyname*} **)**
| **interval =** < **none** | *intervalname* >
| < **childsuspend** | **suspend** | **nosuspend** >
| **merge mode =** < **nomerge** | **merge local** | **merge global** | **failure** >
| **mode =** < **and** | **or** >
| **nicevalue =** < **none** | *signed_integer* >
| **priority =** < **none** | *signed_integer* >
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| < **static** | **dynamic** >
| **translation =** < **none** | *transname* >

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:
condition = < **none** | *string* >
| **dependency** *dependencyname*
| **expired =** < **none** | *signed_period_rj* >
| *folderpath* **.** *jobname*

| **mode =** < **all final** | **job final** >
| **resolve =** < **internal** | **external** | **both** >
| **select-statement condition =** < **none** | *string* >
| **state = none**
| **state = (** JRQ_REQ_STATE {**,** JRQ_REQ_STATE} **)**
| **state = all reachable**
| **state = default**
| **state = unreachable**
| **unresolved =** JRQ_UNRESOLVED

JRD_ITEM:

amount = *integer*
| **expired =** < **none** | *signed_period* >
| < **nokeep** | **keep** | **keep final** >
| **condition =** < *string* | **none** >
| **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** >]

JRQ_UNRESOLVED:

defer
| **defer ignore**
| **error**
| **ignore**
| **suspend**

LOCKMODE:

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

Beschreibung

Beschreibung Das *alter job definition* Kommando kennt zwei verschiedene Varianten.

- Die erste ähnelt dem *create job definition* Statement und wird benutzt um die Job Definition erneut zu definieren. Alle betroffenen Optionen werden überschrieben. Alle nichtadressierten Optionen verbleiben wo sie sind.
- Die zweite Variante wird benutzt um Einträge aus den Listen der Children, Resource-Anforderungen, Abhängigkeiten oder Parameter hinzuzufügen, zu ändern oder zu löschen.

Die Optionen werden ausführlich in dem *create job definition* Kommando auf Seite 128 beschrieben. Dieses gilt auch für die Optionen in Child-, Resource-, Anforderungs-, Dependency- und Parameter-Definitionen.
Wird das **existing** Schlüsselwort verwendet, wird kein Fehler erzeugt wenn die adressierte Job Definition nicht existiert. Ist das **existing** Schlüsselwort während der Löschung oder Änderung der Listeneinträge in Benutzung, gilt auch für sie dasselbe.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter named resource

Zweck

Das *alter named resource* Statement wird eingesetzt um die Eigenschaften einer Named Resource zu ändern. *Zweck*

Syntax

Die Syntax des *alter named resource* Statements ist *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*]

Beschreibung

Beschreibung Das *alter named resource* Statement wird verwendet um Eigenschaften der Named Resources zu ändern. (Für eine ausführliche Beschreibung der Optionen siehe die Beschreibung des *create named resource* Statements auf der Seite 147.)
Wenn das Schlüsselwort **existing** spezifiziert wird, führt der Versuch eine nicht existierende Named Resource zu ändern *nicht* zu einem Fehler.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter resource

Zweck

Das *alter resource* Statement wird eingesetzt um die Eigenschaften von Resources zu ändern. *Zweck*

Syntax

Die Syntax des *alter resource* Statements ist *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** >

Beschreibung

Das *alter resource* Statement wird verwendet um die Eigenschaften von Resources zu ändern.(Für eine ausführliche Beschreibung der Optionen siehe die Beschreibung des *create resource* Statements auf Seite 150.) *Beschreibung*
Wenn das **existing** Schlüsselwort spezifiziert ist, wird der Versuch eine nicht existierende Resource zu ändern *nicht* zu einem Fehler führen.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter resource state mapping

Zweck

Zweck Das *alter resource state mapping* Statement wird eingesetzt um die Eigenschaften eines Mappings zu ändern.

Syntax

Syntax Die Syntax des *alter resource state mapping* Statements ist

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

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

Beschreibung

Beschreibung Das *alter resource state mapping* Statement wird verwendet um die Eigenschaften des Resource State Mappings zu ändern. (Für eine ausführliche Beschreibung der Optionen siehe die Beschreibung des *create resource state mapping* Statements auf Seite 155.)
Wenn das **existing** Schlüsselwort spezifiziert ist, wird der Versuch ein nicht existierendes Resource State Mapping zu ändern *nicht* zu einem Fehler führen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter resource state profile

Zweck

Zweck

Das *alter resource state profile* Statement wird eingesetzt die Eigenschaften des spezifizierten Resource State Profiles zu ändern.

Syntax

Syntax

Die Syntax des *alter resource state profile* Statements ist

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

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

Beschreibung

Beschreibung

Das *alter resource state profile* Statement wird werwendet um Eigenschaften der Resource State Profiles zu ändern. (Für eine ausführliche Beschreibung der Optionen siehe die Beschreibung der *resource state profile* Statements auf Seite 156.)
Wenn das **existing** Schlüsselwort spezifiziert ist, wird der Versuch ein nicht existierendes Resource State Profile zu ändern, *nicht* zu einem Fehler führen.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter schedule

Zweck

Zweck Das *alter schedule* Statement wird eingesetzt um die Eigenschaften des spezifizierten Zeitplans zu ändern.

Syntax

Syntax Die Syntax des *alter schedule* Statements ist

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**

Beschreibung

Beschreibung Das *alter schedule* Statement wird verwendet um die Eigenschaften eines Schedules zu ändern. (Für eine ausführliche Beschreibung der Optionen des *create schedule* Statements siehe Seite 157.)
Wenn das **existing** Schlüsselwort spezifiziert ist, wird der Versuch ein nicht existierendes schedule zu ändern, *nicht* zu einem Fehler führen.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter scheduled event

Zweck

Zweck Das *alter scheduled event* Statement wird eingesetzt um Eigenschaften des spezifizierten scheduled Events zu ändern.

Syntax

Syntax Die Syntax des *alter scheduled event* Statements ist

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*

Beschreibung

Beschreibung Das *alter scheduled event* Statement wird verwendet um die Eigenschaften eines scheduled Events zu ändern. (Für eine ausführliche Beschreibung der Optionen des *create scheduled event* Statements siehe Seite 159.)
Wenn das **existing** Schlüsselwort spezifiziert ist, wird der Versuch ein nicht existierendes scheduled Event zu ändern, *nicht* zu einem Fehler führen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter scope

Zweck

Das *alter scope* Statement wird eingesetzt um die Eigenschaften eines spezifizierten Scopes zu ändern. *Zweck*

Syntax

Die Syntax des *alter scope* Statements ist *Syntax*

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

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

alter [**existing**] **jobserver**
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* >

Beschreibung

Beschreibung Das *alter scope* Kommando ist ein Benutzerkommando. Dieses Kommando wird verwendt um die Konfiguration oder andere Eigenschaften eines Scopes zu ändern.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter server

Zweck

Das *alter server* Statement wird eingesetzt um die Benutzerverbindungen zu aktivieren oder deaktivieren oder um den Trace Level festzulegen. *Zweck*

Syntax

Die Syntax des *alter server* Statements ist *Syntax*

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

alter server with schedule

alter server with trace level = *integer*

alter server with < **suspend** | **resume** > *integer*

Beschreibung

Das *alter server* Kommando kann verwendet werden um die Möglichkeit sich mit dem Server zu verbinden ein- und auszuschalten. Wurde die Möglichkeit sich mit dem Server zu verbinden ausgeschaltet, kann sich nur der Benutzer "System" verbinden. *Beschreibung*
Das *alter server* Kommando wird ebenso benutzt um die prokotollierten Typen von Server-Nachrichten zu definieren. Die folgenden Informationstypen werden definiert:

Type	Bedeutung
Fatal	Ein fataler Fehler ist aufgetreten. Der Server wird runtergefahren.
Error	Ein Fehler ist aufgetreten.
Info	Eine wichtige informative Nachricht, die nicht aufgrund eines Fehlers geschrieben wird.
Warning	Eine Warnung.
Message	Eine informative Nachricht.
Debug	Meldungen die für die Fehlersuche benutzt werden können.

Fatal-Nachrichten, Error-Nachrichten und Info-Nachrichten werden immer ins Server Logfile geschrieben. Warnings werden geschrieben, wenn der Trace Level 1 oder höher ist. Normale Messages werden mit einem Trace Level von 2 oder höher

geschrieben. Debug-Nachrichten liefern sehr viel Output und werden ausgegeben wenn der Trace Level 3 ist.
Die **schedule** Option wird benutzt um den Scheduling Thread einen vollständigen Reschedule ausführen zu lassen.
Mit der **suspend/resume** Option können interal Threads angehalten oder wieder gestartet werden.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter session

Zweck

Das *alter session* Statement wird eingesetzt um das genutzten Protokoll, den session timeout Wert oder den Trace Level für die spezifizierte Session zu setzen. *Zweck*

Syntax

Die Syntax des *alter session* Statements ist *Syntax*

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

alter session set user = *username* [**with** WITHITEM {**,** WITHITEM}]

alter session set user = *username* **for** *username* [**with** WITHITEM {**,** WITHITEM}]

alter session set user is default

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

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

Beschreibung

Das *alter session* Kommando kann verwendet werden um die Trace ein- und auszuschalten. Ist die Trace eingeschaltet, werden alle abgesetzten Befehle ins Logfile *Beschreibung*

protokolliert. Des Weiteren kann ein Kommunikationsprotokoll gewählt werden. Eine Übersicht der derzeitig definierten Protokolle wird in der untenstehenden Tabelle angezeigt.

Protokoll	Bedeutung
Line	Reines ASCII Output
Perl	Die Ausgabe wird als Perl-Struktur angeboten, die mittels `eval()` auf einfache Weise dem Perl Script bekannt gemacht werden kann.
Python	Wie Perl, nur handelt es sich hier um eine Python-Struktur.
Serial	Serialisierte Java Objekte
Xml	Eine xml Struktur wird ausgegeben.

Als letzte Möglichkeit kann der Timeout Parameter für die Session festgelegt werden. Ein Timeout von 0 bedeutet, dass kein Timeout aktiv ist. Jede Zahl größer als 0 gibt die Anzahl Sekunden an, nach der eine Session automatisch disconnected wird.
Die zweite Form des *alter session* Statements ist ausschließlich Mitgliedern der Gruppe ADMIN vorbehalten. Sie dient dazu vorübergehend den Benutzer, und die damit zusammenhängenden Rechte, zu wechseln. Mit der dritten Form wird der Benutzer, sowie alle Rechte, wieder zurückgesetzt.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter trigger

Zweck

Das *alter trigger* Statement wird eingesetzt um Eigenschaften des spezifizierten Triggers zu ändern. *Zweck*

Syntax

Die Syntax des *alter trigger* Statements ist *Syntax*

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

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

WITHITEM:
< **active** | **inactive** >
| **check =** *period*
| **condition =** < **none** | *string* >
| < **nowarn** | **warn** >
| **event = (** CT_EVENT {**,** CT_EVENT} **)**
| **group event**
| **limit state =** < **none** | *statename* >
| **main none**
| **main** *folderpath*
| < **nomaster** | **master** >
| **parameter = none**
| **parameter = (** *identifier* **=** *expression* {**,** *identifier* **=** *expression*} **)**
| **parent none**
| **parent** *folderpath*
| **rerun**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **single event**
| **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:
after final
| **before final**
| **finish child**
| **immediate local**
| **immediate merge**
| **until final**
| **until finished**
| **warning**

Beschreibung

Beschreibung

Das *alter trigger* Kommando wird benutzt um die Eigenschaften eines definierten Triggers zu ändern. Wenn das **existing** Schlüsselwort spezifiziert ist, führt das Ändern eines nicht existierenden Triggers *nicht* zu einem Fehler.
(Für eine ausführliche Beschreibung der Optionen siehe das *create trigger* Statement auf Seite 164.)

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

alter user

Zweck

Das *alter user* Statement wird eingesetzt um Eigenschaften des spezifizierten Benutzers zu ändern. *Zweck*

Syntax

Die Syntax des *alter user* Statements ist *Syntax*

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

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

WITHITEM:
connect type = < **plain** | **ssl** | **ssl authenticated** >
| **default group =** *groupname*
| < **enable** | **disable** >
| **equivalent = none**
| **equivalent = (** < *username* | *serverpath* > {**,** < *username* | *serverpath* >} **)**
| **group = (** *groupname* {**,** *groupname*} **)**
| **parameter = none**
| **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

ADD_DELITEM:
add [**or alter**] **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**
| < **add** | **delete** > **group = (** *groupname* {**,** *groupname*} **)**
| **alter** [**existing**] **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**
| **delete** [**existing**] **parameter = (** *parmlist* **)**

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

Beschreibung

Beschreibung Das *alter user* Kommando wird verwendet um die Eigenschaften eines definierten Users zu ändern. Wenn das **existing** Schlüsselwort spezifiziert ist, führt der Versuch einen nicht existierenden User zu ändern *nicht* zu einem Fehler.
(Für eine ausführliche Beschreibung der Optionen siehe das *create user* Statement auf Seite 174.)
Die zweite Form des Statements wird benutzt um den User von den spezifizierten Gruppen zu löschen oder zuzufügen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 4

connect commands

connect

Zweck

Zweck Das *connect* Statement wird eingesetzt um Benutzer vom Server authentifizieren zu lassen.

Syntax

Syntax Die Syntax des *connect* Statements ist

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

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

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

Beschreibung

Beschreibung Das *connect* Kommando wird benutzt um den verbundenen Prozess am Server zu authentifizieren. Es kann wahlweise ein Kommunikationsprotokoll spezifiziert werden. Das Default-Protokoll ist **line**.
Das ausgewählte Protokoll definiert die Form des Outputs. Alle Protokolle, außer **serial**, liefern ASCII Output. Das Protokoll **serial** liefert ein Serialized Java Objekt zurück.
Beim *connect* Kommando kann auch gleich ein auszuführendes Kommando mitgegeben werden. Als Output des *connect* Kommandos wird in diesem Fall der Output des mitgegebenen Kommandos genutzt. Falls das Kommando fehlschlägt, der *connect* aber gültig war, bleibt die Connection bestehen.

Im Folgenden ist für alle Protokolle, außer für das **serial** Protokoll, ein Beispiel gegeben.

line protocol Das line protocol liefert nur einen ASCII-Text als Ergebnis von einem Kommando zurück.

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

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

XML protocol Das XML protocol liefert eine XML-Struktur als Ergebnis eines Kommandos zurück.

```
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>
```

python protocol Das python protocol liefert eine Python-Struktur, welche durch die *python eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

perl protocol Das perl protocol liefert eine Perl-Struktur, welche durch die *perl eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 5

copy commands

copy folder

Zweck

Zweck Das *copy folder* Statement wird eingesetzt um einen oder mehrere Folder und/ oder Job Definitions mitsamt des Inhalts an eine andere Stelle in der Ordnerhierarchie zu kopieren.

Syntax

Syntax Die Syntax des *copy folder* Statements ist

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* >]

Beschreibung

Beschreibung Wenn ein Folder kopiert wurde, wird jedes Objekt das der Folder enthält mitkopiert. Wenn Beziehungen zwischen Objects bestehen, welche durch eine *copy folder* Operation kopiert wurden, wie z. B. Abhängigkeiten, Cildren, Trigger etc., werden diese entsprechend geändert und den resultierenden Objekten von der Kopie zugeordnet.
Wenn z. B. ein Ordner SYSTEM.X.F zwei Jobs A und B enthält mit SYSTEM.X.F.B abhängig von SYSTEM.X.F.A, in den Ordner SYSTEM.Y kopiert wird, wird der neu angelegte Job SYSTEM.Y.F.B von dem neu angelegten Job SYSTEM.Y.F.A abhängig sein.
Beachten Sie, dass wenn die Jobs mit einem *copy job definition* Kommando kopiert wurden, der neue Job SYSTEM.Y.F.B immer noch vom SYSTEM.X.F.A abhängig wäre. Es kann sein, dass dies *nicht* der Ansicht des Users entspricht.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

copy named resource

Zweck

Das *copy named resource* Statement wird eingesetzt um eine Named Resource in eine andere Kategorie zu kopieren. *Zweck*

Syntax

Die Syntax des *copy named resource* Statements ist *Syntax*

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

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

Beschreibung

Das *copy named resource* Kommando wird benutzt um eine Kopie von einer Named Resource oder einer ganzen Kategorie zu machen. *Beschreibung*
Wenn der spezifizierte "target resourcepath" bereits als Kategorie existiert, wird eine Named Resource oder Kategorie mit demselben Namen wie das Source Objekt innerhalb dieser Kategorie angelegt.
Wenn das spezifizierte "target resourcepath" bereits als Named Resource existiert, wird dies als Fehler betrachtet.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

copy scope

Zweck

Zweck Das *copy scope* Statement wird eingesetzt um um einen Scope mitsamt seines Inhalts an eine andere Stelle in der Scope-Hierarchie zu kopieren.

Syntax

Syntax Die Syntax des *copy scope* Statements ist

copy < **scope** *serverpath* | **jobserver** *serverpath* > **to** *serverpath*

copy < **scope** *serverpath* | **jobserver** *serverpath* > **to** *scopename*

Beschreibung

Beschreibung Das *copy scope* Kommando wird benutzt um eine Kopie von ganzen Scopes zu machen. Diese Kopie umfasst auch Resource- und Parameter Definitions.
Wenn der spezifizierte "target serverpath" bereits als Scope existiert, wird ein Scope mit demselben Namen wie das Source Object innerhalb dieses Scopes angelegt.
Wenn das spezifizierte "target serverpath" bereits als Jobserver existiert, wird dies als Fehler betrachtet.
Da ein Jobserver nur als eine spezielle Art von Scope angesehen wird, ist es möglich Jobserver mit diesem Kommando zu kopieren. In diesem Fall ist dieses Kommando identisch mit dem *copy jobserver* Kommando.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 6

create commands

create comment

Zweck

Zweck Das *create comment* Statement wird eingesetzt um einen Kommentar zum spezifizierten Objekt anzulegen.

Syntax

Syntax Die Syntax des *create comment* Statements ist

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*
| **exit state translation** *transname*
| **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* | **jobserver** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

CC_WITHITEM:
    CC_TEXTITEM {**,** CC_TEXTITEM}
  | **url =** *string*

PARAM_LOC:
    folder *folderpath*
  | **job definition** *folderpath*
  | **named resource** *resourcepath*
  | < **scope** *serverpath* | **jobserver** *serverpath* >

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

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

Beschreibung

Beschreibung

Das *create comment* Statement wird benutzt um die Kurzbeschreibung bzw. die URL der Beschreibung des zu kommentierenden Objektes zu erstellen.
Das optionale Schlüsselwort **or alter** wird benutzt um den Kommentar, wenn einer existiert, zu aktualisieren. Wenn es nicht spezifiziert ist, führt das Vorhandensein eines Kommentars zu einem Fehler.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create environment

Zweck

Zweck Das *create environment* Statement wird eingesetzt um eine Anzahl von Static Named Resources, welche in den Scopes, in dem ein Job laufen will, gebraucht werden, zu definieren.

Syntax

Syntax Die Syntax des *create environment* Statements ist

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** >]

Beschreibung

Beschreibung Das *create environment* Statement wird benutzt um eine Reihe von Static Resource Requests festzulegen, welche die notwendige Umgebung, die ein Job braucht, beschreiben. Da die Environments nicht von normalen Benutzern angelegt werden können und Jobs den Environment, den sie zum Ablauf benötigen, beschreiben müssen, können Environments benutzt werden um Jobs zu zwingen einen bestimmten Jobserver zu benutzen.

Resources Die *Resources* Klausel wird benutzt um die Required (Static) Resources zu spezifizieren. Spezifizierte Resourcen, welche nicht static sind, führen zu einem Fehler. Da nur statische Resources spezifiziert werden, werden keine weiteren Informationen benötigt. Es ist zulässig einen leeren Environment (ein Environment ohne Resource-Anforderungen) zu spezifizieren. Dies wird *nicht* empfohlen, da es den Verlust an Kontrolle bedeutet.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create event

Zweck

Zweck

Das *create event* Statement wird eingesetzt um eine Aktion, die vom Time Scheduling Modul ausgeführt wird, zu definieren.

Syntax

Syntax

Die Syntax des *create event* Statements ist

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* >

Beschreibung

Beschreibung

Das *create event* Statement wird benutzt um eine Aktion, die vom Time Scheduling System scheduled werden kann, zu definieren. Die definierte Aktion ist die Submission eines Master Submittable Jobs oder Batches.

action Der Submit-Teil von dem Statement ist eine eingeschränkte Form des Submit-Kommandos (Siehe Seite 412).

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create exit state definition

Zweck

Zweck Das *create exit state definition* Statement wird eingesetzt um einen symbolischen Namen für den State eines Jobs zu erstellen.

Syntax

Syntax Die Syntax des *create exit state definition* Statements ist

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

Beschreibung

Beschreibung Das *create exit state definition* Statement wird benutzt um einen symbolischen Namen für den Exit State eines Jobs, Milestones oder Batches zu definieren.
Das optionale Schlüsselwort **or alter** wird benutzt um das Auftreten von Fehlermeldungen und das Abbrechen der laufenden Transaktion, wenn eine Exit State Definition bereits existiert, zu verhindern. Das ist in Kombination mit *multicommands* besonders nützlich. Wenn es nicht spezifiziert ist, führt das Existieren einer Exit State Definition mit dem spezifizierten Namen zu einem Fehler.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel In den folgenden Beispielen wurden symbolische Namen für Job States erstellt.

```
create exit state definition erfolg;
create exit state definition fehler;
create exit state definition erreicht;
create exit state definition warnung;
create exit state definition warten;
create exit state definition ueberspringen;
create exit state definition unerreichbar;
```

create exit state mapping

Zweck

Zweck

Das *create exit state mapping* Statement wird eingesetzt um ein Mapping zwischen dem numerischen Exit Code eines Prozesses und einem symbolischen Exit State zu erstellen.

Syntax

Syntax

Die Syntax des *create exit state mapping* Statements ist

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

Beschreibung

Beschreibung

Das *create exit state mapping* Statement definiert das Mapping von Exit Codes zu logischen Exit States. Die einfachste Form des Statements spezifiziert nur einen Exit State. Das bedeutet, dass der Job automatisch diesen Exit State nach Ablauf bekommt, ungeachtet seines Exit Codes. Komplexere Definitionen spezifizieren mehr als einen Exit State und mindestens eine Abgrenzung.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel

Das nachfolgende Beispiel zeigt ein relativ einfaches, jedoch realistisches, Mapping von Exit Codes nach logischen Exit States.
Das Statement

```
create exit state mapping beispiel1
with map = (   fehler,
            0, erfolg,
            1, warnung,
            4, fehler);
```

definiert folgendes Mapping:

Exit code Range von	Exit code Range bis	Resultierende exit state
$-\infty$	-1	fehler
0	0	erfolg
1	3	warnung
4	∞	fehler

create exit state profile

Zweck

Zweck Das *create exit state profile* Statement wird eingesetzt um eine Reihe von gültigen Exit States zu definieren.

Syntax

Syntax Die Syntax des *create exit state profile* Statements ist

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:
batch default
| **broken**
| **dependency default**
| **disable**
| **unreachable**

Beschreibung

Beschreibung Das *create exit state profile* Statement wird benutzt um eine Menge von gültigen Exit States für einen Job, Milestone oder Batch zu definieren.

default mapping Mit der default mapping Klausel ist es möglich zu definieren welches Exit State Mapping benutzt werden soll, wenn kein anderes Mapping spezifiziert ist. Dieses vereinfacht die Erstellung von Jobs wesentlich.

force Während ein Exit State Profile erstellt wird, hat die force Option keinen Effekt und wird ignoriert. Wenn "**or alter**" spezifiziert ist und der Exit State Profile, der erstellt werden soll, bereits existiert, verschiebt die force Option die Integritätsprüfung auf später.

state Die state Klausel definiert welche Exit State Definitions innerhalb dieses Profiles gültig sind. Jede Exit State Definition muss als **final**, **restartable** oder **pending** klassifiziert sein. Wenn ein Job ein State der **final** ist erreicht, kann dieser Job nicht mehr gestartet werden. Das bedeutet, der State kann sich nicht mehr ändern. Wenn ein Job ein State, der **restartable** ist erreicht, kann er noch einmal gestartet werden. Das bedeutet, dass sich der State von solch einem Job ändern kann. **Pending** bedeutet, dass ein Job nicht neu gestartet werden kann, aber auch nicht **final** ist. Der Status muss von außerhalb gesetzt werden.
Die Reihenfolge, in der die Exit States definiert sind, ist relevant. Der erste spezifizierte Exit State hat die höchste, und der zuletzt spezifizierte Exit State hat die niedrigste Präferenz. Normalerweise werden **final** States später als **restartable** States spezifiziert. Die Präferenz eines States wird benutzt um zu entscheiden welcher State sichtbar ist, wenn mehrere verschiedene Exit States von Children zusammengeführt werden.
Man kann genau einen Exit State als **unreachable** State deklarieren. Das bedeutet, ein Job, Batch oder Milestone mit diesem Profile bekommt den spezifizierten State, sobald er unreachable geworden ist. Dieses Exit State muss **final** sein.
Maximal ein Exit State innerhalb eines Profiles kann als **broken** State gekennzeichnet werden. Das bedeutet, ein Job wird diesen State erreichen, sobald der Job in den **error** oder **broken_finished** State gewechselt ist. Dieses kann mittels eines Triggers behandelt werden. Der Exit State der als **broken** State definiert ist, muss **restartable** sein.
Maximal ein State kann als **batch default** deklariert werden. Ein leerer Batch wird diesen Status annehmen. Damit kann explizit vom Standardverhalten abgewichen werden. Wird kein State als **batch default** gekennzeichnet, wird ein leerer Batch automatisch den, nicht als **unreachable** gekennzeichneten, finalen State mit niedrigster Präferenz annehmen. Gibt es einen solchen State nicht, wird auch der **unreachable** State als Kandidat betrachtet.
Beliebig viele final States können als **dependency default** gekennzeichnet werden. Dependencies, die eine Default-Abhängigkeit definieren, werden dann erfüllt, wenn der required Job eine der States, die als **dependency default** gekennzeichnet sind, annimmt.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

Beispiel

In den folgenden Beispielen werden die Exit State Profiles `beispiel_1` und `beispiel_2` erstellt. *Beispiel*
Im ersten, sehr einfachen Beispiel, soll der Exit State `erfolg` ein final State sein.

```
create exit state profile beispiel_1
with
```

```
    state = ( erfolg final
            );
```

Im zweiten Beispiel wird der Exit State `fehler` als restartable definiert. Dieser State hat eine höhere Priorität als der (final) State `success` und muss dementsprechend als erste aufgeführt werden.

```
create exit state profile beispiel_2
with
    state = ( fehler restartable,
              erfolg final
            );
```

create exit state translation

Zweck

Zweck

Das *create exit state translation* Statement wird eingesetzt um eine Übersetzung zwischen den Child und Parent Exit States zu erstellen.

Syntax

Syntax

Die Syntax des *create exit state translation* Statements ist

create [**or alter**] **exit state translation** *transname*
with translation = (*statename* **to** *statename* {**,** *statename* **to** *statename*}
)

Beschreibung

Beschreibung

Das *create exit state translation* Statement wird benutzt um eine Übersetzung zwischen zwei Exit State Profiles zu definieren. Eine solche Übersetzung kann, muss aber nicht, in Parent-Child-Beziehungen benutzt werden, wenn die beiden beteiligten Exit State Profiles inkompatibel sind. Die Default-Übersetzung ist die Identität. Dies bedeutet, Exit States werden nach Exit States mit demselben Namen übersetzt, solange nichts anderes spezifiziert ist.
Es ist nicht möglich einen Final State nach einem Restartable State zu übersetzen.
Wenn die Exit State Translation schon existiert und das Schlüsselwort "**or alter**" spezifiziert ist, ist die spezifizierte Exit State Translation geändert. Andererseits führt eine schon existierende Exit State Translation, mit demselben Namen, zu einem Fehler.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel

Im folgenden Beispiel wird der Exit State des Childs `warnung` nach dem Exit State des Parents `ueberspringen` übersetzt.

```
create exit state translation beispiel1
with translation = ( warnung to ueberspringen );
```

create folder

Zweck

Zweck Das *create folder* Statement wird eingesetzt um eine Mappe für Job Definitions und/oder für andere Folder zu erstellen.

Syntax

Syntax Die Syntax des *create folder* Statements ist

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**

Beschreibung

Beschreibung Dieses Kommando erstellt einen Folder und kennt folgende Optionen:

environment Wenn ein Environment einem Folder zugewiesen wurde, wird jeder Job in diesem Folder, und seinen Subfolders, alle Resource-Anforderungen von der Environment Definition erben.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

parameter Mit der parameter Option ist es möglich key/value pairs für den Folder zu definieren. Die vollständige Liste der Parameter muss innerhalb eines Kommandos spezifiziert werden.

inherit grant Die inherit grant Klausel ermöglicht es zu definieren welche Privilegien über die Hierarchie geerbt werden sollen. Wird diese Klausel nicht spezifiziert, werden per Default alle Rechte geerbt.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create footprint

Zweck

Zweck Das *create footprint* Statement wird eingesetzt um eine Anzahl von häufig benutzten System Resource-Anforderungen zu erstellen.

Syntax

Syntax Die Syntax des *create footprint* Statements ist

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

REQUIREMENT:
ITEM { ITEM}

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

Beschreibung

Beschreibung Das *create footprint* Kommando erstellt eine Menge von Resource-Anforderungen welche wiederbenutzt werden können. Die Required Resources sind alle System Resources. Die Required Resources werden durch ihren Namen beschrieben, eine Menge, per Default Null und optional eine keep Option.

keep Die keep Option in einer Resource-Anforderung definiert den Zeitpunkt zu dem die Resource freigegeben wird. Die keep Option ist sowohl für System als auch für Synchronizing Resources gültig. Es gibt drei mögliche Werte. Ihre Bedeutung wird in der folgenden Tabelle erklärt:

Wert	Bedeutung
nokeep	Die Resource wird am Jobende freigegeben. Dies ist das Default-Verhalten.
keep	Die Resource wird freigegeben, sobald der Job den final State erreicht hat.
keep final	Die Resource wird freigegeben, wenn der Job und alle seine Children final sind.

amount Die amount Option ist nur mit Anforderungen für Named Resources von der Art system oder synchronizing gültig. Der Amount in einer Resource-Anforderung drückt aus, wieviele Einheiten von der Required Resource belegt werden.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create group

Zweck

Zweck Das *create group* Statement wird eingesetzt um ein Objekt zu erstellen, an das man Rechte vergeben kann.

Syntax

Syntax Die Syntax des *create group* Statements ist

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

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

Beschreibung

Beschreibung Das *create group* Statement wird benutzt um eine Gruppe zu erstellen. Ist das Schlüsselwort "**or alter**" spezifiziert, wird eine bereits existierende Gruppe geändert. Ansonsten wird eine bereits existierende Gruppe als Fehler betrachtet.

user Die user Klausel wird benutzt um zu spezifizieren, welche Benutzer Gruppenmitglieder sind.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create interval

Zweck

Das *create interval* Statement wird eingesetzt um ein periodisches oder nicht periodisches Muster, auf welchem Events getriggert werden können, zu definieren. *Zweck*

Syntax

Die Syntax des *create interval* Statements ist *Syntax*

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

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

IVAL_DISPATCHITEM:
dispatchname < **active** | **inactive** > IVAL_DISPATCHDEF

CINTERVALNAME:
(*intervalname*
with WITHITEM {**,** WITHITEM} **)**
| *intervalname*

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

IVAL_DISPATCHDEF:
none CINTERVALNAME < **enable** | **disable** >
| CINTERVALNAME CINTERVALNAME < **enable** | **disable** >
| CINTERVALNAME < **enable** | **disable** >

Beschreibung

Beschreibung

Die Intervalle sind das Herzstück des Time Schedulings. Sie können als Muster von Blöcken gesehen werden. Diese Muster können periodisch oder aber auch aperiodisch sein. Innerhalb einer **Periode** (**Base**), die im aperiodischen Fall eine Länge unendlich (∞) hat, gibt es Blöcke einer bestimmten Länge (**Duration**). Der letzte Block kann dabei unvollständig sein falls die Periodenlänge kein ganzzahliges Vielfaches der Duration ist. Die Duration kann ebenfalls eine Länge ∞ haben. Das bedeutet, dass die Blöcke dieselbe Länge wie die Perioden haben.

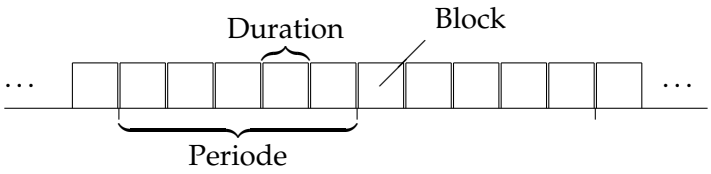

Abbildung 6.1: Darstellung von Perioden und Blöcken

Nun müssen nicht alle Blöcke auch tatsächlich vorhanden sein. Welche Blöcke vorhanden sind, kann gewählt werden. Dieses **Wählen** kann durch Angabe der Blocknummer relativ zum Anfang oder Ende einer Periode ($1, 2, 3$ bzw. $-1, -2, -3$) oder durch "von - bis" Angaben (alle Tage zwischen 3.4. und 7.6.) erfolgen.
Dadurch entstehen kompliziertere Muster wie in Abbildung 6.2.

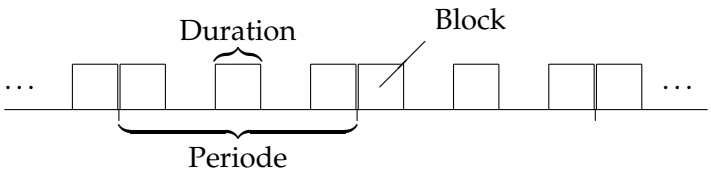

Abbildung 6.2: Komplexeres Muster

Das Auswählen ist 1-based, d.h. der erste Block hat die Nummer 1. Der letzte Block wird mittels Nummer -1 addressiert. Ein Block 0 existiert somit nicht.
Ein Intervall kann im Wesentlichen mit folgenden Parametern beschrieben werden: Basisfrequenz (Periodenlänge), Duration und Auswahl. Da ein Intervall aber nicht zwingend immer gültig sein muss, kann weiterhin noch ein Start- und Endzeitpunkt angegeben werden.

Unendliche Intervalle Bei einem aperiodischen Intervall ohne Duration (Unendlichkeit) kommt dem Startzeitpunkt eine Sonderrolle zu: Er definiert dann die einzige positive Flanke dieses Intervalls. Analog dazu legt ein Endzeitpunkt die einzige negative Flanke fest.
Wird eine Auswahl getroffen, führt diese Auswahl jeweils zu der Entstehung von Blöcken. Eine Auswahl "-0315T18:40" führt jedes Jahr am 15. März zu einem Block von 18h40 bis 18h41.
Es ist selbstverständlich, dass eine Auswahl von Blöcken über die Position (erster, zweiter, ..) natürlich unsinnig ist. Dies wird bei unendlichen Intervallen dann auch ignoriert.

Inverse Wenn etwa die Zeit zwischen Weihnachten und Neujahr zu irgendeinem Zweck positiv definiert wurde, gibt es bisher keine Möglichkeit die komplementierende Zeit leicht zu definieren. Im aktuellen Beispiel ist das noch nicht gravierend, bei komplexeren Mustern führt diese Unmöglichkeit zu aufwendigen und fehlerträchtigen Doppeldefinitionen.
Es wird daher ein Inverse Flag eingeführt, welches zur Folge hat, dass die angegebene Auswahlliste komplementär interpretiert wird, d.h. es werden nur die Blöcke ausgewählt, die ohne gesetztes Invert-Flag nicht ausgewählt würden. Im Falle des Monatsultimo (letzter Arbeitstag des Monats) resultiert das Setzen des Inverse-Flags also in allen Arbeitstagen, mit Ausnahme des Monatsultimo.

Filter Die Auswahl von Blöcken kann noch weiter eingeschränkt werden. Hat man zum Beispiel einen Intervall "Tag des Monats" definiert (d.h. die Base ist ein Monat, die Duration ein Tag) und dann den zweiten Block ausgewählt, würde ein solcher Intervall jeweils am zweiten Tag eines Monats einen Block haben. Möchte man, dass dies nur für die ungeraden Monate (Januar, März, Mai, ...) definieren, dann wäre dies ohne Filterfunktionalität wegen der Schaltjahre nicht möglich.
Die Lösung des Problems besteht darin, dass ein weiterer Intervall (Monat des Jahres), mit der Selektion $1, 3, 5, 7, 9, 11$ definiert wird ind dieser Intervall als Filter des ersten Intervalls angegeben wird.
Es gilt dann, dass der erste Intervall nur dann einen Block zeigt, wenn auch der zweite zu der "Zeit" einen Block zeigt.
Werden mehrere Intervalle als Filter angegeben, reicht es, wenn einer dieser Intervalle einen Block zum verlangten Zeitpunkt hat (ODER). Für die Abbildung einer AND-Beziehung zwischen den Filterintervallen werden die Filterintervalle als Kette angelegt (A filtert B filtert C ... usw.). Die Reihenfolge der Filter ist dabei unwichtig.

Embedded Leider ist die Welt nicht immer ganz so einfach. Insbesondere ist es nicht unwichtig, ob man zuerst eine Operation ausführt, und dann seine Auswahl trifft, oder zuerst wählen muss und anschließend die Operation anwendet. Anders

gesagt, ist es ein großer Unterschied, ob man über den letzten Tag des Monats – falls dies ein Arbeitstag ist – redet, oder über den letzten Arbeitstag des Monats.
Diese Möglichkeit der Differenzierung wollen wir natürlich auch in unser Modell aufnehmen. Dazu wird die Möglichkeit des **Einbettens** eingeführt.
Beim Einbetten werden zuerst alle Parameter des eingebetteten Intervalls übernommen. Anschließend wird die Auswahlliste evaluiert. Obwohl erlaubt, hat dabei die Angabe einer "von - bis" Auswahl natürlich keinen Sinn, da diese Funktionalität mittels einer einfachen Multiplikation ebenfalls erzielbar ist. Viel interessanter ist die Möglichkeit der relativen Auswahl. Wenn etwa die Arbeitstage eines Monats eingebettet werden und anschließend der Tag -1 ausgewählt wird, haben wir insgesamt ein Intervall, welches den letzten Arbeitstag eines Monats definiert. Wird dagegen das Intervall mit den Arbeitstagen eines Monats mit einem Intervall, das den letzten Tag eines Monats liefert, multipliziert, erhalten wir nur dann einen Treffer, wenn der letzte Tag des Monats ein Arbeitstag ist.
Das Einbetten kann auch so verstanden werden, dass bei der Auswahl der Blöcke nicht *alle* eingebetteten Blöcke betrachet (und vor allem gezählt) werden, sondern statt dessen nur die *aktiven* Blöcke berücksichtigt werden.

Synchronisation Nicht berücksichtigt sind noch die Situationen, in denen Vielfache einzelner Perioden im Spiel sind. Eine Periode von z.B. 40 Tagen könnte ihre steigende Flanke zu jeder beliebigen Mitternacht (00:00h) haben. Daher wird ein Synchronisationszeitpunkt (**synctime**) eingeführt, der die früheste Flanke auswählt, die $\geq$ diesem Termin ist. Wird kein solcher Zeitpunkt explizit angegeben, wird das Datum der Definitionserstellung (*create*) verwendet.
Der erste Block einer Periode beginnt zunächst grundsätzlich an deren Beginn. In Fällen, in denen das nicht möglich ist (Periode = ∞, Duration $>$ Periode, Periode XOR Duration haben die Einheit Woche), wird der Beginn der Periode als Synchronisationszeitpunkt verwendet. Ist auch das nicht möglich (Periode = ∞), kommt der normale Synchronisationszeitpunkt zum Tragen. Dieses Vorgehen hat zur Folge, dass auch der *erste* Block einer Periode unvollständig sein kann (und dann *niemals* aktiv ist).

Dispatcher Obwohl die bisherigen Syntaxkomponenten sehr mächtig sind, und nahezu jeden Rhythmus beschreiben können, ist ihre Anwendung nicht immer intuitiv. Dies ist zum Zeitpunkt der Erstellung des Intervalls noch kein Problem, kann aber bei der späteren Wartung zu einem Problem werden.
Der Dispatcher ermöglicht es dem Anwender Intervalldefinitionen zu entwickeln, die deutlich einfacher zu verstehen sind.
Als Beispiel nehmen wir an, dass ein Job Montags um 10:00 gestartet werden soll, aber an den restlichen Wochentagen bereits um 09:00.
Als erstes entwickeln wir ein Intervall, welches Montags um 10:00 feuert:

```
create or alter interval MONDAY10
```

```
with
        base = none,
        duration = none,
        selection = ('T10:00'),
        filter = (
                (MONDAYS
                 with
                        base = 1 week,
                        duration = 1 day,
                        selection = (1)
                )
        );
```

Die Möglichkeit Filter und embedded Intervalle ”inline” definieren zu können führt hier zu einer schlanken Definition.
Das Intervall, welches an den restlichen Wochentagen um 09:00 feuert, sieht ähnlich aus:

```
create or alter interval WEEKDAY09
with
        base = none,
        duration = none,
        selection = ('T09:00'),
        filter = (
                (WEEKDAYS
                 with
                        base = 1 week,
                        duration = 1 day,
                        selection = (2, 3, 4, 5)
                )
        );
```

Das kombinierte Intervall ohne Dispatcher sieht damit folgendermaßen aus:

```
create or alter interval MO10_DI_FR09
with
        base = none,
        duration = none,
        selection = ('T09:00', 'T10:00'),
        filter = (MONDAY10, WEEKDAY09);
```

Die beiden möglichen Uhrzeiten werden selektiert und beide Filter werden evaluiert. Montags wird nur die Uhrzeit 10:00 durchgelassen, an anderen Tagen nur die Uhrzeit 09:00.
Die selbe Funktionalität, aber jetzt mit Dispatcher, sieht verständlicher aus:

```
create or alter interval D_MO10_DI_FR09
with
        base = none,
        duration = none,
        filter = none,
```

```
        selection = none,
        dispatch = (
                MONDAY_RULE
                        active
                        (MONDAYS
                         with
                                base = 1 week,
                                duration = 1 day,
                                selection = (1)
                        )
                        (MONDAY_TIME
                         with
                                base = none,
                                duration = none,
                                selection = ('T10:00')
                        )
                        enable,
                WEEKDAY_RULE
                        active
                        (WEEKDAYS
                         with
                                base = 1 week,
                                duration = 1 day,
                                selection = (2, 3, 4, 5)
                        )
                        (WEEKDAY_TIME
                         with
                                base = none,
                                duration = none,
                                selection = ('T09:00')
                        )
                        enable
        );
```

Die Anforderung ist in dieser Form klar dargestellt, leicht verständlich und ebenso leicht wartbar.
Eine Dispatcher-Definition ist relativ einfach. Sie besteht erstmal aus einer Liste mit Regeln. Die Reihenfolge dieser Regeln hat dabei eine Bedeutung. Wenn zwei oder mehr Regeln ”zuständig” sind, gewinnt die erste Regel aus der Liste.
Im obigen Beispiel könnte das WEEKDAYS Intervall so geändert werden, dass auch der Montag selektiert wird:

```
...
                WEEKDAY_RULE
                        active
                        (WEEKDAYS
                         with
                                base = 1 week,
                                duration = 1 day,
                                selection = (1, 2, 3, 4, 5)
                        )
...
```

Da aber die erste Regel, `MONDAY_RULE`, den Montag bereits behandelt, hätte die Änderung keinen Effekt.
Eine Dispatch-Regel besteht aus 5 Angaben. Sie fängt an mit einem Namen, der den üblichen Regeln für einen Identifier entsprechen muss. Der Name hat keine Auswirkung und dient im Wesentlichen als Möglichkeit die Idee hinter der Regel zu verdeutlichen. Innerhalb des Dispatchers muss der Name (als Name einer Regel) eindeutig sein.
Die nächste Angabe is der **active** Flag. Steht sie auf **inactive** werden keine Blöcke erzeugt, bzw. alle Blöcke herausgefiltert. Steht sie auf **active**, wird das Filter Intervall ausgewertet.
Die dritte Angabe ist das "Select Intervall". Dieses Interval definiert, zu welchen Zeiten die Regel gültig ist. Ist die Regel gültig, wird das Filter Intervall ausgewertet, so fern die Regel als active gekennzeichnet ist.
Wird als Select Intervall das Schlüsselwort **none** angegeben, ist dies gleichbedeutend mit einem unendlichen Interval, ohne weitere Eigenschaften. Dies wiederum bedeutet, salopp gesagt, immer gültig.
Die vierte Angabe ist das "Filter Intervall". Dieses Interval macht die eigentliche Arbeit. Im obigen Beispiel erzeugt es einen Block mit einer Startzeit von 09:00 (Montags).
Das Filter Intervall darf weggelassen werden. Auch hier ist dies gleichbedeutend mit einem unendlichen Intervall ohne weitere Eigenschaften. Als Driver gibt es keine Blöcke, als Filter lässt es alles durch.
Die Kombination von **none** als Select Intervall und das Weglassen des Filter Intervalls ist nicht zulässig.
Die letzte Angabe ist der **enable** Flag. Mit diesem Schalter können Regeln ein- oder augeschaltet werden. Ist eine Regel disabled wird sie nicht berücksichtigt.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create job definition

Zweck

Zweck Das *create job definition* Statement wird eingesetzt um ein Scheduling Entity Objekt zu erstellen welches selbstständig oder als Teil einer größeren Hierarchie submitted werden kann.

Syntax

Syntax Die Syntax des *create job definition* Statements ist

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* >
| **condition =** < **none** | *string* >
| < **enable** | **disable** >
| *folderpath* **.** *jobname*
| **ignore dependency = none**
| **ignore dependency = (** *dependencyname* {**,** *dependencyname*} **)**
| **interval =** < **none** | *intervalname* >

| < **childsuspend** | **suspend** | **nosuspend** >
| **merge mode =** < **nomerge** | **merge local** | **merge global** | **failure** >
| **mode =** < **or** | **and** >
| **nicevalue =** < **none** | *signed_integer* >
| **priority =** < **none** | *signed_integer* >
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| < **static** | **dynamic** >
| **translation =** < **none** | *transname* >

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:

condition = < **none** | *string* >
| **dependency** *dependencyname*
| **expired =** < **none** | *signed_period_rj* >
| *folderpath* **.** *jobname*
| **mode =** < **all final** | **job final** >
| **resolve =** < **internal** | **external** | **both** >
| **select-statement condition =** < **none** | *string* >
| **state = none**
| **state = (** JRQ_REQ_STATE {**,** JRQ_REQ_STATE} **)**
| **state = all reachable**
| **state = default**
| **state = unreachable**
| **unresolved =** JRQ_UNRESOLVED

JRD_ITEM:

amount = *integer*
| **expired =** < **none** | *signed_period* >
| < **nokeep** | **keep** | **keep final** >

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

JRQ_UNRESOLVED:
defer
| **defer ignore**
| **error**
| **ignore**
| **suspend**

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

Beschreibung

Beschreibung

Dieses Kommando erzeugt oder (wahlweise) ändert Job, Batch oder Milestone Definitions.
Da Jobs, Batches und Milestones eine Menge gemeinsam haben, benutzen wir im Folgenden meistens den generellen Fachausdruck "Scheduling Entity", wann immer das Verhalten für alle drei Typen von Job Definitions das gleiche ist. Die Aus-

drücke ”Job”, ”Batch” und ”Milestone” werden für Scheduling Entities des entsprechenden Typs Job, Batch und Milestone benutzt. Wenn der **”or alter”** Modifikator benutzt wird, wird das Kommando, falls bereits ein Scheduling Entity mit dem gleichen Namen existiert, dieses entsprechend den spezifizierten Optionen ändern.

aging Das aging beschreibt die Geschwindigkeit mit der die Priorität hochgestuft wird.

children Der children Abschnitt eines Job Definition Statements definiert eine Liste von Child-Objekten, und wird benutzt um eine Hierarchie aufzubauen, die das Modellieren von komplexen Job-Strukturen ermöglicht.

Wann immer ein Scheduling Entity submitted wird, werden alle statischen Children rekursiv submitted.

Zusätzlich können Children, die nicht statisch sind, während der Ausführung durch einen running Job oder Trigger submitted werden.

Die Children werden durch eine Kommagetrennte Liste von Scheduling Entity Pfadnamen und zusätzliche Eigenschaften spezifiziert.

Die Eigenschaften der Child Definitions sind im Folgenden beschrieben:

ALIAS Mittels dieser Option kann die Implementierung des submitteten Jobs unabhängig von der Folder-Struktur gehalten werden und es wird unabhängig davon, ob Objekte innerhalb der Folder-Struktur verschoben werden, funktionieren.

Die alias einer Child Definition wird nur benutzt wenn Jobs dynamische Children submitten.

IGNORE DEPENDENCY Normalerweise werden Abhängigkeiten der Parents auf die Children vererbt. In wenigen besonderen Situationen kann dies jedoch unerwünscht sein. Mit der ignore dependency Option können solche Abhängigkeiten daher ignoriert werden.

MERGE MODE Ein einzelnes Scheduling Entity kann als Child von mehr als einem Parent Scheduling Entity benutzt werden. Wenn zwei oder mehr solcher Parents ein Teil eines Master Runs sind, werden dieselben Children mehrmals innerhalb dieses Master Runs instanziiert. Das ist nicht immer eine gewünschte Situation. Das Setzen des merge modes kontrolliert wie das System damit umgeht.

Die folgende Tabelle gibt einen Überblick über die möglichen merge modes und ihrer Bedeutung:

merge mode	Beschreibung
nomerge	Eine doppelte Instanz von dem Scheduling Entity wird erstellt. Das ist das Default-Verhalten.
merge global	Es wird keine doppelte Instanz erstellt. Ein Link wird zwischen dem Parent Submitted Entity und dem bereits existierendem Child Submitted Entity erstellt.
merge local	Wie merge global, aber nur Submitted Entities, die in einem einzelnen Submit erzeugt wurden, werden zusammengeführt.
failure	Der Submit, der versucht einen doppelten Submitted Entity zu erstellen schlägt fehl.

NICEVALUE Die nicevalue definiert ein Offset der Priorität, der für die Berechnung der Prioritäten des Childs und seiner Children herangezogen wird. Werte von -100 bis 100 sind zulässig.

PRIORITY Die spezifizierte priority in einer Child Definition überschreibt die Priorität von der Child Scheduling Entity Definition. Werte von 0 (hohe Priorität) bis 100 (niedrige Priorität) sind zugelassen.

TRANSLATION Das Setzen der Exit State translation für ein Child resultiert in einer Übersetzung des Exit States des Childs zu einem Exit State, welche in der resultierenden Exit States von dem Parent Submitted Entity gemerged wird.
Wenn keine Übersetzung gegeben ist, wird ein Child State, der nicht gleichzeitig ein gültiger Parent State ist, ignoriert.
Wenn eine Übersetzung gegeben ist, müssen alle Child States nach einem gültigen Parent State übersetzt werden.

SUSPEND Die childsuspend Klausel definiert, ob ein, im Kontext dieser Child Definition, neuer Submitted Job suspended wird.
Die folgende Tabelle zeigt die möglichen Werte und deren Bedeutung von der suspend Klausel an:

suspend clause	Beschreibung
suspend	Das Child wird unabhängig von dem Wert des suspend Flags, spezifiziert in der Child Scheduling Entities, suspended.
nosuspend	Das Child wird unabhängig von dem Wert des suspend Flags, spezifiziert in der Child Scheduling Entities Definition, nicht suspended.
childsuspend	Das Child wird suspended, wenn das suspended Flag in dem Child Scheduling Entity gesetzt ist.

Falls **suspend** spezifiziert wird, kann wahlweise auch eine resume Klausel mitgegeben werden, die ein automatisches Resume zum angegebenen Zeitpunkt, bzw. nach Ablauf des angegebenen Intervalls zur Folge hat.

Für teilqualifizierte Zeitpunkte wird die Submit-Zeit als Referenz genommen. So bedeutet etwa T16:00, dass der Job bei einer Submit-Zeit von 15:00 nach etwa einer Stunde anläuft. Liegt der Submit-Zeitpunkt jedoch nach 16:00, wird der Job bis zum nächsten Tag warten.

DYNAMIC CLAUSE Die dynamic Klausel definiert, ob das Child immer dann vom System automatisch submitted wird, wenn auch der Parent submitted wird.
Um imstande zu sein zur Laufzeit eines Jobs ein Child zu submitten, muss dieses Child als dynamic Child definiert sein.
Die folgende Tabelle zeigt die möglichen Werte in der dynamic Klausel und ihre Bedeutung an.

dynamic clause	Beschreibung
static	Das Child wird automatisch mit dem Parent submitted.
dynamic	Das Child wird nicht automatisch mit dem Parent submitted.

Milestones haben eine andere Semantik für ihre Children. Wann immer ein Scheduling Entity in einem Master Run dynamisch submitted wird, welches auch ein Child von einem Milestone im selben Master Run ist, wird der Submitted Scheduling Entity als Child an diesen Milestone gebunden. Somit kann ein Milestone nur dann final werden, wenn seine Abhängigkeiten erfüllt, und alle Children final sind. In anderen Worten, ein Milestone sammelt Child-Instanzen die bei anderen Submitted Entities dynamisch submitted werden und wartet auf die Beendigung von diesen Submitted Entities. Um dies korrekt funktionieren zu lassen, sollte eine Abhängigkeit, von dem Scheduling Entity der submitted, definiert sein.

dependency mode Der dependency mode definiert welche Required Submitted Entities einen final State erreichen müssen, bevor der abhängige Submitted Entity den 'Dependency Wait' System State verlassen kann.
Die folgende Tabelle zeigt die möglichen Dependency Modes und ihre Bedeutung.

dependency mode	Beschreibung
all	Der Submitted Entity verlässt den Dependency Wait State, nachdem alle Abhängigkeiten erfüllt sind.
any	Der Submitted Entity verlässt den Dependency Wait State, nachdem mindestens eine Abhängigkeit erfüllt ist.

environment Jeder Job muss definieren welches Environment für die Jobausführung gebraucht wird.
Der Job kann nur von Jobservern ausgeführt werden, die alle Static Resource-Anforderungen, die in der Environment Definition gelistet sind, erfüllen.
Die environment Option ist nur für Jobs gültig.

errlog Die errlog Option definiert die Datei in die Fehlerausgaben (stderr) des auszuführenden Prozesses geschrieben werden. Wenn der Dateiname relativ ist, wird die Datei relativ zum Working Directory des Jobs angelegt.
Diese Option ist nur für Jobs gültig.

footprint footprints sind Mengen von Anforderungen für System Resources. Wenn mehrere Jobs mit ähnlichen Anforderungen definiert werden, wird dies durch die Benutzung von footprints viel einfacher.
Der Job kann nur von Jobservern ausgeführt werden, die alle System Resource-Anforderungen erfüllen, die in der footprint Definition gelistet sind.
Die footprint Option gilt nur für Jobs.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

inherit grant Die inherit grant Klausel ermöglicht es zu definieren welche Privilegien über die Hierarchie geerbt werden sollen. Wird diese Klausel nicht spezifiziert, werden per Default alle Rechte geerbt.

kill program Diese Option wird benutzt um die Möglichkeit zu schaffen, laufende Prozesse aus dem Scheduling System heraus vorzeitig terminieren zu können. Üblicherweise beinhaltet das kill Programm die PId des running Jobs als Parameter (z.B. `kill -9 ${PID}`).
(Für Details über Kommandozeilen parsing, Ausführungen und Parameter Substitutionen siehe die ”run program” Option auf Seite 145.)

logfile Die logfile Option definiert die Datei in die der Standard Output (stdout) des auszuführenden Prozesses geschrieben wird. Wenn der Dateiname relativ ist, wird die Datei relativ zum Working Directory des Jobs angelegt.
Diese Option ist nur für Jobs gültig.

mapping Die mapping Option definiert das Exit State Mapping, das benutzt wird um Betriebssystem-Exit Codes eines ausführenden Programms in einen Exit State zu übersetzen. Wenn ein Job kein mapping hat, wird das Default Exit State Mapping des Exit State Profiles des Jobs, benutzt.
(Für eine ausführliche Beschreibung des exit state mappings siehe das ’create exit state mapping’ Kommando auf Seite 111.)

nicevalue Die nicevalue Option definiert eine Korrektur die für die Berechnung der Prioritäten des Jobs und seiner Children benutzt wird. Es sind Werte von -100 bis 100 erlaubt.

parameter Die parameter Section definiert welche Parameter und Input-Werte ein Job benötigt und wie der Job Daten mit anderen Jobs und dem Scheduling System auswechselt.
Die Parameter können in der Spezifikation des Run Programms, Rerun Programms, Kill Programms, Workdir, Logfile und error Logfile, sowie in Trigger und Dependency Conditions genutzt werden.
Zusätzlich kann ein Job zur Laufzeit Parameter abfragen oder setzen. Variablen die zur Laufzeit gesetzt und nicht von der Job Definition definiert wurden, sind nur für den Job selbst sichtbar und können nicht referenziert werden. Dasselbe gilt selbstverständlich auch für alle Variablen die als **local** definiert sind, sowie für die unten genannte Systemvariablen.
Gelegentlich gibt es jedoch die Notwendigkeit eine oder mehrere der (z.B.) Systemvariablen nach außen bekannt zu machen. Dies kann mittels eines kleinen Tricks gemacht werden. Enthählt der Wert eines Parameters eine Zeichenkette der Form `$irgendwas` (also ein $-Zeichen gefolgt von einem Namen), wird dies als der Name einer Variablen interpretiert und es wird versucht diese Variable *im Scope des Objektes der den ursprünglichen Wert des Parameters geliefert hat* aufzulösen.
So definiert z.B. ein Job `SYSTEM.A` eine Konstante namens `MYJOBNAME` mit als Inhalt `$JOBNAME`. Wird nun etwa über eine Reference die Konstante `MYJOBNAME` von außen angesprochen, bekommt man als Ergebnis den Wert `SYSTEM.A`.
Es sind für jeden Job immer eine Anzahl von Systemvariablen definiert. Diese werden vom System gesetzt und stehen dem Job für einen lesenden Zugriff zur Verfügung.
Diese System Variablen sind:

Name	Description
JOBID	Submitted Entity Id des Jobs
MASTERID	Submitted Entity Id des Master Jobs oder Batches
KEY	"Passwort" des Jobs für die Verbindung zum Scheduling System als Job mit "JOBID"
PID	Die Betriebssystem Prozess Id des Jobs. Dieser Parameter ist nur bei Kill Programs gesetzt
LOGFILE	Name des Logfiles (stdout)
ERRORLOG	Name des Error Logfiles (stderr)
SDMSHOST	Hostname des Scheduling Servers
SDMSPORT	Listen Port des Scheduling Servers
JOBNAME	Name des Jobs
JOBTAG	Childtag des Jobs ist gegeben wenn der Job dynamisch Submitted wird
TRIGGERNAME	Name des Triggers
TRIGGERTYPE	Typ des Triggers (JOB_DEFINITION oder NAMED_RESOURCE)

Continued on next page

Continued from previous page

Name	Description
TRIGGERBASE	Name des triggernden Objektes der den Trigger zum Feuern veranlasst
TRIGGERBASEID	Id der triggernden Objekt-Definition die den Trigger zum Feuern veranlasst
TRIGGERBASEJOBID	Id des triggernden Objektes der den Trigger zum Feuern veranlasst
TRIGGERORIGIN	Name des triggernden Objektes welches den Trigger definiert
TRIGGERORIGINID	Id der triggernden Objekt-Definition welche den Trigger definiert
TRIGGERORIGINJOBID	Id des triggernden Objektes welches den Trigger definiert
TRIGGERREASON	Name des triggernden Objektes welches den Trigger direkt oder indirekt feuert
TRIGGERREASONID	Id der triggernden Objekt-Definition die den Trigger direkt oder indirekt feuert
TRIGGERREASONJOBID	Id des triggernden Objektes welches den Trigger direkt oder indirekt feuert
TRIGGERSEQNO	Die Anzahl mal die der Trigger feuerte
TRIGGEROLDSTATE	Der alte Status des Objektes, verursacht durch den Trigger für Resource Trigger
TRIGGERNEWSTATE	(Neuer) Status des Objektes welches bewirkt, dass der Trigger ausgelöst wird
SUBMITTIME	Submit-Zeitpunkt
STARTTIME	Startzeitpunkt
EXPRUNTIME	Erwartete Laufzeit
JOBSTATE	Exit State des Jobs
MERGEDSTATE	Merged Exit State des Jobs
PARENTID	Id des Parent Jobs (Submission Baum)
STATE	Aktueller Status des Jobs (Running, Finished, ...)
ISRESTARTABLE	Ist der Job Restartable? 1 = ja, 0 = nein
SYNCTIME	Zeitpunkt des Übergangs nach Synchronize Wait
RESOURCETIME	Zeitpunkt des Übergangs nach Resource Wait
RUNNABLETIME	Zeitpunkt des Übergangs nach Runnable
FINISHTIME	Abschlusszeitpunkt
SYSDATE	Aktuelles Datum
SEID	Id der Job Definition
TRIGGERWARNING	Der Text der Warnung die diesen Trigger ausgelöst hat
LAST_WARNING	Der Text der zuletzt ausgegebenen Warnung. Wenn keine aktuelle Warnung vorliegt ist er leer

Continued on next page

Continued from previous page

Name	Description
RERUNSEQ	Die Anzahl der Reruns bis jetzt
SCOPENAME	Name des Scopes (Jobservers) in dem der Job läuft oder zuletzt lief

Tabelle 6.1: List of System Variables

Die TRIGGER... System Variablen sind nur gefüllt, wenn der Job von einem Trigger submitted wurde. (Für eine ausführlichere Beschreibung der TRIGGER... System Variablen, siehe das create trigger Statement auf Seite 164.)
Wenn ein Job ausgeführt wird, werden die Parameter, die in Kommandos, workdir und Datei Spezifikationen benutzt werden, aufgelöst, konform untenstehender Reihenfolge:

1. System Variable
2. Der Jobeigene Adressraum
3. Der Adressraum des Jobs und der submittenden Parents, von unten nach oben
4. Der Adressraum des Jobservers der den Job ausführt
5. Der Adressraum der Parent Scopes des Jobservers, der den Job ausführt, von unten nach oben
6. Die Job Definitions Parent Folders, von unten nach oben
7. Die Parent Folders des Parent Jobs, von unten nach oben

Wenn der Konfigurations-Parameter 'ParameterHandling' des Servers auf 'strict' (default) gesetzt ist, führt der Zugriff auf Variablen, die in der Job Definition nicht definiert sind, zu einer Fehlermeldung. Es sei denn es handelt sich um eine Systemvariable.
Wenn im Inhalt einer Variablen eine Referenz auf einen weiteren Parameter auftritt, wird dieser Parameter im Kontext des definierenden Jobs ausgewertet und ersetzt. Die verschiedenen Parametertypen und ihre Semantik werden im Folgenden beschrieben:

IMPORT import Parametertypen werden benutzt um die Daten eines Job Scheduling Environments einem anderen Job zu übergeben. Dieser Typ ist beinahe wie der Typ parameter, doch import Parametertypen können nicht wie Parameter übergeben werden, wenn ein Job submitted wird. import Parametertypen können einen Default-Wert haben, welcher benutzt wird wenn kein Wert vom Scheduling Environment erworben werden kann.

PARAMETER Parameter vom Typ parameter werden benutzt um Daten von einem Job Scheduling Environment an einen anderen Job zu übergeben. Dieser Typ ist

fast wie der Parametertyp import, doch Parametertypen parameter können als Parameter übergeben werden, wenn ein Job submitted wird. Parametertypen parameter können einen Default-Wert haben, welcher benutzt wird wenn kein Wert vom Scheduling Environment erworben werden kann.

REFERENCE reference Parametertypen werden normalerweise benutzt um Ergebnisse von einem Job zu einem anderen zu übergeben.
Zum Anlegen eines reference Parametertyps werden der vollqualifizierte Name der Job Definition, sowie der Name des referenzierenden Parameters benötigt. Beim Auflösen der reference wird das nächstliegende Submitted Entity gesucht, was der Job Definition der reference entspricht. Wenn diese Zuordnung nicht eindeutig gemacht werden kann, wird ein Fehler ausgegeben. Wird kein entsprechendes Submitted Entity gefunden, wird, soweit definiert, der Default-Wert zurückgegeben.

REFERENCE CHILD reference child Parametertypen werden genutzt um auf Parameter von direkten oder indirekten Children zu verweisen. Dies kann etwa zu Reporting-Zwecken nützlich sein. Die Definition eines reference child Parametertyps erfolgt mittels eines vollqualifizierten Job Definition-Namens, sowie des Namens des zu qualifizierenden Parameters. Bei der Auflösung des Parameters wird in der Submission-Hierarchie nach unten gesucht, anstelle nach oben wie bei den reference Parametertypen. Das Verhalten bei der Auflösung ist ansonsten identisch mit der Auflösung vom reference Parametertypen.

REFERENCE RESOURCE reference resource Parametertypen werden bezugnehmend auf Parameter von allokierten Resourcen benutzt.
Dieser Parametertyp benötigt einen vollqualifizierten Namen einer Named Resource mit einem additionalen Parameternamen um die Referenzvorgabe zu spezifizieren. Voraussetzung für die Nutzung eines reference resource Parametertyps ist, dass die Resource auch angefordert wird. Die Wertermittlung erfolgt im Kontext der allokierten Resource.

RESULT result Parametertypen können vom Job (mittels des API) einen Wert bekommen. Solange dieser Wert nicht gesetzt wurde, wird bei der Wertabfrage der optionale Default-Wert zurückgegeben.

CONSTANT constant Parametertypen sind Parameter mit einem bei der Definition festgelegten Wert. Dieser Wert kann sich also zur Laufzeit nicht ändern.

LOCAL Diese Variablen sind nur aus der Sicht des definierenden Jobs sichtbar.

priority Die priority eines Jobs bestimmt die Ausführungsreihenfolge von Jobs. Werte von 0 (hohe Priorität) bis 100 (niedrige Priorität) sind zugelassen. Die priority Option gilt nur für Jobs.

profile Das profile definiert den Exit State Profile der die gültigen Exit States des Scheduling Entity beschreibt.
(Für eine ausführliche Beschreibung der Exit State Profiles siehe das create exit state profile Kommando auf Seite 112.)

required Die required Section definiert die Abhängigkeiten von anderen submitteten Entities in einem Master Run, die erfüllt sein müssen bis das Submitted Entity fähig ist zu weiterzulaufen.
Ob alle Abhängigkeiten erfüllt sein müssen oder nur eine, wird durch den 'dependency mode' definiert.
Abhängigkeiten werden in einer kommagetrennten Liste von vollqualifizierten Namen von Scheduling Entities (inklusive Pfandnamen von Foldern) definiert.
Abhängigkeiten wirken nur zwischen Submitted Entities des Master Runs. Die Synchronizierung der Submitted Entities aus unterschiedlichen Master Runs muss mittels Synchronizing Resources implementiert werden.
Nach dem Anlegen der Submitted Entity-Instanzen der Submitted Scheduling Entity-Hierarchie, sucht das System die Abhängigkeiten wie folgt: Beginnend bei dem Parent des abhängigen Submitted Entity, wird in allen Children eine Instanz des Required Scheduling Entity gesucht, dabei wird natürlich der Ast mit dem abhängigen Submitted Entity ausgelassen. Wenn keine Instanz gefunden wird, geht die Suche bei den Submit Hierarchy Parents weiter, bis genau eine Instanz gefunden wird. Wenn keine Instanz gefunden wird, definiert die Eigenschaft 'unresolved' wie diese Situation vom System gehandhabt wird. Wird mehr als ein Submitted Entity gefunden, schlägt der Submit mit einem 'ambigous dependency resolution' Fehler fehl.
Während der Ausführung eines Master Runs kann ein Scheduling Entity einen 'unreachable' State bekommen, da die Abhängigkeiten nicht mehr erfüllt werden können. Dies kann passieren, wenn ein Required Scheduling Entity einen Final State erreicht, der nicht in der Liste von benötigten States für Dependencies eingetragen ist oder durch das Canceln eines Submitted Entities der von einem anderen Submitted Entity benötigt wird. Diese zwei Fälle werden unterschiedlich gehandhabt.
Wenn die unreachable Situation durch ein Submitted Entity verursacht wird, der mit einem nicht passenden Exit State beendet wird, ermittelt das System den Exit State Profile des abhängigen Submitted Entities und setzt den Exit State in den State der als 'unreachable' im Profile markiert ist.
Wenn keiner der Profile States als unreachable State markiert ist oder der unreachable Zustand durch das Canceln eines Submitted Entity verursacht wurde, wird das abhängige Submitted Entity in den Zustand 'unreachable' versetzt, welcher nur von einem Operator dadurch aufgelöst werden kann, dass er die Abhängigkeit ignoriert oder das abhängige Entity cancelt.
Alle direkten oder indirekten Children eines Jobs oder Batches erben alle Abhängigkeiten des Parents. Das heißt, dass kein Child eines Jobs oder Batches den Zustand dependency wait verlassen kann, solange der Parent selbst im Zustand 'dependency wait' ist. Children von Milestones erben die Abhängigkeiten vom Parent nicht.
Die Eigenschaften der dependency definitions werden im Folgenden beschrieben.

CONDITION Es ist möglich zu einer Dependency eine condition anzugeben. Die Dependency ist erst dann erfüllt, wenn die Evaluation der condition den Wahr-

heitswert TRUE ergibt. Wird keine condition spezifiziert, gilt die Bedingung immer als erfüllt.

DEPENDENCY NAME Optional kann beim Definieren einer Abhängigkeit ein Name für die Abhängigkeit spezifiziert werden. Children, sowohl direkte als auch indirekte, können auf den Namen Bezug nehmen, um diese Abhängigkeit zu ignorieren.

MODE Die mode Eigenschaft ist nur relevant, wenn das benötigte Scheduling Entity ein Job mit Children ist. In diesem Fall definiert das Dependency mode den Zeitpunkt, in dem die Abhängigkeit erfüllt wird.
Die folgende Tabelle zeigt die zwei möglichen Werte und ihre Bedeutung:

dependency mode	Beschreibung
all_final	Der benötigte Job und alle seine Children müssen einen final State erreicht haben.
job_final	Nur der benötigte Job selbst muss einen final State erreichen, der Zustand der Children ist irrelevant.

STATE Die state Eigenschaft einer Abhängigkeit definiert eine Liste von Final States, die das benötigte Scheduling Entity erreichen kann um die Abhängigkeit zu erfüllen.
Ohne diese Option ist die Abhängigkeit erfüllt, wenn das benötigte Scheduling Entity einen Final State erreicht.
Zusätzlich ist es möglich bei einem State eine condition anzugeben. Wenn eine condition spezifiziert wurde, gilt eine Abhängigkeit nur dann als erfüllt, wenn die condition erfüllt ist. Die syntaktischen Regeln für die Spezifikation von conditions sind dieselben die auch für Trigger gelten. (Siehe dazu das create trigger Statement auf Seite 164.)
Als weitere Möglichkeiten stehen einige implizite Definitionen zur Verfügung:

- **default** — Die Dependency ist erfüllt wenn der Vorgänger einer der States, die in seinem Profile als dependency Default gekennzeichnet sind, erreicht hat.
- **all reachable** — Die Dependency ist erfüllt wenn der Vorgänger einer der States, die nicht als unreachable gekennzeichnet sind, erreicht hat.
- **reachable** — Die Dependency ist erfüllt wenn der Vorgänger den als unreachable gekennzeichneten State erreicht hat.

UNRESOLVED Die unresolved Eigenschaft spezifiziert wie das System die Situation behandeln soll, wenn keine Submitted Entity-Instanz für ein benötigtes Scheduling Entity während einer Submit Operation gefunden wird.
In der folgenden Tabelle werden die möglichen Verhaltensweisen beschrieben:

unresolved	Beschreibung
error	Die Submit Operation schlägt fehl mit einer Fehlermeldung.
ignore	Die Abhängigkeit wird stillschweigend ignoriert.
suspend	Die Abhängigkeit wird ignoriert, aber der abhängige Submitted Entity wird in Suspended gesetzt und benötigt eine Benutzeraktion um fortzufahren.
defer	Diese Einstellung verspricht, dass der gesuchte Vorgänger später noch dynamisch submitted wird.
defer ignore	Diese Einstellung hofft, dass der gesuchte Vorgänger später noch dynamisch submitted wird. Ist dies nicht der Fall, wird die Abhängigkeit einfach ignoriert.

rerun program Wenn eine rerun program Kommandozeile für einen Job definiert wurde, wird diese Kommandozeile anstelle der run Kommandozeile bei einem Neustart des Jobs nach einem failure ausgeführt.
(Für Details in Bezug auf commandline parsing, Ausführungen und Parameter Substitution siehe die run program Option auf Seite 145.)

resource Die resource Section einer Job Definition definiert Resource-Anforderungen zusätzlich zu diesen Anforderungen die indirekt durch die environment und footprint Optionen definiert wurden.
Wenn hier dieselbe Named Resource wie im footprint angefordert wird, dann überschreibt die Anforderung in der Resource Section die Anforderung in footprint.
Da Evironments nur Named Resources mit der Usage static benötigen und footprints nur Named Resources mit der Usage system, ist die Resource Section einer Job Definition der einzige Platz um Resource-Anforderungen, für Named Resources mit der Usage synchronizing, zu definieren.
Resource-Anforderungen werden durch den vollqualifizierten Pfadnamen zu einer Named Resource mit den folgenden zusätzlichen Anforderungsoptionen definiert:

AMOUNT Die amount Option ist nur mit Anforderungen für Named Resources von der Art system oder synchronizing gültig. Der Amount in einer Resource-Anforderung drückt aus, wieviele Einheiten von der Required Resource belegt werden.

EXPIRED Die expired Option ist nur für synchronizing Resources mit einem definierten Resource State Profile gültig. Ist die expired Option spezifiziert, darf der Zeitpunkt, in der der Resource State von der Resource gesetzt wurde, nicht länger her sein als die expire Option angibt. Nun hat eine negative Expiration zur Folge, dass eine Resource mindestens so alt sein muss wie angegeben. Der Resource State kann nur von dem alter resource Kommando gesetzt werden (siehe Seite 81) oder automatisch beim Definieren eines Resource State Mappings, welcher den Exit State und Resource State in einem neuen Resource State umwandelt. Selbst wenn

in so einem Fall der neue Resource State dem alten Resource State gleicht, gilt der Resource State als gesetzt.

LOCKMODE Die lockmode Option in einer Resource-Anforderung ist nur für Synchronizing Resources gültig. Es sind fünf mögliche Lockmodes definiert:

Name	Bedeutung
X	exclusive lock
S	shared lock
SX	shared exclusive lock
SC	shared compatible lock
N	nolock

Wichtig ist die Kompatibilitäts 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

Das Ziel vom exclusive lock ist es, die Resource exclusiv zu haben und um fähig zu sein den Resource State und eventuell Parameter-Werte zu setzen. Ein häufiges Beispiel für das Benutzen des exclusive lock ist das neue Laden einer Datenbanktabelle.
Das Ziel vom shared lock ist es, anderen zu erlauben die Resource auf die gleiche Weise zu nutzen, aber Änderungen zu verhindern. Das gebräuchlichste Beispiel für das Benutzen der shared locks ist ein großer laufender Leseprozess einer Datenbanktabelle. Andere lesende Prozesse können einfach toleriert werden, aber es werden keine schreibenden Transaktionen erlaubt.
Das Ziel vom shared exclusive lock ist es ein zweites shared lock zu haben, welcher nicht kompatibel mit dem normalen shared lock ist. Wenn wir den normalen shared lock für große lesende Transaktionen benutzen, benutzen wir den shared exclusive lock für kleine schreibende Transaktionen. Kleine schreibende Transaktionen können problemlos parallel zueinander laufen, doch laufen sie parallel zu einer großen lesenden Transaktion führen sie fast sicher zu einem "Snapshot too old" oder zu anderen ähnlichen Problemen.
Das Ziel vom shared compatible lock ist es einen Typ von shared lock zu haben, welcher sowohl zu dem shared als auch zu dem shared exclusive lock kompatibel ist. Dieser lock-Typ ist für kurze lesende Transaktionen gedacht, welche keine

Konflikte mit kleinen schreibenden Transaktionen oder mit großen lesenden Transaktionen haben. Selbstverständlich haben kleine lesende Transaktionen keine Konflikte mit anderen kleinen lesenden Transaktionen. Das parallele Laufen von kleinen lesenden und großen schreibenden Transaktionen kann eventuell zu Problemen führen.
Das Ziel vom nolock ist es zu gewährleisten, dass die Resource existiert und alle anderen Eigenschaften der Resource den Bedarf decken. Die Resource ist nolocked und alles kann passieren, einschließlich Statusänderungen.

STATE Die state Option ist nur für Synchronizing Resources mit einem Resource State Profile gültig. Die Option wird benutzt um gültige Resource States für diesen Job zu spezifizieren. Eine Resource kann nur allokiert werden, wenn sie in einer von dem angeforderten States ist.

STATE MAPPING Die state mapping Option ist nur für Synchronizing Resources, die ein Resource State Profile spezifizieren und mit einem exclusive Lockmode angefordert werden gültig. Das Mapping definiert eine Funktion die Kombinationen von Exit States und Resource States in einem neuen Resource State abbilden. (Für ausführliche Informationen über Resource State Mappings siehe das create resource state mapping Statement auf Seite 155.)

KEEP Die keep Option in einer Resource-Anforderung definiert den Zeitpunkt zu dem die Resource freigegeben wird. Die keep Option ist sowohl für System als auch für Synchronizing Resources gültig. Es gibt drei mögliche Werte. Ihre Bedeutung wird in der folgenden Tabelle erklärt:

Wert	Bedeutung
nokeep	Die Resource wird am Jobende freigegeben. Dies ist das Default-Verhalten.
keep	Die Resource wird freigegeben, sobald der Job den final State erreicht hat.
keep final	Die Resource wird freigegeben, wenn der Job und alle seine Children final sind.

STICKY Die sticky Option ist nur für Synchronizing Resources gültig. Wenn sticky spezifiziert ist, wird die Resource in dem Master Batch (dies wird MASTER_RESERVATION genannt) allokiert, so lange innerhalb des Batches weitere Jobs existieren, die die Resource sticky benötigen. Der Amount und Lockmode für die Master Reservation wird aus allen sticky-Anforderungen aller Children abgeleitet. Der Amount ist das Maximum der Anforderungsmenge.
Der Lockmode ist abhängig von den weiteren Anforderungen. Im Normalfall ist der Lockmode exclusive, wenn mindestens zwei Jobs vorhanden sind, welche die Resource mit einem anderen Lockmode ungleich nolock anfordern. Die Ausnahme ist die Kombination von Shared und Shared Compatible Anforderungen. Diese resultiert in einen Lockmode Shared.

Es wird versucht alle Anforderungen aus der Master Reservation zu erfüllen.
Optional kann ein Name für die sticky Allocation vergeben werden. Es werden grundsätzlich jeweils nur die Anforderungen mit demselben Namen für das vorher beschriebene Verfahren berücksichtigt. Es kann daher mehrere MASTER_RESERVATIONS für einen Master Batch gleichzeitig geben. Mit Hilfe der Namen können innerhalb eines Ablaufs mehrere, von einander getrennte critical Regions realisiert werden.
Zusätzlich oder aber auch alternativ zum Namen kann ein Parent Job oder Batch spezifiziert werden. At Runtime wird über die Submission Hierarchie dann die zugehörige Instanz des Parents ermittelt. Die sticky Anforderung gilt nur ab dem Parent abwärts. Prinzipiell kann dies so interpretiert werden, als würde die Id des Parents einen Teil des Namens der sticky Anforderung darstellen. Mit diesem Mechanimus können auf einfache Weise getrennte critical Regions in dynamisch submittete Teilabläufe realisiert werden.

runtime Die runtime Option wird benutzt um die geschätzte Laufzeit eines Jobs zu definieren. Diese Zeit kann beim Auslösen von Trigger ausgewertet werden.

run program Die run programm Kommandozeile ist obligatorisch für Jobs, da sie das auszuführende Kommando für diesen Jobs spezifiziert.
Die Kommandozeile ist in einem Kommando und einer Liste mit Argumenten durch whitespace Characters getrennt. Das erste Element der Kommandozeile wird als Name des auszuführenden ablauffähigen Programms betrachtet und der Rest als Parameter des Programms.
Ob der Jobserver die Umgebungsvariable PATH beim Suchen nach der ausführbaren Datei benutzt, ist eine Eigenschaft des Jobservers.
System- und Jobparameter können mit $ Notation angesprochen werden.
Mittels Quoting können whitespace Characters, sowie $-Zeichen als Teil der Kommandozeile weitergeleitet werden. Das Quoting befolgt die Unix Bourne Shell-Regeln. Das bedeutet, doppelte Quotes verhindern, dass whitespace Characters als Trennzeichen interpretiert werden. Einzelne Quotes verhindern auch die Auflösung von Variablen. Es ist möglich Backticks für das Quoting zu verwenden. Teile der Kommandozeile die von Backticks gequoted wurden, werden als einzeln gequoted betrachtet, aber die Backticks bleiben ein Teil des Arguments. Andere Quotes werden entfernt. Sollen Backticks ohne ihre spezielle Bedeutung in der Kommandozeile vorkommen, müssen diese entwertet (mit '\') werden.
Beispiel:
Die run Kommandozeile `'sh -c "example.sh ${JOBID} \$HOME" '$SHELL''` wird das Programm `'sh'` mit dem Parameter `'-c'`, `'example.sh 4711 $HOME'` und `'$SHELL'` ausführen (in der Annahme, dass das Submitted Entity die Id 4711 hat).
Ist das auszuführenden Programm (erstes Element der Kommandozeile) eine gültige Ganzzahl, so wird die Kommonadozeile nicht vom Jobserver ausgeführt, son-

dern der Job so behandelt, als hätte er sich mit der Ganzzahl als Exit Code beendet. Dummy Jobs mit 'true' oder 'false' als Programm können nun als '0' statt 'true' bzw. '1' statt 'false' implementiert und so vom System wesentlich effizienter und schneller verarbeitet werden.
Sollte es tatsächlich einmal nötig sein ein Executable mit einer Zahl als Namen auszuführen, so kann dies durch einen Pfad Prefix ('./42' statt '42') erreicht werden.

suspend Die suspend Option definiert, ob ein Submitted Entity zur Submit-Zeit suspended wird.
Wenn die suspend Option spezifiziert wird, kann optional die resume Klausel verwendet werden. Damit kann ein automatischer resume zum angegebenen Zeitpunkt, bzw. nach der angegebenen Zeit, bewirkt werden.
Wenn der resume Zeitpunkt mittels des unvollständigen Datumsformates (siehe dazu auch Seite 21) spezifiziert wird, erfolgt der Resume zum ersten passenden Zeitpunkt nach dem Submit-Zeitpunkt.
Wenn etwa ein Submit um 16:00 erfolgt, und als Resume-Zeit ist T17:30 eingetragen, wird der Resume am gleichen Tag um 17:30 erfolgen. Wurde jedoch als Resume-Zeit T15:55 angegeben, wird der Job bis zum nächsten Tag 15:55 warten müssen.

timeout Die timeout Klausel einer Job Definition definiert die maximale Zeit die ein Job wartet, bis seine Resource-Abhängigkeiten erfüllt sind.
Wenn die Timeout-Bedingung erreicht ist, bekommt der Job den Exit State der in der timeout Klausel spezifiziert wurde. Dieser Exit State muss Bestandteil des Exit State Profiles sein.
Wenn keine timeout Option gegeben ist, wird der Job warten bis alle Anforderungen erfüllt sind.

type Die type Option spezifiziert den Typ des Scheduling Entity der erstellt oder geändert wird.

workdir Die workdir eines Jobs des Typs Scheduling Entity definiert das Verzeichnis in dem das run, rerun oder kill Programm ausgeführt wird.

master Die master Option definiert ob dieses Scheduling Entity submitted werden kann um einen Master Run zu erstellen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create named resource

Zweck

Das *create named resource* Statement wird eingesetzt um eine Klasse von Resources zu definieren. *Zweck*

Syntax

Die Syntax des *create named resource* Statements ist *Syntax*

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**

Beschreibung

Beschreibung Das *create named resource* Statement wird benutzt um Klassen von Resources zu definieren. Diese Klassen definieren den Namen, den Usage Type und wahlweise das benutzte Resource State Profile, sowie die Parameter.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

parameter Es kann nützlich sein Parameter in Zusammenhang mit der Belegung von Resources zu benutzen. Zum Beispiel könnte eine Resource wie RESOURCE.TEMP_SPACE einen Parameter namens LOCATION haben. Auf diese Weise kann ein Job so eine Resource benutzen und temporären Speicherplatz an einer Stelle, die von der aktuellen Instanz der Named Resource abhängt, allokieren.
Es existieren drei Typen von Parameter in einem Resource Kontext:

Typ	Bedeutung
constant	Dieser Parametertyp definiert den Wert, welcher für alle Resources konstant ist.
local constant	Dieser Parametertyp definiert einen nicht variablen Parameter, dessen Wert zwischen Instanzen derselben Named Resource abweichen kann.
parameter	Der Wert eines solchen Parameters kann durch Jobs, die diese Resource exclusiv gesperrt haben, geändert werden.

Tabelle 6.2: Named Resource Parametertypen

state profile Im Fall von Synchronizing Resources kann ein Resource state profile spezifiziert werden. Dieses erlaubt Jobs die Resource in einem bestimmten State anzufordern. Resource State Änderungen können genutzt werden um Trigger auszulösen.

usage Die usage der Named Resource kann eine der folgenden sein:

Usage	Bedeutung
category	Kategorien verhalten sich wie Folder und können benutzt werden um die Named Resources in eine übersichtliche Hierarchie einzuordnen.
static	Static Resources sind Resources die, falls angefordert, in dem Scope, in dem der Job läuft, vorhanden sein müssen, aber nicht verbraucht werden können. Mögliche Beispiele von Static Resources sind ein bestimmtes Betriebssystem, shared libraries für DBMS-Zugriffe oder das Vorhandensein eines C-Compilers.
system	System Resources sind Resources die gezählt werden können. Mögliche Beispiele sind die Anzahl Prozesse, die Menge der Zwischenspeicher oder die Verfügbarkeit von (einer Anzahl von) Bandlaufwerken.
synchronizing	Synchronizing Resources sind die komplexesten Resources und werden benutzt um den Mehrfachzugriff zu synchronizieren. Ein mögliches Beispiel ist eine Datenbank-Tabelle. Abhängig von der Art des Zugriffs (große lesende Transaktionen, große schreibende Transaktionen, mehrere kleine schreibende Transaktionen, ...) kann der Mehrfachzugriff toleriert werden oder auch nicht.
pool	Named Resources vom Typ Pool werden benutzt um sogenannte Resource Pools anzulegen. Diese Pools bieten die Möglichkeit die Verteilung von Amounts für System Resources zentral und flexibel zu regeln.

Tabelle 6.3: Named Resource Usage

factor Beim Anlegen einer Named Resource kann der factor, mit der die Mengenangaben in einer Resource-Anfrage multipliziert werden, spezifiziert werden. Dieser factor ist per Default 1. Bei jeder Instanz dieser Named Resource, jede Resource also, kann der factor überschrieben werden.

inherit grant Die inherit grant Klausel ermöglicht es zu definieren welche Privilegien über die Hierarchie geerbt werden sollen. Wird diese Klausel nicht spezifiziert, werden per Default alle Rechte geerbt.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create resource

Zweck

Zweck Das *create resource* Statement wird eingesetzt um eine Instanz von Named Resources innerhalb eines Scopes, Folders oder einer Job Definition zu erstellen.

Syntax

Syntax Die Syntax des *create resource* Statements ist

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** >

Beschreibung

Beschreibung Das *create resource* Statement wird benutzt um Named Resources innerhalb von Scopes, Folders oder Job Definitions zu instanziieren. Im letzteren Fall wird nur ein Template angelegt, welcher materialisiert wird, sobald der Job submitted wird und automatisch zerstört wird, sobald der Master Run final oder cancelled ist.
Ist die **or alter** Option spezifiziert, wird eine bereits existierende Resource geändert, andernfalls wird es als Fehler betrachtet wenn die Resource bereits existiert.

amount Die amount Klausel definiert den Available Amount von dieser Resource. Im Falle von statischen Resources, wird die Amount-Option nicht spezifiziert.

base multiplier Der base multiplier ist nur relevant wenn das Resource Tracing genutzt wird. Der base multiplier bestimmt den Multiplikationsfaktor von **trace base**. Wenn der trace base mit B und der trace multiplier mit M bezeichnet wird gilt, dass über die Zeiträume $B*M^0$, $B*M^1$ und $B*M^2$ die Durschschnittsbelegung

ermittelt wird. Der Default ist 600 (10 Minuten), sodass die Werte für B, $10B$ und $100B$ (in Minuten) ermittelt werden.

factor Um Resource-Anforderungen von Jobs von außen justieren zu können, wurde ein Resource factor eingeführt. Dieser kann sowohl an der Named Resource als auch individuell an der Resource gesetzt werden. Bei der Bestimmung, ob ein Job eine bestimmte Resource belegen kann wird durch den Vergleich der ursprünglichen Anforderung mit dem Requestable Amount bestimmt. Bei der tatsächlichen Belegung wird jedoch

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

hergenommen.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

online Die online Klausel definiert, ob die Resource online oder offline ist.
Wenn eine Resource offline ist, steht sie nicht zur Verfügung. Das bedeutet, dass ein Job der diese Resource benötigt nicht innerhalb dieses Scopes laufen kann. Doch da die Resource online gesetzt werden kann, wird der Job warten und nicht in einen Fehlerstatus versetzt werden.
Das ist ebenfalls für statische Resources gültig.

parameter Die parameter Klausel wird benutzt um die Werte vom Parameter, wie sie für die Named Resource definiert wurden, zu spezifizieren.
Parameter die als Konstante auf Named Resource Level deklariert sind, sind hier nicht erlaubt. Alle anderen Parameter können, müssen aber nicht, spezifiziert werden. Wenn kein Parameter und kein Default, für den Parameter auf dem Named Resource Level, spezifiziert ist, wird bei der Auflösung ein leerer String zurückgegeben.
Wird beim Ändern der Resource der Parametername = default spezifiziert, hat dies zur Folge, dass der Wert des Parameters den Default-Wert, so wie bei der Named Resource spezifiziert, erhält.
Wenn der Parameter auf der Named Resource-Ebene geändert wird, wird dies auf der Resource-Ebene für alle Parameter, die auf Default gesetzt wurde, sichtbar sein.
Es sind für jede Resource immer eine Anzahl von System Variablen definiert. Diese werden vom System gesetzt und stehen Jobs, welche die Resource allokieren über "RESOURCEREFERENCES" für einen lesenden Zugriff zur Verfügung.
Diese System Variable sind:

Name	Beschreibung
STATE	Der Resource State einer ”synchronizing” Resource mit einem Status Modell.
AMOUNT	Der insgesamt zur Verfügung stehende Resources-Betrag
FREE_AMOUNT	Der freie zur Verfügung stehende Resources-Betrag
REQUESTABLE_AMOUNT	Der von einem Job maximal allokierbare Betrag
REQUESTED_AMOUNT	Der vom Job angeforderte Betrag
TIMESTAMP	Touch Zeitstempel einer ”synchronizing” Resource mit einem Status Modell.

Tabelle 6.4: Liste der System Variablen

requestable amount Die requestable amount Klausel definiert den Amount von dieser Resource, die von einem einzelnen Job angefordert werden darf. Dieses kann von dem available Amount verschieden sein. Wenn die angeforderte Menge kleiner als der Amount ist, ist es sicher, dass ein Job nicht alle verfügbaren Resources allokieren kann. Wenn der requestable amount größer als der Amount ist, können Jobs mehr anfordern als an Amount zur Verfügung steht, ohne ein ”cannot run in any Scope” Fehler zu erzeugen.
Wenn der requestable amount nicht spezifiziert ist, gleicht er dem Amount.
Im Falle von statischen Resources wird die requestable amount Option nicht spezifiziert.

state Die state Klausel definiert den Status in dem die Resource ist.
Diese Option ist nur für synchronizing Resources mit einem Resource State Profile gültig.

tag Um die Trace Tabelle leichter auswerten zu können, können Resources und Pools nun mit einem tag versehen werden. Dieser tag soll innerhalb Resources und Pools eindeutig sein. (D.h. auch das Benutzen eines tags sowohl für eine Resource als auch für einen Pool ist verboten).

touch Die touch Klausel definiert den letzten Zeitpunkt in dem der Status der Resource (von einem Job) geändert wurde. Dieser Timestamp wird nicht gesetzt wenn ein Resource State manuell gesetzt wurde.
Diese Option ist nur für Synchronizing Resources, mit einem Resource State Profile, gültig.

trace base Falls der trace base **none** ist, ist Tracing ausgeschaltet. Ansonsten ist es die Basis für den Auswertungszeitraum.

trace interval Das trace interval ist die minimale Zeit zwischen dem Schreiben von Trace Records in Sekunden. Ist das trace interval **none**, ist Tracing ausgeschaltet.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create resource state definition

Zweck

Zweck Das *create resource state definition* Statement wird eingesetzt um einen symbolischen Namen für den Resource State zu erstellen.

Syntax

Syntax Die Syntax des *create resource state definition* Statements ist

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

Beschreibung

Beschreibung Das *create resource state definition* Statement wird benutzt um einen symbolischen Namen für einen Resource State zu definieren.
Das optionale Schlüsselwort **or alter** wird benutzt um das Auftreten von Fehlermeldungen und das Abbrechen der laufenden Transaktion, wenn eine Resource State Definition bereits existiert, zu verhindern. Ist es nicht spezifiziert, führt die Existenz einer Resource State Definition mit dem spezifizierten Namen zu einem Fehler.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel In diesen Beispielen werden eine Anzahl Namen für Resource States definiert.

```
create resource state definition leer;

create resource state definition gueltig;

create resource state definition ungueltig;

create resource state definition stadium1;

create resource state definition stadium2;

create resource state definition stadium3;
```

create resource state mapping

Zweck

Zweck

Das *create resource state mapping* Statement wird eingesetzt um ein Mapping zwischen den Exit States eines Jobs und dem resultierenden Resource State einer Resource zu definieren.

Syntax

Syntax

Die Syntax des *create resource state mapping* Statements ist

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

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

Beschreibung

Beschreibung

Das *create resource state mapping* Statement definiert das Mapping von Exit States in Kombination mit Resource States zu neuen Resource States.
Der erste State Name muss ein Exit State sein. Der zweite und dritte State jeweils ein Resource State. Terminiert ein Job mit dem genannte Exit State, wird der Status der Resources auf den neuen State gesetzt, wenn der aktuelle State mit dem erstgenannten State übereinstimmt. Wird als Anfangs-State **any** spezifiziert, wird jeder beliebige Resource State auf den neuen abgebildet. Wenn sowohl ein spezifisches Mapping als auch ein generelles Mapping spezifiziert sind, hat das spezifische Mapping die höchste Priorität.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel

Das nachfolgende Beispiel zeigt ein Mapping welches den State einer Resource bei jeder Anwendung in die nächste "PHASE" versetzt. Also `PHASE1` → `PHASE2` → `PHASE3` → `PHASE1` → ...

```
create or alter resource state mapping 'PHASE_MODEL'
with map = (
                'SUCCESS' maps 'PHASE1' to 'PHASE2',
                'SUCCESS' maps 'PHASE2' to 'PHASE3',
                'SUCCESS' maps 'PHASE3' to 'PHASE1'
);
```

create resource state profile

Zweck

Zweck Das *create resource state profile* Statement wird eingesetzt um eine Anzahl von gültigen Resource States zu erstellen.

Syntax

Syntax Die Syntax des *create resource state profile* Statements ist

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

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

Beschreibung

Beschreibung Das *create resource state profile* Statement wird benutzt um eine Menge von gültige Resource States für eine (Named) Resource zu definieren.

state Die state Klausel definiert welche Resource State Definitions innerhalb dieses Profils gültig sind.

initial state Die initial state Klausel bestimmt den initialen State einer Resource mit diesem Profile. Der initial state muss nicht in der Liste der States aus der state Klausel enthalten sein. Dies erlaubt das Anlegen einer Resource, ohne dass diese Resource sofort eine aktive Rolle im System spielt.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Beispiel

Beispiel In diesem Beispiel soll der Exit State leer ungültig werden.

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

create schedule

Zweck

Das *create schedule* Statement wird eingesetzt um einen aktiven Container für scheduled Events zu erstellen. *Zweck*

Syntax

Die Syntax des *create schedule* Statements ist *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**

Beschreibung

Mit dem *create schedule* Statement kann man mittels einfacher Definitionen komplexe Zeitpläne für Jobs und Batches erstellen. *Beschreibung*

active Die Angabe active bewirkt, dass der Schedule im Takt des spezifizierten Intervalls prinzipiell Ereignissen auslöst (vorausgesetzt, es sind welche definiert). Die Angabe inactive bewirkt dagegen, dass der Schedule im Takt des spezifizierten Intervalls gerade das Auslösen von Ereignisse verhindert. Durch die hierarchische Anordnung von Schedules können auf diese Weise etwa Ausnahmeperioden (wie Downtimes) gebildet werden.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

interval Das angegebene interval fungiert als Taktgeber für den Schedule. Wird ein Event mit dem Schedule verknüpft, wird dieser Event im Rhythmus des Intervalls ausgelöst.

inherit grant Die inherit grant Klausel ermöglicht es zu definieren welche Privilegien über die Hierarchie geerbt werden sollen. Wird diese Klausel nicht spezifiziert, werden per Default alle Rechte geerbt.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create scheduled event

Zweck

Der Zweck des *create scheduled event* Statements ist es eine Verbindung zwischen einem Event und einem Zeitplan zu definieren. *Zweck*

Syntax

Die Syntax des *create scheduled event* Statements ist *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*

Beschreibung

Scheduled Events stellen eine Verbindung zwischen Events (was ist zu tun) und Schedules (wann soll es gemacht werden) dar. *Beschreibung*

backlog handling Das backlog handling gibt an, wie mit Events die in Zeiten in denen der Server down war, aufgetreten sind, umgegangen werden soll. In der untenstehende Tabelle sind die drei Möglichkeiten aufgeführt:

Möglichkeit	Bedeutung
last	Nur der letzte Event wird ausgelöst
all	Alle zwischenzeitlich eingetretene Events werden ausgelöst
none	Keiner der zwischenzeitlich eingetretenen Events wird ausgelöst

Group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

active Scheduled Events können als active oder inactive gekennzeichnet werden. Wenn sie als active gekennzeichnet sind, werden Events ausgelöst. Dementsprechend werden Events nicht ausgelöst, wenn der scheduled Event als inactive gekennzeichnet ist. Mittels dieser Option können scheduled Events deaktiviert werden, ohne dass die Definition verloren geht.

suspend limit Das suspend limit gibt an, nach wieviel Verspätung ein zu einem Event gehörender Job automatisch mit der suspend-Option submitted wird. Eine Verspätung kann auftreten wenn, aus welchem Grund auch immer, der Scheduling Server für einige Zeit down ist. Nach dem Hochfahren des Servers werden die zwischenzeitliche Events, abhängig von der **backlog handling** Option, ausgelöst. Die Ausführungszeit ist damit später als die geplante Ausführungszeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

create scope

Zweck

Das *create scope* Statement wird eingesetzt um einen Scope innerhalb der Scope-Hierarchie zu erstellen. *Zweck*

Syntax

Die Syntax des *create scope* Statements ist *Syntax*

create [**or alter**] < **scope** *serverpath* | **jobserver** *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* >

Beschreibung

Beschreibung Das *create scope* Kommando wird benutzt um einen Scope oder Jobserver und seine Eigenschaften zu definieren.

Config Die config Option erlaubt die Konfiguration eines Jobservers mittels Key/-Value Pairs. Die Konfiguration wird nach unten vererbt, sodass generelle Konfigurations-Parameter bereits auf Scope-Ebene gesetzt werden können und damit für alle darunter angelegten Jobserver ihre Gültigkeit haben, sofern die Parameter auf unterer Ebene nicht überschrieben werden.
Bei der Anmeldung eines Jobservers wird diesem die Liste mit Konfigurations-Parametern übergeben.

Enable Die enable Option erlaubt dem Jobserver die Verbindung zu dem Repository Server. Diese Option ist für Scopes ungültig und wird, wenn sie spezifiziert wird, stillschweigend ignoriert.

Disable Die disable Option verbietet dem Jobserver die Verbindung zum Repository Server. Diese Option ist für Scopes ungültig und wird, wenn sie spezifiziert wird, stillschweigend ignoriert.

Group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

Node Der node gibt an, auf welchem Rechner der Jobserver läuft. Dieses Feld hat einen rein dokumentativen Charakter.

Parameter parameter können zur Kommunikation und Datenübermittlung zwischen Jobs verwendet werden. Sie stehen den Jobs und den Programmen, welche innerhalb der Jobs ausgeführt werden, zur Verfügung.
Die Parameter von Scopes und Jobservers können dazu benutzt werden Information über die Laufzeitumgebung eines Jobs zu spezifizieren.
Bei dem Dynamic Parameter wird der Parameter nach der Anmeldung des Jobservers aus seiner eigenen Prozessumgebung gefüllt. Beim Ändern der Prozessumgebung eines Jobservers muss auf diese Dynamic Variable geachtet werden, da ansonsten leicht Race Conditions entstehen.

Inherit grant Die inherit grant Klausel ermöglicht es zu definieren welche Privilegien über die Hierarchie geerbt werden sollen. Wird diese Klausel nicht spezifiziert, werden per Default alle Rechte geerbt.

Password Die password Option wird benutzt, um das Passwort des Jobservers zu setzen. Diese Option ist für Scopes ungültig und wird, wenn sie spezifiziert wird, stillschweigend ignoriert.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create trigger

Zweck

Zweck Das *create trigger* Statement wird eingesetzt um ein Objekt, welches einen Job dynamisch submitted, wenn eine bestimmte Kondition gegeben ist, zu erstellen.

Syntax

Syntax Die Syntax des *create trigger* Statements ist

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

CT_OBJECT:
job definition *folderpath*
| **named resource** *resourcepath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** < *folderpath* | *serverpath* >

WITHITEM:
< **active** | **inactive** >
| **check =** *period*
| **condition =** < **none** | *string* >
| < **nowarn** | **warn** >
| **event = (** CT_EVENT {**,** CT_EVENT} **)**
| **group event**
| **limit state =** < **none** | *statename* >
| **main none**
| **main** *folderpath*
| < **nomaster** | **master** >
| **parameter = none**
| **parameter = (** *identifier* **=** *expression* {**,** *identifier* **=** *expression*} **)**
| **parent none**
| **parent** *folderpath*
| **rerun**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **single event**
| **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:
after final
| **before final**
| **finish child**
| **immediate local**
| **immediate merge**
| **until final**
| **until finished**
| **warning**

Beschreibung

Beschreibung

Das *create trigger* Statement wird benutzt um ein Objekt zu erstellen, welches darauf wartet, dass ein bestimmtes Ereignis eintritt, nach welches, als Reaktion auf dieses Event, ein Job oder Batch submitted wird.
Ist die **or alter** Option spezifiziert, wird ein bereits existierender Trigger geändert, andernsfalls führt es zu Fehlern, wenn der Trigger bereits existiert.
Trigger können für Scheduling Entities oder Synchronizing (Named) Resources definiert werden. Im letzteren Fall wird der Trigger bei jedem Statuswechsel der Resource oder Instanz der Named Resource ausgewertet. Resource Trigger werden immer sogenannte Master Trigger sein. Dies bedeutet, sie submitten einen neuen Master Batch oder Job. Trigger in Scheduling Entities können Master Batches submitten, aber submitten standardmäßig neue Children. Diese Children müssen als (dynamische) Children des triggernden Scheduling Entities definiert sein.

active Die active Option ermöglicht das aktivieren beziehungsweise deaktivieren des Triggers. Damit kann das Triggern vorübergehend unterbunden werden, ohne den Trigger löschen zu müssen.

check Die check Option ist nur für **until final** und **until finished** Trigger gültig. Es definiert die Zeitintervalle zwischen zwei Auswertungen der Bedingungen.
Ohne Rücksicht auf die definierten Intervalle zu nehmen, wird die Condition auf jeden Fall überprüft, wenn ein Job beendet wird.

condition Die condition Option kann spezifiziert werden um eine zusätzliche Bedingung, welche geprüft werden muss bevor der Trigger feuert, zu definieren. Diese Bedingung ist ein boolscher Ausdruck und das Feuern findet statt wenn diese Bedingung true ergibt.
BOOLSCHE OPERATOREN Da diese Bedingung ein boolscher Ausdruck ist, können Boolsche Operatoren benutzt werden um mehrere komplexe Bedingungen zu erstellen. Diese Boolsche Operatoren sind:

- **not** (unärer Negations Operator)
- **and**
- **or**

Dabei gelten die üblichen Priotitätsregeln. Der not-Operator hat vor dem and-Operator Vorrang, welcher vor dem or-Operator Vorrang hat. Es ist erlaubt Klammern zu benutzen um eine Auswertungsreihenfolge zu erzwingen.
Ferner ist es erlaubt die boolesche Konstante **false** und **true** zu benutzen.
VERGLEICHS OPERATOREN Vergleiche können als Teil von booleschen Ausdrücken benutzt werden. Folgende Vergleichs Operatoren werden definiert.

- $==$ (gleich)
- $>=$ (größer oder gleich)
- $<=$ (kleiner oder gleich)
- $!=$ (ungleich)
- $>$ (größer als)
- $<$ (kleiner als)
- $=\sim$ (pattern matches)
- $!\sim$ (pattern matches nicht)

Alle Vergleichs Operatoren können mit Zeichenketten arbeiten. Die größer und kleiner als Operatoren benutzen im Fall von Zeichenketten den ascii-Wert von den Charaktern. Die matching-Operatoren arbeiten nicht mit Zahlen.
(Für eine vollständige Beschreibung der regulären Ausdrücke, welche von den match-Operatoren benutzt werden können, verweisen wir auf die originale Java Dokumentation der java.util.regexp.)

NUMERISCHE OPERATOREN Da nicht garantiert werden kann, dass Entscheidungen nicht nur vom Abgleich zweier Werte getroffen werden kann, ist es erlaubt (numerische) Operatoren zu benutzen. Die gültigen Operatoren sind:

- + (unärer Operator)
- − (unärer Negation Operator)
- ∗ (multiplications Operator)
- / (divisions Operator)
- % (modulo Operator)
- + (binärer Additions Operator)
- − (binärer Subtractions Operator)

LITERALE UND VARIABLEN Literals sind Zahlen (ganze Zahlen und Fließkomma Zahlen) oder Zeichenketten. Zeichenketten werden durch doppelte quotes (") abgegrenzt. Es ist möglich Variable zu benutzen, welche innerhalb des Kontext des triggernden Jobs oder Resource aufgelöst werden. Variable werden adressiert, indem man ihren Namen ein Dollarzeichen ($) voranstellt.
Bei der Auflösung der Variablen wird zuerst angenommen, dass es sich um eine Triggervariable handelt. Trifft dies nicht zu, wird die Variable als Jobvariable interpretiert. Diese Art der Auflösung ist zwar häufig richtig, aber leider nicht immer. Durch das Voranstellen von `job.`, `trigger.` oder `resource.`, sowie, im Kontext von Abhängigkeiten, `dependent.` und `required.`, kann explizit angegeben werden bei welchem Objekt nach der Variablen gesucht werden soll.
Normalerweise werden Variablen in Großbuchstaben angelegt. Dies kann durch Quoting der Namen verhindert werden. Allerdings wird beim Ansprechen der Variablen in Conditions der Name wieder in Großbuchstaben konvertiert. Um dies zu verhindern, muss der Name sowie ein eventuelles Prefix in geschweiften Klammern geschrieben werden.
Abhängig von dem Operator und dem ersten Operand, werden die Operands als Zeichenketten oder Zahlen interpretiert. Multiplikationen, Divisionen, Modulo und Subtractionen sowie die unären Abläufe sind nur für numerische Werte definiert. Der Additions-Operator in einem Zeichenketten-Kontext bewirkt das Aneinanderreihen der Operanden.

FUNKTIONEN Weil nicht alles einfach mittels (numerischer) Ausdrücke ausgedrückt werden kann, sind einige Funktionen dazugekommen. Zur Zeit sind folgende Funktionen definiert:

- **abs(***expression***)** – der absolute Wert des Ausdrucks wird zurückgegeben
- **int(***expression***)** – der ganzzahlige Wert des Ausdrucks wird zurückgegeben

- **lowercase(***expression***)** – das Ergebnis des Ausdrucks wird in Kleinbuchstaben umgewandelt und zurückgegeben
- **round(***expression***)** – der Ausdruck wird gerundet und zurückgegeben
- **str(***expression***)** – der Ausdruck wird als Zeichenkette zurückgegeben
- **substr(***source***,** *from* [**,** *until*]**)** – gibt einen Teil der Zeichenkette *source*, beginnend in der Position *from* bis zum Ende der Zeichenkette, oder wenn *until* spezifiziert ist, bis zur Position *until*, zurück.
- **trim(***expression***)** – der Ausdruck wird ohne Leerzeichen am Schluß zurückgegeben
- **uppercase(***expression***)** – das Ergebnis des Ausdrucks wird in Großbuchstaben umgewandelt und zurückgegeben

Funktionen können ohne Beschränkung ineinander verschachtelt werden.

BEISPIELE Zur Verdeutlichung folgen jetzt einige Statements die Conditions spezifizieren. Da Conditions nicht ausschließlich in der Definition von Triggers vorkommen, gibt es auch andere Beispiele. Die Syntax ist jedoch immer dieselbe.
Das erste Beispiel zeigt einen Trigger, der dann ausgelöst wird, wenn der Job auf `WARNING` oder `FAILURE` geht, aber schon Zeilen verarbeitet hat (`$NUM_ROWS > 0$`).

```
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';
```

Das zweite Beispiel zeigt ein Environment, welches fordert, dass der Wert der Resource Variablen `AVAILABLE` mit einem `T` anfangen soll (wie etwa TRUE, True, true oder Tricky).

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

Das dritte Beispiel zeigt dasselbe wie das zweite, nur ist der Parametername in mixed case definiert.

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

event Die event Option ist nur für Object Monitor Triggers relevant. Sie spezifiziert bei welcher Art von Ereignissen der Trigger gefeuert werden soll.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

main Die main Option ist nur für Object Monitor Triggers relevant. Wenn die main Option spezifiziert wird, wird der angegebene Job oder Batch beim Auslösen des Triggers submitted.
Falls keine parent Option spezifiziert wird, muss der eigentliche Triggerjob als dynamisches Child des main Jobs definiert sein. Für alle Object Instances die sich entsprechend der Triggerdefinition geändert haben (neu angelegt, geändert und/ oder gelöscht), wird eine Instanz des Triggerjobs als Child unter den main Job gehängt.
Falls die master Option nicht spezifiziert wurde, muss der main Job für sich als (dynamisches) Child des Watchers definiert sein. Ist die master Option spezifiziert, muss der main Job master submittable sein.

master Die master Option wird benutzt um festzulegen, ob der Job als master submitted wird oder nicht. Diese Option hat nur für Job Trigger eine Bedeutung, da Resource Trigger immer als master submitted werden.

parameter Die parameter option dient der Spezifikation von Parametern des zu triggernden Jobs.
Die Ausdrücke werden im Kontext des triggernden Objektes ausgewertet. Beim Submit des getriggerten Jobs werden die Ergebnisse dann als Wert des spezifizierten Parameters übergeben.
Die Syntax der Audrücke entspricht die der Conditions. Dabei sind selbstverständlich nicht nur Boolsche Ausdrücke, sondern auch numerische oder Zeichenkettenmanipulierende Ausdrücke erlaubt.
Die Operanden werden abhängig vom Operator numerisch oder als Zeichenketten interpretiert. Dabei ist im Zweifelsfall der implizite Datentyp des ersten Operandes maßgeblich.
Um dies zu verdeutlichen folgen einige Beispiele von Ausdrücken. Dazu nehmen wir an, dass der triggernde Job einige Parameter definiert hat:

```
$A = "5"
$B = "10"
$C = "hallo"
$D = "Welt"
```

Mit diesen Parametern gelten folgende Gleichungen (d.h. als Condition würden sie als Wahr evaluiert werden):

```
$A + $B == 15
"" + $A + $B == "510"
$A + "0" + $B == 15
$C + " " + $D == "hallo Welt"
$A + $C == "5hallo"
int("" + $A + $B) * 2 == 1020
$C + ($A + $B) == "hallo15"
```

Fehler liefern Ausdrücke wie

```
$C * $A
$C - $D
$B / ($A - 5)
```

Die erste beide Ausdrücke sind falsch, da $C nicht als numerischer Wert interpretiert werden kann. Im letzten Ausdruck wird versucht durch 0 zu teilen.
Läuft die Evaluierung eines Ausdrucks in einen Fehler, schlägt auch das Triggern fehl.

parent Die parent Option ist nur für Object Monitor Triggers relevant. Sie kann auch nur in Kombination mit der main Option spezifiziert werden.
Wenn sie spezifiziert ist, hat es zur Folge, dass der entsprechende Job (oder Batch) innerhalb des über den main Job submitteden Baumes gesucht wird und die Triggerjobs unter den Parent gehängt werden.

rerun Die rerun Option kann nur reagieren auf restartable States und hat einen automatischen rerun zur Folge. In vielen Fällen wird es sinnvoll sein auch die suspend/resume Optionen zu spezifizieren um eine gewisse Zeit zwischen den Wiederholungen zu lassen.
Entweder die submit Option, oder die rerun Option muss spezifiziert werden.

resume Die resume Option kann zusammen mit der suspend Option verwendet werden um eine verzögerte Ausführung zu bewirken. Es gibt dabei zwei Möglichkeiten. Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
Bei der unvollständigen Angabe eines Zeitpunktes, wie etwa `T16:00`, wird der Zeitpunkt der Trigger-Auslösung als Referenzzeit hergenommen.

state Die state Option ist für alle, außer **until final** und **until finished**, Trigger gültig. Im Falle von Trigger auf Jobs, kann eine Liste von Exit States spezifiziert werden. Wenn der Job, in dem der Trigger definiert ist, einen Exit State, welcher in der Trigger Definition gelistet ist, erreicht, dann feuert der Trigger (es sei denn, es ist eine Konditon spezifiziert die mit **false** bewertet wird).
Im Falle von einem Trigger auf einer (Named) Resource, kann eine Liste von State-Wechseln spezifiziert werden. Auf diesem Weg kann jeder State-Wechsel explizit adressiert werden. Es ist möglich zu triggern, wenn ein Status verlassen wird, durch die Benutztung des Schlüsselwortes **any** auf der rechten Seite. Es ist jederzeit möglich auf das Erreichen eines bestimmten States, durch das Spezifizieren von **any** auf der linken Seite, zu triggern. Um auf jeden Status-Wechsel zu triggern, wird die State Option ausgelassen.

submit Die submit Option definiert welcher Job oder Batch submitted wird, wenn der Trigger feuert.
Entweder die submit Option, oder die rerun Option muss spezifiziert werden.

submitcount Die submitcount Option ist nur bei Trigger auf Jobs zulässig. Es definiert die Anzahl mal die ein Trigger feuern kann. Wenn diese Option nicht spezifiziert ist, wird ein submitcount von 1 genommen.
Wenn ein submitcount von 0 spezifiziert ist, wird der submitcount auf dem Serverparameter TriggerSoftLimit (dessen default-Wert 50 ist) gesetzt. Handelt es sich jedoch um einen rerun Trigger, bedeutet ein submitcount von 0, dass es kein Limit bezüglich der Anzahl restart Versuche gibt.
Ist ein submitcount, der größer als der TriggerSoftLimit ist, spezifiziert, wird der submitcount auf dem Serverparameter TriggerHardLimit (dessen Wert per Default 100 ist) beschränkt. Dies wird gemacht, um Endlosschleifen zu vermeiden. Der TriggerHardLimit kann in der Serverkonfiguration auf $2^{31} - 1$ gesetzt werden um die obere Schranke praktisch zu eliminieren.

suspend Die suspend Option wird benutzt um den Job oder Batch suspended zu submitten. Diese Option ist für alle Triggertypen gültig.

type Es gibt mehrere typs von Triggern in Jobs. Die wichtigste Differenz zwischen ihnen ist die Zeit, zu der sie überprüft werden. Die folgende Tabelle zeigt eine Liste von allen typs mit einer kurzen Beschreibung ihres Verhaltens.
Es muss hervorgehoben werden, dass die type Option nicht für (named) resource trigger gültig ist. Im Falle von recource Triggers wird der Trigger bei einer Statusänderung immer sofort ausgelöst.

Feld	Beschreibung
Typ	Prüfungszeit
after final	Nur nachdem ein final state erreicht wurde, wird überprüft, ob der definierte Trigger feuern muss. Wenn der Trigger kein Master Trigger ist, wird der neu submittete Job den gleichen Parent wie der triggernde Job haben. Eine besondere Situation tritt ein, wenn der triggernde Job seinen eigenen submit triggert. In diesem Fall ersetzt der neu submittete Job den triggernden Job. Da dieser Austausch stattfindet, bevor die Abhängigkeit überprüft wurde, warten alle abhängigen Jobs, bis der neu submittete Job final ist.
before final	Direkt bevor ein final state erreicht wird, wird überprüft, ob der definierte Trigger feuern soll. Das ist die letzte Möglichkeit neue Children zu submitten. Wird das gemacht, dann wird der Job oder Batch zu diesem Zeitpunkt keinen final state erreichen.
finish child	Ein finisch child Trigger prüft jedesmal, wenn ein direktes oder indirektes Child sich beendet, ob gefeuert werden soll.
immediate local	Der immediate local Trigger prüft, ob er feuern muss, wenn ein Job terminiert. Nur der exit state des Jobs wird berücksichtigt.
immediate merge	Der immediate merge Trigger prüft, ob er feuern muss, sobald der merged exit state wechselt.
until final	Der until final Trigger prüft periodisch, ob er feuern muss. Diese Prüfung startet sobald ein Job oder Batch submitted wurde und hält nicht an, bevor er final ist. Der until final Trigger benötigt zwingend eine Condition. Diese Condition wird zumindest einmal geprüft. Diese Prüfung findet statt wenn der Job oder Batch in den State finished wechselt.

Fortsetzung auf der nächsten Seite

Fortsetzung von der vorherigen Seite

Feld	Beschreibung
until finished	Der until finished Trigger ähnelt dem until final Trigger. Der einzige Unterschied ist, dass der until finished Trigger die Prüfung beendet, sobald der Job finished ist. Der until finished Trigger benötigt zwingend eine Condition. Diese Condition wird zumindest einmal geprüft. Diese Prüfung findet statt wenn der Job oder Batch in den State finished wechselt.

Tabelle 6.5: Beschreibung der verschiedenen Trigger Typen

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

create user

Zweck

Zweck Das *create user* Statement wird eingesetzt um ein Wertepaar zu erstellen, welches benutzt werden kann um sich beim Server zu authentifizieren.

Syntax

Syntax Die Syntax des *create user* Statements ist

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

WITHITEM:
connect type = < **plain** | **ssl** | **ssl authenticated** >
| **default group =** *groupname*
| < **enable** | **disable** >
| **equivalent = none**
| **equivalent = (** < *username* | *serverpath* > {**,** < *username* | *serverpath* >} **)**
| **group = (** *groupname* {**,** *groupname*} **)**
| **parameter = none**
| **parameter = (** PARAMETERSPEC {**,** PARAMETERSPEC} **)**
| **password =** *string*
| **rawpassword =** *string* [**salt =** *string*]

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

Beschreibung

Beschreibung Das *create user* Statement wird benutzt um einen Benutzer anzulegen. Ist "**or alter**" spezifiziert, wird ein bereits existierender Benutzer geändert. Andernfalls führt ein bereits existierender Benutzer zu einem Fehler.
Die *default group* Klausel wird benutzt um die Default Group zu spezifizieren.

connect type Die connect type Klausel spezifiziert welche Art von Verbindung vom Benutzer zumindest genutzt werden muss.

Wert	Bedeutung
plain	Jede Art von Verbindung ist erlaubt
ssl	Nur SSL-Verbindungen sind erlaubt
ssl authenticated	Nur SSL-Verbindungen mit Client Authentifizierung sind erlaubt

default group Die default group Klausel definiert die Gruppe welche als Eigentümer für alle seine Objekte, die der User erstellt, benutzt wird, wenn keine explizite Gruppe bei der Objekterstellung spezifiziert wurde.

enable Die enable Option erlaubt dem Benutzer die Verbindung zu dem Repository Server.

disable Die disable Option verbietet dem Benutzer die Verbindung zum Repository Server.

group Die group Klausel wird benutzt um die Gruppen, zu der der User gehört, zu spezifizieren. Jeder User ist ein Mitglied der Systemgruppe PUBLIC.

password Die password Option wird benutzt, um das Passwort des Users zu setzen.

rawpassword Das rawpassword wird benutzt um das Passwort des Users zu setzen, das nötig ist, um mit dem Repository Server verbunden zu werden. Das rawpassword ist das bereits verschlüsselte Passwort. Die rawpassword Option ist für create user Kommandos vom dump Kommando erzeugt worden.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

Kapitel 7

deregister commands

deregister

Zweck

Zweck Das *deregister* Statement wird eingesetzt um den Server zu benachrichtigen, das der Jobserver keine Jobs mehr ausführt. Siehe das *register* Statement auf Seite 280.

Syntax

Syntax Die Syntax des *deregister* Statements ist

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

Beschreibung

Beschreibung Das *deregister* Statement wird genutzt um den Server über einen, mehr oder weniger, permanenten Ausfall eines Jobservers zu informieren.
Diese Nachricht hat verschiedene Serveraktionen zur Folge. Als Erstes werden alle running Jobs des Jobservers, d.h. Jobs im Status **started**, **running**, **to_kill** und **killed**, auf den Status **broken_finished** gesetzt. Jobs im Status **starting** werden wieder auf **runnable** gesetzt. Dann wird der Jobserver aus der Liste der Jobserver, die Jobs verarbeiten können, entfernt, sodass dieser Jobserver im Folgenden auch keine Jobs mehr zugeteilt bekommt. Als Nebeneffekt werden Jobs, die aufgrund Resource-Anforderungen nur auf diesem Jobserver laufen können, in den Status **error**, mit der Meldung "Cannot run in any scope because of resource shortage", versetzt. Als Letztes wird ein komplettes Reschedule ausgeführt um eine Neuverteilung von Jobs auf Jobservern herbeizuführen.
Durch erneutes Registrieren (siehe *register* Statement auf Seite 280), wird der Jobserver erneut in die Liste der Jobs-verarbeitenden Jobserver eingetragen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 8

disconnect commands

disconnect

Zweck

Zweck Das *disconnect* Statement wird eingesetzt um die Serververbindung zu beenden.

Syntax

Syntax Die Syntax des *disconnect* Statements ist

disconnect

Beschreibung

Beschreibung Mit dem *disconnect* Statement kann die Verbindung zum Server beendet werden.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 9

drop commands

drop comment

Zweck

Zweck Das *drop comment* Statement wird eingesetzt um einen Kommentar zu einem Objekt zu löschen.

Syntax

Syntax Die Syntax des *drop comment* Statements ist

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*
| **exit state translation** *transname*
| **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* | **jobserver** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

PARAM_LOC:

folder *folderpath*
| **job definition** *folderpath*
| **named resource** *resourcepath*
| < **scope** *serverpath* | **jobserver** *serverpath* >

TRIGGEROBJECT:

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

Beschreibung

Beschreibung

Das *drop comment* Statement löscht den zu dem angegebenen Objekt vorhandenen Kommentar. Wenn das Schlüsselwort **existing** nicht spezifiziert wird, wird das Fehlen eines Kommentars als Fehler betrachtet.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop environment

Zweck

Zweck Das *drop environment* Statement wird eingesetzt um das spezifizierte Environment zu löschen.

Syntax

Syntax Die Syntax des *drop environment* Statements ist

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

Beschreibung

Beschreibung Das *drop environment* Statement wird benutzt um eine Definition von einem Environment zu löschen. Es führt zu einem Fehler, wenn Jobs immer noch dieses Environment benutzen. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Environment nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop event

Zweck

Das *drop event* Statement wird eingesetzt um das spezifizierte Event zu löschen. *Zweck*

Syntax

Die Syntax des *drop event* Statements ist *Syntax*

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

Beschreibung

Das *drop event* Statement wird benutzt um eine Definition eines Events zu löschen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Event nicht existiert. *Beschreibung*
Ein Event kann nicht gelöscht werden wenn es dazugehörige scheduled Events gibt.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

drop exit state definition

Zweck

Zweck Das *drop exit state definition* Statement wird eingesetzt um die spezifizierte Exit State Definition zu löschen.

Syntax

Syntax Die Syntax des *drop exit state definition* Statements ist

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

Beschreibung

Beschreibung Das *drop exit state definition* Statement wird benutzt um eine Exit State Definition zu löschen. Es wird als Fehler betrachtet wenn Exit State Profiles diese Exit State Definition immer noch benutzen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn die spezifizierte Exit State Definition nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop exit state mapping

Zweck

Zweck

Das *drop exist state mapping* Statement wird eingesetzt um das spezifizierte Mapping zu löschen.

Syntax

Syntax

Die Syntax des *drop exit state mapping* Statements ist

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

Beschreibung

Beschreibung

Das *drop exit state mapping* Statement wird benutzt um Exit State Mappings zu löschen. Es wird als Fehler betrachtet, wenn Jobs oder Exit State Profiles immer noch dieses Exit State Mapping benutzen. Wenn das Schlüsselwort **existing** benutzt wird, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Exit State Mapping nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop exit state profile

Zweck

Zweck Das *drop exit state profile* Statement wird eingesetzt um das spezifizierte Profile zu löschen.

Syntax

Syntax Die Syntax des *drop exit state profile* Statements ist

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

Beschreibung

Beschreibung Das *drop exit state profile* Statement wird benutzt um eine Definition eines Exit State Profiles zu löschen. Es wird als Fehler betrachtet, wenn Jobs immer noch dieses Exit State Profile benutzen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Exit State Profile nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop exit state translation

Zweck

Zweck

Das *drop exit state translation* Statement wird eingesetzt um die spezifizierte Exit State Translation zu löschen.

Syntax

Syntax

Die Syntax des *drop exit state translation* Statements ist

drop [**existing**] **exit state translation** *transname*

Beschreibung

Beschreibung

Das *drop exit state translation* Statement wird benutzt um Exit State Translations zu löschen. Es wird als Fehler betrachtet, wenn die Translation immer noch in Parent-Child-Beziehungen verwendet wird. Wenn das **existing** Schlüsselwort in Benutzung ist, wird es *nicht* als Fehler betrachtet, wenn die spezifizierte Exit State Translation nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop folder

Zweck

Zweck Das *drop folder* Statement wird eingesetzt um einen Folder und seinen Inhalt aus dem System zu entfernen.

Syntax

Syntax Die Syntax des *drop folder* Statements ist

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

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

Beschreibung

Beschreibung Mit dem *drop folder* Statement werden Folder und deren Inhalte aus dem System gelöscht. Es gibt zwei Optionen:

Cascade Mit der cascade Option werden Folder, Job Definitions und Subfolder gelöscht, allerdings nur wenn nicht an die Job Definitions referenziert wird z. B. als required Job.

Force Mit der force Option werden Referenzen an Job Definitions ebenfalls entfernt. Force impliziert cascade.
Folder können nicht gelöscht werden wenn sie nicht leer sind, es sei denn cascade oder force wird spezifiziert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop footprint

Zweck

Zweck

Das *drop footprint* Statement wird eingesetzt um den spezifizierten Footprint zu löschen.

Syntax

Syntax

Die Syntax des *drop footprint* Statements ist

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

Beschreibung

Beschreibung

Das *drop footprint* Statement wird benutzt um Footprints und Resource-Anforderungen zu löschen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Footprint nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop group

Zweck

Zweck Das *drop group* Statement wird eingesetzt um eine Gruppe aus dem System zu entfernen.

Syntax

Syntax Die Syntax des *drop group* Statements ist

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

Beschreibung

Beschreibung Das *drop group* Statement wird benutzt um eine Gruppe zu löschen. Wenn dort noch Gruppenmitglieder existieren, wird die Mitgliedschaft automatisch beendet. Es wird als Fehler betrachtet, wenn die Gruppe immer noch Eigentümer eines Objektes ist.
Es ist nicht möglich eine Gruppe, die als Default-Gruppe eines Users definiert ist, zu löschen.
Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler angesehen, wenn die spezifizierte Gruppe nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop interval

Zweck

Zweck

Das *drop interval* Statement wird eingesetzt um das spezifizierte Intervall zu löschen.

Syntax

Syntax

Die Syntax des *drop interval* Statements ist

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

Beschreibung

Beschreibung

Das *drop interval* Statement wird benutzt um Intervalle zu löschen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Intervall nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop job definition

Zweck

Zweck Das *drop job definition* Statement wird eingesetzt um das spezifizierte Scheduling Entity Objekt zu löschen.

Syntax

Syntax Die Syntax des *drop job definition* Statements ist

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

Beschreibung

Beschreibung Das *drop job definition* Statement löscht die angegebene Job Definition.
Falls eine Job Definition referenziert wird (etwa als Required Job), kann sie nicht gelöscht werden, es sei denn man spezifiziert die force Option. Wird die force Option genutzt, werden alle Referenzen auf eine Job Definition ebenfalls gelöscht.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop named resource

Zweck

Das *drop named resource* Statement wird eingesetzt um eine Klasse von Resources zu löschen. *Zweck*

Syntax

Die Syntax des *drop named resource* Statements ist *Syntax*

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

Beschreibung

Das *drop named resource* Statement wird benutzt um Named Resources zu löschen. Es wird als Fehler betrachtet, wenn die Named Resource immer noch in Scopes, Job Definitions und/oder Folder instanziiert ist und die **cascade** Option nicht spezifiziert ist. *Beschreibung*
Auf der anderen Seite werden Scope Resources, sowie Folder und Job Definition Resources gelöscht wenn die **cascade** Option spezifiziert ist.
Wenn irgendwelche Anforderungen für die Named Resource, die gelöscht werden sollen, existieren, schlägt das Statement fehl.
Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler angesehen wenn die spezifizierte Named Resource nicht existiert.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

drop resource

Zweck

Zweck Das *drop resource* Statement wird eingesetzt um die Instanz einer Named Resource von einem Scope, Folder oder Job Definition zu löschen.

Syntax

Syntax Die Syntax des *drop resource* Statements ist

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

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

Beschreibung

Beschreibung Das *drop resource* Statement wird benutzt um eine Resource zu löschen. Es wird als Fehler betrachtet wenn die Resource immer noch von Running Jobs allokiert ist. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler angesehen, wenn die spezifizierte Resource nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop resource state definition

Zweck

Zweck

Das *drop resource state definition* Statement wird eingesetzt um die Definition zu löschen.

Syntax

Syntax

Die Syntax des *drop resource state definition* Statements ist

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

Beschreibung

Beschreibung

Das *drop resource state definition* Statement wird benutzt um Resource State Definitions zu löschen. Es wird als Fehler betrachtet, wenn Resource State Profiles immer noch diese Resource State Definition verwenden. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler angesehen, wenn die spezifizierte Resource State Definition nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop resource state mapping

Zweck

Zweck Das *drop resource state mapping* Statement wird eingesetzt um ein Mapping zu löschen.

Syntax

Syntax Die Syntax des *drop resource state mapping* Statements ist

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

Beschreibung

Beschreibung Das *drop resource state mapping* Statement wird benutzt um ein Resource State Mapping zu löschen. Es wird als Fehler betrachtet, wenn Job Definitions dieses Resource State Mapping benutzen. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler gesehen, wenn das Resource State Mapping nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop resource state profile

Zweck

Zweck

Das *drop resource state profile* Statement wird eingesetzt um ein Resource State Profile zu löschen.

Syntax

Syntax

Die Syntax des *drop resource state profile* Statements ist

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

Beschreibung

Beschreibung

Das *drop resource state profile* Statement wird benutzt um die Definition eines Resource State Profiles zu löschen. Es wird als Fehler betrachtet, wenn Named Resources immer noch dieses Resource State Profile benutzen. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Resource State Profile nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop schedule

Zweck

Zweck Das *drop schedule* Statement wird eingesetzt um den spezifizierten Zeitplan zu löschen.

Syntax

Syntax Die Syntax des *drop schedule* Statements ist

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

Beschreibung

Beschreibung Das *drop schedule* Statement wird benutzt um Schedules zu löschen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte Schedule nicht existiert.
Wenn ein Schedule ein zugehöriges scheduled Event hat, kann es *nicht* gelöscht werden. Löschen ist ebenfalls dann unmöglich, wenn Child Objects vorhanden sind.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop scheduled event

Zweck

Zweck

Der Zweck des *drop scheduled event* Statements ist es das spezifizierte scheduled Event zu löschen.

Syntax

Syntax

Die Syntax des *drop scheduled event* Statements ist

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

Beschreibung

Beschreibung

Das *drop interval* Statement wird benutzt um scheduled Events zu löschen. Wird das **existing** Schlüsselwort benutzt, wird es *nicht* als Fehler betrachtet, wenn das spezifizierte scheduled Event nicht existiert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop scope

Zweck

Zweck Das *drop scope* Statement wird eingesetzt um einen Scope und seinen Inhalt aus der Scope-Hierarchie zu entfernen.

Syntax

Syntax Die Syntax des *drop scope* Statements ist

drop [**existing**] < **scope** *serverpath* | **jobserver** *serverpath* > [**cascade**]

Beschreibung

Beschreibung Dieses Statement ist synonym zu dem *drop jobserver* Statement. Die **cascade** Option bewirkt, dass der Scope samt Inhalt gelöscht wird.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

drop trigger

Zweck

Das *drop trigger* Statement wird eingesetzt um den spezifizierten Trigger zu löschen. *Zweck*

Syntax

Die Syntax des *drop trigger* Statements ist *Syntax*

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

TRIGGEROBJECT:

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

Beschreibung

Das *drop trigger* Statement wird benutzt um Trigger zu löschen. *Beschreibung*
Ist das **existing** Schlüsselwort in Benutzung, wird es *nicht* als Fehler angesehen, wenn der spezifizierte Trigger nicht existiert.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

drop user

Zweck

Zweck Das *drop user* Statement wird eingesetzt um den Benutzer aus dem System zu entfernen.

Syntax

Syntax Die Syntax des *drop user* Statements ist

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

Beschreibung

Beschreibung Das *drop user* Statement wird benutzt um einen User logisch zu löschen. Wenn das **existing** Schlüsselwort benutzt wird, wird es *nicht* als Fehler angesehen, wenn der spezifizierte User nicht existiert.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 10

finish commands

finish job

Zweck

Zweck Der Zweck des *finish job command* ist es den Server über den Ablauf eines Jobs zu informieren.

Syntax

Syntax Die Syntax des *finish job* Statements ist

finish job *jobid*
with exit code = *signed_integer*

finish job
with exit code = *signed_integer*

Beschreibung

Beschreibung Das *finish job* Kommando wird vom Jobserver genutzt um den Exit Code eines Prozesses dem Server zu melden. Im Rahmen von Reparaturarbeiten kann es auch für einen Administrator notwendig sein auf diese Weise das Terminieren eines Jobs dem Server mitzuteilen. Jobs können sich selbst fertig melden. Dazu verbinden sie sich mit dem Server und benutzten die zweite Form des Statements.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 11

get commands

get parameter

Zweck

Zweck Das *get parameter* Statement wird eingesetzt um den Wert des spezifizierten Parameters innerhalb des Kontext des anfordernden Jobs, entsprechend seiner Spezifikation, zu bekommen.

Syntax

Syntax Die Syntax des *get parameter* Statements ist

get parameter *parametername* [< **strict** | **warn** | **liberal** >]

get parameter of *jobid parametername* [< **strict** | **warn** | **liberal** >]

Beschreibung

Beschreibung Das *get parameter* Statement wird eingesetzt um den Wert des spezifierten Parameters innerhalb des Kontextes eines Jobs zu bekommen.
Die Zusatzoption hat dabei folgende Bedeutung:

Option	Bedeutung
strict	Der Server liefert einen Fehler, wenn der gefragte Parameter nicht explizit in der Job Definition deklariert ist
warn	Es wird eine Meldung ins Logfile des Server geschrieben, wenn versucht wird den Wert eines nicht deklarierten Parameters zu ermitteln.
liberal	Der Versuch nicht deklarierte Parameter abzufragen wird stillschweigend erlaubt.

Das Default-Verhalten hängt von der Serverkonfiguration ab.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
VALUE	Wert des angeforderten Parameters

Tabelle 11.1: Beschreibung der Output-Struktur des get parameter Statements

get submittag

Zweck

Das *get submittag* Statement wird eingesetzt um eine eindeutige Identifikation vom Server zu bekommen. Diese Identifikation kann benutzt werden, um *race conditions* zwischen Frontend und Backend während des Submits zu verhindern. *Zweck*

Syntax

Die Syntax des *get submittag* Statements ist *Syntax*

get submittag

Beschreibung

Mit dem *get submittag* Statement bekommt man eine Identifikation vom Server. Damit verhindert man Race Conditions zwischen Frontend und Backend wenn Jobs submitted werden. *Beschreibung*
Eine solche Situation entsteht, wenn aufgrund eines Fehlers die Rückmeldung des Submits nicht ins Frontend eintrifft. Durch Benutzung eines Submit Tags kann das Frontend gefahrlos einen zweiten Versuch starten. Der Server erkennt, ob der betreffende Job bereits submitted wurde und antwortet dementsprechend. Ein doppeltes Submitten des Jobs wird damit zuverlässig verhindert.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
VALUE	Das angeforderte Submit Tag

Tabelle 11.2: Beschreibung der Output-Struktur des get submittag Statements

Kapitel 12

kill commands

kill session

Zweck

Zweck Das Ziel der *kill session* ist, die spezifizierte Session zu beenden.

Syntax

Syntax Die Syntax des *kill session* Statements ist

kill session *sid*

Beschreibung

Beschreibung Mittels *list session* Kommandos kann eine Liste von aktiven Sessions gezeigt werden. Die angezeigte Session-Id kann benutzt werden um mittels des *kill session* Kommandos die betreffende Session zu terminieren. Nur Administratoren, das heißt Mitglieder der Gruppe ADMIN, dürfen dieses Statement benutzen. Es ist *nicht* möglich die eigene Session zu terminieren.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 13

link commands

link resource

Zweck

Zweck Der Zweck des *link resource* Statements ist es in ein Scope eine Referenz auf eine Resource aus einem anderen Scope zu bekommen.

Syntax

Syntax Die Syntax des *link resource* Statements ist

> **link resource** *resourcepath* **in** *serverpath* **to** < **scope** *serverpath* |
> **jobserver** *serverpath* > [**force**]

Beschreibung

Beschreibung Mit dem *link resource* Statement ist es möglich in einem Scope Resources eines anderen Scopes sichtbar und benutzbar zu machen. Dies ist dann notwendig, wenn ein logischer Prozess Ressourcen aus mehr als einem Scope benötigt. Dies ist etwa bei Prozessen die mit einem Datenbanksystem kommunizieren durchaus der Fall.
Aus Sicht des Systems kann ein Resource Link kaum von der Resource auf die verwiesen wird unterschieden werden. Alle Operationen, wie etwa Allokieren, Sperren, das Lesen oder Setzen von Variablen erfolgen auf die Basis-Resource. Damit verhält sich der Link als wäre es die Basis-Resource. Der einzige Unterschied liegt in der Ansicht der Allocations. Bei der Basis-Resource werden alle Allocations gezeigt. Bei einem Link werden nur die Allocations gezeigt die über den Link erfolgen.
Es ist ebenfalls möglich Links auf Links anzulegen.
Mit Hilfe der **force** Option wird ein bereits vorhandener Link überschrieben. Eine bereits vorhandenen Resource wird gelöscht und der Link wird angelegt. Natürlich sind diese Operationen nur dann möglich, wenn die Resource bzw. Link nicht in Benutzung ist, wenn also keine Allocations oder Reservierungen vorliegen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 14

list commands

list calendar

Zweck

Zweck Der Zweck des *list calendar* Statements ist es eine Übersicht über die anstehenden Jobs zu bekommen.

Syntax

Syntax Die Syntax des *list calendar* Statements ist

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*

Beschreibung

Beschreibung Mit dem *list calendar* Statement bekommt man eine Liste aller Kalendereinträge. Die Liste ist sortiert nach Startdatum des Ausführungsobjektes.
Wenn eine Periode spezifiziert wird, werden auch solche Objekte angezeigt, deren Starzeit plus Expected Final Time in der selektierten Periode hineinfällt.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SE_NAME	Name des Scheduling Entities
SE_TYPE	Type des Scheduling Entities (Job oder Batch)
SE_ID	Id des Scheduling Entities
SE_OWNER	Eigentümer des Scheduling Entities
SE_PRIVS	Privilegien auf das Scheduling Entity
SCE_NAME	Name des Schedules
SCE_ACTIVE	Flag, ob Schedule active ist
EVT_NAME	Name des Events
STARTTIME	Startzeitpunkt
EXPECTED_FINAL_TIME	Erwarteter Endzeitpunkt
TIME_ZONE	Die Zeitzone in der die Zeiten ausgegeben werden

Tabelle 14.1: Beschreibung der Output-Struktur des list calendar Statements

list dependency definition

Zweck

Zweck Das *list dependency definition* Statement wird eingesetzt um eine Liste aller Abhängigkeiten einer Job Definition zu erstellen.

Syntax

Syntax Die Syntax des *list dependency definition* Statements ist

list dependency definition *folderpath*

Beschreibung

Beschreibung Mit dem *list dependency definition* Statement bekommt man eine Liste aller Abhängigkeiten einer Job Definition.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SE_DEPENDENT_PATH	Der Folder in dem das abhängige Scheduling Entity liegt
DEPENDENT_NAME	Der Name des abhängigen Scheduling Entities
SE_REQUIRED_PATH	Der Folder in dem das benötigte Scheduling Entity liegt
REQUIRED_NAME	Der Name des benötigten Scheduling Entities
NAME	Der Name des Objektes
UNRESOLVED_HANDLING	Im Feld Unresolved Handling wird beschrieben was zu tun ist, wenn eine abhängige Objektinstanz im aktuellen Master Batch nicht vorhanden ist. Es gibt folgende Optionen: Ignore, Error und Suspend

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
STATE_SELECTION	Die State Selection gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigte Exit States expliziert aufgeführt sein.
ALL_FINALS	Dieses Feld gibt an, ob die Abhängigkeit bereits beim Erreichen eines final States erfüllt ist (true) oder die benötigten States explizit aufgeführt sind (false).
CONDITION	Im Feld Condition wird die Bedingung, die erfüllt werden muss, eingetragen.
STATES	Hier steht die Liste von allen gültigen Exit States, welche das benötigte Objekt haben muss, damit die Abhängigkeit erfüllt wird und der abhängige Job starten kann.
RESOLVE_MODE	Der Resolve Mode definiert den Kontext in dem die Dependency aufgelöst werden soll. Mögliche Werte sind: Wert — Bedeutung **internal** — Die Dependency wird innerhalb des Masters aufgelöst. **both** — Die Dependency wird, wenn möglich innerhalb des Masters aufgelöst. Gelingt dies nicht, wird außerhalb des Masters gesucht. **external** — Die Dependency wird außerhalb des Masters aufgelöst.
EXPIRED_AMOUNT	Bei der Auflösung eines external Dependencies spielt es eine Rolle wann der benötigte Job oder Batch aktiv war. Die expired amount definiert wie viele Zeiteinheiten dies in die Vergangenheit liegen darf.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPIRED_BASE	Die exipired base definiert die Zeiteinheit für die expired amount.
SELECT_CONDITION	Die select condition definiert eine Bedingung die erfüllt sein muss, damit ein Job oder Batch als required Job betrachtet werden kann.

Tabelle 14.2: Beschreibung der Output-Struktur des list dependency definition Statements

list dependency hierarchy

Zweck

Das *list dependency hierarchy* Statement wird eingesetzt um eine Liste aller Abhängigkeiten eines Submitted Entities zu bekommen. *Zweck*

Syntax

Die Syntax des *list dependency hierarchy* Statements ist *Syntax*

list dependency hierarchy *jobid* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Mit dem *list dependency hierarchy* Statement bekommt man eine Liste aller Abhängigkeiten eines Submitted Entities. *Beschreibung*

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Id der Dependency Instance
DD_ID	Die Id der Dependency Definition
DEPENDENT_ID	Hierbei handelt es sich um die Id des abhängigen Jobs.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
DEPENDENT_NAME	Das ist der vollqualifizierte Name des abhängigen Jobs.
REQUIRED_ID	Hierbei handelt es sich um die Id des benötigten Jobs.
REQUIRED_NAME	Hierbei handelt es sich um den vollqualifizierten Namen des benötigten Jobs.
DEP_STATE	Hierbei handelt es sich um den aktuellen Status der Abhängigkeitsbeziehung. Es gibt folgende Ausprägungen: OPEN, FULLFILLED und FAILED.
DEPENDENCY_PATH	Hierbei handelt es sich um eine durch ';' getrennte Liste von Job Hierarchies (Parent-Child-Beziehungen). Jede Job Hierarchie ist eine Liste von Pfadnamen jeweils durch ':' getrennt.
SE_DEPENDENT_ID	Die Id des abhängigen Scheduling Entities
SE_DEPENDENT_NAME	Der vollqualifizierte Name des abhängigen Scheduling Entities
SE_REQUIRED_ID	Die Id des benötigten Scheduling Entities
SE_REQUIRED_NAME	Der vollqualifizierte Name des benötigten Scheduling Entities
DD_NAME	Name der Dependency Definition
UNRESOLVED_HANDLING	Im Feld Unresolved Handling wird beschrieben was zu tun ist, wenn eine abhängige Objektinstanz im aktuellen Master Batch nicht vorhanden ist. Es gibt folgende Optionen: Ignore, Error und Suspend
MODE	Gibt den aktuell verwendeten Dependency Mode an (ALL_FINAL oder JOB_FINAL)
STATE_SELECTION	Die State Selection gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigte Exit States expliziert aufgeführt sein.
MASTER_ID	Hierbei handelt es sich um die Id des Master Jobs, welcher submitted wurde, um dieses Laufzeitobjekt zu erzeugen.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
SE_TYPE	Hierbei handelt es sich um den Typ des Scheduling Entities (Job, Batch oder Milestone).
PARENT_ID	Hierbei handelt es sich um die Id des Parent-Laufzeitobjektes welche den aktuellen Job submitted hat. Hat der Job kein Parent, wird NONE angezeigt.
PARENT_NAME	Hierbei handelt es sich um den vollqualifizierten Namen des Parent-Laufzeitobjektes welche den aktuellen Job submitted hat.
OWNER	Die Gruppe die Eigentümer des Objektes ist
SCOPE	Hierbei handelt es sich um den vollqualifizierten Namen des Jobservers auf dem der Job gestartet wurde. Wurde der Job noch nicht gestartet, wird 'null' angezeigt.
EXIT_CODE	Beim Exit Code handelt es sich um den Exit-Wert, den das Run Program bei der Beendigung des Prozesses hatte.
PID	Das ist die Prozess Id des Job Executors.
EXTPID	Das ist die Id des Prozesses der ausgeführt wird.
JOB_STATE	Der aktuelle Job State
JOB_ESD	Hierbei handelt es sich um den Exit State des Jobs. Wenn der Job noch nicht beendet ist, wird 'null' angezeigt.
FINAL_ESD	Hierbei handelt es sich um den Merged Exit State.
JOB_IS_FINAL	Gibt an, ob der Job final ist (true) oder nicht (false).
CNT_REQUIRED	Die Anzahl der Jobs von denen der aktuelle Job abhängt, wenn der Job im Status dependency_wait ist.
CNT_RESTARTABLE	Die Anzahl der Children im Status restartable
CNT_SUBMITTED	Die Anzahl der Children im Status submitted
CNT_DEPENDENCY_WAIT	Die Anzahl der Children im Status dependency_wait
CNT_RESOURCE_WAIT	Die Anzahl der Children im Status resource_wait
CNT_RUNNABLE	Die Anzahl der Children im Status runnable
CNT_STARTING	Die Anzahl der Children im Status starting

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CNT_STARTED	Die Anzahl der Children im Status started
CNT_RUNNING	Die Anzahl der Children im Status running
CNT_TO_KILL	Die Anzahl der Children im Status to_kill
CNT_KILLED	Die Anzahl der Children im Status killed
CNT_CANCELLED	Die Anzahl der Children im Status cancelled
CNT_FINAL	Die Anzahl der Children im Status final
CNT_BROKEN_ACTIVE	Die Anzahl der Children im Status broken_active
CNT_BROKEN_FINISHED	Die Anzahl der Children im Status broken_finished
CNT_ERROR	Die Anzahl der Children im Status error
CNT_SYNCHRONIZE_WAIT	Die Anzahl der Children im Status synchronize_wait
CNT_FINISHED	Die Anzahl der Children im Status finished
SUBMIT_TS	Der Zeitpunkt zu dem der Job submitted wurde.
SYNC_TS	Der Zeitpunkt zu dem der Job in den Status synchronize_wait gewechselt ist
RESOURCE_TS	Der Zeitpunkt zu dem der Job in den Status resource_wait gewechselt ist
RUNNABLE_TS	Der Zeitpunkt an dem der Job den Status runnable erreicht hat
START_TS	Der Zeitpunkt zu dem der Job vom Jobserver als gestarted gemeldet wurde
FINSH_TS	Der Zeitpunkt zu der der Job in den State finished übergegangen ist
FINAL_TS	Der Zeitpunkt zu der der Job in den State final übergegangen ist
ERROR_MSG	Die Fehlermeldung, die beim Erreichen des Status Error ausgegeben wurde
DEPENDENT_ID_ORIG	Die Id des Objektes das die Abhängigkeit definiert hat
DEPENDENCY_OPERATION	Die Dependency Operation gibt an, ob alle Abhängigkeiten (all) oder nur eine einzige Abhängigkeit erfüllt sein muss.
CHILD_TAG	Marker zur Unterscheidung mehrerer dynamic submitted Children

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CHILDREN	Die Anzahl der Children des Jobs
REQUIRED	Die Anzahl der abhängigen Jobs
DD_STATES	Eine kommagetrennte Liste der benötigten Exit States
IS_SUSPENDED	Dieses Feld gibt an, ob der Job suspended (true) ist oder nicht (false).
PARENT_SUSPENDED	Dieses Feld gibt an, ob der Job über einen seiner Parents suspended ist (true) oder nicht (false).
CNT_UNREACHABLE	Die Anzahl Children deren Dependencies nicht erfüllt werden können
DEPENDENT_PATH_ORIG	Der vollqualifizierte Name des Objektes das die Abhängigkeit definiert hat
IGNORE	Ignore gibt an, ob diese Abhängigkeit ignoriert wird (true) oder nicht (false)
RESOLVE_MODE	Der Resolve Mode definiert den Kontext in dem die Dependency aufgelöst werden soll. Mögliche Werte sind: Wert — Bedeutung **internal** — Die Dependency wird innerhalb des Masters aufgelöst. **both** — Die Dependency wird, wenn möglich innerhalb des Masters aufgelöst. Gelingt dies nicht, wird außerhalb des Masters gesucht. **external** — Die Dependency wird außerhalb des Masters aufgelöst.
EXPIRED_AMOUNT	Bei der Auflösung eines external Dependencies spielt es eine Rolle wann der benötigte Job oder Batch aktiv war. Die expired amount definiert wie viele Zeiteinheiten dies in die Vergangenheit liegen darf.
EXPIRED_BASE	Die exipired base definiert die Zeiteinheit für die expired amount.
SELECT_CONDITION	Die select condition definiert eine Bedingung die erfüllt sein muss, damit ein Job oder Batch als required Job betrachtet werden kann.

Tabelle 14.3: Beschreibung der Output-Struktur des list dependency hierarchy Statements

list environment

Zweck

Zweck Das *list environment* Statement wird eingesetzt um eine Liste von definierten Environments zu bekommen.

Syntax

Syntax Die Syntax des *list environment* Statements ist

list environment

Beschreibung

Beschreibung Das *list environment* Statement wird benutzt um eine Liste von definierten Environments, die für den Benutzer sichtbar sind, zu bekommen.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Environments
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.4: Beschreibung der Output-Struktur des list environment Statements

list event

Zweck

Das *list event* Statement wird eingesetzt um eine Liste von allen definierten Events zu bekommen. *Zweck*

Syntax

Die Syntax des *list event* Statements ist *Syntax*

list event

Beschreibung

Das *list event* Statement erzeugt eine Liste aller definierten Events. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
SCHEDULING_ENTITY	Batch oder Job der submitted wird wenn dieses Event eintritt
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.5: Beschreibung der Output-Struktur des list event Statements

list exit state definition

Zweck

Zweck Das *list exit state definition* Statement wird eingesetzt um eine Liste aller definierten Exit States zu bekommen.

Syntax

Syntax Die Syntax des *list exit state definition* Statements ist

list exit state definition

Beschreibung

Beschreibung Mit dem *list exit state definition* Statement bekommt man eine Liste aller Exit States.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.6: Beschreibung der Output-Struktur des list exit state definition Statements

list exit state mapping

Zweck

Das *list exit state mapping* Statement wird eingesetzt um eine Liste aller definerten Mappings zu bekommen. *Zweck*

Syntax

Die Syntax des *list exit state mapping* Statements ist *Syntax*

list exit state mapping

Beschreibung

Mit dem *list exit state mapping* Statement bekommt man eine Liste aller definierten Mappings. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.7: Beschreibung der Output-Struktur des list exit state mapping Statements

list exit state profile

Zweck

Zweck Das *list exit state profile* Statement wird eingesetzt um eine Liste von allen definierten Exit State Profiles zu bekommen.

Syntax

Syntax Die Syntax des *list exit state profile* Statements ist

list exit state profile

Beschreibung

Beschreibung Mit dem *list exit state profile* Statement bekommt man eine Liste aller definierten Exit State Profiles.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
DEFAULT_ESM_NAME	Default Exit State Mapping ist aktiv wenn der Job selbst nichts anderes angibt.
IS_VALID	Flag Anzeige über die Gültigkeit dieses Exit State Profiles
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.8: Beschreibung der Output-Struktur des list exit state profile Statements

list exit state translation

Zweck

Zweck

Der Zweck des *list exit state translation* Statements ist es eine Liste von allen definierten Exit State Translations zu bekommen.

Syntax

Syntax

Die Syntax des *list exit state translation* Statements ist

list exit state translation

Beschreibung

Beschreibung

Mit dem *list exit state translation* Statement bekommt man eine Liste aller definierten Exit State Translations.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.9: Beschreibung der Output-Struktur des list exit state translation Statements

list folder

Zweck

Zweck Das *list folder* Statement wird eingesetzt um eine Liste mit allen Folder die im System definiert sind zu bekommen.

Syntax

Syntax Die Syntax des *list folder* Statements ist

list [**condensed**] **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*} **)**

Beschreibung

Beschreibung Mit dem *list folder* Statement bekommt man eine Liste des angegebenen Folders mit allen direkten Child Folders.

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

filter Die Child Folders können nach ihrem Namen selektiert werden. Für die genaue Syntax der Regular Expressions sei auf die offizielle Java Dokumentation verwiesen. Die verschiedenen Bedingungen können mittels **and** und **or** miteinander kombiniert werden. Dabei gilt die übliche Auswertungsreihenfolge der Operatoren (**and** vor **or**).

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Type gibt die Art des Objektes an. Es gibt folgende Optionen: Batch, Milestone, Job und Folder.
RUN_PROGRAM	Im Feld Run_Program kann eine Kommandozeile angegeben werden, die das Skript oder Programm startet.
RERUN_PROGRAM	Das Feld Rerun_Program gibt das Kommando an, welches bei einer wiederholten Ausführung des Jobs nach einem Fehlerzustand (rerun) ausgeführt werden soll.
KILL_PROGRAM	Das Kill_Program bestimmt welches Programm ausgeführt werden soll, um einen aktuell laufenden Job zu beenden.
WORKDIR	Hierbei handelt es sich um das Working Directory des aktuellen Jobs.
LOGFILE	Das Feld Logfile gibt an in welche Datei alle normalen Ausgaben des Run_Programs ausgegeben werden sollen. Unter normalen Ausgaben sind alle Ausgaben gemeint, die den normalen Ausgabekanal (STDOUT unter UNIX) benutzen.
TRUNC_LOG	Gibt an, ob das Logfile erneuert werden soll oder nicht
ERRLOGFILE	Das Feld Errorlogfile gibt an, in welcher Datei alle Fehlerausgaben des Run_Programs ausgegeben werden sollen.
TRUNC_ERRLOG	Gibt an, ob das Error Logfile erneuert werden soll oder nicht

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPECTED_RUNTIME	Die Expected_Runtime beschreibt die erwartete Zeit die ein Job für seine Ausführung benötigt.
EXPECTED_FINALTIME	Die Expected Finaltime beschreibt die erwartete Zeit die ein Job oder Batch samt Children für seine Ausführung benötigt.
GET_EXPECTED_RUNTIME	Hierbei handelt es sich um ein reserviertes Feld für zukünftige Erweiterungen.
PRIORITY	Das Feld Priority gibt an mit welcher Dringlichkeit ein Prozess, falls er gestartet werden soll, vom Scheduling System berücksichtigt wird.
MIN_PRIORITY	Minimale effektive Priorität die durch das natürliche Altern erreicht werden kann
AGING_AMOUNT	Die Anzahl Zeiteinheiten nach der die effektive Priorität um 1 erhöht wird
AGING_BASE	Die Zeiteinheit die für das Alterungsintervall genutzt wird
SUBMIT_SUSPENDED	Flag das angibt, ob das Objekt nach dem Submit suspended werden soll.
MASTER_SUBMITTABLE	Der Job der durch den Trigger gestartet wird, wird als eigener Master Job submitted und hat keinen Einfluss auf den aktuellen Master Job-Lauf des triggernden Jobs.
SAME_NODE	Obsolete
GANG_SCHEDULE	Obsolete
DEPENDENCY_MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
ESP_NAME	Hierbei handelt es sich um den Namen des Exit State Profiles.
ESM_NAME	Hierbei handelt es sich um den Namen des Exit State Mappings.
ENV_NAME	Hierbei handelt es sich um den Namen des Environments.
FP_NAME	Hierbei handelt es sich um den Namen des Footprints.
SUBFOLDERS	Hierbei handelt es sich um die Anzahl Folder unterhalb des Folders.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ENTITIES	Hierbei handelt es sich um die Anzahl Jobs und Batches unterhalb des Folders.
HAS_MSE	In dem Folder befindet sich mindestens ein Job der als Master submittable ausgeführt werden kann.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
IDPATH	Id Pfad zum Objekt
HIT	Zeile ist ein Suchtreffer Y/N

Tabelle 14.10: Beschreibung der Output-Struktur des list folder Statements

list footprint

Zweck

Zweck Das *list footprint* Statement wird eingesetzt um eine Liste aller definierten Footprints zu bekommen.

Syntax

Syntax Die Syntax des *list footprint* Statements ist

list footprint

Beschreibung

Beschreibung Mit dem *list footprint* Statement bekommt man eine Liste aller definierten Footprints.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.11: Beschreibung der Output-Struktur des list footprint Statements

list group

Zweck

Zweck

Das *list group* Statement wird eingesetzt um eine Liste von allen definierten Gruppen zu bekommen.

Syntax

Syntax

Die Syntax des *list group* Statements ist

list group

Beschreibung

Beschreibung

Mit dem *list group* Statement bekommt man eine Liste aller definierten Gruppen.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.12: Beschreibung der Output-Struktur des list group Statements

list interval

Zweck

Zweck Das *list interval* Statement wird eingesetzt um eine Liste aller definierten Intervalle zu bekommen.

Syntax

Syntax Die Syntax des *list interval* Statements ist

list interval

list interval all

Beschreibung

Beschreibung Mit dem *list interval* Statement bekommt man eine Liste aller definierten Intervalle.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
STARTTIME	Der Anfang des Intervalls. Vor dieser Zeit werden keine Flanken generiert.
ENDTIME	Das Ende des Intervalls. Nach dieser Zeit werden keine Flanken generiert.
BASE	Die Periode des Intervalls
DURATION	Die Dauer eines Blocks
SYNCTIME	Die Zeit mit der das Intervall synchronisiert wird. Die erste Periode des Intervalls startet zu dieser Zeit.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
INVERSE	Die Angabe, ob die Auswahlliste positiv oder negativ aufgefasst werden soll
EMBEDDED	Das Intervall aus dem nachträglich eine Auswahl getroffen wird
OBJ_TYPE	Der object type ist der Typ des Objektes, zu dem das Intervall gehört.
OBJ_ID	Die object id ist die Id des Objektes, zu dem das Intervall gehört.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
SE_ID	Dieses Feld wurde noch nicht beschrieben

Tabelle 14.13: Beschreibung der Output-Struktur des list interval Statements

list job

Zweck

Zweck Das *list job* Statement wird eingesetzt um eine Liste von Submitted Entities zu bekommen, basierend auf spezifizierte Selektionskriterien.

Syntax

Syntax Die Syntax des *list job* Statements ist

list [**condensed**] **job** [*jobid* {**,** *jobid*}] [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
enabled only
| **expand = none**
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >
| FILTERTERM {**or** FILTERTERM}
| **mode =** < **list** | **tree** >
| **parameter = (** *parametername* {**,** *parametername*} **)**

FILTERTERM:
FILTERITEM {**and** FILTERITEM}

FILTERITEM:
(FILTERTERM {**or** FILTERTERM} **)**
| < **enable** | **disable** >
| < **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*} **)**
| **jobserver 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**
| **SUSPENDED**
| **synchronize wait**
| **to kill**
| **unreachable**

Beschreibung

Beschreibung

Mit dem *list job* Statement bekommt man eine Liste von Submitted Entities. Die Auswahl der Jobs kann duch Angabe eines Filters beliebig fein spezifiziert werden. Zudem können Job Parameter-Namen spezifiziert werden, die daraufhin in der Ausgabe sichtbar werden.
Das Statement `list job` ohne weitere Angaben ist gleichbedeutend mit dem Statement `list job with master` und gibt also die Liste aller Master Jobs und Batches aus.

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

mode Im Mode **list** wird einfach eine Liste von selektierten Jobs ausgegeben. Wird dagegen **tree** als Mode angegeben, werden zu jedem selektierten Job alle Parents ausgegeben.

parameter Durch Angabe von Parameter-Namen kann zusätzliche Information zu den selektierten Jobs ausgegeben werden. Die Parameter werden im jeweiligen Jobkontext ausgewertet und der Wert des Parameters wird in der Ausgabe sichtbar gemacht. Falls dies fehlschlägt, wird ein Leerstring ausgegeben. Das heißt, dass die Angabe von nicht existierenden Parameter-Namen keine negativen Folgen hat.
Auf diese Weise können Status- oder Fortschrittinformationen von Jobs leicht übersichtlich dargestellt werden.

filter Zur Filterung aller im System vorhandenen Jobs kann von einer Vielzahl an Filter Gebrauch gemacht werden. Die einzelne Filter können mittels boolschen Operatoren miteinander kombiniert werden. Dabei gilt die übliche Prioritätsreihenfolge der Operatoren.
Im Folgenden werden die einzelnen Filtermöglichkeiten kurz beschrieben.

FINAL, RESTARTABLE, PENDING Dieser Filter selektiert alle Jobs die **final** bzw. **restartable** oder **pending** sind.

EXIT STATE Alle Jobs, die einen Exit State haben, der in der spezifizierte Liste vorkommt, werden selektiert. Es handelt sich hier um den jobeigenen Exit State, nicht den merged Exit State, der auch die Exit States der Children berücksichtigt.

HISTORY Durch Angabe einer History werden nur die Jobs selektiert, die frühestens vor der angegebenen Zeit **final** geworden sind. **Nonfinal** Jobs werden alle selektiert.

FUTURE Durch Angabe einer Future werden ebenfalls geplante zukünftige Jobs ausgegeben. Diese Ereignisse werden auf Basis von scheduled Events sowie Kalendereinträgen ermittelt. Als Status solcher Jobs wird "SCHEDULED" ausgegeben.

JOB.IDENTIFIER Mittels dieses Filters werden alle die Jobs selektiert, deren angegebene Parameter die Bedingung erfüllen. Auf diese Weise können etwa alle Jobs eines Entwicklers leich selektiert werden. (Unter der Annahme, dass natürlich jeder Job einen Parameter mit dem Enwicklernamen hat.)
Mit Hilfe der **expr** Funktion können auch Berechnungen durchgeführt werden. Der Ausdruck

```
job.starttime < expr('job.sysdate - job.expruntime * 1.5')
```

ermittelt die Jobs, die ihre zu erwartende Laufzeit um mehr als 50% überschritten haben.

JOB IN (ID, ...) Diese Filteroption ist gleichbedeutend mit der Angabe von Job-Ids nach "**list job**". Nur die Jobs mit einer der angegebenen Ids werden selektiert.

JOBSERVER Nur die Jobs die auf dem angegebenen Jobserver laufen werden selektiert.

JOB STATUS Dieser Filter selektiert nur die Jobs die einer der angegebenen Job-States haben. Es ist z.B. leicht alle Jobs im Status **broken_finished** zu finden.

MASTER Nur die Master Jobs und -batches werden selektiert.

MASTER_ID Nur Jobs die zu den spezifizierten Master Jobs und -batches gehören werden selektiert.

MERGED EXIT STATE Alle Jobs die einen Merged Exit State haben, der in der spezifizierte Liste vorkommt, werden selektiert. Es handelt sich hier um den Exit State, der aus dem eigenen Exit State in Kombination mit den Exit States der Children resultiert.

NAME IN (FOLDERPATH, ...) Die Jobs deren zugehöriger Scheduling Entity in der spezifizierten Liste vorkommt, werden selektiert.

NAME LIKE STRING Die Jobs deren zugehöriger Scheduling Entity den passenden Namen hat, werden selektiert. (Für nähere Information bezüglich der Syntax von Regular Expressions sei auf die offizielle Java Dokumentation verwiesen.)

NODE Jobs die auf einem der spezifizierten Nodes laufen werden selektiert. In diesem Kontext bezeichnet der Node den Eintrag für **node** des Jobservers.

OWNER Nur die Jobs der angegebenen Owners (Gruppen) werden selektiert.

SUBMITTING USER Nur die Jobs die vom angegebenen Benutzer submitted wurden, werden selektiert.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
MASTER_ID	Hierbei handelt es sich um die Id des Master Jobs.
HIERARCHY_PATH	Der Hierarchy_Path stellt den kompletten Pfad des aktuellen Eintrags dar. Die einzelnen Hierarchiestufen werden mittels eines Punkts getrennt.
SE_TYPE	Hierbei handelt es sich um den Typ des Scheduling Entities (Job, Batch oder Milestone).
PARENT_ID	Hierbei handelt es sich um die Id des Parents.
OWNER	Die Gruppe die Eigentümer des Objektes ist

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
SCOPE	Der Scope, bzw. Jobserver, dem der Job zugeordnet ist
HTTPHOST	Der Hostname des Scopes für den Zugriff auf Logfiles via HTTP
HTTPPORT	Die HTTP Portnummer des Jobservers für den Zugriff auf Logfiles via HTTP
EXIT_CODE	Der Exit_Code des ausgeführten Prozesses
PID	Bei der PID handelt es sich um die Prozessidentifikationsnummer des überwachenden Jobserverprozesses auf dem jeweiligen Hostsystem.
EXTPID	Die EXT_PID ist die Prozessidentifikationsnummer des Nutzprozesses.
STATE	Der State ist der aktuelle Status des Jobs.
IS_DISABLED	Zeigt an, ob der Job bzw. Batch disabled ist.
IS_CANCELLED	Zeigt an, ob ein Cancel auf den Job ausgeführt wurde
JOB_ESD	Der job_esd ist der Exit State des Jobs.
FINAL_ESD	Der FINAL_ESD ist der zusammengefasste Exit State vom Job oder Batch mit allen Child Exit States.
JOB_IS_FINAL	Dieses Feld gibt an, ob der Job selbst final ist.
CNT_RESTARTABLE	Die Anzahl der Children im Status restartable
CNT_SUBMITTED	Die Anzahl der Children im Status submitted
CNT_DEPENDENCY_WAIT	Die Anzahl der Children im Status dependency_wait
CNT_SYNCHRONIZE_WAIT	Die Anzahl der Children im Status synchronize_wait
CNT_RESOURCE_WAIT	Die Anzahl der Children im Status resource_wait
CNT_RUNNABLE	Die Anzahl der Children im Status runnable
CNT_STARTING	Die Anzahl der Children im Status starting
CNT_STARTED	Die Anzahl der Children im Status started
CNT_RUNNING	Die Anzahl der Children im Status running
CNT_TO_KILL	Die Anzahl der Children im Status to_kill
CNT_KILLED	Die Anzahl der Children im Status killed
CNT_CANCELLED	Die Anzahl der Children im Status cancelled

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CNT_FINISHED	Die Anzahl der Children im Status finished
CNT_FINAL	Die Anzahl der Children im Status final
CNT_BROKEN_ACTIVE	Die Anzahl der Children im Status broken_active
CNT_BROKEN_FINISHED	Die Anzahl der Children im Status broken_finished
CNT_ERROR	Die Anzahl der Children im Status error
CNT_UNREACHABLE	Die Anzahl der Children im Status unreachable
CNT_WARN	Die Anzahl der Children für die eine Warnung vorliegt
SUBMIT_TS	Der Zeitpunkt zu dem der Job submitted wurde.
RESUME_TS	Der Zeitpunkt zu dem der Job automatisch resumed wird
SYNC_TS	Der Zeitpunkt zu dem der Job in den Status synchronize_wait gewechselt ist
RESOURCE_TS	Der Zeitpunkt zu dem der Job in den Status resource_wait gewechselt ist
RUNNABLE_TS	Der Zeitpunkt an dem der Job den Status runnable erreicht hat
START_TS	Der Zeitpunkt zu dem der Job vom Jobserver als gestarted gemeldet wurde
FINISH_TS	Hierbei handelt es sich um den Zeitpunkt zu dem der Job beendet wird.
FINAL_TS	Der Zeitpunkt zu der der Job in den State final übergegangen ist
PRIORITY	Die statische Priorität eines Jobs. Diese setzt sich zusammen aus der definierten Priorität und den Nice Values der Parents.
DYNAMIC_PRIORITY	Die dynamische Priorität des Jobs. Diese ist die abhängig von der Wartezeit korrigierte statische Priorität.
NICEVALUE	Nice Value ist die Korrektur der Priorität der Children.
MIN_PRIORITY	Der minimale Wert der dynamischen Priorität
AGING_AMOUNT	Der Aging_Amount gibt an nach wievielen Zeiteinheiten die dynamische Priorität eines Jobs um einen Punkt hochgesetzt wird.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
AGING_BASE	Die Aging_Base gibt an um welche Zeiteinheit es beim Aging Amount geht.
ERROR_MSG	Die Fehlermeldung die beschreibt warum der Job in den Status error gewechselt ist
CHILDREN	Die Anzahl Children des Jobs oder Batches
HIT	Dieses Feld gibt an, ob der Job aufgrund Filterkriterien ausgewählt wurde oder nicht.
HITPATH	Dieses Feld gibt an, dass der Job ein direkter oder indirekter Parent eines selektierten Jobs ist.
SUBMITPATH	Dies ist die Liste der submitting Parents. Im Gegensatz zu der allgemeinen Parent Child- Hierarchie ist diese immer eindeutig.
IS_SUSPENDED	Dieses Feld gibt an, ob der Job oder Batch selbst suspended ist.
IS_RESTARTABLE	Dieses Feld gibt an, ob der Job restartable ist.
PARENT_SUSPENDED	Dieses Feld gibt an, ob der Job über einen seiner Parents suspended ist (true) oder nicht (false).
CHILDTAG	Der Tag womit mehrere Children voneinander unterschieden werden können
IS_REPLACED	Dieses Feld gibt an, ob der Job oder Batch durch einen anderen ersetzt wurde.
WARN_COUNT	Dies ist die Anzahl unbehandelter Warnings.
CHILD_SUSPENDED	Die Anzahl der Children die suspended wurden
CNT_PENDING	Die Anzahl der Children im Status pending
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
WORKDIR	Name des Working Directorys des Nutzprozesses
LOGFILE	Name des Logfiles des Nutzprozesses. Hier werden die Ausgaben nach `stdout` protokolliert.
ERRLOGFILE	Name des Error Logfiles des Nutzprozesses. Hier werden die Ausgaben nach `stderr` protokolliert.

Tabelle 14.14: Beschreibung der Output-Struktur des list job Statements

list job definition hierarchy

Zweck

Das *list job definition hierarchy* Statement wird eingesetzt um den kompletten Job Tree des spezifizierten Jobs zu bekommen. *Zweck*

Syntax

Die Syntax des *list job definition hierarchy* Statements ist *Syntax*

list job definition hierarchy *folderpath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Mit dem *list job definition hierarchy* Statement bekommt man die komplette Baumstruktur des spezifizierten Jobs. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Type gibt die Art des Objektes an. Es gibt folgende Optionen: Batch, Milestone, Job und Folder.
RUN_PROGRAM	Im Feld Run_Program kann eine Kommandozeile angegeben werden, die das Skript oder Programm startet.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RERUN_PROGRAM	Das Feld Rerun_Program gibt das Kommando an, welches bei einer wiederholten Ausführung des Jobs nach einem Fehlerzustand (rerun) ausgeführt werden soll.
KILL_PROGRAM	Das Kill_Program bestimmt welches Programm ausgeführt werden soll, um einen aktuell laufenden Job zu beenden.
WORKDIR	Hierbei handelt es sich um das Working Directory des aktuellen Jobs.
LOGFILE	Das Feld Logfile gibt an in welche Datei alle normalen Ausgaben des Run_Programs ausgegeben werden sollen. Unter normalen Ausgaben sind alle Ausgaben gemeint, die den normalen Ausgabekanal (STDOUT unter UNIX) benutzen.
TRUNC_LOG	Gibt an, ob das Logfile erneuert werden soll oder nicht
ERRLOGFILE	Das Feld Errorlogfile gibt an, in welcher Datei alle Fehlerausgaben des Run_Programs ausgegeben werden sollen.
TRUNC_ERRLOG	Gibt an, ob das Error Logfile erneuert werden soll oder nicht
EXPECTED_RUNTIME	Die Expected_Runtime beschreibt die erwartete Zeit die ein Job für seine Ausführung benötigt.
GET_EXPECTED_RUNTIME	Hierbei handelt es sich um ein reserviertes Feld für zukünftige Erweiterungen.
PRIORITY	Das Feld Priority gibt an mit welcher Dringlichkeit ein Prozess, falls er gestartet werden soll, vom Scheduling System berücksichtigt wird.
SUBMIT_SUSPENDED	Der Parameter Submit_Suspended gibt an in welcher Form das Child Objekt beim Start verzögert wird oder aber sofort gestartet werden kann. Es gibt folgende Optionen: Yes, No und Childsuspend.
MASTER_SUBMITTABLE	Der Job der durch den Trigger gestartet wird, wird als eigener Master Job submitted und hat keinen Einfluss auf den aktuellen Master Job-Lauf des triggernden Jobs.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
SAME_NODE	Obsolete
GANG_SCHEDULE	Obsolete
DEPENDENCY_MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
ESP_NAME	Hierbei handelt es sich um den Namen des Exit State Profiles.
ESM_NAME	Hierbei handelt es sich um den Namen des Exit State Mappings.
ENV_NAME	Hierbei handelt es sich um den Namen des Environments.
FP_NAME	Hierbei handelt es sich um den Namen des Footprints.
CHILDREN	Hierbei handelt es sich um die Anzahl direkter Children.
SH_ID	Die Id der Hierarchy Definition
IS_STATIC	Flag, das statische oder dynamische Submits von diesem Job anzeigt
IS_DISABLED	Flag das angibt, ob das Child ausgeführt oder übersprungen wird
INT_NAME	**int_name** ist der Name des Intervalls, welches verwendet wird, um zu prüfen ob das Child enabled ist.
ENABLE_CONDITION	**int_name** ist der Name des Intervalls, welches verwendet wird, um zu prüfen ob das Child enabled ist.
ENABLE_MODE	**int_name** ist der Name des Intervalls, welches verwendet wird, um zu prüfen ob das Child enabled ist.
SH_PRIORITY	Das Feld Priority gibt an mit welcher Dringlichkeit ein Prozess, falls er gestartet werden soll, vom Scheduling System berücksichtigt wird.
SH_SUSPEND	Der Schalter Submit Suspended ermöglicht den tatsächlichen Start eines Ablaufes zu verzögern.
SH_ALIAS_NAME	Mit dem Alias kann einem Child eine neue logische Benennung zugeordnet werden.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
MERGE_MODE	Der Merge_Mode gibt an, ob ein Child-Objekt mehrfach innerhalb eines Master Job-Laufes gestartet wird oder nicht. Es gibt folgende Optionen: No Merge, Failure, Merge Local und Merge Global.
EST_NAME	Hierbei handelt es sich um die Exit State Translation.
IGNORED_DEPENDENCIES	Hier kann eine Liste von Abhängigkeiten hinzugefügt werden, welche innerhalb dieser Parent-Child-Beziehung vom Child ignoriert werden soll.
HIERARCHY_PATH	Der Path beschreibt die darüberliegende Ordnerhierarchie eines Objektes. Alle übergeordneten Ordner werden punktgetrennt angezeigt.
STATES	Der State ist der aktuelle Status des Jobs.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.15: Beschreibung der Output-Struktur des list job definition hierarchy Statements

list named resource

Zweck

Das *list named resource* Statement wird eingesetzt um eine Liste von allen definierten Named Resources zu bekommen. *Zweck*

Syntax

Die Syntax des *list named resource* Statements ist *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**

Beschreibung

Mit dem *list named resource* Statement bekommt man eine Liste aller definierten Named Resources. Wenn eine Resource spezifiziert wird, wird diese Named Resource, sowie wenn es sich bei der Named Resource um eine Category handelt, auch alle seine Children gezeigt. Die Liste der Named Resources kann durch die Spezifikation eines Filters entsprechend verkürzt werden. *Beschreibung*

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

filter Durch die Spezifikation von Filter können Named Resources nach Name und oder Usage gefiltert werden. (Für die Syntax von Regular Expressions wird auf die ofiizielle Java Dokumentation verwiesen.)

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
USAGE	Die Usage gibt an um welchen Typ Resource es sich handelt.
RESOURCE_STATE_PROFILE	Es handelt sich hier um das zur Resource zugeordnete Resource State Profile.
FACTOR	Dies ist der Standard Factor mit der Resource Requirement Amounts multipliziert werden, wenn bei der betreffenden Resource nichts anders spezifiziert ist.
SUBCATEGORIES	Dies ist die Anzahl Categories die als Children unterhalb der gezeigten Named Resource vorhanden sind.
RESOURCES	Das sind die Instanzen der Named Resource.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
IDPATH	Dieses Feld wurde noch nicht beschrieben

Tabelle 14.16: Beschreibung der Output-Struktur des list named resource Statements

list resource state definition

Zweck

Der Zweck der *list resource state definition* ist es eine Liste von allen definierten Resource States zu bekommen. *Zweck*

Syntax

Die Syntax des *list resource state definition* Statements ist *Syntax*

list resource state definition

Beschreibung

Mit dem *list resource state definition* Statement bekommt man eine Liste aller definierten Resource States. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.17: Beschreibung der Output-Struktur des list resource state definition Statements

list resource state mapping

Zweck

Zweck Das *list resource state mapping* Statement wird eingesetzt um ein Liste von allen definierten Resource State Mappings zu bekommen.

Syntax

Syntax Die Syntax des *list resource state mapping* Statements ist

list resource state mapping

Beschreibung

Beschreibung Mit dem *list resource state mapping* Statement bekommt man eine Liste aller definierten Resource State Mappings.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.18: Beschreibung der Output-Struktur des list resource state mapping Statements

list resource state profile

Zweck

Das *list resource state profile* Statement wird eingesetzt um eine Liste von allen derzeit definierten Resource State Profiles zu bekommen. *Zweck*

Syntax

Die Syntax des *list resource state profile* Statements ist *Syntax*

list resource state profile

Beschreibung

Mit dem *list resource state profile* Statement bekommt man eine Liste aller definierten Resource State Profiles. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
INITIAL_STATE	Dieses Feld definiert den initialen Status der Resource. Dieser Resource State muss nicht in der Liste gültiger Resource States vorhanden sein.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.19: Beschreibung der Output-Struktur des list resource state profile Statements

list schedule

Zweck

Zweck Das *list schedule* Statement wird eingesetzt um eine Liste von allen definierten Zeitplänen zu erhalten.

Syntax

Syntax Die Syntax des *list schedule* Statements ist

list schedule *schedulepath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Beschreibung Das *list schedule* Statement liefert eine Liste mit dem angegebenen Schedule sowie aller seiner Children.

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
INTERVAL	Der Name des zum Schedule gehörenden Intervalls

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
TIME_ZONE	Die Zeitzone in der der Schedule gerechnet werden soll
ACTIVE	Dieses Feld gibt an, ob der Schedule als active markiert ist.
EFF_ACTIVE	Dieses Feld gibt an, ob der Schedule tatsächlich active ist. Dies kann aufgrund der hierarchischen Anordnug von "active" abweichen.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.20: Beschreibung der Output-Struktur des list schedule Statements

list scheduled event

Zweck

Zweck Der Zweck des *list scheduled event* Statements ist es eine Liste von allen definierten scheduled Events zu bekommen.

Syntax

Syntax Die Syntax des *list scheduled event* Statements ist

list scheduled event

Beschreibung

Beschreibung Mit dem *list scheduled event* Statement bekommt man eine Liste aller definierten scheduled Events.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
SCHEDULE	Der Schedule der den Zeitplan für den Scheduled Event bestimmt
EVENT	Der Event der ausgelöst wird
ACTIVE	Dieses Feld gibt an, ob der Schedule als active markiert ist.
EFF_ACTIVE	Dieses Flag gibt an, ob der scheduled Event auch tatsächlich active ist.
BROKEN	Mittels des Broken Feldes kann geprüft werden, ob beim Submit des Jobs ein Fehler aufgetreten ist.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ERROR_CODE	Im Feld Error_Code wird, falls ein Fehler bei der Ausführung des Jobs im Time Scheduling aufgetreten ist, der übermittelte Fehlercode angezeigt. Ist kein Fehler aufgetreten, bleibt das Feld leer.
ERROR_MSG	Im Feld Error Message wird, falls ein Fehler bei der Ausführung des Jobs im Time Scheduling aufgetreten ist, die übermittelte Fehlermeldung angezeigt. Ist kein Fehler aufgetreten, bleibt das Feld leer.
LAST_START	Hier wird der letzte Ausführungszeitpunkt des Jobs durch das Scheduling System angezeigt.
NEXT_START	Hier wird der nächste geplante Ausführungszeitpunkt des Tasks durch das Scheduling System angezeigt.
NEXT_CALC	Wenn der Next_Start leer ist gibt der Next_Calc den Zeitpunkt an wann nach einem nächsten Startzeitpunkt gesucht wird. Ansonsten findet die neue Berechnung zum Next_Start Zeitpunkt statt.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
BACKLOG_HANDLING	Das Backlog_Handling beschreibt den Umgang mit Events die zu Downtimes ausgelöst hätten werden sollen.
SUSPEND_LIMIT	Das Suspend_Limit gibt an nach welcher Verspätung ein Job als suspended submitted wird.
EFFECTIVE_SUSPEND_LIMIT	Das Suspend Limit gibt an nach welcher Verspätung ein Job als suspended submitted wird.
CALENDAR	Dieses Flag gibt an, ob Kalendereinträge erzeugt werden.
CALENDAR_HORIZON	Die definierte Länge der Periode in Tagen für die ein Kalender erstellt wird
EFFECTIVE_CALENDAR_HORIZON	Die effektive Länge der Periode in Tagen für die ein Kalender erstellt wird

Tabelle 14.21: Beschreibung der Output-Struktur des list scheduled event Statements

list scope

Zweck

Zweck Das *list scope* Statement wird eingesetzt um eine Liste aller definierten Scopes zu bekommen.

Syntax

Syntax Die Syntax des *list scope* Statements ist

list < **scope** *serverpath* | **jobserver** *serverpath* > [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Beschreibung Mit dem *list scope* Statement bekommt man die Liste des beantragten Scopes mit seinen Children angezeigt.

expand Mit der expand Option kann die Hierarchie nach unten sichtbar gemacht werden. Dazu werden die Id's von den Knoten deren Children sichtbar sein sollen in der Liste spezifiziert. Falls **none** als expand Option spezifiziert wird, wird nur die Ebene unterhalb des beantragten Knoten sichtbar gemacht.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Typ des Scopes

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_TERMINATE	Dieses Flag zeigt an, ob ein Terminierungsauftrag vorliegt.
HAS_ALTERED_CONFIG	Die Konfiguration im Server weicht von der aktuellen im Jobserver ab.
IS_SUSPENDED	Zeigt an, ob der Scope suspended ist
IS_ENABLED	Nur wenn das Enable Flag YES gesetzt ist kann sich der Jobserver am Server anmelden
IS_REGISTERED	Gibt an, ob der Jobserver einen register Befehl gesendet hat
IS_CONNECTED	Zeigt an, ob der Jobserver connected ist.
STATE	Hierbei handelt es sich um den aktuellen Status der Resource in diesem Scope.
PID	Bei PID handelt es sich um die Prozess Identifikationsnummer des Jobserverprozesses auf dem jeweiligen Hostsystem.
NODE	Der Node gibt an auf welchem Rechner der Jobserver läuft. Dieses Feld hat einen rein dokumentativen Charakter.
IDLE	Die Zeit die seit dem letzten Befehl vergangen ist. Dies gilt nur für Jobserver.
NOPDELAY	Die Zeit die ein Jobserver nach einem NOP wartet
ERRMSG	Hierbei handelt es sich um die zuletzt ausgegebene Fehlermeldung.
SUBSCOPES	Die Anzahl Scopes und Jobserver die unter diesem Scope vorhanden sind
RESOURCES	Hier werden die Resourcen die in diesem Scope vorhanden sind angezeigt.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
IDPATH	Dieses Feld wurde noch nicht beschrieben

Tabelle 14.22: Beschreibung der Output-Struktur des list scope Statements

list session

Zweck

Zweck Das *list session* Statement wird eingesetzt um eine Liste von den Connected Sessions zu bekommen.

Syntax

Syntax Die Syntax des *list session* Statements ist

list session

Beschreibung

Beschreibung Mit dem *list session* Statement bekommt man eine Liste der Connected Sessions.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
THIS	Die aktuelle Session wird in diesem Feld mit einem Asterisk (*) gekennzeichnet.
SESSIONID	Die serverinterne Id der Session
PORT	Der TCP/IP Portnummer an dem die Session connected ist
START	Zeitpunkt an dem die Connection hergestellt wurde
TYPE	Typ der Connection: User, Jobserver oder Job
USER	Name des connecting Users, Jobservers oder Jobs (Job Id)
UID	Id des Users, Jobservers oder Jobs
IP	IP-Adresse der connecting Sessions
TXID	Nummer der letzten Transaktion die von der Session ausgeführt wurde
IDLE	Die Anzahl Sekunden seit dem letzten Statement einer Session

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
STATE	Der Status der Session. Dies ist einer der folgenden: IDLE (keine Aktivität), QUEUED (Statement wartet auf Ausführung), ACTIVE (Statement wird gerade ausgeführt), COMMITTING (Änderungen einer schreibenden Transaktion werden geschrieben), CONNECTED (noch nicht authentifiziert).
TIMEOUT	Die Idle Zeit nach der die Session automatisch disconnected wird
INFORMATION	Zusätzliche Information zu der Session (optional)
STATEMENT	Das Statement das gerade ausgeführt wird
WAIT	Das Feld wait zeigt, ob die Session wartet (auf eine Sperre).

Tabelle 14.23: Beschreibung der Output-Struktur des list session Statements

list trigger

Zweck

Zweck Das *list trigger* Statement wird eingesetzt um eine Liste von definierten Triggers zu bekommen.

Syntax

Syntax Die Syntax des *list trigger* Statements ist

list trigger

list trigger for *folderpath*

list trigger of *folderpath*

list trigger for CT_OBJECT

CT_OBJECT:
job definition *folderpath*
| **named resource** *resourcepath*
| **object monitor** *objecttypename*
| **resource** *resourcepath* **in** < *folderpath* | *serverpath* >

Beschreibung

Beschreibung Mit dem *list trigger* Statement bekommt man eine Liste aller definierten Trigger.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
OBJECT_TYPE	Der Typ des Objektes in dem der Trigger definiert ist
OBJECT_SUBTYPE	Der Subtyp des Objektes in dem der Trigger definiert ist
OBJECT_NAME	Kompletter Pfadname des Objektes in dem der Trigger definiert ist
ACTIVE	Das Flag gibt an, ob der Trigger momentan aktiv ist.
ACTION	Typ der ausgelöste Aktion: SUBMIT oder RERUN
STATES	Eine Liste mit States die zum Auslösen des Triggers führen
SUBMIT_TYPE	Der Objekttyp der submitted wird, wenn getriggert wird
SUBMIT_NAME	Name der Job Definition die submitted wird
SUBMIT_SE_OWNER	Der Besitzer des Objektes das submitted wird
SUBMIT_PRIVS	Die Privilegien auf das zu submittende Objekt
MAIN_TYPE	Typ des Main Jobs (Job/Batch)
MAIN_NAME	Name des Main Jobs
MAIN_SE_OWNER	Owner des Main Jobs
MAIN_PRIVS	Privilegien auf den Main Job
PARENT_TYPE	Typ des Parent Jobs (Job/Batch)
PARENT_NAME	Name des Parent Jobs
PARENT_SE_OWNER	Owner des Parent Jobs
PARENT_PRIVS	Privilegien auf den Parent Job
TRIGGER_TYPE	Der Trigger Typ der beschreibt wann gefeuert wird
MASTER	Zeigt an, ob der Trigger einen Master oder ein Child submitted
IS_INVERSE	Im Falle eines Inverse Triggers gehört der Trigger dem getriggerten Job. Der Trigger kann so als Art Callback-Funktion gesehen werden. Das Flag hat keinen Einfluß auf die Funktion des Triggers.
SUBMIT_OWNER	Die Eigentümergruppe die beim Submitted Entity eingesetzt wird

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_CREATE	Zeigt an, ob der Trigger auf create Events reagiert
IS_CHANGE	Zeigt an, ob der Trigger auf change Events reagiert
IS_DELETE	Zeigt an, ob der Trigger auf delete Events reagiert
IS_GROUP	Zeigt an, ob der Trigger die Events als Gruppe behandelt
MAX_RETRY	Die maximale Anzahl von Trigger Auslösungen in einem einzelnen Submitted Entity
SUSPEND	Spezifiziert, ob das submittete Objekt suspended wird
RESUME_AT	Zeitpunkt des automatischen Resume
RESUME_IN	Anzahl Zeiteinheiten bis zum automatischen Resume
RESUME_BASE	Zeiteinheitsangabe für RESUME_IN
WARN	Spezifiziert, ob eine Warnung ausgegeben werden muss wenn das Feuerlimit erreicht ist
LIMIT_STATE	Spezifiziert den Status der vom auslösenden Jobs angenommen wird, wenn das Fire Limit erreicht wird. Hat der Job bereit einen finalen Status, wird diese Einstellung ignoriert. Steht der Wert auf NONE, wird keine Statusänderung vorgenommen.
CONDITION	Konditionaler Ausdruck um die Trigger Condition zu definieren
CHECK_AMOUNT	Die Menge der CHECK_BASE Einheiten um die Kondition bei nicht synchronen Triggern zu überprüfen
CHECK_BASE	Einheiten für den CHECK_AMOUNT
PARAMETERS	Mit der **parameter** Klausel können Parameter für den zu submittenden Job oder Batch festgelegt werden. Die Namen der Parameter werden als solche übernommen. Die Ausdrücke werden im Kontext des triggernden Jobs oder Batches evaluiert.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
TAG	Einheiten für den CHECK_AMOUNT
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 14.24: Beschreibung der Output-Struktur des list trigger Statements

list user

Zweck

Zweck Das *list user* Statement wird eingesetzt um eine Liste von allen definierten Benutzern zu erhalten.

Syntax

Syntax Die Syntax des *list user* Statements ist

list user

Beschreibung

Beschreibung Mit dem *list user* Statement bekommt man eine Liste aller definierten Benutzer.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
IS_ENABLED	Flag, das anzeigt, ob es dem Benutzer erlaubt ist sich anzumelden
DEFAULT_GROUP	Die Default-Gruppe der Benutzer die die Eigentümer des Objektes benutzen

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CONNECTION_TYPE	Gibt an welche Sicherheitsstufe für eine Verbindung gefordert wird. 1. **plain** – Jede Art von Verbindung ist erlaubt 2. **ssl** – Nur SSL-Verbindungen sind erlaubt 3. **ssl_auth** – Nur SSL-Verbindungen mit Client Authentifizierung sind erlaubt
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 14.25: Beschreibung der Output-Struktur des list user Statements

Kapitel 15

move commands

move folder

Zweck

Zweck Das *move folder* Statement wird eingesetzt um den Folder umzubenennen und/oder ihn an eine andere Stelle in der Ordnerhierarchie zu verschieben.

Syntax

Syntax Die Syntax des *move folder* Statements ist

move folder *folderpath* **to** *folderpath*

Beschreibung

Beschreibung Der *move folder* Befehl verschiebt den angegebenen Folder an eine andere Stelle oder er benennt ihn um.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

move job definition

Zweck

Zweck

Das *move job definition* Statement wird eingesetzt um ein Scheduling Entity Objekt umzubenennen und/oder es in einen anderen Folder zu verschieben.

Syntax

Syntax

Die Syntax des *move job definition* Statements ist

move job definition *folderpath* **to** *folderpath*

Beschreibung

Beschreibung

Der *move job definition* Befehl verschiebt die angegebene Job Definition in den angegebenen Folder. Wenn der Ziel-Folder nicht existiert, wird das letzte Glied des vollqualifizierten Namens als neuer Name für die Job Definition interpretiert. Die Beziehungen zu anderen Objekten werden nicht geändert.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

move named resource

Zweck

Zweck Das *move named resource* Statement wird eingesetzt um die Named Resource umzubenennen und/oder um die Resource in eine andere Kategorie zu verschieben.

Syntax

Syntax Die Syntax des *move named resource* Statements ist

move named resource *resourcepath* **to** *resourcepath*

Beschreibung

Beschreibung Das *move named resource* Statement wird benutzt um Named Resources umzubenennen, bzw. zu verschieben und/oder um Kategorien neu zu ordnen.
Wenn eine Named Resource verschoben wird, muss das spezifizierte Ziel eine Kategorie sein oder es darf nicht existieren und sein Parent muss ebenfalls eine Kategorie sein.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

move schedule

Zweck

Das *move schedule* Statement wird eingesetzt um den Zeitplan umzubenennen oder ihn an einen anderen Ort in der Hierarchie zu verschieben. *Zweck*

Syntax

Die Syntax des *move schedule* Statements ist *Syntax*

move schedule *schedulepath* **.** *schedulename* **to** *schedulepath*

Beschreibung

Der *move schedule* Befehl verschiebt den angegebenen Schedule an einen anderen Ort und/oder er benennt ihn um. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

move scope

Zweck

Zweck Das *move scope* Statement wird eingesetzt um einen Scope umzubenennen und/oder an eine andere Stelle in der Scope-Hierarchie zu verschieben.

Syntax

Syntax Die Syntax des *move scope* Statements ist

move < **scope** *serverpath* | **jobserver** *serverpath* > **to** *serverpath*

Beschreibung

Beschreibung Der *move scope* Befehl verschiebt den angegebenen Scope an einen anderen Ort und/oder er benennt ihn um.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 16

multicommand commands

multicommand

Zweck

Zweck Der Zweck des *multicommands* ist es mehrere SDMS-Kommandos als Einheit zuzuführen.

Syntax

Syntax Die Syntax des *multicommand* Statements ist

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Beschreibung

Beschreibung Mit den *multicommands* ist es möglich mehrere SDMS-Kommandos zusammen, d.h. in einer Transaktion auszuführen. Damit wird gewährleistet, dass entweder alle Statements fehlerfrei ausgeführt werden, oder nichts passiert. Des Weiteren wird die Transaktion nicht von anderen schreibenden Transaktionen unterbrochen. Wird das Keyword **rollback** spezifiziert, wird die Transaktion am Ende der Verarbeitung rückgängig gemacht. Auf diese Weise kann getestet werden, ob die Statements (technisch) korrekt verarbeitet werden können.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 17

register commands

register

Zweck

Zweck Das *register* Statement wird eingesetzt um den Server zu benachrichtigen, dass der Jobserver bereit ist Jobs auszuführen.

Syntax

Syntax Die Syntax des *register* Statements ist

register *serverpath* **.** *servername*
with pid = *pid* [**suspend**]

register with pid = *pid*

Beschreibung

Beschreibung Die erste Form wird vom Operator benutzt um das Aktivieren von Jobs auf dem spezifizierten Jobserver zu ermöglichen.
Die zweite Form wird vom Jobserver selbst benutzt um den Server über seine Bereitschaft Jobs auszuführen zu informieren.
Unabhängig davon, ob der Jobserver connected ist oder nicht, werden Jobs für diesen Server eingeplant, es sei den der Jobserver ist suspended.
(Siehe Statement *'deregister'* auf Seite 178.)

pid Die pid Option liefert dem Server Informationen über die Prozess-Id des Jobservers auf Betriebsebene.

suspend Die suspend Option bewirkt, dass der Jobserver in den suspended Zustand überführt wird.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 18

rename commands

rename environment

Zweck

Zweck Das *rename environment* Statement wird eingesetzt um den spezifizierten Environment umzubenennen.

Syntax

Syntax Die Syntax des *rename environment* Statements ist

rename environment *environmentname* **to** *environmentname*

Beschreibung

Beschreibung Das *rename environment* Statement wird benutzt um Environments umzubenennen. Die Umbenennung eines Environments hat keinen Einfluss auf Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename event

Zweck

Der Zweck des *rename event* Statements ist es dem spezifizierten Event einen anderen Namen zu geben. *Zweck*

Syntax

Die Syntax des *rename event* Statements ist *Syntax*

rename event *eventname* **to** *eventname*

Beschreibung

Mit dem *rename event* Statement gibt man einem spezifizierten Event einen anderen Namen. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename exit state definition

Zweck

Zweck Das *rename exit state definition* Statement wird eingesetzt um die spezifizierte Exit State Definition umzubenennen.

Syntax

Syntax Die Syntax des *rename exit state definition* Statements ist

rename exit state definition *statename* **to** *statename*

Beschreibung

Beschreibung Das *rename exit state definition* Statement wird benutzt um Exit State Definitions umzubenennen. Das Umbenennen einer Exit State Definition hat keinen Einfluss auf Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename exit state mapping

Zweck

Das *rename exit state mapping* Statement wird eingesetzt um das spezifizierte Mapping umzubenennen. *Zweck*

Syntax

Die Syntax des *rename exit state mapping* Statements ist *Syntax*

rename exit state mapping *mappingname* **to** *profilename*

Beschreibung

Das *rename exit state mapping* Statement wird benutzt um Exit State Mappings umzubenennen. Das Umbenennen eines Exit State Mappings hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename exit state profile

Zweck

Zweck Das *rename exit state profile* Statement wird eingesetzt um das spezifizierten Profile umzubenennen.

Syntax

Syntax Die Syntax des *rename exit state profile* Statements ist

rename exit state profile *profilename* **to** *profilename*

Beschreibung

Beschreibung Das *rename exit state profile* Statement wird benutzt um Exit State Profiles umzubenennen. Das Umbenennen der Exit State Profiles hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename exit state translation

Zweck

Das *rename exit state translation* Statement wird eingesetzt um die spezifizierte Exit State Translation umzubenennen. *Zweck*

Syntax

Die Syntax des *rename exit state translation* Statements ist *Syntax*

rename exit state translation *transname* **to** *transname*

Beschreibung

Das *rename exit state translation* Statement wird benutzt um Exit State Translations umzubenennen. Das Umbenennen einer Exit State Translation hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename folder

Zweck

Zweck Das *rename folder* Statement dient zum Umbenennen eines Folders.

Syntax

Syntax Die Syntax des *rename folder* Statements ist

rename folder *folderpath* **to** *foldername*

Beschreibung

Beschreibung Der *rename folder* Befehl benennt den angegebenen Folder um. Dies geschieht innerhalb des gleichen Parent Folders. Wenn bereits ein Objekt mit dem neuen Namen vorhanden ist, wird eine Fehlermeldung ausgegeben.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename footprint

Zweck

Das *rename footprint* Statement wird eingesetzt um den spezifizierten Footprint umzubenennen. *Zweck*

Syntax

Die Syntax des *rename footprint* Statements ist *Syntax*

rename footprint *footprintname* **to** *footprintname*

Beschreibung

Mit dem *rename footprint* Statement vergibt man einem spezifizierten Footprint einen anderen Namen. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename group

Zweck

Zweck Das *rename group* Statement wird eingesetzt um den Namen einer Gruppe zu ändern, ohne das andere Eigenschaften davon betroffen werden .

Syntax

Syntax Die Syntax des *rename group* Statements ist

rename group *groupname* **to** *groupname*

Beschreibung

Beschreibung Das *rename group* Statement wird benutzt um Gruppen umzubenennen. Das Umbenennen einer Gruppe hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename interval

Zweck

Das *rename interval* Statement wird eingesetzt um den spezifizierten Intervall umzubenennen. *Zweck*

Syntax

Die Syntax des *rename interval* Statements ist *Syntax*

rename interval *intervalname* **to** *intervalname*

Beschreibung

Mit dem *rename interval* Statement gibt man dem spezifizierten IntervalL einen anderen Namen. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename job definition

Zweck

Zweck Das *rename job definition* Statement dient zum Umbenennen einer Job Definition.

Syntax

Syntax Die Syntax des *rename job definition* Statements ist

rename job definition *folderpath* **to** *jobname*

Beschreibung

Beschreibung Der *rename job definition* Befehl benennt die angegebene Job Definition um.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename named resource

Zweck

Zweck

Das *rename named resource* Statement dient zum Umbenennen einer Named Resource.

Syntax

Syntax

Die Syntax des *rename named resource* Statements ist

rename named resource *resourcepath* **to** *resourcename*

Beschreibung

Beschreibung

Das *move named resource* Statement wird benutzt um eine Named Resource umzubenennen.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename resource state definition

Zweck

Zweck Das *rename resource state definition* Statement wird eingesetzt um den Resource State umzubenennen.

Syntax

Syntax Die Syntax des *rename resource state definition* Statements ist

rename resource state definition *statename* **to** *statename*

Beschreibung

Beschreibung Das *rename resource state definition* Statement wird benutzt um Resource State Definitions umzubenennen. Das Umbenennen einer Resource State Definition hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename resource state mapping

Zweck

Zweck

Das *rename resource state mapping* Statement wird eingesetzt um dem spezifizierten Mapping einen neuen Namen zu geben.

Syntax

Syntax

Die Syntax des *rename resource state mapping* Statements ist

rename resource state mapping *mappingname* **to** *profilename*

Beschreibung

Beschreibung

Das *rename resource state mapping* Statement wird benutzt um Resource State Mappings umzubenennen. Das Umbenennen eines Resource State Mappings hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename resource state profile

Zweck

Zweck Der Zweck des *rename resource state profiles* ist es dem spezifizierten Resource State Profile einen neuen Namen zu geben.

Syntax

Syntax Die Syntax des *rename resource state profile* Statements ist

rename resource state profile *profilename* **to** *profilename*

Beschreibung

Beschreibung Das *rename resource state profile* Statement wird benutzt um Resource State Profiles umzubenennen. Das Umbenennen eines Resource State Profiles hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename schedule

Zweck

Das *rename schedule* Statement dient zum Umbenennen eines Schedules. *Zweck*

Syntax

Die Syntax des *rename schedule* Statements ist *Syntax*

rename schedule *schedulepath* **.** *schedulename* **to** *schedulename*

Beschreibung

Der *rename schedule* Befehl benennt den angegebenen Schedule um. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename scope

Zweck

Zweck Das *rename scope* Statement dient zum Umbenennen eines Scopes.

Syntax

Syntax Die Syntax des *rename scope* Statements ist

rename < **scope** *serverpath* | **jobserver** *serverpath* > **to** *scopename*

Beschreibung

Beschreibung Der *rename scope* Befehl benennt den angegebenen Scope um.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

rename trigger

Zweck

Das *rename trigger* Statement wird eingesetzt um dem spezifizierten Trigger einen anderen Namen zu geben. *Zweck*

Syntax

Die Syntax des *rename trigger* Statements ist *Syntax*

rename trigger *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >] **to** *triggername*

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

Beschreibung

Das *rename trigger* Statement wird benutzt um den Trigger umzubenennen. Das Umbenennen eines Triggers hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

rename user

Zweck

Zweck Das *rename user* Statement wird eingesetzt um den Benutzernamen umzubenennen, ohne eine seiner anderen Eigenschaften zu ändern.

Syntax

Syntax Die Syntax des *rename user* Statements ist

rename user *username* **to** *username*

Beschreibung

Beschreibung Das *rename user* Statement wird benutzt um User umzubenennen. Das Umbenennen eines Users hat keinen Einfluss auf die Funktionalitäten und dient nur der Übersichtlichkeit.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 19

resume commands

resume

Zweck

Zweck Das *resume* Statement wird eingesetzt um den Jobserver zu reaktivieren. Siehe das *suspend* Statement auf Seite 416.

Syntax

Syntax Die Syntax des *resume* Statements ist

resume *serverpath*

Beschreibung

Beschreibung Mit dem *resume* Statement reaktiviert man einen Jobserver.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 20

select commands

select

Zweck

Zweck Das *select* Statement wird eingesetzt um es dem Benutzer zu ermöglichen nahezu beliebige Queries auszuführen.

Syntax

Syntax Die Syntax des *select* Statements ist

select-statement [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
identifier **category** [**quoted**]
| *identifier* **folder** [**quoted**]
| *identifier* **job** [**quoted**]
| *identifier* **resource** [**quoted**]
| *identifier* **schedule** [**quoted**]
| *identifier* **scope** [**quoted**]
| **sort (** *signed_integer* {**,** *signed_integer*} **)**

Beschreibung

Beschreibung Das *select* Statement ermöglicht es nahezu beliebige Datenbank Select Statements vom Scheduling Server ausführen zu lassen. (Für den Syntax des Select Statements wird dazu auf die Dokumentation des eingesetzten Datenbanksystem verwiesen.)
Da das Absetzen von beliebigen *select* Statements prinzipiell eine Sicherheitslücke darstellt, werden für dieses Statement Administratorrechte benötigt. Das heißt, dass nur Benutzer der Gruppe **ADMIN** dieses Statement nutzen können.
Die Benutzung der *withitems* führt zum Übersetzen von Ids nach Namen. Dies wird für alle hierarchisch strukturierte Objekttypen angeboten, da diese Operation mit SQL-Mitteln nicht immer einfach durchzuführen ist.
Wird das optionale keyword **quoted** spezifiziert, werden die einzelne Identifiers mit Quotes versehen. Dies ist insbesondere für das Generieren von Statements gedacht.
Es ist ebenfalls möglich die Ergebnismenge nach dem Ersetzen der Ids zu sortieren. Die Spalten nach denen sortiert werden soll, werden durch ihre Position in der Ergebnismenge addressiert (zero based, d.h. die erste Spalte hat Nummer 0).

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 21

set commands

set parameter

Zweck

Zweck Das *set parameter* Statement wird eingesetzt um den Wert des spezifizierten Parameters innerhalb des Kontext eines Jobs zu setzen.

Syntax

Syntax Die Syntax des *set parameter* Statements ist

set parameter *parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,**
parametername **=** *string*} [**with comment =** *string*]

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,**
parametername **=** *string*} **identified by** *string* [**with comment =** *string*]

Beschreibung

Beschreibung Mittels des *set parameter* Statements können Jobs oder Benutzer Parameterwerte im Kontext des Jobs setzen.
Falls die **identified by** Option spezifiziert ist, wird der Parameter nur dann gesetzt, wenn das Paar *jobid* und *string* eine Anmeldung ermöglichen würden.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 22

show commands

show comment

Zweck

Zweck Das *show comment* Statement wird eingesetzt um den Kommentar zu dem spezifizierten Objekt anzuzeigen.

Syntax

Syntax Die Syntax des *show comment* Statements ist

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*
| **exit state translation** *transname*
| **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* | **jobserver** *serverpath* >
| **trigger** *triggername* **on** TRIGGEROBJECT [< **noinverse** | **inverse** >]
| **user** *username*

PARAM_LOC:

folder *folderpath*
| **job definition** *folderpath*
| **named resource** *resourcepath*
| < **scope** *serverpath* | **jobserver** *serverpath* >

TRIGGEROBJECT:

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

Beschreibung

Beschreibung

Das *show comment* Statement dient der Anzeige des, zum spezifizierten Objekt, abgelegten Kommentars. Wenn kein Kommentar zu dem Objekt vorhanden ist, wird dies *nicht* als Fehler gesehen, sondern es wird eine leere Output-Struktur erzeugt und zurückgegeben. Diese leere Output-Struktur entspricht natürlich der unten beschriebenen Output-Struktur, sodass sie auch leicht von Programmen, ohne Ausnahmebehandlung, ausgewertet werden kann.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Systemweit eindeutige Objektnummer
TAG	Der Comment Tag ist eine Kopfzeile für den aktuellen Kommentarblock. Das Feld ist optional.
COMMENT	Der Kommentar zum spezifizierten Objekt
COMMENTTYPE	Typ des Kommentars, Text oder URL
CREATOR	Name des Benutzers der diesen Pool angelegt hat
CREATE_TIME	Zeitpunkt des Anlegens
CHANGER	Name des Benutzers der diesen Pool zuletzt geändert hat

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CHANGE_TIME	Zeitpunkt der letzten Änderung
PRIVS	Abkürzung für die Privilegien die der anfragende Benutzer für dieses Objekt hat

Tabelle 22.1: Beschreibung der Output-Struktur des show comment Statements

show environment

Zweck

Das *show environment* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Environment zu bekommen. *Zweck*

Syntax

Die Syntax des *show environment* Statements ist *Syntax*

show environment *environmentname* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Mit dem *show environment* Statement bekommt man ausführliche Informationen über das spezifizierte Environment. *Beschreibung*

expand Da die Anzahl Job Definitions in der Tabelle JOB_DEFINITIONS sehr groß werden kann, werden sie per Default nicht angezeigt. Wenn die Option **expand = all** benutzt wird, werden alle Job Definitions, sowie die Folder, in die sie liegen, samt Folderhierarchie ausgegeben. Durch die Spezifikation einzelner (Folder) Id's können einzelne Pfade der Hierarchie selektiert werden.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Environments
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
RESOURCES	Tabelle von statischen Resourcen die dieses Environment formen Siehe auch Tabelle 22.3 auf Seite 312
JOB_DEFINITIONS	Tabelle von Jobs und Folder die dieses Environment nutzen Siehe auch Tabelle 22.4 auf Seite 313

Tabelle 22.2: Beschreibung der Output-Struktur des show environment Statements

RESOURCES Das Layout der RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
NR_NAME	Kompletter Pfadname von statischen Named Resources
CONDITION	Die Condition die zur Belegung erfüllt sein muss
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.3: Output-Struktur der show environment Subtabelle

JOB_DEFINITIONS Das Layout der JOB_DEFINITIONS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SE_PATH	Kompletter Folder-Pfadname von Job Definitions oder Folder
TYPE	Der Objekttyp. Die möglichen Werte sind FOLDER und JOB_DEFINITION
ENV	Ein Asterisk zeigt an, dass das aktuelle Environment hier spezifiziert wurde.
HAS_CHILDREN	True bedeutet, dass es weiter unten im Baum noch Environment-Benutzer gibt.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.4: Output-Struktur der show environment Subtabelle

show event

Zweck

Zweck Das *show event* Statement wird eingesetzt um detaillierte Informationen über das spezifizierte Event zu bekommen.

Syntax

Syntax Die Syntax des *show event* Statements ist

show event *eventname*

Beschreibung

Beschreibung Mit dem *show event* Statement bekommt man ausführliche Informationen über das spezifizierte Event.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Show Events
OWNER	Die Gruppe die Eigentümer des Objektes ist
SCHEDULING_ENTITY	Batch oder Job der submitted wird wenn dieses Event eintritt
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PARAMETERS	Parameter die beim Submit des Jobs oder Batches benutzt werden Siehe auch Tabelle 22.6 auf Seite 315
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 22.5: Beschreibung der Output-Struktur des show event Statements

PARAMETERS Das Layout der PARAMETERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
KEY	Name des Parameters
VALUE	Wert des Parameters

Tabelle 22.6: Output-Struktur der show event Subtabelle

show exit state definition

Zweck

Zweck Das *show exit state definition* Statement wird eingesetzt um detaillierte Informationen über die spezifizierte Exit State Definition zu bekommen.

Syntax

Syntax Die Syntax des *show exit state definition* Statements ist

show exit state definition *statename*

Beschreibung

Beschreibung Mit dem *show exit state definition* Statement bekommt man ausführliche Informationen über die spezifizierte Exit State Definition.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name der Exit State Definiton
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.7: Beschreibung der Output-Struktur des show exit state definition Statements

show exit state mapping

Zweck

Zweck

Das *show exist state mapping* Statement wird eingesetzt um detaillierte Informationen über das spezifizierte Mapping zu bekommen.

Syntax

Syntax

Die Syntax des *show exit state mapping* Statements ist

show exit state mapping *mappingname*

Beschreibung

Beschreibung

Mit dem *show exit state mapping* Statement bekommt man ausführliche Informationen über das spezifizierte Mapping.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
COMMENT	Ein Kommentar, vom Benutzer frei wählbar
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
RANGES	Die Zuordnung der jeweiligen Wertebereiche, in einer Tabelle dargestellt Siehe auch Tabelle 22.9 auf Seite 318

Tabelle 22.8: Beschreibung der Output-Struktur des show exit state mapping Statements

RANGES Das Layout der RANGES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ECR_START	Untere Grenze des Bereiches (inklusiv)
ECR_END	Obere Grenze des Bereiches (inklusiv)
ESD_NAME	Name des Exit States auf den dieser Bereich abgebildet wird

Tabelle 22.9: Output-Struktur der show exit state mapping Subtabelle

show exit state profile

Zweck

Das *show exist state profile* Statement wird eingesetzt um detaillierte Informationen über das spezifizierten Profile zu bekommen. *Zweck*

Syntax

Die Syntax des *show exit state profile* Statements ist *Syntax*

show exit state profile *profilename*

Beschreibung

Mit dem *show exit state profile* Statement bekommt man ausführliche Informationen über den spezifizierten Profile. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
DEFAULT_ESM_NAME	Default Exit State Mapping ist aktiv wenn der Job selbst nichts anderes angibt.
IS_VALID	Flag Anzeige über die Gültigkeit dieses Exit State Profiles
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
STATES	Tabelle beinhaltet Exit States die für dieses Profil gültig sind Siehe auch Tabelle 22.11 auf Seite 320

Tabelle 22.10: Beschreibung der Output-Struktur des show exit state profile Statements

STATES Das Layout der STATES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
PREFERENCE	Die Präferenz die Verbindung der Child Exit States zu kontrollieren
TYPE	Zeigt an, ob der State FINAL, PENDING oder RESTARTABLE ist
ESD_NAME	Name der Exit State Definition
IS_UNREACHABLE	Zeigt an, dass dieser Exit State benutzt wird wenn ein Job unreachable wird
IS_DISABLED	Normalerweise wird ein disabled Job denselben Exit State annehmen wie ein leerer Batch. Wenn jedoch ein FINAL State als Disabled gekennzeichnet wird, wird dadurch das Default Verhalten ausgehebelt und ein Disabled Job wird diesen State annehmen.
IS_BROKEN	Zeigt an, dass dieser Exit State benutzt wird wenn ein Job fehlerhaft ist
IS_BATCH_DEFAULT	Zeigt an, dass dieser Exit State benutzt wird wenn ein Batch oder Milestane keine Children hat
IS_DEPENDENCY_DEFAULT	Zeigt an, dass dieser Exit State benutzt wird wenn bei der Dependency Definition die State Selection DEFAULT gewählt wurde

Tabelle 22.11: Output-Struktur der show exit state profile Subtabelle

show exit state translation

Zweck

Zweck

Das *show exit state translation* Statement wird eingesetzt um detaillierte Informationen über die spezifizierte Exit State Translation zu bekommen.

Syntax

Syntax

Die Syntax des *show exit state translation* Statements ist

show exit state translation *transname*

Beschreibung

Beschreibung

Mit dem *show exit state translation* Statement bekommt man ausführliche Informationen über die spezifizierte Exit State Translation.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name der Exit State Translation
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
TRANSLATION	Tabelle der Exit State Translations vom Child zum Parent Siehe auch Tabelle 22.13 auf Seite 322

Tabelle 22.12: Beschreibung der Output-Struktur des show exit state translation Statements

TRANSLATION Das Layout der TRANSLATION Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
FROM_ESD_NAME	Child Exit State
TO_ESD_NAME	Parent Exit State

Tabelle 22.13: Output-Struktur der show exit state translation Subtabelle

show folder

Zweck

Zweck

Das *show folder* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Folder zu bekommen.

Syntax

Syntax

Die Syntax des *show folder* Statements ist

show folder *folderpath*

Beschreibung

Beschreibung

Mit dem *show folder* Statement bekommt man ausführliche Informationen über den spezifizierten Folder.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Folders
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Type gibt die Art des Objektes an. Es gibt folgende Optionen: Batch, Milestone, Job und Folder.
ENVIRONMENT	Der Name des optionalen Environments
INHERIT_PRIVS	Vom übergeordneten Ordner zu erbende Privilegien
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
PARAMETERS	Die Parameter Tabelle zeigt alle definierten Konstanten für diesen Folder an.
DEFINED_RESOURCES	Die Defined_Resources Tabelle zeigt alle Resource Instanzen an, die für diesen Folder definiert sind.

Tabelle 22.14: Beschreibung der Output-Struktur des show folder Statements

show footprint

Zweck

Das *show footprint* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Footprint zu bekommen. *Zweck*

Syntax

Die Syntax des *show footprint* Statements ist *Syntax*

show footprint *footprintname* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Mit dem *show footprint* Statement bekommt man ausführliche Informationen über den spezifizierten Footprint. *Beschreibung*

expand Da die Anzahl Job Definitions in der Tabelle JOB_DEFINITIONS sehr groß werden kann, werden sie per Default nicht angezeigt. Wenn die Option **expand = all** benutzt wird, werden alle Job Definitions, sowie die Folder, in die sie liegen, samt Folderhierarchie ausgegeben. Durch die Spezifikation einzelner (Folder) Id's können einzelne Pfade der Hierarchie selektiert werden.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Footprints
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
RESOURCES	Tabelle von System Resources die diesen Footprint formen Siehe auch Tabelle 22.16 auf Seite 326
JOB_DEFINITIONS	Tabelle von Job Definitionen die diesen Footprint nutzen Siehe auch Tabelle 22.17 auf Seite 327

Tabelle 22.15: Beschreibung der Output-Struktur des show footprint Statements

RESOURCES Das Layout der RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
RESOURCE_NAME	Vollqualifizierter Pfadname von System Named Resources
AMOUNT	Menge der Resource-Einheiten die allokiert werden
KEEP_MODE	Keep_Mode spezifiziert wann die Resource freigegeben wird (FINISHED, JOB_FINAL oder FINAL)

Tabelle 22.16: Output-Struktur der show footprint Subtabelle

JOB_DEFINITIONS Das Layout der JOB_DEFINITIONS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SE_PATH	Ordner Pfadname des Objektes
TYPE	Typ des Objekts
HAS_CHILDREN	True bedeutet, dass es weiter unten im Baum noch Environment-Benutzer gibt.
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.17: Output-Struktur der show footprint Subtabelle

show group

Zweck

Zweck Das *show group* Statement wird eingesetzt um detaillierte Informationen über die spezifizierte Gruppe zu bekommen.

Syntax

Syntax Die Syntax des *show group* Statements ist

show group *groupname*

Beschreibung

Beschreibung Mit dem *show group* Statement bekommt man ausführliche Informationen über die spezifizierten Gruppe.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name der Gruppe
COMMENTTYPE	Typ des Kommentars
COMMENT	Kommentar zum Objekt, wenn vorhanden
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
MANAGE_PRIVS	Tabelle der Manage Privilegien Siehe auch Tabelle 22.19 auf Seite 329

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
USERS	Tabelle der Benutzergruppen Siehe auch Tabelle 22.20 auf Seite 329

Tabelle 22.18: Beschreibung der Output-Struktur des show group Statements

MANAGE_PRIVS Das Layout der MANAGE_PRIVS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.19: Output-Struktur der show group Subtabelle

USERS Das Layout der USERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
UID	Id des Users
NAME	Der Name des Objektes
IS_ENABLED	Dieses Flag teilt dem Benutzer mit, ob er verbunden werden kann.
DEFAULT_GROUP	Die Default-Gruppe von diesem Benutzer
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.20: Output-Struktur der show group Subtabelle

show interval

Zweck

Zweck Das *show interval* Statement wird eingesetzt um detaillierte Informationen über den Intervall zu bekommen.

Syntax

Syntax Die Syntax des *show interval* Statements ist

show interval *intervalname* [**(** *id* **)**]

Beschreibung

Beschreibung Das *show interval* Statement zeigt detaillierte Informationen zu einem Intervall. Ohne *expand* Klausel werden keine steigende Flanken (Edges) angezeigt. Mit der *expand* Klausel kann eine Periode, für die die Edges gezeigt werden sollen, spezifiziert werden.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
STARTTIME	Der Anfang des Intervalls. Vor dieser Zeit werden keine Flanken generiert.
ENDTIME	Das Ende des Intervalls. Nach dieser Zeit werden keine Flanken generiert.
BASE	Die Periode des Intervalls
DURATION	Die Dauer eines Blocks
SYNCTIME	Die Zeit mit der das Intervall synchronisiert wird. Die erste Periode des Intervalls startet zu dieser Zeit.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
INVERSE	Die Angabe, ob die Auswahlliste positiv oder negativ aufgefasst werden soll
EMBEDDED	Das Intervall aus dem nachträglich eine Auswahl getroffen wird
SELECTION	Mit Selection werden einzelne Blöcke selektiert. Siehe auch Tabelle 22.22 auf Seite 332
FILTER	Name(n) der Intervalle die den Output dieses Intervalls weiter filtern (Multiplikation) Siehe auch Tabelle 22.23 auf Seite 332
DISPATCHER	Die Dispatch Tabelle ist nur für Dispatchintervalle relevant. Sie gibt Detailinformation zu der Dispatch Funktionalität. Siehe auch Tabelle 22.24 auf Seite 332
HIERARCHY	Die Hierarchy Tabelle zeigt die hierarchische Struktur eines Intervalls. Siehe auch Tabelle 22.25 auf Seite 333
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
OWNER_OBJ_TYPE	Falls ein Intervall zu einem anderen Objekt gehört, steht in diesem Feld der Typ des übergeordneten Objekts.
OWNER_OBJ_ID	Falls ein Intervall zu einem anderen Objekt gehört, steht in diesem Feld die Id des übergeordneten Objekts.
SE_ID	Dieses Feld wurde noch nicht beschrieben
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
EDGES	Dieses Feld wurde noch nicht beschrieben Siehe auch Tabelle 22.26 auf Seite 336

Tabelle 22.21: Beschreibung der Output-Struktur des show interval Statements

SELECTION Das Layout der SELECTION Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
VALUE	Nummer des ausgewählten Edges
PERIOD_FROM	Anfang der Periode in der alle auftretenden Edges als ausgewählt gelten
PERIOD_TO	Ende der Periode in der alle auftretenden Edges als ausgewählt gelten

Tabelle 22.22: Output-Struktur der show interval Subtabelle

FILTER Das Layout der FILTER Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
CHILD	Name des filternden Intervalls

Tabelle 22.23: Output-Struktur der show interval Subtabelle

DISPATCHER Das Layout der DISPATCHER Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SEQNO	Das Feld **seqno** definiert die Reihenfolge der Dispatch-Regeln.
NAME	Um die Dispatch-Regeln einfacher verstehen zu können, hat jede Regel einen Namen.
SELECT_INTERVAL_ID	Die Id des Intervalls, das die Zeiträume definiert, in denen die Regel zutrifft.
SELECT_INTERVAL_NAME	Der Name des Intervalls, das die Zeiträume definiert, in denen die Regel zutrifft.
FILTER_INTERVAL_ID	Die Id des Intervalls, das zu den vom Select Intervall definierten Zeiten evaluiert werden soll.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
FILTER_INTERVAL_NAME	Der Name des Intervalls, das zu den vom Select Intervall definierten Zeiten evaluiert werden soll.
IS_ENABLED	Dieses Feld gibt an ob die Regel ausgewertet werden soll oder nicht.
IS_ACTIVE	Dieses Feld definiert ob das Filter Intervall evaluiert wird oder nicht. Wird das Filter Intervall nicht evaluiert, wird nichts durchgelassen.

Tabelle 22.24: Output-Struktur der show interval Subtabelle

HIERARCHY Das Layout der HIERARCHY Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
LEVEL	Das **level** gibt an auf welcher Hierarchieebene sich das beschriebene Objekt befindet.
ROLE	Das Feld **role** gibt an welche Rolle das Objekt hat. Es gibt folgende Möglichkeiten:
PARENT	Das Feld **parent** gibt an welches Objekt das übergeordnete Objekt in der Hierarchie ist.
NAME	Der Name des Intervalls
SEQNO	Das Feld **seqno** definiert die Reihenfolge der Dispatch-Regeln.
SELECT_INTERVAL_NAME	Der Name des Intervalls, das die Zeiträume definiert, in denen die Regel zutrifft.

Rolle	Bedeutung
HEAD	Top level Objekt
FILTER	Filter Intervall
EMBEDDED	Embedded Intervall
DISPATCH	Dispatch Intervall
DISPATCH_SELECT	Select Intervall einer Dispatch Regel
DISPATCH_FILTER	Filter Intervall einer Dispatch Regel

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
FILTER_INTERVAL_NAME	Der Name des Intervalls, das zu den vom Select Intervall definierten Zeiten evaluiert werden soll.
IS_ENABLED	Dieses Feld gibt an ob die Regel ausgewertet werden soll oder nicht.
IS_ACTIVE	Dieses Feld definiert ob das Filter Intervall evaluiert wird oder nicht. Wird das Filter Intervall nicht evaluiert, wird nichts durchgelassen.
OWNER	Die Gruppe die Eigentümer des Objektes ist
STARTTIME	Der Anfang des Intervalls. Vor dieser Zeit werden keine Flanken generiert.
ENDTIME	Das Ende des Intervalls. Nach dieser Zeit werden keine Flanken generiert.
BASE	Die Periode des Intervalls
DURATION	Die Dauer eines Blocks
SYNCTIME	Die Zeit mit der das Intervall synchronisiert wird. Die erste Periode des Intervalls startet zu dieser Zeit.
INVERSE	Die Angabe, ob die Auswahlliste positiv oder negativ aufgefasst werden soll
EMBEDDED	Das Intervall aus dem nachträglich eine Auswahl getroffen wird
SELECTION	Mit Selection werden einzelne Blöcke selektiert.
FILTER	Name(n) der Intervalle die den Output dieses Intervalls weiter filtern (Multiplikation)
DISPATCHER	Name(n) der Intervalle die den Output dieses Intervalls weiter filtern (Multiplikation)
OWNER_OBJ_TYPE	Falls ein Intervall zu einem anderen Objekt gehört, steht in diesem Feld der Typ des übergeordneten Objekts.
OWNER_OBJ_ID	Falls ein Intervall zu einem anderen Objekt gehört, steht in diesem Feld die Id des übergeordneten Objekts.

Tabelle 22.25: Output-Struktur der show interval Subtabelle

EDGES Das Layout der EDGES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung

Tabelle 22.26: Output-Struktur der show interval Subtabelle

show job

Zweck

Das *show job* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Job zu bekommen. *Zweck*

Syntax

Die Syntax des *show job* Statements ist *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**
- **clone**
- **comment**
- **disable**
- **enable**
- **ignore named resource**
- **ignore resource**
- **ignore dependency** [**recursive**]
- **job in error**
- **kill**
- **renice**
- **rerun** [**recursive**]
- **restartable**
- **resume**
- **set exit state**
- **set parameter**
- **set resource state**
- **set state**
- **set warning**
- **submit** [**suspend**]

| **suspend**
| **timeout**
| **trigger failure**
| **trigger submit**
| **unreachable**

Beschreibung

Beschreibung Mit dem *show job* Statement bekommt man ausführliche Informationen über den spezifizierten Job. Der Job kann mittels seiner Id, oder aber, wenn beim Submit ein Submit Tag spezifiziert wurde, mittels des Submit Tags spezifiziert werden.
Die Filter-Option dient zum Selektieren von Audit-Einträgen. Ohne Angabe der Filter-Option werden alle Audit-Einträge gezeigt. Ansonsten werden nur die Einträge der im Filter spezifizierten Typen ausgegeben.
Die **recursive audit** Option sammelt alle Audit-Meldungen des gezeigten Jobs, sowie die seiner direkten oder indirekten Children.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SE_NAME	Der komplette Pfadname des Objektes
SE_OWNER	Eigentümer des Objekts
SE_TYPE	Der Se_Type ist der Objekttyp (JOB, BATCH oder MILESTONE).
SE_RUN_PROGRAM	Die Run_Program Zeile der Job Definition
SE_RERUN_PROGRAM	Die Rerun_Program Zeile der Job Definition
SE_KILL_PROGRAM	Die Kill_Program Zeile der Job Definition
SE_WORKDIR	Die Workdir der Job Definition
SE_LOGFILE	Das Logfile der Job Definition
SE_TRUNC_LOG	Gibt an, ob das Logfile gekürzt werden soll, bevor der Prozess startet, oder ob die Loginformation angehängt werden soll.
SE_ERRLOGFILE	Das Error Logfile der Job Definition

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
SE_TRUNC_ERRLOG	Gibt an, ob das Logfile gekürzt werden soll, bevor der Prozess startet, oder ob die Loginformation angehängt werden soll.
SE_EXPECTED_RUNTIME	Die erwartete Laufzeit der Job Definition
SE_PRIORITY	Priorität/Nice Value der Job Definition
SE_SUBMIT_SUSPENDED	Das Suspend Flag des Objekts
SE_MASTER_SUBMITTABLE	Das Master_Submittable Flag des Objekts
SE_DEPENDENCY_MODE	Der Dependency_Mode des Objekts
SE_ESP_NAME	Der Exit State Profile des Objekts
SE_ESM_NAME	Das Exit State Mapping der Job Definition
SE_ENV_NAME	Das Environment der Job Definition
SE_FP_NAME	Der Footprint der Job Definition
MASTER_ID	Hierbei handelt es sich um die Id des Master Jobs.
TIME_ZONE	Dieses Feld wurde noch nicht beschrieben
CHILD_TAG	Tag zur ausschließlichen Erkennung von Jobs die mehrmals als Children desselben Jobs submitted wurden
SE_VERSION	Die Ausführung von Definitionen die für dieses Submitted Entity gültig sind
OWNER	Die Gruppe die Eigentümer des Objektes ist
PARENT_ID	Hierbei handelt es sich um die Id des Parents.
SCOPE_ID	Der Scope, bzw. Jobserver, dem der Job zugeordnet ist
HTTPHOST	Der Hostname des Scopes für den Zugriff auf Logfiles via HTTP
HTTPPORT	Die HTTP Portnummer des Jobservers für den Zugriff auf Logfiles via HTTP
IS_STATIC	Flag, das statische oder dynamische Submits von diesem Job anzeigt
MERGE_MODE	Zeigt an wie mehrere Submits von demselben definierten Objekt im aktuellen Master Run gehandhabt werden
STATE	Der State ist der aktuelle Status des Jobs.
IS_DISABLED	Zeigt an, ob der Job bzw. Batch disabled ist.
IS_PARENT_DISABLED	Zeigt an, ob der Job bzw. Batch disabled ist.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_CANCELLED	Zeigt an, ob ein Cancel auf den Job ausgeführt wurde
JOB_ESD_ID	Der job_esd ist der Exit State des Jobs.
JOB_ESD_PREF	Die Präferenz zum Mischen der Job Exit States mit den Child States
JOB_IS_FINAL	Dieses Feld gibt an, ob der Job selbst final ist.
JOB_IS_RESTARTABLE	Flag das anzeigt das dieser Job restartable ist
FINAL_ESD_ID	Der finale (merged) Exit State des Objekts
EXIT_CODE	Der Exit_Code des ausgeführten Prozesses
COMMANDLINE	Die erstellte Kommandozeile die bei der ersten Ausführung genutzt wird
RR_COMMANDLINE	Erstellte Rerun Kommandozeile die bei der letzten Rerun-Ausführung genutzt wird
WORKDIR	Name des Working Directorys des Nutzprozesses
LOGFILE	Name des Logfiles des Nutzprozesses. Hier werden die Ausgaben nach `stdout` protokolliert.
ERRLOGFILE	Das erstellte Error Logfile
PID	Bei der PID handelt es sich um die Prozessidentifikationsnummer des überwachenden Jobserverprozesses auf dem jeweiligen Hostsystem.
EXT_PID	Die EXT_PID ist die Prozessidentifikationsnummer des Nutzprozesses.
ERROR_MSG	Die Fehlermeldung die beschreibt warum der Job in den Status error gewechselt ist
KILL_ID	Die Submitted Entity Id des submitteten Kill Jobs
KILL_EXIT_CODE	Der Exit Code der letzten Kill Program Ausführung
IS_SUSPENDED	Dieses Feld gibt an, ob der Job oder Batch selbst suspended ist.
IS_SUSPENDED_LOCAL	Flag, das anzeigt, ob das Objekt lokal suspended ist (bei Restart Triggern mit suspend)
PRIORITY	Die statische Priorität eines Jobs. Diese setzt sich zusammen aus der definierten Priorität und den Nice Values der Parents.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RAW_PRIORITY	Der uninterpretierte Prioritätswert des Jobs. Anders als die Priorität, ist dieser Wert praktisch nicht limitiert. Er wird benötigt um nach Manipulationen über Nice Profiles die korrekte Priorität wiederherstellen zu können.
NICEVALUE	Die aktuelle Nice Value des Jobs
NP_NICEVALUE	Der Np_Nicevalue ist der Nice Value, der als Folge von Aktivierungen (und Deaktivierungen) von Nice Profiles entsteht.
MIN_PRIORITY	Der minimale Wert der dynamischen Priorität
AGING_AMOUNT	Der Aging_Amount gibt an nach wievielen Zeiteinheiten die dynamische Priorität eines Jobs um einen Punkt hochgesetzt wird.
AGING_BASE	Die Aging_Base gibt an um welche Zeiteinheit es beim Aging Amount geht.
DYNAMIC_PRIORITY	Die dynamische Priorität des Jobs. Diese ist die abhängig von der Wartezeit korrigierte statische Priorität.
PARENT_SUSPENDED	Dieses Feld gibt an, ob der Job über einen seiner Parents suspended ist (true) oder nicht (false).
SUBMIT_TS	Hierbei handelt es sich um den Zeitpunkt zu dem der Job submitted wird.
RESUME_TS	Der Zeitpunkt zu dem der Job automatisch resumed wird
SYNC_TS	Der Zeitpunkt zu dem der Job in den Status synchronize_wait gewechselt ist
RESOURCE_TS	Der Zeitpunkt zu dem der Job in den Status resource_wait gewechselt ist
RUNNABLE_TS	Der Zeitpunkt an dem der Job den Status runnable erreicht hat
START_TS	Der Zeitpunkt zu dem der Job vom Jobserver als gestarted gemeldet wurde
FINISH_TS	Hierbei handelt es sich um den Zeitpunkt zu dem der Job beendet wird.
FINAL_TS	Der Zeitpunkt zu der der Job in den State final übergegangen ist
CNT_SUBMITTED	Die Anzahl der Children im Status submitted

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CNT_DEPENDENCY_WAIT	Die Anzahl der Children im Status dependency_wait
CNT_SYNCHRONIZE_WAIT	Die Anzahl der Children im Status synchronize_wait
CNT_RESOURCE_WAIT	Die Anzahl der Children im Status resource_wait
CNT_RUNNABLE	Die Anzahl der Children im Status runnable
CNT_STARTING	Die Anzahl der Children im Status starting
CNT_STARTED	Die Anzahl der Children im Status started
CNT_RUNNING	Die Anzahl der Children im Status running
CNT_TO_KILL	Die Anzahl der Children im Status to_kill
CNT_KILLED	Die Anzahl der Children im Status killed
CNT_CANCELLED	Die Anzahl der Children im Status cancelled
CNT_FINISHED	Die Anzahl der Children im Status finished
CNT_FINAL	Die Anzahl der Children im Status final
CNT_BROKEN_ACTIVE	Die Anzahl der Children im Status broken_active
CNT_BROKEN_FINISHED	Die Anzahl der Children im Status broken_finished
CNT_ERROR	Die Anzahl der Children im Status error
CNT_RESTARTABLE	Die Anzahl der Children im Status restartable
CNT_UNREACHABLE	Die Anzahl der Children im Status unreachable
CNT_WARN	Die Anzahl der Children für die eine Warnung vorliegt
WARN_COUNT	Dies ist die Anzahl unbehandelter Warnings.
IDLE_TIME	Die Zeit die der Job idle war, beziehungsweise gewartet hat
DEPENDENCY_WAIT_TIME	Die Zeit in der der Job im Status Dependency_Wait war
SUSPEND_TIME	Die Zeit in der der Job Suspended war
SYNC_TIME	Die Zeit in der der Job im Status Synchronize_Wait war
RESOURCE_TIME	Die Zeit in der der Job im Status Resource_Wait war
JOBSERVER_TIME	Die Zeit in der der Job unter der Kontrolle eines Jobservers war

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RESTARTABLE_TIME	Die Zeit in der der Job in einem Restartable State war, wartend auf einen Rerun oder Cancel
CHILD_WAIT_TIME	Die Zeit in der der Job gewartet hat, bis seine Children einen Final State erreicht haben
PROCESS_TIME	Die Zeit in der ein Job ausgeführt wurde, oder ausgeführt hätte werden können, wenn genug Resourcen zur Verfügung gestanden hätten. Somit die Zeit zwischen Submit und Final ohne die Zeit in der er auf Abhängigkeiten gewartet hat.
ACTIVE_TIME	Die Zeit in der der Job aktiv war
IDLE_PCT	Der Prozentsatz der gesamten Zeit in der der Job als aktiv galt
CHILDREN	Die Anzahl Children des Jobs oder Batches Siehe auch Tabelle 22.28 auf Seite 344
PARENTS	Tabelle der Parents Siehe auch Tabelle 22.29 auf Seite 345
PARAMETER	Tabelle der Parameter Siehe auch Tabelle 22.30 auf Seite 346
REQUIRED_JOBS	Tabelle der benötigten Jobs Siehe auch Tabelle 22.31 auf Seite 346
DEPENDENT_JOBS	Tabelle der abhängigen Jobs Siehe auch Tabelle 22.32 auf Seite 349
REQUIRED_RESOURCES	Tabelle der benötigten Resources Siehe auch Tabelle 22.33 auf Seite 351
SUBMIT_PATH	Der Pfad vom Job bis zum Master über die Submit Hierarchy
IS_REPLACED	Dieses Feld gibt an, ob der Job oder Batch durch einen anderen ersetzt wurde.
TIMEOUT_AMOUNT	Die Zeit die der Job maximal auf seine Resource wartet
TIMEOUT_BASE	Die Einheit die genutzt wird um das Timeout in Sekunden, Minuten, Stunden oder Tagen zu spezifizieren
TIMEOUT_STATE	Das Timeout des Scheduling Entities
RERUN_SEQ	Reihenfolge des Rerun
AUDIT_TRAIL	Tabelle der Protokolleinträge

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
	Siehe auch Tabelle 22.34 auf Seite 353
CHILD_SUSPENDED	Die Anzahl der Children die suspended wurden
CNT_PENDING	Die Anzahl der Children im Status pending
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
SE_PRIVS	Privilegien auf das Scheduling Entity
SUBMITTAG	Einmaliger Marker der zur Submit-Zeit gegeben ist
UNRESOLVED_HANDLING	Bestimmt wie man im Fall, dass das benötigte Objekt nicht gefunden werden kann, handeln soll.
DEFINED_RESOURCES	Tabelle der Defined_Resources des Objekts
	Siehe auch Tabelle 22.35 auf Seite 354
RUNS	Tabelle der Defined_Resources des Objekts
	Siehe auch Tabelle 22.36 auf Seite 354

Tabelle 22.27: Beschreibung der Output-Struktur des show job Statements

CHILDREN Das Layout der CHILDREN Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
CHILDID	Die Submitted Entity Id des Childs
CHILDPRIVS	Die Privilegien auf das Child-Objekt
CHILDSENAME	Der Name des Child-Objekts
CHILDSETYPE	Die Art des Child-Objekts
CHILDSEPRIVS	Die Privilegien auf die Definition des Child-Objekts
PARENTID	Die Id des Parents
PARENTPRIVS	Die Privilegien auf das Parent-Objekt
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
PARENTSENAME	Der Name des Parent-Objekts
PARENTSETYPE	Die Art des Parent-Objekts
PARENTSEPRIVS	Die Privilegien für die Job Definition die zum Parent gehört
IS_STATIC	Statisches Flag der Hierarchy Definition
PRIORITY	Die Priorität auf die Hierarchy Definition
SUSPEND	Der Suspend Modus der Hierarchy Definition
MERGE_MODE	Der Merge Modus der Hierarchy Definition
EST_NAME	Der Name der Exit State Translation der Hierarchy Definition
IGNORED_DEPENDENCIES	Ignored Dependencies Flag der Hierarchy Definition

Tabelle 22.28: Output-Struktur der show job Subtabelle

PARENTS Das Layout der PARENTS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
CHILDID	Die Submitted Entity Id des Childs
CHILDPRIVS	Die Privilegien auf das Child-Objekt
CHILDSENAME	Der Name des Child-Objekts
CHILDSETYPE	Die Art des Child-Objekts
CHILDSEPRIVS	Die Privilegien auf die Definition des Child-Objekts
PARENTID	Die Id des Parents
PARENTPRIVS	Die Privilegien auf das Parent-Objekt
PARENTSENAME	Der Name des Parent-Objekts
PARENTSETYPE	Die Art des Parent-Objekts
PARENTSEPRIVS	Die Privilegien für die Job Definition die zum Parent gehört
IS_STATIC	Statisches Flag der Hierarchy Definition
PRIORITY	Die Priorität auf die Hierarchy Definition
SUSPEND	Der Suspend Modus der Hierarchy Definition
MERGE_MODE	Der Merge Modus der Hierarchy Definition
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
EST_NAME	Der Name der Exit State Translation der Hierarchy Definition
IGNORED_DEPENDENCIES	Ignored Dependencies Flag der Hierarchy Definition

Tabelle 22.29: Output-Struktur der show job Subtabelle

PARAMETER Das Layout der PARAMETER Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
NAME	Der Name des Parameters, der Variablen oder des Ausdrucks
TYPE	Die Art des Parameters, der Variablen oder des Ausdrucks
VALUE	Der Wert des Parameters, der Variablen oder des Ausdrucks

Tabelle 22.30: Output-Struktur der show job Subtabelle

REQUIRED_JOBS Das Layout der REQUIRED_JOBS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
DEPENDENT_ID	Id des abhängigen Submitted Entities
DEPENDENT_PATH	Der Pfad vom Job bis zum Master über die Submit Hierarchy
DEPENDENT_PRIVS	Die Privilegien auf das abhängige Objekt
DEPENDENT_ID_ORIG	Id des originalen abhängigen Submitted Entities, auf dem die Abhängigkeit für Abhängigkeiten, die von den Parents geerbt wurden, definiert ist.
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite

Feld	Beschreibung
DEPENDENT_PATH_ORIG	Der Pfad vom abhängigen Objekt bis zum Master über die Submit Hierarchy
DEPENDENT_PRIVS_ORIG	Die Privilegien auf das originale abhängige Objekt
DEPENDENCY_OPERATION	Definiert, ob alle oder nur manche Abhängigkeiten des originalen Objektes erfüllt sein müssen
REQUIRED_ID	Id des benötigten Submitted Entities
REQUIRED_PATH	Der Pfad vom benötigten Objekt bis zum Master über die Submit Hierarchy
REQUIRED_PRIVS	Die Privilegien auf das benötigte Objekt
STATE	Der Status der Abhängigkeit (OPEN, FULFILLED oder FAILED)
DD_ID	Id des Dependency Definition Objekts
DD_NAME	Name der Dependency Definition
DD_DEPENDENTNAME	Der komplette Pfadname des Objekts
DD_DEPENDENTTYPE	Die Art des abhängigen Objekts
DD_DEPENDENTPRIVS	Privilegien auf das abhängige Objekt
DD_REQUIREDNAME	Pfadname der Definition des abhängigen Objektes
DD_REQUIREDTYPE	Die Art des benötigten Objektes
DD_REQUIREDPRIVS	Die Privilegien auf das benötigte Objekt
DD_UNRESOLVED_HANDLING	Spezifiziert wie man mit nicht auflösbaren Abhängigkeiten beim Submit umgehen soll
DD_STATE_SELECTION	Gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigten Exit States expliziert aufgeführt sein.
DD_MODE	Gibt an, ob nur der benötigte Job selbst, oder der benötigte Job mit seinen Children, final sein muss
DD_STATES	Liste mit Exit States die das benötigte Objekt erreichen muss um die Abhängigkeit zu erfüllen
JOB_STATE	In der Liste Job State kann nach Jobs gefiltert werden, die den eingetragenen Job State haben.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_SUSPENDED	Dieses Feld gibt an, ob der Job oder Batch selbst suspended ist.
PARENT_SUSPENDED	Dieses Feld gibt an, ob der Job über einen seiner Parents suspended ist (true) oder nicht (false).
CNT_SUBMITTED	Die Anzahl der Children im Status submitted
CNT_DEPENDENCY_WAIT	Die Anzahl der Children im Status dependency_wait
CNT_SYNCHRONIZE_WAIT	Die Anzahl der Children im Status synchronize_wait
CNT_RESOURCE_WAIT	Die Anzahl der Children im Status resource_wait
CNT_RUNNABLE	Die Anzahl der Children im Status runnable
CNT_STARTING	Die Anzahl der Children im Status starting
CNT_STARTED	Die Anzahl der Children im Status started
CNT_RUNNING	Die Anzahl der Children im Status running
CNT_TO_KILL	Die Anzahl der Children im Status to_kill
CNT_KILLED	Die Anzahl der Children im Status killed
CNT_CANCELLED	Die Anzahl der Children im Status cancelled
CNT_FINISHED	Die Anzahl der Children im Status finished
CNT_FINAL	Die Anzahl der Children im Status final
CNT_BROKEN_ACTIVE	Die Anzahl der Children im Status broken_active
CNT_BROKEN_FINISHED	Die Anzahl der Children im Status broken_finished
CNT_ERROR	Die Anzahl der Children im Status error
CNT_RESTARTABLE	Die Anzahl der Children im Status restartable
CNT_UNREACHABLE	Die Anzahl der Children im Status unreachable
JOB_IS_FINAL	Die Anzahl der Children im Is_Final Status
CHILD_TAG	Tag zur ausschließlichen Erkennung von Jobs die mehrmals als Children desselben Jobs submitted wurden
FINAL_STATE	Der endgültige Status eines Jobs
CHILDREN	Die Anzahl Children des Jobs oder Batches
IGNORE	Flag, das anzeigt, ob die Resource Allocation ignoriert wird
CHILD_SUSPENDED	Die Anzahl der Children die suspended wurden

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CNT_PENDING	Die Anzahl der Children im Status pending
DD_CONDITION	Die Condition die zusätzlich erfüllt sein muss damit die Abhängigkeit erfüllt ist

Tabelle 22.31: Output-Struktur der show job Subtabelle

DEPENDENT_JOBS Das Layout der DEPENDENT_JOBS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
DEPENDENT_ID	Id des abhängigen Submitted Entities
DEPENDENT_PATH	Der Pfad vom Job bis zum Master über die Submit Hierarchy
DEPENDENT_PRIVS	Die Privilegien auf das abhängige Objekt
DEPENDENT_ID_ORIG	Id des originalen abhängigen Submitted Entities, auf dem die Abhängigkeit für Abhängigkeiten, die von den Parents geerbt wurden, definiert ist.
DEPENDENT_PATH_ORIG	Der Pfad vom abhängigen Objekt bis zum Master über die Submit Hierarchy
DEPENDENT_PRIVS_ORIG	Die Privilegien auf das originale abhängige Objekt
DEPENDENCY_OPERATION	Definiert, ob alle oder nur manche Abhängigkeiten des originalen Objektes erfüllt sein müssen
REQUIRED_ID	Id des benötigten Submitted Entities
REQUIRED_PATH	Der Pfad vom benötigten Objekt bis zum Master über die Submit Hierarchy
REQUIRED_PRIVS	Die Privilegien auf das benötigte Objekt
STATE	Der Status der Abhängigkeit (OPEN, FULFILLED oder FAILED)
DD_ID	Id des Dependency Definition Objekts
DD_NAME	Name der Dependency Definition
DD_DEPENDENTNAME	Der komplette Pfadname des Objekts
DD_DEPENDENTTYPE	Die Art des abhängigen Objekts

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
DD_DEPENDENTPRIVS	Privilegien auf das abhängige Objekt
DD_REQUIREDNAME	Pfadname der Definition des abhängigen Objektes
DD_REQUIREDTYPE	Die Art des benötigten Objektes
DD_REQUIREDPRIVS	Die Privilegien auf das benötigte Objekt
DD_UNRESOLVED_HANDLING	Spezifiziert wie man mit nicht auflösbaren Abhängigkeiten beim Submit umgehen soll
DD_STATE_SELECTION	Gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigten Exit States expliziert aufgeführt sein.
DD_MODE	Gibt an, ob nur der benötigte Job selbst, oder der benötigte Job mit seinen Children, final sein muss
DD_STATES	Liste mit Exit States die das benötigte Objekt erreichen muss um die Abhängigkeit zu erfüllen
JOB_STATE	In der Liste Job State kann nach Jobs gefiltert werden, die den eingetragenen Job State haben.
IS_SUSPENDED	Dieses Feld gibt an, ob der Job oder Batch selbst suspended ist.
PARENT_SUSPENDED	Dieses Feld gibt an, ob der Job über einen seiner Parents suspended ist (true) oder nicht (false).
CNT_SUBMITTED	Die Anzahl der Children im Status submitted
CNT_DEPENDENCY_WAIT	Die Anzahl der Children im Status dependency_wait
CNT_SYNCHRONIZE_WAIT	Die Anzahl der Children im Status synchronize_wait
CNT_RESOURCE_WAIT	Die Anzahl der Children im Status resource_wait
CNT_RUNNABLE	Die Anzahl der Children im Status runnable
CNT_STARTING	Die Anzahl der Children im Status starting
CNT_STARTED	Die Anzahl der Children im Status started
CNT_RUNNING	Die Anzahl der Children im Status running
CNT_TO_KILL	Die Anzahl der Children im Status to_kill
CNT_KILLED	Die Anzahl der Children im Status killed
CNT_CANCELLED	Die Anzahl der Children im Status cancelled

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CNT_FINISHED	Die Anzahl der Children im Status finished
CNT_FINAL	Die Anzahl der Children im Status final
CNT_BROKEN_ACTIVE	Die Anzahl der Children im Status broken_active
CNT_BROKEN_FINISHED	Die Anzahl der Children im Status broken_finished
CNT_ERROR	Die Anzahl der Children im Status error
CNT_RESTARTABLE	Die Anzahl der Children im Status restartable
CNT_UNREACHABLE	Die Anzahl der Children im Status unreachable
JOB_IS_FINAL	Die Anzahl der Children im Is_Final Status
CHILD_TAG	Tag zur ausschließlichen Erkennung von Jobs die mehrmals als Children desselben Jobs submitted wurden
FINAL_STATE	Der endgültige Status eines Jobs
CHILDREN	Die Anzahl Children des Jobs oder Batches
IGNORE	Flag, das anzeigt, ob die Resource Allocation ignoriert wird
CHILD_SUSPENDED	Die Anzahl der Children die suspended wurden
CNT_PENDING	Die Anzahl der Children im Status pending
DD_CONDITION	Die Condition die zusätzlich erfüllt sein muss damit die Abhängigkeit erfüllt ist

Tabelle 22.32: Output-Struktur der show job Subtabelle

REQUIRED_RESOURCES Das Layout der REQUIRED_RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
SCOPE_ID	Id des Scopes der die Resource allokiert hat
SCOPE_NAME	Der vollqualifizierte Name des Scopes
SCOPE_TYPE	Die Art des Scopes (SCOPE oder SERVER, FOLDER, BATCH oder JOB)
SCOPE_PRIVS	Die Privilegien auf dem Scope
RESOURCE_ID	Id der Required Resource
RESOURCE_NAME	Kategorischer Pfadname der Requested Resource
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RESOURCE_USAGE	Die Anwendung der benötigten Resource (STATIC, SYSTEM oder SYNCHRONIZING)
RESOURCE_OWNER	Name des Eigentümers der Requested Resource
RESOURCE_PRIVS	Die Privilegien auf die Requested Resource
RESOURCE_STATE	Der Status der Requested Resource
RESOURCE_TIMESTAMP	Der letzte Zeitpunkt zu dem der Resource Status dieser Resource gesetzt wurde
REQUESTABLE_AMOUNT	Die Menge der Resourcen die maximal von einem Job angefordert werden darf
TOTAL_AMOUNT	Der komplette Amount der allokiert werden kann
FREE_AMOUNT	Die freie Menge die allokiert werden darf
REQUESTED_AMOUNT	Hierbei handelt es sich um die Anforderungsmenge
REQUESTED_LOCKMODE	Der beantragte Lockmode
REQUESTED_STATES	Der beantragte Resource State
RESERVED_AMOUNT	Die Menge die von der Requested Resource reserviert ist
ALLOCATED_AMOUNT	Die Menge die von der Requested Resource allokiert wurde
ALLOCATED_LOCKMODE	Der, von der Requested Resource aktuell allokierte Lockmode
IGNORE	Flag, das anzeigt, ob die Resource Allocation ignoriert wird
STICKY	Flag, das anzeigt, ob es eine Sticky Resource Allocation ist
STICKY_NAME	Optionaler Name der Sticky Anforderung
STICKY_PARENT	Parent Job innerhalb dessen die Sticky Anforderung gilt
STICKY_PARENT_TYPE	Typ des Parents innerhalb dessen die Sticky Anforderung gilt
ONLINE	Flag, das anzeigt, ob die Resource für eine Allocation erhältlich ist
ALLOCATE_STATE	Der Status der Alocation (RESERVED, ALLOCATED, AVAILABLE oder BLOCKED)

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPIRE	Zeitpunkt, der angibt, wie alt eine Resource maximal bzw. minimal sein darf, je nachdem, ob der Expire positiv oder negativ ist
EXPIRE_SIGN	Sign of the expiration condition, +/- indicating younger/older than Vorzeichen der Expiration Bedingung, +/-
IGNORE_ON_RERUN	Dieser Flag gibt an ob die Aktualitätsbedingung im Falle eines Reruns ignoriert werden soll.
DEFINITION	Die Speicherstelle der Resource Definition

Tabelle 22.33: Output-Struktur der show job Subtabelle

AUDIT_TRAIL Das Layout der AUDIT_TRAIL Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
USERNAME	Benutzername der diese Audit-Eingabe verursacht
TIME	Der Zeitpunkt zu dem diese Audit-Eingabe erstellt wurde
TXID	Transaktionsnummer der Änderung
ACTION	Aktion die diese Audit-Eingabe verursacht
ORIGINID	Die originale Object Id die diese Audit-Eingabe verursacht
JOBID	Die Id des Jobs für den der Auditeintrag gilt
JOBNAME	Der Name des Jobs für den der Auditeintrag gilt
COMMENT	Kommentar zum Objekt, wenn vorhanden
INFO	Zusätzliche Systeminformation über das Action Event das die Audit-Eingabe verursacht hat

Tabelle 22.34: Output-Struktur der show job Subtabelle

DEFINED_RESOURCES Das Layout der DEFINED_RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Id der Defined Resource
RESOURCE_NAME	Kompletter Pfadname des Defined Objects
RESOURCE_USAGE	Die Anwendung der benötigten Resource (STATIC, SYSTEM oder SYNCHRONIZING)
RESOURCE_OWNER	Der Eigentümer der Resource
RESOURCE_PRIVS	Die Privilegien auf die Resource
RESOURCE_STATE	Der aktuelle Status der Resource
RESOURCE_TIMESTAMP	Der letzte Zeitpunkt zu dem der Resource Status dieser Resource gesetzt wurde
REQUESTABLE_AMOUNT	Die Menge der Resourcen die maximal von einem Job angefordert werden darf
TOTAL_AMOUNT	Der komplette Amount der allokiert werden kann
FREE_AMOUNT	Die freie Menge die allokiert werden darf
ONLINE	Zeigt an, ob die Resource allokiert werden kann oder nicht

Tabelle 22.35: Output-Struktur der show job Subtabelle

RUNS Das Layout der RUNS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
RERUN_SEQ	Reihenfolge des Rerun
SCOPE_ID	Der Scope, bzw. Jobserver, dem der Job zugeordnet ist
HTTPHOST	Der Hostname des Scopes für den Zugriff auf Logfiles via HTTP
HTTPPORT	Die HTTP Portnummer des Jobservers für den Zugriff auf Logfiles via HTTP
JOB_ESD_ID	Der job_esd ist der Exit State des Jobs.
EXIT_CODE	Der Exit_Code des ausgeführten Prozesses
COMMANDLINE	Die erstellte Kommandozeile die bei der ersten Ausführung genutzt wird
WORKDIR	Name des Working Directorys des Nutzprozesses

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
LOGFILE	Name des Logfiles des Nutzprozesses. Hier werden die Ausgaben nach `stdout` protokolliert.
ERRLOGFILE	Das erstellte Error Logfile
EXT_PID	Die EXT_PID ist die Prozessidentifikationsnummer des Nutzprozesses.
SYNC_TS	Der Zeitpunkt zu dem der Job in den Status synchronize_wait gewechselt ist
RESOURCE_TS	Der Zeitpunkt zu dem der Job in den Status resource_wait gewechselt ist
RUNNABLE_TS	Der Zeitpunkt an dem der Job den Status runnable erreicht hat
START_TS	Der Zeitpunkt zu dem der Job vom Jobserver als gestarted gemeldet wurde
FINISH_TS	Hierbei handelt es sich um den Zeitpunkt zu dem der Job beendet wird.

Tabelle 22.36: Output-Struktur der show job Subtabelle

show job definition

Zweck

Zweck Das *show job definition* Statement wird eingesetzt um detaillierte Informationen über die definierte Job Definition zu bekommen.

Syntax

Syntax Die Syntax des *show job definition* Statements ist

show job definition *folderpath*

Beschreibung

Beschreibung Mit dem *show job definition* Statement bekommt man ausführliche Informationen über die spezifizierte Job Definition.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der vollständige Pfadname der Job Definition
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Type gibt die Art des Objektes an. Es gibt folgende Optionen: Batch, Milestone, Job und Folder.
INHERIT_PRIVS	Vom übergeordneten Ordner zu erbende Privilegien
RUN_PROGRAM	Im Feld Run_Program kann eine Kommandozeile angegeben werden, die das Skript oder Programm startet.
RERUN_PROGRAM	Das Feld Rerun_Program gibt das Kommando an, welches bei einer wiederholten Ausführung des Jobs nach einem Fehlerzustand (rerun) ausgeführt werden soll.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
KILL_PROGRAM	Das Kill_Program bestimmt welches Programm ausgeführt werden soll, um einen aktuell laufenden Job zu beenden.
WORKDIR	Hierbei handelt es sich um das Working Directory des aktuellen Jobs.
LOGFILE	Das Feld Logfile gibt an in welche Datei alle normalen Ausgaben des Run_Programs ausgegeben werden sollen. Unter normalen Ausgaben sind alle Ausgaben gemeint, die den normalen Ausgabekanal (STDOUT unter UNIX) benutzen.
TRUNC_LOG	Gibt an, ob das Logfile erneuert werden soll oder nicht
ERRLOGFILE	Das Feld Errorlogfile gibt an, in welcher Datei alle Fehlerausgaben des Run_Programs ausgegeben werden sollen.
TRUNC_ERRLOG	Gibt an, ob das Error Logfile erneuert werden soll oder nicht
EXPECTED_RUNTIME	Die Expected_Runtime beschreibt die erwartete Zeit die ein Job für seine Ausführung benötigt.
EXPECTED_FINALTIME	Die Expected Finaltime beschreibt die erwartete Zeit die ein Job oder Batch samt Children für seine Ausführung benötigt.
PRIORITY	Das Feld Priority gibt an mit welcher Dringlichkeit ein Prozess, falls er gestartet werden soll, vom Scheduling System berücksichtigt wird.
MIN_PRIORITY	Minimale effektive Priorität die durch das natürliche Altern erreicht werden kann
AGING_AMOUNT	Die Anzahl Zeiteinheiten nach der die effektive Priorität um 1 erhöht wird
AGING_BASE	Die Zeiteinheit die für das Alterungsintervall genutzt wird
SUBMIT_SUSPENDED	Flag das angibt, ob das Objekt nach dem Submit suspended werden soll.
RESUME_AT	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume zum angegebenen Zeitpunkt.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RESUME_IN	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume nach der angegebenen Anzahl Zeiteinheiten.
RESUME_BASE	Zeiteinheitsangabe für RESUME_IN
MASTER_SUBMITTABLE	Der Job der durch den Trigger gestartet wird, wird als eigener Master Job submitted und hat keinen Einfluss auf den aktuellen Master Job-Lauf des triggernden Jobs.
TIMEOUT_AMOUNT	Die Anzahl Zeiteinheiten die gewartet wird bis das Timeout eintritt
TIMEOUT_BASE	Die Einheit die genutzt wird um das Timeout in Sekunden, Minuten, Stunden oder Tagen zu spezifizieren
TIMEOUT_STATE	Das Timeout des Scheduling Entities
DEPENDENCY_MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
ESP_NAME	Hierbei handelt es sich um den Namen des Exit State Profiles.
ESM_NAME	Hierbei handelt es sich um den Namen des Exit State Mappings.
ENV_NAME	Hierbei handelt es sich um den Namen des Environments.
FP_NAME	Hierbei handelt es sich um den Namen des Footprints.
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
CHILDREN	Tabelle der Children

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
	Siehe auch Tabelle 22.38 auf Seite 359
PARENTS	Tabelle der Parents
	Siehe auch Tabelle 22.39 auf Seite 361
PARAMETER	Tabelle der Parameter und Variablen die für dieses Objekt definiert sind
REFERENCES	Tabelle der Parameter References auf dieses Objekt
REQUIRED_JOBS	Tabelle der benötigten Jobs
	Siehe auch Tabelle 22.40 auf Seite 363
DEPENDENT_JOBS	Tabelle der Objekte die vom gezeigten Objekt abhängig sind
	Siehe auch Tabelle 22.41 auf Seite 364
REQUIRED_RESOURCES	Tabelle der Resource Anforderungen die nicht im Environent und Footprint enthalten sind
	Siehe auch Tabelle 22.42 auf Seite 366
DEFINED_RESOURCES	Tabelle von Resourcen zum Instanziieren in die Submit Zeit, sichtbar um Children zu submitten

Tabelle 22.37: Beschreibung der Output-Struktur des show job definition Statements

CHILDREN Das Layout der CHILDREN Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
CHILDNAME	Kompletter Pfadname des Child Objekts
CHILDTYPE	Der Child Typ (JOB, BATCH oder MILESTONE)
CHILDPRIVS	Ein String der die Benutzerprivilegien des Child Objekts enthält
PARENTNAME	Der Name des Parent-Objekts
PARENTTYPE	Der Parent Typ (JOB, BATCH oder MILESTONE)
PARENTPRIVS	Ein String der die Benutzerprivilegien des Parent Objekts enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ALIAS_NAME	Name zum Referieren auf Child Definitions bei dynamischen Submits
IS_STATIC	Das is_static Flag gibt an, ob der Job statisch oder dynamisch submitted werden soll.
IS_DISABLED	Flag das angibt, ob das Child ausgeführt oder übersprungen wird
INT_NAME	**int_name** ist der Name des Intervalls, welches verwendet wird, um zu prüfen ob das Child enabled ist.
ENABLE_CONDITION	Die Enable Condition, wenn ausgefüllt, bestimmt ob ein Child enabled oder disabled ist. Die Condition wird zum Zeitpunkt des Submits ausgewertet; eventuelle Parameterwerte müssen daher bereits zu der Zeit bekannt sein. Die Grundidee ist mit Hilfe der Condition parametergesteuert Ablaufvarianten zu ermöglichen.
ENABLE_MODE	Die Enable Mode bestimmt wie das Ergebnis der Enable Condition und das Ergebnis des Enable Intervalls miteinander verknüpft werden. Die Möglichkeiten sind AND und OR. Im ersten Fall wird ein Child nur dann enabled, wenn sowohl das Enable Intervall als auch die Condition dazu Anlass geben. Im letzeren Fall muss nur einer der beiden grünes Licht geben. Wenn einer der beiden Bedingungen fehlt, ist der Wert der Enable Mode irrelevant.
PRIORITY	Der Nicevalue der den Children zugefügt wurde
SUSPEND	Bestimmt, ob das Child beim Submit suspended werden soll
RESUME_AT	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume zum angegebenen Zeitpunkt.
RESUME_IN	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume nach der angegebenen Anzahl Zeiteinheiten.
RESUME_BASE	Zeiteinheitsangabe für RESUME_IN

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
MERGE_MODE	Legt fest wie die Kondition dasselbe Objekt behandelt das mehr als einmal in der Submit Hierarchy auftritt
EST_NAME	Eine Exit State Translation die benutzt wird um die Exit States der Children nach den Exit States der Parents zu Übersetzen
IGNORED_DEPENDENCIES	Liste mit den Namen der Abhängigkeiten zum Ignorieren der Abhängigkeiten der Parents

Tabelle 22.38: Output-Struktur der show job definition Subtabelle

PARENTS Das Layout der PARENTS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
CHILDNAME	Kompletter Pfadname des Child Objekts
CHILDTYPE	Der Child Typ (JOB, BATCH oder MILESTONE)
CHILDPRIVS	Ein String der die Benutzerprivilegien des Child Objekts enthält
PARENTNAME	Der Name des Parent-Objekts
PARENTTYPE	Der Parent Typ (JOB, BATCH oder MILESTONE)
PARENTPRIVS	Ein String der die Benutzerprivilegien des Parent Objekts enthält
ALIAS_NAME	Name zum Referieren auf Child Definitions bei dynamischen Submits
IS_STATIC	Das is_static Flag gibt an, ob der Job statisch oder dynamisch submitted werden soll.
IS_DISABLED	Flag das angibt, ob das Child ausgeführt oder übersprungen wird
INT_NAME	**int_name** ist der Name des Intervalls, welches verwendet wird, um zu prüfen ob das Child enabled ist.
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ENABLE_CONDITION	Die Enable Condition, wenn ausgefüllt, bestimmt ob ein Child enabled oder disabled ist. Die Condition wird zum Zeitpunkt des Submits ausgewertet; eventuelle Parameterwerte müssen daher bereits zu der Zeit bekannt sein. Die Grundidee ist mit Hilfe der Condition parametergesteuert Ablaufvarianten zu ermöglichen.
ENABLE_MODE	Die Enable Mode bestimmt wie das Ergebnis der Enable Condition und das Ergebnis des Enable Intervalls miteinander verknüpft werden. Die Möglichkeiten sind AND und OR. Im ersten Fall wird ein Child nur dann enabled, wenn sowohl das Enable Intervall als auch die Condition dazu Anlass geben. Im letzeren Fall muss nur einer der beiden grünes Licht geben. Wenn einer der beiden Bedingungen fehlt, ist der Wert der Enable Mode irrelevant.
PRIORITY	Der Nicevalue der den Children zugefügt wurde
SUSPEND	Bestimmt, ob das Child beim Submit suspended werden soll
RESUME_AT	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume zum angegebenen Zeitpunkt.
RESUME_IN	Falls der Job suspended submitted werden soll, erfolgt ein automatischer Resume nach der angegebenen Anzahl Zeiteinheiten.
RESUME_BASE	Zeiteinheitsangabe für RESUME_IN
MERGE_MODE	Legt fest wie die Kondition dasselbe Objekt behandelt das mehr als einmal in der Submit Hierarchy auftritt
EST_NAME	Eine Exit State Translation die benutzt wird um die Exit States der Children nach den Exit States der Parents zu Übersetzen
IGNORED_DEPENDENCIES	Liste mit den Namen der Abhängigkeiten zum Ignorieren der Abhängigkeiten der Parents

Tabelle 22.39: Output-Struktur der show job definition Subtabelle

REQUIRED_JOBS Das Layout der REQUIRED_JOBS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
DEPENDENTNAME	Der komplette Pfadname des abhängigen Objekts
DEPENDENTTYPE	Der Typ des abhängigen Objekts (JOB, BATCH oder MILESTONE)
DEPENDENTPRIVS	String der die Benutzerprivilegien des abhängigen Objekts enthält
REQUIREDNAME	Der komplette Pfadname des benötigten Objekts
REQUIREDTYPE	Der Typ des benötigten Objekts (JOB, BATCH oder MILESTONE)
REQUIREDPRIVS	String der die Benutzerprivilegien des benötigten Objektes enthält
UNRESOLVED_HANDLING	Bestimmt wie man im Fall, dass das benötigte Objekt nicht gefunden werden kann, handeln soll.
MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
STATE_SELECTION	Gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigten Exit States expliziert aufgeführt sein.
CONDITION	Zusätzlich spezifizierte Conditions müssen ebenfalls erfüllt sein
STATES	Kommas trennen Listen von zugelassenen Exit States, die das benötigte Objekt erreichen muss, um die Abhängigkeiten zu erfüllen.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
RESOLVE_MODE	Der Resolve Mode definiert den Kontext in dem die Dependency aufgelöst werden soll. Mögliche Werte sind: Wert — Bedeutung **internal** — Die Dependency wird innerhalb des Masters aufgelöst. **both** — Die Dependency wird, wenn möglich innerhalb des Masters aufgelöst. Gelingt dies nicht, wird außerhalb des Masters gesucht. **external** — Die Dependency wird außerhalb des Masters aufgelöst.
EXPIRED_AMOUNT	Bei der Auflösung eines external Dependencies spielt es eine Rolle wann der benötigte Job oder Batch aktiv war. Die expired amount definiert wie viele Zeiteinheiten dies in die Vergangenheit liegen darf.
EXPIRED_BASE	Die exipired base definiert die Zeiteinheit für die expired amount.
SELECT_CONDITION	Die select condition definiert eine Bedingung die erfüllt sein muss, damit ein Job oder Batch als required Job betrachtet werden kann.

Tabelle 22.40: Output-Struktur der show job definition Subtabelle

DEPENDENT_JOBS Das Layout der DEPENDENT_JOBS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
DEPENDENTNAME	Der komplette Pfadname des abhängigen Objekts
DEPENDENTTYPE	Der Typ des abhängigen Objekts (JOB, BATCH oder MILESTONE)
DEPENDENTPRIVS	String der die Benutzerprivilegien des abhängigen Objekts enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
REQUIREDNAME	Der komplette Pfadname des benötigten Objekts
REQUIREDTYPE	Der Typ des benötigten Objekts (JOB, BATCH oder MILESTONE)
REQUIREDPRIVS	String der die Benutzerprivilegien des benötigten Objektes enthält
UNRESOLVED_HANDLING	Bestimmt wie man im Fall, dass das benötigte Objekt nicht gefunden werden kann, handeln soll.
MODE	Der Dependency Mode gibt an in welchem Zusammenhang die Liste der Dependencies gesehen werden muss. Es gibt folgende Optionen: ALL und ANY.
STATE_SELECTION	Gibt an, wie die benötigten Exit States ermittelt werden. Es gibt die Optionen FINAL, ALL_REACHABLE, UNREACHABLE und DEFAULT. Im Falle von FINAL können die benötigten Exit States expliziert aufgeführt sein.
CONDITION	Zusätzlich spezifizierte Conditions müssen ebenfalls erfüllt sein
STATES	Kommas trennen Listen von zugelassenen Exit States, die das benötigte Objekt erreichen muss, um die Abhängigkeiten zu erfüllen.
RESOLVE_MODE	Der Resolve Mode definiert den Kontext in dem die Dependency aufgelöst werden soll. Mögliche Werte sind: Wert — Bedeutung **internal** — Die Dependency wird innerhalb des Masters aufgelöst. **both** — Die Dependency wird, wenn möglich innerhalb des Masters aufgelöst. Gelingt dies nicht, wird außerhalb des Masters gesucht. **external** — Die Dependency wird außerhalb des Masters aufgelöst.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
EXPIRED_AMOUNT	Bei der Auflösung eines external Dependencies spielt es eine Rolle wann der benötigte Job oder Batch aktiv war. Die expired amount definiert wie viele Zeiteinheiten dies in die Vergangenheit liegen darf.
EXPIRED_BASE	Die exipired base definiert die Zeiteinheit für die expired amount.
SELECT_CONDITION	Die select condition definiert eine Bedingung die erfüllt sein muss, damit ein Job oder Batch als required Job betrachtet werden kann.

Tabelle 22.41: Output-Struktur der show job definition Subtabelle

REQUIRED_RESOURCES Das Layout der REQUIRED_RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
RESOURCE_NAME	Kompletter Pfadname der benötigten Named Resource
RESOURCE_USAGE	Die Anwendung der benötigten Resource (STATIC, SYSTEM oder SYNCHRONIZING)
RESOURCE_PRIVS	String der die Benutzerprivilegien der Named Resource beinhaltet
AMOUNT	Die benötigte Menge bei System oder Synchronizing Resources
KEEP_MODE	Keep_Mode spezifiziert wann die Resource freigegeben wird (FINISHED, JOB_FINAL oder FINAL)
IS_STICKY	Zeigt an, ob die Resource-Zuordnung für nachfolgende Jobs beibehalten wird
STICKY_NAME	Optionaler Name der Sticky Anforderung
STICKY_PARENT	Parent Job Definition innerhalb der die Sticky Anforderung gilt
RESOURCE_STATE_MAPPING	Gibt an, wie und ob der State der Resource nach Beendigung des Jobs geändert werden soll

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPIRED_AMOUNT	Die Anzahl der Einheiten. Wenn der Expired Amount positiv ist, bedeutet das, dass der Statuswechsel maximal die angegebene Zeit her sein darf. Ist der Amount negativ, muss es minimal so lange her sein
EXPIRED_BASE	Die Einheit um den Expired-Zeitpunkt zu spezifizieren
IGNORE_ON_RERUN	Dieser Flag gibt an ob die Aktualitätsbedingung im Falle eines Reruns ignoriert werden soll.
LOCKMODE	Der Sperrmodus zum Allokieren von Synchronizing Resourcen (N, S, SX, X)
STATES	Kommas trennen Listen von zugelassenen Exit States, die das benötigte Objekt erreichen muss um die Abhängigkeiten zu erfüllen.
DEFINITION	(REQUIREMENT, FOOTPRINT, FOLDER oder ENVIRONMENT)
ORIGIN	Name der Resource Anforderungsdefinition, ungültig im Falle einer vollständigen Anforderung
CONDITION	Die optionale Condition die für Anforderungen von Static Resources definiert sein darf

Tabelle 22.42: Output-Struktur der show job definition Subtabelle

show named resource

Zweck

Zweck Das *show named resource* Statement wird eingesetzt um detaillierte Informationen über die Named Resource zu bekommen.

Syntax

Syntax Die Syntax des *show named resource* Statements ist

show [**condensed**] **named resource** *resourcepath* [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Beschreibung Mit dem *show named resource* Statement bekommt man ausführliche Informationen über die Named Resource.

expand Da die Anzahl Job Definitions in der Tabelle JOB_DEFINITIONS sehr groß werden kann, werden sie per Default nicht angezeigt. Wenn die Option **expand = all** benutzt wird, werden alle Job Definitions, sowie die Folder, in die sie liegen, samt Folderhierarchie ausgegeben. Durch die Spezifikation einzelner (Folder) Id's können einzelne Pfade der Hierarchie selektiert werden.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name der Named Resource
OWNER	Eigentümer der Named Resource

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
USAGE	Die Usage gibt an um welchen Typ Resource es sich handelt.
INHERIT_PRIVS	Vom übergeordneten Ordner zu erbende Privilegien
RESOURCE_STATE_PROFILE	Es handelt sich hier um das zur Resource zugeordnete Resource State Profile.
FACTOR	Dies ist der Standard Factor mit der Resource Requirement Amounts multipliziert werden, wenn bei der betreffenden Resource nichts anders spezifiziert ist.
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
RESOURCES	Das sind die Instanzen der Named Resource. Siehe auch Tabelle 22.44 auf Seite 370
PARAMETERS	Das sind die definierten Parameter der Named Resource Siehe auch Tabelle 22.45 auf Seite 370
JOB_DEFINITIONS	Die Job Definitions, die die Named Resource anfordern Siehe auch Tabelle 22.46 auf Seite 371

Tabelle 22.43: Beschreibung der Output-Struktur des show named resource Statements

RESOURCES Das Layout der RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
SCOPE	Hier erscheinen die Namen der Scopes, Submitted Entities, Scheduling Entities der Folder, welche die jeweilige Named Resource anbieten.
TYPE	Hierbei handelt es sich um den Resource Typ.
OWNER	Die Gruppe die Eigentümer des Objektes ist
STATE	Zeigt den Status der Resource an
REQUESTABLE_AMOUNT	Die Menge der Resourcen die maximal von einem Job angefordert werden darf
AMOUNT	Der Amount gibt die aktuelle Anzahl von Instanzen der Named Resource für diesen Scope oder Jobserver an.
FREE_AMOUNT	Der Free Amount bezeichnet die Anzahl aller noch nicht von Jobs belegten Instanzen einer Resource innerhalb des gewählten Scopes oder Jobservers.
IS_ONLINE	Zeigt an, ob die Resource online ist oder nicht
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.44: Output-Struktur der show named resource Subtabelle

PARAMETERS Das Layout der PARAMETERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters
TYPE	Beim Type handelt es sich um den Parametertyp. Local oder Local Constant
DEFAULT_VALUE	Beim Default_Value unterscheiden wir zwischen Constants und Local Constants. Bei Constants ist er der Wert des Parameters und bei Local Constants der Default-Wert.
TAG	Die Überschrift für den nachfolgenden Kommentar

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 22.45: Output-Struktur der show named resource Subtabelle

JOB_DEFINITIONS Das Layout der JOB_DEFINITIONS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
NAME	Name der Job Definition
AMOUNT	Die Resource-Menge die vom Job benötigt wird
KEEP_MODE	Wert des Keep Parameters für die Resource-Anforderung des Jobs
IS_STICKY	Gibt an, ob es Sticky Request ist oder nicht
STICKY_NAME	Optionaler Name der Sticky Anforderung
STICKY_PARENT	Parent Job Definition innerhalb der die Sticky Anforderung gilt
RESOURCE_STATE_MAPPING	Wurde bei der Resource-Anforderung ein Resource State Mapping angegeben, wird dies hier angezeigt.
EXPIRED_AMOUNT	Die Anzahl der Einheiten. Wenn der Expired Amount positiv ist, bedeutet dies, dass der Statuswechsel maximal die angegebene Zeit her sein darf. Ist der Amount negativ, muss es minimal so lange her sein.
EXPIRED_BASE	Die Einheit in Minuten, Stunden, Tage, Wochen, Monate und Jahre
IGNORE_ON_RERUN	Dieser Flag gibt an ob die Aktualitätsbedingung im Falle eines Reruns ignoriert werden soll.
LOCKMODE	Der Lockmode beschreibt den Zugriffsmodus auf diese Resource (exclusiv, shared etc.).
STATES	Falls mehrere States für diesen Job akzeptabel sind, werden die einzelnen States durch Komma getrennt.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CONDITION	Die Condition die für Anforderungen von Static Resources definiert sein darf
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.46: Output-Struktur der show named resource Subtabelle

show resource

Zweck

Das *show resource* Statement wird eingesetzt um detaillierte Informationen über die Resource zu bekommen. *Zweck*

Syntax

Die Syntax des *show resource* Statements ist *Syntax*

show RESOURCE_URL

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

Beschreibung

Mit dem *show resource* Statement bekommt man ausführliche Informationen über die Resource. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name der Resource
SCOPENAME	Name des Scopes in dem der Pool angelegt wurde
OWNER	Die Gruppe die Eigentümer des Objektes ist
LINK_ID	Id der Resource auf die verwiesen wird
LINK_SCOPE	Scopename der Resource auf die verwiesen wird
BASE_ID	Id der Resource auf die letztendlich verwiesen wird

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
BASE_SCOPE	Scopename der Resource auf die letztendlich verwiesen wird
MANAGER_ID	Id des Managing Pools
MANAGER_NAME	Name des Managing Pools
MANAGER_SCOPENAME	Name des Scopes in dem der Managing Pool angelegt wurde
USAGE	Die Usage gibt an um welchen Typ Resource es sich handelt.
RESOURCE_STATE_PROFILE	Es handelt sich hier um das zur Resource zugeordnete Resource State Profile.
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
TAG	Optionaler Kurzname für die Resource.
STATE	Bei State handelt es sich um den aktuellen Status der Resource in diesem Scope oder Jobserver.
TIMESTAMP	Der Timestamp gibt die Zeit des letzten Statuswechsels einer Resource an.
REQUESTABLE_AMOUNT	Die Menge der Resourcen die maximal von einem Job angefordert werden darf
DEFINED_AMOUNT	Die Menge die vorhanden ist wenn die Resource nicht pooled ist
AMOUNT	Die tatsächliche Menge die vorhanden ist
FREE_AMOUNT	Der Free_Amount bezeichnet die Anzahl aller noch nicht von Jobs belegten Instanzen einer Resource.
IS_ONLINE	Is_Online ist ein Indikator, der angibt, ob die Resource online ist oder nicht.
FACTOR	Dies ist der Korrekturfaktor mit dem angeforderte Amounts multipliziert werden.
TRACE_INTERVAL	Trace_Interval ist die minimale Zeit zwischen dem Schreiben von Trace Records in Sekunden.
TRACE_BASE	Trace_Base ist die Basis für den Auswertungszeitraum.
TRACE_BASE_MULTIPLIER	Base_Multiplier bestimmt den Multiplikationsfaktor von der Trace_Base.
TD0_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^0$ Sekunden

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
TD1_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^1$ Sekunden
TD2_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^2$ Sekunden
LW_AVG	Die durchschnittliche Resource-Belegung seit dem letzten Schreiben eines Trace Records
LAST_WRITE	Zeitpunkt des letzen Schreibens eines Trace Records
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
ALLOCATIONS	Das ist eine Tabelle mit Resource-Belegungen. Siehe auch Tabelle 22.48 auf Seite 375
PARAMETERS	Im Tab Parameters können zusätzliche Informationen zu einer Resource gespeichert werden. Siehe auch Tabelle 22.49 auf Seite 376

Tabelle 22.47: Beschreibung der Output-Struktur des show resource Statements

ALLOCATIONS Das Layout der ALLOCATIONS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
JOBID	Hierbei handelt es sich um die Id der Jobinstanz, welche durch ein direktes Submit des Jobs oder durch ein Submit des Master Batches oder Jobs gestartet wurde.
MASTERID	Hierbei handelt es sich um die Id der Job- oder Batchinstanz, welche als Master Job gestartet wurde und den aktuellen Job als Child beinhaltet.
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
JOBTYPE	Hierbei handelt es sich um den Id des Jobs.
JOBNAME	Hierbei handelt es sich um den Namen des Jobs.
AMOUNT	Der Amount ist die Menge die vorhanden ist.
KEEP_MODE	Der Keep Parameter gibt an, ob der Job die aktuelle Resource hält oder nicht. Es gibt folgende Ausführungen: KEEP, NO KEEP und KEEP FINAL.
IS_STICKY	Die Resource wird nur freigegeben wenn im gleichen Batch keine weiteren Sticky Requests für diese Named Resource vorhanden sind.
STICKY_NAME	Optionaler Name der Sticky Anforderung
STICKY_PARENT	Parent Job innerhalb dessen die Sticky Anforderung gilt
STICKY_PARENT_TYPE	Typ des Parents innerhalb dessen die Sticky Anforderung gilt
LOCKMODE	Der Lockmode gibt an mit welchem Zugriffsmodus die Resource vom aktuellen Job belegt wird.
RSM_NAME	Der Name des Resource State Mappings
TYPE	Die Art der Allokierung: Available, Blocked, Allocations, Master_Reservation, Reservation
TYPESORT	Hilfe für die Sortierung der Allocations
P	Die Priorität des Jobs
EP	Die effektive Priorität des Jobs
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.48: Output-Struktur der show resource Subtabelle

PARAMETERS Das Layout der PARAMETERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPORT_NAME	Der Export Name definiert den Namen unter dem der Wert des Parameters in die Prozessumgebung exportiert wird.
TYPE	Hierbei handelt es sich um die Art des Parameters.
IS_LOCAL	True für lokale Parameter die nur für den Job selbst sichtbar sind
EXPRESSION	Name der Aggregat Funktion
DEFAULT_VALUE	Der Default Value des Parameters
REFERENCE_TYPE	Typ des Objektes auf das referenziert wird
REFERENCE_PATH	Der Pfad des Objektes auf das referenziert wird
REFERENCE_PRIVS	Die Privilegien des Benutzers auf das Objekt auf das referenziert wird
REFERENCE_PARAMETER	Name des Parameters auf den referenziert wird
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters
TYPE	Hierbei handelt es sich um die Art des Parameters.
IS_LOCAL	True für lokale Parameter die nur für den Job selbst sichtbar sind
REFERENCE_TYPE	Typ des Objekts welches den Parameter referenziert
REFERENCE_PATH	Der Pfad des Objekts welches den Parameter referenziert
REFERENCE_PRIVS	Die Privilegien des Benutzers auf das Objekt welches den Parameter referenziert
REFERENCE_PARAMETER	Name des Parameters auf den referenziert wird
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 22.49: Output-Struktur der show resource Subtabelle

show resource state definition

Zweck

Zweck Der Zweck der *show resource state definition* ist es detaillierte Informationen über die spezifizierte Resource State Definition zu bekommen.

Syntax

Syntax Die Syntax des *show resource state definition* Statements ist

show resource state definition *statename*

Beschreibung

Beschreibung Mit dem *show resource state definition* Statement bekommt man ausführliche Informationen über die Resource State Definition.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.50: Beschreibung der Output-Struktur des show resource state definition Statements

show resource state mapping

Zweck

Zweck

Das *show resource state mapping* Statement wird eingesetzt um detaillierte Informationen über das spezifizierte Mapping zu bekommen.

Syntax

Syntax

Die Syntax des *show resource state mapping* Statements ist

show resource state mapping *profilename*

Beschreibung

Beschreibung

Mit dem *show resource state mapping* Statement bekommt man ausführliche Informationen über das spezifizierte Mapping.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Resource State Mappings
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
MAPPINGS	Eine Tabelle mit Übersetzungen von Exit State nach Resource State Siehe auch Tabelle 22.52 auf Seite 380

Tabelle 22.51: Beschreibung der Output-Struktur des show resource state mapping Statements

MAPPINGS Das Layout der MAPPINGS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ESD_NAME	Name der Exit State Definition
RSD_FROM	Der ursprüngliche State der Resource
RSD_TO	Der aktuelle State der Resource

Tabelle 22.52: Output-Struktur der show resource state mapping Subtabelle

show resource state profile

Zweck

Der Zweck des *show resource state profile* Statements ist es detaillierte Informationen über die spezifizierten Resorce State Profiles zu bekommen. *Zweck*

Syntax

Die Syntax des *show resource state profile* Statements ist *Syntax*

show resource state profile *profilename*

Beschreibung

Mit dem *show resource state profile* Statement bekommt man ausführliche Informationen über die spezifizierten Resource State Profiles. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
INITIAL_STATE	Dieses Feld definiert den initialen Status der Resource. Dieser Resource State muss nicht in der Liste gültiger Resource States vorhanden sein.
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
STATES	In der Tabelle States stehen in der Spalte Resource State die gültigen Resource States. Siehe auch Tabelle 22.54 auf Seite 382

Tabelle 22.53: Beschreibung der Output-Struktur des show resource state profile Statements

STATES Das Layout der STATES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
RSD_NAME	Name der Resource State Definition
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.54: Output-Struktur der show resource state profile Subtabelle

show schedule

Zweck

Das *show schedule* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Zeitplan zu erhalten. *Zweck*

Syntax

Die Syntax des *show schedule* Statements ist *Syntax*

show schedule *schedulepath*

Beschreibung

Mit dem *show schedule* Statement bekommt man ausführliche Informationen über den spezifizierten Zeitplan. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
INHERIT_PRIVS	Vom übergeordneten Ordner zu erbende Privilegien
INTERVAL	Der Name des zum Schedule gehörenden Intervalls
TIME_ZONE	Die Zeitzone in der der Schedule gerechnet werden soll
ACTIVE	Dieses Feld gibt an, ob der Schedule als active markiert ist.
EFF_ACTIVE	Dieses Feld gibt an, ob der Schedule tatsächlich active ist. Dies kann aufgrund der hierarchischen Anordnug von ”active” abweichen.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 22.55: Beschreibung der Output-Struktur des show schedule Statements

show scheduled event

Zweck

Der Zweck des *show scheduled event* Statements ist es detaillierte Informationen über das spezifizierte Event zu bekommen. *Zweck*

Syntax

Die Syntax des *show scheduled event* Statements ist *Syntax*

show scheduled event *schedulepath* **.** *eventname*

Beschreibung

Mit dem *show scheduled event* Statement bekommt man ausführliche Informationen über das spezifizierte Event. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
SCHEDULE	Der Schedule der den Zeitplan für den Scheduled Event bestimmt
EVENT	Der Event der ausgelöst wird
ACTIVE	Dieses Feld gibt an, ob der Schedule als active markiert ist.
EFF_ACTIVE	Dieses Flag gibt an, ob der scheduled Event auch tatsächlich active ist.
BROKEN	Mittels des Broken Feldes kann geprüft werden, ob beim Submit des Jobs ein Fehler aufgetreten ist.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ERROR_CODE	Im Feld Error_Code wird, falls ein Fehler bei der Ausführung des Jobs im Time Scheduling aufgetreten ist, der übermittelte Fehlercode angezeigt. Ist kein Fehler aufgetreten, bleibt das Feld leer.
ERROR_MSG	Im Feld Error Message wird, falls ein Fehler bei der Ausführung des Jobs im Time Scheduling aufgetreten ist, die übermittelte Fehlermeldung angezeigt. Ist kein Fehler aufgetreten, bleibt das Feld leer.
LAST_START	Hier wird der letzte Ausführungszeitpunkt des Jobs durch das Scheduling System angezeigt.
NEXT_START	Hier wird der nächste geplante Ausführungszeitpunkt des Tasks durch das Scheduling System angezeigt.
NEXT_CALC	Wenn der Next_Start leer ist gibt der Next_Calc den Zeitpunkt an wann nach einem nächsten Startzeitpunkt gesucht wird. Ansonsten findet die neue Berechnung zum Next_Start Zeitpunkt statt.
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
BACKLOG_HANDLING	Das Backlog_Handling beschreibt den Umgang mit Events die zu Downtimes ausgelöst hätten werden sollen.
SUSPEND_LIMIT	Das Suspend_Limit gibt an nach welcher Verspätung ein Job als suspended submitted wird.
EFFECTIVE_SUSPEND_LIMIT	Das Suspend Limit gibt an nach welcher Verspätung ein Job als suspended submitted wird.
CALENDAR	Dieses Flag gibt an, ob Kalendereinträge erzeugt werden.

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
CALENDAR_HORIZON	Die definierte Länge der Periode in Tagen für die ein Kalender erstellt wird
EFFECTIVE_CALENDAR_HORIZON	Die effektive Länge der Periode in Tagen für die ein Kalender erstellt wird
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CALENDAR_TABLE	Die Tabelle mit nächsten Startzeitpunkten

Tabelle 22.56: Beschreibung der Output-Struktur des show scheduled event Statements

show scope

Zweck

Zweck Das *show scope* Statement wird eingesetzt um detaillierte Informationen über einen Scope zu bekommen.

Syntax

Syntax Die Syntax des *show scope* Statements ist

show < **scope** *serverpath* | **jobserver** *serverpath* > [**with** EXPAND]

EXPAND:
expand = none
| **expand =** < **(** *id* {**,** *id*} **)** | **all** >

Beschreibung

Beschreibung Mit dem *show scope* Statement bekommt man ausführliche Informationen über den Scope.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OWNER	Die Gruppe die Eigentümer des Objektes ist
TYPE	Der Typ des Scopes
INHERIT_PRIVS	Vom übergeordneten Ordner zu erbende Privilegien
IS_TERMINATE	Dieses Flag zeigt an, ob ein Terminierungsauftrag vorliegt.
IS_SUSPENDED	Zeigt an, ob der Scope suspended ist

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_ENABLED	Nur wenn das Enable Flag YES gesetzt ist kann sich der Jobserver am Server anmelden
IS_REGISTERED	Gibt an, ob der Jobserver einen register Befehl gesendet hat
IS_CONNECTED	Zeigt an, ob der Jobserver connected ist.
HAS_ALTERED_CONFIG	Die Konfiguration im Server weicht von der aktuellen im Jobserver ab.
STATE	Hierbei handelt es sich um den aktuellen Status der Resource in diesem Scope.
PID	Bei PID handelt es sich um die Prozess Identifikationsnummer des Jobserverprozesses auf dem jeweiligen Hostsystem.
NODE	Der Node gibt an auf welchem Rechner der Jobserver läuft. Dieses Feld hat einen rein dokumentativen Charakter.
IDLE	Die Zeit die seit dem letzten Befehl vergangen ist. Dies gilt nur für Jobserver.
ERRMSG	Hierbei handelt es sich um die zuletzt ausgegebene Fehlermeldung.
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
RESOURCES	Hier werden die Resourcen die in diesem Scope vorhanden sind angezeigt. Siehe auch Tabelle 22.58 auf Seite 390
CONFIG	Im Tab Config steht die Konfiguration des Jobservers beschrieben. Siehe auch Tabelle 22.59 auf Seite 392

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CONFIG_ENVMAPPING	In diesem Tab wird konfiguriert, ob und unter welchem Namen die Umgebungsvariablen sichtbar sind. Siehe auch Tabelle 22.60 auf Seite 392
PARAMETERS	Im Tab Parameters können zusätzliche Informationen zu einer Resource gespeichert werden. Siehe auch Tabelle 22.61 auf Seite 392

Tabelle 22.57: Beschreibung der Output-Struktur des show scope Statements

RESOURCES Das Layout der RESOURCES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
NR_ID	Id der Named Resource
NAME	Name der Named Resource
USAGE	Hierbei handelt es sich um den Gebrauch der Named Resource (STATIC, SYSTEM oder SYNCHRONIZING)
NR_PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf diese Named Resource enthält
TAG	Optionaler Kurzname für die Resource.
OWNER	Die Gruppe die Eigentümer des Objektes ist
LINK_ID	Id der Resource auf die verwiesen wird
LINK_SCOPE	Scopename der Resource auf die verwiesen wird
STATE	Der Resource State in dem sich die Resource befindet
REQUESTABLE_AMOUNT	Die Menge der Resources die maximal von einem Job angefordert werden darf
AMOUNT	Die aktuelle Menge die zur Verfügung steht
FREE_AMOUNT	Die freie Menge die allokiert werden darf
TOTAL_FREE_AMOUNT	Der Free_Amount für Allocations inklusive dem Free_Amount der pooled Resources, wenn diese ein Pool sind

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
IS_ONLINE	Hierbei handelt es sich um den Verfügbarkeitsstatus der Resource
FACTOR	Dies ist der Korrekturfaktor mit dem angeforderte Amounts multipliziert werden.
TIMESTAMP	Der Timestamp gibt die Zeit des letzten Statuswechsels einer Resource an.
SCOPE	Der Scope in dem die Resource angelegt ist
MANAGER_ID	Id des Managing Pools
MANAGER_NAME	Name des Managing Pools
MANAGER_SCOPENAME	Name des Scopes in dem der Managing Pool angelegt wurde
HAS_CHILDREN	Flag, das anzeigt, ob ein Pool Child Resources/-Pools managed. Wenn es kein Pool ist, ist es immer FALSE.
POOL_CHILD	Dieses Flag zeigt an, ob die gezeigte Resource ein Child vom Pool ist.
TRACE_INTERVAL	Trace_Interval ist die minimale Zeit zwischen dem Schreiben von Trace Records in Sekunden.
TRACE_BASE	Die Trace_Base ist die Basis für den Auswertungszeitraum (B).
TRACE_BASE_MULTIPLIER	Der Base_Multiplier bestimmt den Multiplikationsfaktor (M) von der Trace_Base.
TD0_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^0$ Sekunden
TD1_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^1$ Sekunden
TD2_AVG	Die durchschnittliche Resource-Belegung der letzten $B * M^2$ Sekunden
LW_AVG	Die durchschnittliche Resource-Belegung seit dem letzten Schreiben eines Trace Records
LAST_WRITE	Zeitpunkt des letzen Schreibens eines Trace Records
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.58: Output-Struktur der show scope Subtabelle

CONFIG Das Layout der CONFIG Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
KEY	Der Name der Konfigurationsvariablen
VALUE	Der Wert der Konfigurationsvariablen
LOCAL	Zeigt an, ob der Key Value Pairs local definiert ist oder übergeordnet
ANCESTOR_SCOPE	Das ist der Scope in dem der Key Value Pairs definiert ist.
ANCESTOR_VALUE	Das ist der Wert der übergeordnet definiert ist.

Tabelle 22.59: Output-Struktur der show scope Subtabelle

CONFIG_ENVMAPPING Das Layout der CONFIG_ENVMAPPING Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
KEY	Name der Umgebungsvariablen
VALUE	Name der Umgebungsvariablen die gesetzt werden soll
LOCAL	Zeigt an, ob der Key Value Pairs local definiert ist oder übergeordnet
ANCESTOR_SCOPE	Das ist der Scope in dem der Key Value Pairs definiert ist.
ANCESTOR_VALUE	Das ist der Wert der übergeordnet definiert ist.

Tabelle 22.60: Output-Struktur der show scope Subtabelle

PARAMETERS Das Layout der PARAMETERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
EXPORT_NAME	Der Export Name definiert den Namen unter dem der Wert des Parameters in die Prozessumgebung exportiert wird.
TYPE	Hierbei handelt es sich um die Art des Parameters.
IS_LOCAL	True für lokale Parameter die nur für den Job selbst sichtbar sind
EXPRESSION	Name der Aggregat Funktion
DEFAULT_VALUE	Der Default Value des Parameters
REFERENCE_TYPE	Typ des Objektes auf das referenziert wird
REFERENCE_PATH	Der Pfad des Objektes auf das referenziert wird
REFERENCE_PRIVS	Die Privilegien des Benutzers auf das Objekt auf das referenziert wird
REFERENCE_PARAMETER	Name des Parameters auf den referenziert wird
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters
TYPE	Hierbei handelt es sich um die Art des Parameters.
IS_LOCAL	True für lokale Parameter die nur für den Job selbst sichtbar sind
REFERENCE_TYPE	Typ des Objekts welches den Parameter referenziert
REFERENCE_PATH	Der Pfad des Objekts welches den Parameter referenziert
REFERENCE_PRIVS	Die Privilegien des Benutzers auf das Objekt welches den Parameter referenziert
REFERENCE_PARAMETER	Name des Parameters auf den referenziert wird
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars

Tabelle 22.61: Output-Struktur der show scope Subtabelle

show session

Zweck

Zweck Das *show session* Statement wird eingesetzt um mehr detaillierte Informationen über die spezifizierte oder die akutelle Session zu bekommen.

Syntax

Syntax Die Syntax des *show session* Statements ist

show session [*sid*]

Beschreibung

Beschreibung Mit dem *show session* Statement bekommt man ausführliche Informationen über die spezifizierte oder aktuelle Session.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
THIS	Die aktuelle Session wird in diesem Feld mit einem Asterisk (*) gekennzeichnet.
SESSIONID	Die serverinterne Id der Session
START	Zeitpunkt an dem die Connection hergestellt wurde
USER	Name des Users mit dem die Session angemeldet wurde
UID	Id des Users, Jobservers oder Jobs
IP	IP-Adresse der connecting Sessions
IS_SSL	Gibt an ob die Verbindung über SSL/TLS erfolgt
IS_AUTHENTICATED	Gibt an ob der Client sich authentifiziert hat
TXID	Nummer der letzten Transaktion die von der Session ausgeführt wurde
IDLE	Die Anzahl Sekunden seit dem letzten Statement einer Session

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
TIMEOUT	Die Idle Zeit nach der die Session automatisch disconnected wird
STATEMENT	Das Statement das gerade ausgeführt wird

Tabelle 22.62: Beschreibung der Output-Struktur des show session Statements

show system

Zweck

Zweck Das *show system* Statement wird eingesetzt um Informationen über die aktuelle Konfiguration des laufenden Servers zu bekommen.

Syntax

Syntax Die Syntax des *show system* Statements ist

show system

show system with lock

Beschreibung

Beschreibung Mit dem *show system* Statement bekommt man ausführliche Informationen über die aktuelle Konfiguration des laufenden Servers.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
VERSION	Die aktuelle Version der Software
MAX_LEVEL	Die maximale Kompatibilitätsstufe der Software
NUM_CPU	Die Anzahl Prozessoren die im System vorhanden sind
MEM_USED	Die Menge benutzter Hauptspeicher
MEM_FREE	Die Menge freier Speicher
MEM_MAX	Die maximale Speichermenge die der Server in Anspruch nehmen kann
STARTTIME	Der Zeitpunkt zu dem der Server gestartet wurde
UPTIME	Der Zeitpunkt ab dem der Server schon läuft

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
HITRATE	Der Hitrate im Environment Cache des Scheduling Threads
LOCK_HWM	Der Lock_HWM zeigt die High Water Mark der Anzahl aktiver Sperren im System. Dieses Feld ist nur dann relevant, wenn mehrere Writer Threads aktiv sind.
LOCKS_REQUESTED	Das Feld Locks_Requested zeigt die Anzahl beantragter Sperren seit Server Startup. Dieses Feld ist nur relevant, wenn mehrere Writer Threads aktiv sind.
LOCKS_USED	Dieses Feld zeigt die Anzahl derzeit benutzter Sperren. Es ist nur relevant, wenn mehrere Writer Threads aktiv sind.
LOCKS_DISCARDED	Das Feld Locks_Discarded zeigt die Anzahl Sperren die freigegeben wurden, ohne sie für spätere Wiederbenutzung aufzuheben.
CNT_RW_TX	Die Anzahl R/W Transaktionen seit Server Startup.
CNT_DL	Die Anzahl Deadlocks seit Server Startup.
CNT_WL	Die Anzahl single threaded Write Worker Transaktionen seit Server Startup
WORKER	Eine Tabelle mit einer Liste der Worker Threads Siehe auch Tabelle 22.64 auf Seite 397
LOCKING STATUS	Der Locking State gibt Information über den Zustand des internen Locking Systems. Dieses Feld wird nur dann gezeigt, wenn die **with locks** Option spezifiziert wird.

Tabelle 22.63: Beschreibung der Output-Struktur des show system Statements

WORKER Das Layout der WORKER Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
ID	Die Nummer des Repository Objektes
TYPE	Der Typ des Worker Threads, Read/Write (RW) or Read Only (RO)
Fortsetzung auf der nächsten Seite	

Fortsetzung der vorherigen Seite

Feld	Beschreibung
NAME	Der Name des Objektes
STATE	Der Status des Workers
TIME	Der Zeitpunkt ab dem der Worker in einem Status ist

Tabelle 22.64: Output-Struktur der show system Subtabelle

show trigger

Zweck

Das *show trigger* Statement wird eingesetzt um detaillierte Informationen über den spezifizierten Trigger zu bekommen. *Zweck*

Syntax

Die Syntax des *show trigger* Statements ist *Syntax*

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

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

Beschreibung

Mit dem *show trigger* Statement bekommt man ausführliche Informationen über den spezifizierten Trigger. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
OBJECTTYPE	Der Typ des Objektes in dem der Trigger definiert ist
OBJECTNAME	Kompletter Pfadname des Objektes in dem der Trigger definiert ist

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ACTIVE	Das Flag gibt an, ob der Trigger momentan aktiv ist.
ACTION	Typ der ausgelöste Aktion: SUBMIT oder RERUN
SUBMIT_TYPE	Der Objekttyp der submitted wird, wenn getriggert wird
SUBMIT_NAME	Name der Job Definition die submitted wird
SUBMIT_SE_OWNER	Der Besitzer des Objektes das submitted wird
SUBMIT_PRIVS	Die Privilegien auf das zu submittende Objekt
MAIN_TYPE	Typ des Main Jobs (Job/Batch)
MAIN_NAME	Name des Main Jobs
MAIN_SE_OWNER	Owner des Main Jobs
MAIN_PRIVS	Privilegien auf den Main Job
PARENT_TYPE	Typ des Parent Jobs (Job/Batch)
PARENT_NAME	Name des Parent Jobs
PARENT_SE_OWNER	Owner des Parent Jobs
PARENT_PRIVS	Privilegien auf den Parent Job
TRIGGER_TYPE	Der Trigger Typ der beschreibt wann gefeuert wird
MASTER	Zeigt an, ob der Trigger einen Master oder ein Child submitted
IS_INVERSE	Im Falle eines Inverse Triggers gehört der Trigger dem getriggerten Job. Der Trigger kann so als Art Callback-Funktion gesehen werden. Das Flag hat keinen Einfluß auf die Funktion des Triggers.
SUBMIT_OWNER	Die Eigentümergruppe die beim Submitted Entity eingesetzt wird
IS_CREATE	Zeigt an, ob der Trigger auf create Events reagiert
IS_CHANGE	Zeigt an, ob der Trigger auf change Events reagiert
IS_DELETE	Zeigt an, ob der Trigger auf delete Events reagiert
IS_GROUP	Zeigt an, ob der Trigger die Events als Gruppe behandelt

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
MAX_RETRY	Die maximale Anzahl von Trigger Auslösungen in einem einzelnen Submitted Entity
SUSPEND	Spezifiziert, ob das submittete Objekt suspended wird
RESUME_AT	Zeitpunkt des automatischen Resume
RESUME_IN	Anzahl Zeiteinheiten bis zum automatischen Resume
RESUME_BASE	Zeiteinheitsangabe für RESUME_IN
WARN	Spezifiziert, ob eine Warnung ausgegeben werden muss wenn das Feuerlimit erreicht ist
LIMIT_STATE	Spezifiziert den Status der vom auslösenden Jobs angenommen wird, wenn das Fire Limit erreicht wird. Hat der Job bereit einen finalen Status, wird diese Einstellung ignoriert. Steht der Wert auf NONE, wird keine Statusänderung vorgenommen.
CONDITION	Konditionaler Ausdruck um die Trigger Condition zu definieren
CHECK_AMOUNT	Die Menge der CHECK_BASE Einheiten um die Kondition bei nicht synchronen Triggern zu überprüfen
CHECK_BASE	Einheiten für den CHECK_AMOUNT
COMMENT	Kommentar zum Objekt, wenn vorhanden
COMMENTTYPE	Typ des Kommentars
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
STATES	Eine Liste mit States die zum Auslösen des Triggers führen Siehe auch Tabelle 22.66 auf Seite 402
PARAMETERS	Eine Liste mit States die zum Auslösen des Triggers führen Siehe auch Tabelle 22.67 auf Seite 402

Tabelle 22.65: Beschreibung der Output-Struktur des show trigger Statements

STATES Das Layout der STATES Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
FROM_STATE	Der Trigger feuert wenn der angegebene State der alte Resource State ist.
TO_STATE	Der Trigger feuert wenn dder angegebene State der neue Resource State oder der Exit State des Objektes ist.

Tabelle 22.66: Output-Struktur der show trigger Subtabelle

PARAMETERS Das Layout der PARAMETERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Name des Parameters, der zu Submitzeit gesetzt wird.
EXPRESSION	Ein Ausdruck der im Kontext des triggernden Objektes ausgewertet wird. Die Syntax ist gleich der Syntax in der Triggercondition, nur dass hier generelle Expressions erlaubt sind, nicht nur Boolean Expressions.

Tabelle 22.67: Output-Struktur der show trigger Subtabelle

show user

Zweck

Das *show user* Statement wird eingesetzt um detaillierte Informationen über den Benutzer anzuzeigen. *Zweck*

Syntax

Die Syntax des *show user* Statements ist *Syntax*

show user [*username*]

Beschreibung

Mit dem *show user* Statement bekommt man ausführliche Informationen über den Benutzer. *Beschreibung*

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
IS_ENABLED	Flag, das anzeigt, ob es dem Benutzer erlaubt ist sich anzumelden
DEFAULT_GROUP	Die Default-Gruppe der Benutzer die die Eigentümer des Objektes benutzen
CONNECTION_TYPE	Gibt an welche Sicherheitsstufe für eine Verbindung gefordert wird. 1. **plain** – Jede Art von Verbindung ist erlaubt 2. **ssl** – Nur SSL-Verbindungen sind erlaubt 3. **ssl_auth** – Nur SSL-Verbindungen mit Client Authentifizierung sind erlaubt

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Feld	**Beschreibung**
CREATOR	Name des Benutzers der dieses Objekt angelegt hat
CREATE_TIME	Datum und Uhrzeit der Erstellung
CHANGER	Name des Benutzers der dieses Objekt zuletzt geändert hat
CHANGE_TIME	Datum und Uhrzeit der letzten Änderung
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält
MANAGE_PRIVS	Tabelle der Manage Privilegien Siehe auch Tabelle 22.69 auf Seite 404
GROUPS	Tabelle der Gruppen zu denen der Benutzer gehört Siehe auch Tabelle 22.70 auf Seite 405
EQUIVALENT_USERS	Tabelle mit users und Jobservers die als äquivalent gelten Siehe auch Tabelle 22.71 auf Seite 405
PARAMETERS	Es ist möglich Key-Value Pairs zu einem Benutzer zu speichern. Diese Werte werde zwar von Server selbst nicht benutzt, aber ermöglichen es einem Frontend Benutzerbezogenen Einstellungen zentral abzulegen.
COMMENTTYPE	Typ des Kommentars
COMMENT	Kommentar zum Objekt, wenn vorhanden Siehe auch Tabelle 22.72 auf Seite 405

Tabelle 22.68: Beschreibung der Output-Struktur des show user Statements

MANAGE_PRIVS Das Layout der MANAGE_PRIVS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	**Beschreibung**
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.69: Output-Struktur der show user Subtabelle

GROUPS Das Layout der GROUPS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
ID	Die Nummer des Repository Objektes
NAME	Der Name des Objektes
PRIVS	Zeichenkette die Kürzel für die Benutzerrechte auf dieses Objekt enthält

Tabelle 22.70: Output-Struktur der show user Subtabelle

EQUIVALENT_USERS Das Layout der EQUIVALENT_USERS Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
TYPE	Der Typ des äquivalenten Users (**server** oder **user**)
EQUIVALENT_USER	Der Name des äquivalenten Users

Tabelle 22.71: Output-Struktur der show user Subtabelle

COMMENT Das Layout der COMMENT Tabelle wird in nachfolgender Tabelle gezeigt.

Feld	Beschreibung
TAG	Die Überschrift für den nachfolgenden Kommentar
COMMENT	Kommentar zum Objekt, wenn vorhanden

Tabelle 22.72: Output-Struktur der show user Subtabelle

Kapitel 23

shutdown commands

shutdown

Zweck

Zweck Das *shutdown* Statement wird eingesetzt um den adressierten Jobserver anzuweisen sich zu beenden.

Syntax

Syntax Die Syntax des *shutdown* Statements ist

shutdown *serverpath*

Beschreibung

Beschreibung Mit dem *shutdown* Statement beendet man den adressierten Jobserver.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 24

stop commands

stop server

Zweck

Zweck Das *stop server* Statement wird eingesetzt um den Server anzuweisen sich zu beenden.

Syntax

Syntax Die Syntax des *stop server* Statements ist

stop server

stop server kill

Beschreibung

Beschreibung Mit dem *stop server* Statement beendet man den Server. Sollte dies aus irgendeinem Grund nicht richtig funktionieren, kann der Server auch hart beendet werden, durch **kill** zu spezifizieren.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Kapitel 25

submit commands

submit

Zweck

Zweck Das *submit* Statement wird eingesetzt um einen Master Batch oder Job, sowie alle definierten Children, auszuführen.

Syntax

Syntax Die Syntax des *submit* Statements ist

submit *folderpath* [**with** WITHITEM {**,** WITHITEM}]

submit *aliasname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:

check only
| **childtag =** *string*
| < **enable** | **disable** >
| **master**
| **nicevalue =** *signed_integer*
| **parameter = none**
| **parameter = (** PARAM {**,** PARAM} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **submittag =** *string*
| < **nosuspend** | **suspend** >
| **time zone =** *string*
| **unresolved =** JRQ_UNRESOLVED
| **group =** *groupname*

PARAM:

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

JRQ_UNRESOLVED:

defer
| **defer ignore**
| **error**
| **ignore**
| **suspend**

Beschreibung

Das *submit* Statement wird benutzt um einen Job oder Batch zu submitten. Es existieren zwei Formen des Submit-Kommandos. *Beschreibung*

- Die erste Form wird von Benutzern, welche auch Programme sein können und dem Time Scheduling Module genutzt. Diese Form submitted Master Jobs und Batches.
- Die zweite Form des Statements wird von Jobs genutzt, um dynamische Children zu submitten.

check only Die check only Option wird benutzt, um zu überprüfen, ob ein Master Submittable Batch oder Job submitted werden kann. Das bedeutet, es wird geprüft, ob alle Abhängigkeiten erfüllt werden können und alle referenzierten Parameter definiert sind.
Es wird nicht überprüft, ob die Jobs in irgendeinem Scope ausgeführt werden können oder nicht. Dies ist eine Situation die jederzeit zur Laufzeit auftreten kann.
Eine positive Rückmeldung bedeutet, dass der Job oder Batch aus Sicht des Systems submitted werden kann.
Die check only Option kann nicht in einem Job-Kontext benutzt werden.

childtag Die childtag Option wird von Jobs benutzt, um verschiedene Instanzen von demselben Scheduling Entity zu submitten und um zwischen ihnen unterscheiden zu können.
Es führt zu einem Fehler, wenn der gleiche Scheduling Entity doppelt submitted wird, wenn sich der childtag nicht unterscheidet. Der Inhalt des childtags hat keine weitere Bedeutung für das Scheduling System.
Die maximale Länge eines childtags beträgt 70 Zeichen. Die childtag Option wird im Falle eines Master Submits ignoriert.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

nicevalue Die nicevalue Option definiert eine Korrektur die für die Berechnung der Prioritäten des Jobs und seiner Children benutzt wird. Es sind Werte von -100 bis 100 erlaubt.

parameter Die parameter Option wird benutzt um den Wert von Job Parametern beim Submit zu spezifizieren. Die Parameter werden im Scope des Master Batches oder Jobs gesetzt. Das bedeutet, wenn Parameter, die nicht in dem Master Batch oder Job definiert sind, spezifiziert werden, sind diese Parameter unsichtbar für Children.

submittag Wenn der submittag spezifiziert ist, muss er eine eindeutige Bezeichnung für den Submitted Entity haben. Dieser Tag wurde, um imstande zu sein Jobs und Batches programmatisch zu submitten und um den Job oder Batch, mit demselben Tag, nach einem Absturz von einem der Komponenten neu zu submitten. Wenn die Submission des Jobs das erste Mal erfolgreich war, wird der zweite Submit einen Fehler melden. Wenn nicht, wird der zweite Submit erfolgreich sein.

unresolved Die unresolved Option spezifiziert wie der Server bei nicht auflösbaren Abhängigkeiten reagieren sollte. Diese Option wird hauptsächlich benutzt, wenn Teile eines Batches nach Reparaturarbeiten submitted werden. Der fehlerhafte Teil wird normal gecancelt und dann als Master Run neu submittet. Die vorherigen Abhängigkeiten müssen in diesem Fall ignoriert werden, andernfalls wird der Submit scheitern.

suspend Die suspend Option wird benutzt um Jobs oder Batches zu submitten und sie zur selben Zeit zu suspenden. Wenn nichts festgelegt wurde, wird nicht suspended. Dies kann explizit zur Submit-Zeit spezifiziert werden.
Wenn ein Job oder Batch suspended wurde, wird er, sowie auch seine Children, nicht gestartet. Wenn ein Job bereits läuft, wird er keinen final State erreichen, wenn er suspended ist.

resume Die resume Option kann zusammen mit der suspend Option verwendet werden um eine verzögerte Ausführung zu bewirken. Es gibt dabei zwei Möglichkeiten. Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
Mit dieser Option kann die `at`-Funktionalität ohne das Anlegen eines Schedules nachgebildet werden.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Id des Submitted Entities

Tabelle 25.1: Beschreibung der Output-Struktur des submit Statements

Kapitel 26

suspend commands

suspend

Zweck

Zweck Das *suspend* Statement wird eingesetzt um zu verhindern, dass weitere Jobs von diesem Jobserver ausgeführt werden. Siehe das *resume* Statement auf Seite 302.

Syntax

Syntax Die Syntax des *suspend* Statements ist

suspend *serverpath*

Beschreibung

Beschreibung Mit dem *suspend* Statement wird verhindert, dass weitere Jobs von diesem Jobserver ausgeführt werden.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Teil III

Jobserver Commands

Kapitel 27

Jobserver Commands

alter job

Zweck

Zweck Das *alter job* Statement wird benutzt um Eigenschaften des spezifizierten Jobs zu ändern. Es wird von den Job-Administratoren, Jobservern und vom Job selbst benutzt.

Syntax

Syntax Die Syntax des *alter job* Statements ist

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

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

WITHITEM:
< **disable** | **enable** >
| < **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **clone resume**
| **clone suspend**
| **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 = (** *id* [**recursive**] {**,** *id* [**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**

Beschreibung

Beschreibung

Das *alter job* Kommando wird für mehrere Zwecke genutzt. Als erstes verwenden Jobserver dieses Kommando um den Ablauf eines Jobs zu dokumentieren. Alle Statuswechsel eines Jobs während der Zeit in der der Job innerhalb der Zuständigkeit eines Jobservers fällt, werden mittels des *alter job* Kommandos ausgeführt.
Zweitens werden einige Änderungen, wie z. B. das Ignorieren von Abhängigkeiten oder Resourcen, sowie das Ändern der Priorität eines Jobs, manuell von einem Administrator ausgeführt.
Der Exit State eines Jobs in einem pending State kann vom Job selbst gesetzt werden, bzw. von einem Prozess welcher die Job Id und den Key des zu ändernden Jobs kennt.

cancel Die cancel Option wird benutzt um den adressierten Job und alle nicht final Children zu canceln. Ein Job kann nur gecancelt werden wenn weder der Job selbst noch einer seiner Children aktiv ist.
Wenn ein Scheduling Entity von dem gecancelten Job abhängig ist, kann er unreachable werden. In diesem Fall erhält der abhängige Job nicht den im Exit State Profile definierten unreachable Exit State, sondern wird in den Job Status "unreachable" versetzt. Es ist Aufgabe des Operators diese Jobs nun mittels des Ignorierens von Abhängigkeiten wieder in den Job Status "dependency wait" zu versetzen, oder aber diese Jobs auch zu canceln.
Gecancelte Jobs werden wie final Jobs ohne Exit State betrachtet. Das bedeutet, die Parents eines gecancelten Jobs werden final, ohne den Exit State des gecancelten Jobs zu berücksichtigen. Die abhängigen Jobs der Parents laufen in diesem Fall normal weiter.
Die cancel Option kann nur von Benutzern genutzt werden.

clone Die **clone** option wird benutzt um bereits beendete Jobs noch einmal im selben Kontext aus zu führen. Dies kann in seltene Fällen notwendig sein. Wenn etwa in der Fehlerdiagnose eines Nachfolgers festgestellt wird, dass die Ursache einige Jobs zurück liegt, wird es notwendig sein, nach der Ursachenbeseitigung, die ganze Kette noch einmal aus zu führen.
Um zu gewährleisten, dass die Ausführung kontrolliert anfängt wird spezifiziert ob der Clone suspended werden soll, oder nicht.

comment Die comment Option wird benutzt um eine Aktion zu dokumentieren oder um dem Job einen Kommentar zuzufügen. Comments können maximal 1024 Zeichen lang sein. Es kann eine beliebige Anzahl Comments für einen Job gespeichert werden.
Einige Comments werden automatisch gespeichert. Wenn z. B. ein Job einen restartable State erreicht, wird ein Protokoll geschrieben, um diesen Fakt zu dokumentieren.

error text Die error text Option wird benutzt um Fehlerinformation zu einem Job zu schreiben. Dieses kann von dem verantwortlichen Jobserver oder einem Benutzer gemacht werden. Der Server kann diesen Text auch selbst schreiben.
Diese Option wird normalerweise benutzt, wenn der Jobserver den entsprechenden Prozess nicht starten kann. Mögliche Fälle sind die Unmöglichkeit zum definierten Working Directory zu wechseln, die Unauffindbarkeit des ausführbaren Programmes oder Fehler beim Öffnen des Error Logfiles.

exec pid Die exec pid Option wird ausschließlich vom Jobserver benutzt um die Prozess Id des Kontrollprozesses innerhalb des Servers zu setzen.

exit code Die exit code Option wird vom Jobserver benutzt um dem Repository Server mitzuteilen mit welchem Exit Code sich ein Prozess beendet hat. Der Repository Server berechnet jetzt den zugehörigen Exit State aus dem verwendeten Exit State Mapping.

exit state Die exit state Option wird von Jobs in einem pending State benutzt, um ihren State auf einen anderen Wert zu setzen. Dies wird normalerweise ein restartable oder final State sein. Alternativ dazu kann diese Option von Administratoren benutzt werden, um den State von einem nonfinal Job zu setzen. Sofern das Force Flag nicht benutzt wird, sind die einzigen States die gesetzt werden können, die States, welche, durch die Anwendung des Exit State Mappings auf irgendeinem Exit Code, theoretisch erreichbar sind. Der gesetzte State muss im Exit State Profile vorhanden sein.

ext pid Die ext pid Option wird ausschließlich vom Jobserver genutzt, um die Prozess Id des gestarteten Benutzerprozesses zu setzen.

ignore resource Die ignore resource Option wird benutzt um einzelne Resource Requests aufzuheben. Die ignorierte Resource wird nicht mehr beantragt.
Wenn Parameter einer Resource referenziert werden, kann diese Resource nicht ignoriert werden.
Wenn ungültige Id's spezifiziert wurden, wird dies übergangen. Alle anderen spezifizierten Resources werden ignoriert. Ungültige Id's in diesem Kontext sind Id's von Resources die von dem Job nicht beantragt werden.
Das Ignorieren von Resources wird protokolliert.

ignore dependency Die ignore dependency Option wird benutzt um definierte Dependencies zu ignorieren. Wenn das **recursive** Flag benutzt wird, ignorieren nicht nur der Job oder Batch selbst, sondern auch seine Children die Dependencies.

kill Die kill Option wird benutzt um den definierten Kill Job zu submitten. Wenn kein Kill Job definiert ist, ist es nicht möglich den Job vom BICsuite aus erzwungenermaßen zu terminieren. Natürlich muss der Job aktiv sein, das bedeutet, der Job State muss **running**, **killed** oder **broken_active** sein. Die letzten beiden States sind keine regulären Fälle.
Wenn ein Kill Job submitted wurde, ist der Job State **to_kill**. Nachdem der Kill Job beendet wurde, wird der Job State des killed Jobs in den State **killed** gesetzt, es sei denn er ist beendet, dann wird der Job State **finished** oder **final** sein. Das bedeutet, dass der Job mit dem Job State **killed** immer noch running ist und dass mindestens ein Versuch gemacht wurde, den Job zu terminieren.

nicevalue Die nicevalue Option wird benutzt um die Priorität oder den nicevalue eines Jobs oder Batches und allen seinen Children zu ändern. Hat ein Child mehrere Parents, kann eine Änderung, muss aber nicht, in dem nicevalue von einem der Parents Auswirkungen auf die Priorität des Childs haben. In dem Fall, dass es mehrere Parents gibt wird das maximale nicevalue gesucht.
Also, wenn Job C drei Parents P1, P2 und P3 hat und P1 setzt einen Nicevalue von 0, P2 einen von 10 und P3 einen von -10, ist der effektive nicevalue -10. (Umso niederiger der nicevalue, umso besser). Wenn der nicevalue von P2 auf -5 geändert wird, passiert nichts, weil die -10 von P3 besser als -5 ist. Wenn jetzt der nicevalue von P3 auf 0 sinkt, wird die neue effektive nicevalue für Job C -5.
Die nicevalues können Werte zwischen -100 und 100 haben. Werte die diese Spanne übersteigen, werden stillschweigend angepasst.

priority Die priority Option wird benutzt, um die (statische) Priorität eines Jobs zu ändern. Weil Batches und Milestones nicht ausgeführt werden, haben Prioritäten keine Bedeutung für sie.

Ein Wechsel der Priorität betrifft nur den geänderten Job. Gültige Werte liegen zwischen 0 und 100. Dabei korrespondiert 100 mit der niedrigsten Priorität und 0 mit der höchsten Priorität.
Bei der Berechnung der dynamischen Priorität eines Jobs startet der Scheduler mit der statischen Priorität und passt dies, entsprechend der Zeit in der der Job schon wartet, an. Wenn mehr als ein Job die gleiche dynamische Priorität hat, wird der Job mit der niedrigsten Job Id als erster gescheduled.

renice Die renice Option gleicht der nicevalue Option mit dem Unterschied, dass die renice Option relativ arbeitet, während die nicevalue Option absolut arbeitet. Wenn einige Batches einen nicevalue von 10 haben bewirkt eine renice von -5, dass die nicevalue auf 5 zunimmt. (Zunahme, weil je niedriger die Nummer, desto höher die Piorität).

rerun Die rerun Option wird benutzt um einen Job in einem restartable State neu zu starten. Der Versuch einen Job, der nicht restartable ist, neu zu starten, führt zu einer Fehlermeldung. Ein Job ist restartable, wenn er in einem restartable State oder in einem **error** oder **broken_finished** Job State ist.
Wenn das **recursive** Flag spezifiziert ist, wird der Job selbst und alle direkten und indirekten Children, die in einem restartable State sind, neu gestartet. Wenn der Job selbst final ist, wird das in dem Fall *nicht* als Fehler betrachtet. Es ist also möglich Batches rekursiv neu zu starten.

resume Die resume Option wird benutzt um einen suspended Job oder Batch zu reaktivieren. Es gibt dabei zwei Möglichkeiten. Erstens kann der suspended Job oder Batch sofort reaktiviert werden, und zweitens kann eine Verzögerung eingestellt werden.
Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
(Für die Spezifikation einer Zeit siehe auch die Übersicht auf Seite 21.)
Die resume Option kann zusammen mit der suspend Option verwendet werden. Dabei wird der Job suspended und nach der (bzw. zur) spezifizierten Zeit wieder resumed.

run Die run Option wird vom Jobserver benutzt zwecks der Sicherstellung, dass der geänderte Job mit der aktuellen Version übereinstimmt.
Theoretisch ist es möglich, dass nachdem ein Job von einem Jobserver gestartet wurde, der Computer abstürzt. Um die Arbeit zu erledigen wird der Job mittels eines manuellen Eingriffs, von einem anderen Jobserver, neu gestartet. Nach dem Hochfahren des ersten Systems kann der Jobserver versuchen den Job State nach **broken_finished** zu ändern, ohne über das Geschehen nach dem Absturz Bescheid

zu wissen. Das Benutzen der run Option verhindert nun das fälschliche Setzen des Status.

state Die state Option wird hauptsächlich von Jobservern benutzt, kann aber auch von Administratoren benutzt werden. Es wird nicht empfohlen dies so zu machen, es sei denn Sie wissen genau was Sie tun.
Die übliche Prozedur ist, dass der Jobserver den State eines Jobs von **starting** nach **started**, von **started** nach **running** und von **running** nach **finished** setzt. Im Falle eines Absturzes oder anderen Problemen ist es möglich dass der Jobserver einen Job in einen **broken_active** oder **broken_finished** State setzt. Das bedeutet, der Exit Code von dem Prozess steht nicht zur Verfügung und der Exit State muss manuell gesetzt werden.

suspend Die suspend Option wird benutzt um einen Batch oder Job zu suspendieren. Sie arbeitet nur dann rekursiv wenn **local** nicht spezifiziert ist. Wenn ein Parent suspended ist, sind auch alle Children suspended. Die resume Option wird benutzt um die Situation umzukehren. Die **restrict** Angabe bewirkt, dass nur Benutzer der ADMIN Gruppe die Suspendierung wieder aufheben können.

timestamp Die timestamp Option wird vom Jobserver benutzt um die Timestamps der State-Wechsel zu setzen, gemäß der lokalen Zeit aus Sicht des Jobservers.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

alter jobserver

Zweck

Zweck Das *alter jobserver* Statement wird eingesetzt um die Eigenschaften eines Jobservers zu ändern.

Syntax

Syntax Die Syntax des *alter jobserver* Statements ist

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

alter [**existing**] **jobserver**
with dynamic PARAMETERS

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

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

Beschreibung

Beschreibung Das *alter jobserver* Kommando ist sowohl ein Benutzerkommando als auch ein Jobserver-Kommando. Es wird als Benutzerkommando benutzt um die Konfiguration oder andere Eigenschaften eines Scopes oder Jobservers zu ändern. (Weitere Details sind im *create scope* Kommando auf Seite 161 beschrieben.) Die Syntax von dem Benutzerkommando entspricht der ersten Form des *alter scope* Kommandos.
Als Jobserver Kommando wird es benutzt um den Server über Fehler zu benachrichtigen. Wird der Fatal Flag benutzt, bedeutet dies, dass sich der Jobserver beendet. In dem anderen Fall läuft der Jobserver weiter.
Die dritten Form des *alter jobserver* Kommandos wird auch vom Jobserver benutzt. Der Jobserver veröffentlicht die Werte seines dynamischen Parameters. Der Server verwendet veröffentlichte Werte um Parameter in der Kommandozeile und Logfile-Angaben beim Abholen eines Jobs aufzulösen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

connect

Zweck

Das *connect* Statement wird eingesetzt um einen Jobserver am Server zu authentifizieren. *Zweck*

Syntax

Die Syntax des *connect* Statements ist *Syntax*

connect jobserver *serverpath* **.** *servername* **identified by** *string* [**with** WITHITEM {**,** WITHITEM}]

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

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

Beschreibung

Das *connect* Kommando wird benutzt um den verbundenen Prozess am Server zu authentifizieren. Es kann wahlweise ein Kommunikationsprotokoll spezifiziert werden. Das Default-Protokoll ist **line**. *Beschreibung*
Das ausgewählte Protokoll definiert die Form des Outputs. Alle Protokolle, außer **serial**, liefern ASCII Output. Das Protokoll **serial** liefert ein Serialized Java Objekt zurück.
Beim *connect* Kommando kann auch gleich ein auszuführendes Kommando mitgegeben werden. Als Output des *connect* Kommandos wird in diesem Fall der Output des mitgegebenen Kommandos genutzt. Falls das Kommando fehlschlägt, der *connect* aber gültig war, bleibt die Connection bestehen.

Im Folgenden ist für alle Protokolle, außer für das **serial** Protokoll, ein Beispiel gegeben.

line protocol Das line protocol liefert nur einen ASCII-Text als Ergebnis von einem Kommando zurück.

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

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

XML protocol Das XML protocol liefert eine XML-Struktur als Ergebnis eines Kommandos zurück.

```
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>
```

python protocol Das python protocol liefert eine Python-Struktur, welche durch die *python eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

perl protocol Das perl protocol liefert eine Perl-Struktur, welche durch die *perl eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

deregister

Zweck

Zweck Das *deregister* Statement wird eingesetzt um den Server zu benachrichtigen, das der Jobserver keine Jobs mehr ausführt. Siehe das *register* Statement auf Seite 280.

Syntax

Syntax Die Syntax des *deregister* Statements ist

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

Beschreibung

Beschreibung Das *deregister* Statement wird genutzt um den Server über einen, mehr oder weniger, permanenten Ausfall eines Jobservers zu informieren.
Diese Nachricht hat verschiedene Serveraktionen zur Folge. Als Erstes werden alle running Jobs des Jobservers, d.h. Jobs im Status **started**, **running**, **to_kill** und **killed**, auf den Status **broken_finished** gesetzt. Jobs im Status **starting** werden wieder auf **runnable** gesetzt. Dann wird der Jobserver aus der Liste der Jobserver, die Jobs verarbeiten können, entfernt, sodass dieser Jobserver im Folgenden auch keine Jobs mehr zugeteilt bekommt. Als Nebeneffekt werden Jobs, die aufgrund Resource-Anforderungen nur auf diesem Jobserver laufen können, in den Status **error**, mit der Meldung "Cannot run in any scope because of resource shortage", versetzt. Als Letztes wird ein komplettes Reschedule ausgeführt um eine Neuverteilung von Jobs auf Jobservern herbeizuführen.
Durch erneutes Registrieren (siehe *register* Statement auf Seite 280), wird der Jobserver erneut in die Liste der Jobs-verarbeitenden Jobserver eingetragen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

disconnect

Zweck

Das *disconnect* Statement wird eingesetzt um die Serververbindung zu beenden. *Zweck*

Syntax

Die Syntax des *disconnect* Statements ist *Syntax*

disconnect

Beschreibung

Mit dem *disconnect* Statement kann die Verbindung zum Server beendet werden. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

get next job

Zweck

Zweck Die Zweck des *get next job* Statements ist es das nächste Kommando von dem Server zu holen.

Syntax

Syntax Die Syntax des *get next job* Statements ist

get next job

Beschreibung

Beschreibung Mit dem *get next job* Statement holt der Jobserver das nächste auszuführende Kommando vom Server.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Tabelle.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
COMMAND	Das vom Jobserver auszuführende Kommando. (NOP, ALTER, SHUTDOWN, STARTJOB)
CONFIG	Geänderte Konfiguration. Dieser Wert ist nur im Falle eines ALTER Kommandos vorhanden
ID	Die Id des zu startenden Jobs; nur vorhanden beim Kommando STARTJOB
DIR	Das Working Directory des zu startenden Jobs; nur vorhanden beim Kommando STARTJOB
LOG	Das Logfile des zu startenden Jobs; nur vorhanden beim Kommando STARTJOB
LOGAPP	Indikator, ob das Logfile mit Append geöffnet werden soll; nur vorhanden beim Kommando STARTJOB
ERR	Das Error Logfile des zu startenden Jobs; nur vorhanden beim Kommando STARTJOB

Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite

Feld	Beschreibung
ERRAPP	Indikator, ob das Error Logfile mit Append geöffnet werden soll; nur vorhanden beim Kommando STARTJOB
CMD	File Name des zu startenden Executables; nur vorhanden beim Kommando STARTJOB
ARGS	Die Commandline Parameter des zu startenden Executables; nur vorhanden beim Kommando STARTJOB
ENV	Zusätzliche Einträge für das Environment des zu startenden Executables; nur vorhanden beim Kommando STARTJOB
RUN	Nummer des Runs. (Siehe auch Alter Job Statement auf Seite 68); nur vorhanden beim Kommando STARTJOB
JOBENV	Liste von key value Paaren, die definiert welche in der Job Definition definierten Umgebungsvariablen vor der Jobausführung gesetzt werden sollen.

Tabelle 27.1: Beschreibung der Output-Struktur des get next job Statements

multicommand

Zweck

Zweck Der Zweck des *multicommands* ist es mehrere SDMS-Kommandos als Einheit zuzuführen.

Syntax

Syntax Die Syntax des *multicommand* Statements ist

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Beschreibung

Beschreibung Mit den *multicommands* ist es möglich mehrere SDMS-Kommandos zusammen, d.h. in einer Transaktion auszuführen. Damit wird gewährleistet, dass entweder alle Statements fehlerfrei ausgeführt werden, oder nichts passiert. Des Weiteren wird die Transaktion nicht von anderen schreibenden Transaktionen unterbrochen. Wird das Keyword **rollback** spezifiziert, wird die Transaktion am Ende der Verarbeitung rückgängig gemacht. Auf diese Weise kann getestet werden, ob die Statements (technisch) korrekt verarbeitet werden können.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

reassure

Zweck

Zweck

Das *reassure* Statement wird eingesetzt um eine Bestätigung über die Notwendigkeit einen Job zu starten, nachdem ein Jobserver gestaret wurde, vom Server zu bekommen.

Syntax

Syntax

Die Syntax des *reassure* Statements ist

reassure *jobid* [**with run =** *integer*]

Beschreibung

Beschreibung

Mit dem *reassure* Statement bekommt man vom Server eine Bestätigung, ob ein Job gestartet werden soll. Dieses Statement wird in dem Moment eingesetzt, wenn ein Jobserver beim Hochfahren einen Job im Status **starting** vorfindet.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

register

Zweck

Zweck Das *register* Statement wird eingesetzt um den Server zu benachrichtigen, dass der Jobserver bereit ist Jobs auszuführen.

Syntax

Syntax Die Syntax des *register* Statements ist

register *serverpath* **.** *servername*
with pid = *pid* [**suspend**]

register with pid = *pid*

Beschreibung

Beschreibung Die erste Form wird vom Operator benutzt um das Aktivieren von Jobs auf dem spezifizierten Jobserver zu ermöglichen.
Die zweite Form wird vom Jobserver selbst benutzt um den Server über seine Bereitschaft Jobs auszuführen zu informieren.
Unabhängig davon, ob der Jobserver connected ist oder nicht, werden Jobs für diesen Server eingeplant, es sei den der Jobserver ist suspended.
(Siehe Statement *'deregister'* auf Seite 178.)

pid Die pid Option liefert dem Server Informationen über die Prozess-Id des Jobservers auf Betriebsebene.

suspend Die suspend Option bewirkt, dass der Jobserver in den suspended Zustand überführt wird.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

Teil IV

Job Commands

Kapitel 28

Job Commands

alter job

Zweck

Zweck Das *alter job* Statement wird benutzt um Eigenschaften des spezifizierten Jobs zu ändern. Es wird von den Job-Administratoren, Jobservern und vom Job selbst benutzt.

Syntax

Syntax Die Syntax des *alter job* Statements ist

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

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

WITHITEM:
< **disable** | **enable** >
| < **suspend** | **suspend restrict** | **suspend local** | **suspend local restrict** >
| **cancel**
| **clear warning**
| **clone resume**
| **clone suspend**
| **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 = (** *id* [**recursive**] {**,** *id* [**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**

Beschreibung

Beschreibung

Das *alter job* Kommando wird für mehrere Zwecke genutzt. Als erstes verwenden Jobserver dieses Kommando um den Ablauf eines Jobs zu dokumentieren. Alle Statuswechsel eines Jobs während der Zeit in der der Job innerhalb der Zuständigkeit eines Jobservers fällt, werden mittels des *alter job* Kommandos ausgeführt.
Zweitens werden einige Änderungen, wie z. B. das Ignorieren von Abhängigkeiten oder Resourcen, sowie das Ändern der Priorität eines Jobs, manuell von einem Administrator ausgeführt.
Der Exit State eines Jobs in einem pending State kann vom Job selbst gesetzt werden, bzw. von einem Prozess welcher die Job Id und den Key des zu ändernden Jobs kennt.

cancel Die cancel Option wird benutzt um den adressierten Job und alle nicht final Children zu canceln. Ein Job kann nur gecancelt werden wenn weder der Job selbst noch einer seiner Children aktiv ist.
Wenn ein Scheduling Entity von dem gecancelten Job abhängig ist, kann er unreachable werden. In diesem Fall erhält der abhängige Job nicht den im Exit State Profile definierten unreachable Exit State, sondern wird in den Job Status "unreachable" versetzt. Es ist Aufgabe des Operators diese Jobs nun mittels des Ignorierens von Abhängigkeiten wieder in den Job Status "dependency wait" zu versetzen, oder aber diese Jobs auch zu canceln.
Gecancelte Jobs werden wie final Jobs ohne Exit State betrachtet. Das bedeutet, die Parents eines gecancelten Jobs werden final, ohne den Exit State des gecancelten Jobs zu berücksichtigen. Die abhängigen Jobs der Parents laufen in diesem Fall normal weiter.
Die cancel Option kann nur von Benutzern genutzt werden.

clone Die **clone** option wird benutzt um bereits beendete Jobs noch einmal im selben Kontext aus zu führen. Dies kann in seltene Fällen notwendig sein. Wenn etwa in der Fehlerdiagnose eines Nachfolgers festgestellt wird, dass die Ursache einige Jobs zurück liegt, wird es notwendig sein, nach der Ursachenbeseitigung, die ganze Kette noch einmal aus zu führen.
Um zu gewährleisten, dass die Ausführung kontrolliert anfängt wird spezifiziert ob der Clone suspended werden soll, oder nicht.

comment Die comment Option wird benutzt um eine Aktion zu dokumentieren oder um dem Job einen Kommentar zuzufügen. Comments können maximal 1024 Zeichen lang sein. Es kann eine beliebige Anzahl Comments für einen Job gespeichert werden.
Einige Comments werden automatisch gespeichert. Wenn z. B. ein Job einen restartable State erreicht, wird ein Protokoll geschrieben, um diesen Fakt zu dokumentieren.

error text Die error text Option wird benutzt um Fehlerinformation zu einem Job zu schreiben. Dieses kann von dem verantwortlichen Jobserver oder einem Benutzer gemacht werden. Der Server kann diesen Text auch selbst schreiben.
Diese Option wird normalerweise benutzt, wenn der Jobserver den entsprechenden Prozess nicht starten kann. Mögliche Fälle sind die Unmöglichkeit zum definierten Working Directory zu wechseln, die Unauffindbarkeit des ausführbaren Programmes oder Fehler beim Öffnen des Error Logfiles.

exec pid Die exec pid Option wird ausschließlich vom Jobserver benutzt um die Prozess Id des Kontrollprozesses innerhalb des Servers zu setzen.

exit code Die exit code Option wird vom Jobserver benutzt um dem Repository Server mitzuteilen mit welchem Exit Code sich ein Prozess beendet hat. Der Repository Server berechnet jetzt den zugehörigen Exit State aus dem verwendeten Exit State Mapping.

exit state Die exit state Option wird von Jobs in einem pending State benutzt, um ihren State auf einen anderen Wert zu setzen. Dies wird normalerweise ein restartable oder final State sein. Alternativ dazu kann diese Option von Administratoren benutzt werden, um den State von einem nonfinal Job zu setzen. Sofern das Force Flag nicht benutzt wird, sind die einzigen States die gesetzt werden können, die States, welche, durch die Anwendung des Exit State Mappings auf irgendeinem Exit Code, theoretisch erreichbar sind. Der gesetzte State muss im Exit State Profile vorhanden sein.

ext pid Die ext pid Option wird ausschließlich vom Jobserver genutzt, um die Prozess Id des gestarteten Benutzerprozesses zu setzen.

ignore resource Die ignore resource Option wird benutzt um einzelne Resource Requests aufzuheben. Die ignorierte Resource wird nicht mehr beantragt.
Wenn Parameter einer Resource referenziert werden, kann diese Resource nicht ignoriert werden.
Wenn ungültige Id's spezifiziert wurden, wird dies übergangen. Alle anderen spezifizierten Resources werden ignoriert. Ungültige Id's in diesem Kontext sind Id's von Resources die von dem Job nicht beantragt werden.
Das Ignorieren von Resources wird protokolliert.

ignore dependency Die ignore dependency Option wird benutzt um definierte Dependencies zu ignorieren. Wenn das **recursive** Flag benutzt wird, ignorieren nicht nur der Job oder Batch selbst, sondern auch seine Children die Dependencies.

kill Die kill Option wird benutzt um den definierten Kill Job zu submitten. Wenn kein Kill Job definiert ist, ist es nicht möglich den Job vom BICsuite aus erzwungenermaßen zu terminieren. Natürlich muss der Job aktiv sein, das bedeutet, der Job State muss **running**, **killed** oder **broken_active** sein. Die letzten beiden States sind keine regulären Fälle.
Wenn ein Kill Job submitted wurde, ist der Job State **to_kill**. Nachdem der Kill Job beendet wurde, wird der Job State des killed Jobs in den State **killed** gesetzt, es sei denn er ist beendet, dann wird der Job State **finished** oder **final** sein. Das bedeutet, dass der Job mit dem Job State **killed** immer noch running ist und dass mindestens ein Versuch gemacht wurde, den Job zu terminieren.

nicevalue Die nicevalue Option wird benutzt um die Priorität oder den nicevalue eines Jobs oder Batches und allen seinen Children zu ändern. Hat ein Child mehrere Parents, kann eine Änderung, muss aber nicht, in dem nicevalue von einem der Parents Auswirkungen auf die Priorität des Childs haben. In dem Fall, dass es mehrere Parents gibt wird das maximale nicevalue gesucht.
Also, wenn Job C drei Parents P1, P2 und P3 hat und P1 setzt einen Nicevalue von 0, P2 einen von 10 und P3 einen von -10, ist der effektive nicevalue -10. (Umso niederiger der nicevalue, umso besser). Wenn der nicevalue von P2 auf -5 geändert wird, passiert nichts, weil die -10 von P3 besser als -5 ist. Wenn jetzt der nicevalue von P3 auf 0 sinkt, wird die neue effektive nicevalue für Job C -5.
Die nicevalues können Werte zwischen -100 und 100 haben. Werte die diese Spanne übersteigen, werden stillschweigend angepasst.

priority Die priority Option wird benutzt, um die (statische) Priorität eines Jobs zu ändern. Weil Batches und Milestones nicht ausgeführt werden, haben Prioritäten keine Bedeutung für sie.

Ein Wechsel der Priorität betrifft nur den geänderten Job. Gültige Werte liegen zwischen 0 und 100. Dabei korrespondiert 100 mit der niedrigsten Priorität und 0 mit der höchsten Priorität.
Bei der Berechnung der dynamischen Priorität eines Jobs startet der Scheduler mit der statischen Priorität und passt dies, entsprechend der Zeit in der der Job schon wartet, an. Wenn mehr als ein Job die gleiche dynamische Priorität hat, wird der Job mit der niedrigsten Job Id als erster gescheduled.

renice Die renice Option gleicht der nicevalue Option mit dem Unterschied, dass die renice Option relativ arbeitet, während die nicevalue Option absolut arbeitet. Wenn einige Batches einen nicevalue von 10 haben bewirkt eine renice von -5, dass die nicevalue auf 5 zunimmt. (Zunahme, weil je niedriger die Nummer, desto höher die Piorität).

rerun Die rerun Option wird benutzt um einen Job in einem restartable State neu zu starten. Der Versuch einen Job, der nicht restartable ist, neu zu starten, führt zu einer Fehlermeldung. Ein Job ist restartable, wenn er in einem restartable State oder in einem **error** oder **broken_finished** Job State ist.
Wenn das **recursive** Flag spezifiziert ist, wird der Job selbst und alle direkten und indirekten Children, die in einem restartable State sind, neu gestartet. Wenn der Job selbst final ist, wird das in dem Fall *nicht* als Fehler betrachtet. Es ist also möglich Batches rekursiv neu zu starten.

resume Die resume Option wird benutzt um einen suspended Job oder Batch zu reaktivieren. Es gibt dabei zwei Möglichkeiten. Erstens kann der suspended Job oder Batch sofort reaktiviert werden, und zweitens kann eine Verzögerung eingestellt werden.
Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
(Für die Spezifikation einer Zeit siehe auch die Übersicht auf Seite 21.)
Die resume Option kann zusammen mit der suspend Option verwendet werden. Dabei wird der Job suspended und nach der (bzw. zur) spezifizierten Zeit wieder resumed.

run Die run Option wird vom Jobserver benutzt zwecks der Sicherstellung, dass der geänderte Job mit der aktuellen Version übereinstimmt.
Theoretisch ist es möglich, dass nachdem ein Job von einem Jobserver gestartet wurde, der Computer abstürzt. Um die Arbeit zu erledigen wird der Job mittels eines manuellen Eingriffs, von einem anderen Jobserver, neu gestartet. Nach dem Hochfahren des ersten Systems kann der Jobserver versuchen den Job State nach **broken_finished** zu ändern, ohne über das Geschehen nach dem Absturz Bescheid

zu wissen. Das Benutzen der run Option verhindert nun das fälschliche Setzen des Status.

state Die state Option wird hauptsächlich von Jobservern benutzt, kann aber auch von Administratoren benutzt werden. Es wird nicht empfohlen dies so zu machen, es sei denn Sie wissen genau was Sie tun.
Die übliche Prozedur ist, dass der Jobserver den State eines Jobs von **starting** nach **started**, von **started** nach **running** und von **running** nach **finished** setzt. Im Falle eines Absturzes oder anderen Problemen ist es möglich dass der Jobserver einen Job in einen **broken_active** oder **broken_finished** State setzt. Das bedeutet, der Exit Code von dem Prozess steht nicht zur Verfügung und der Exit State muss manuell gesetzt werden.

suspend Die suspend Option wird benutzt um einen Batch oder Job zu suspendieren. Sie arbeitet nur dann rekursiv wenn **local** nicht spezifiziert ist. Wenn ein Parent suspended ist, sind auch alle Children suspended. Die resume Option wird benutzt um die Situation umzukehren. Die **restrict** Angabe bewirkt, dass nur Benutzer der ADMIN Gruppe die Suspendierung wieder aufheben können.

timestamp Die timestamp Option wird vom Jobserver benutzt um die Timestamps der State-Wechsel zu setzen, gemäß der lokalen Zeit aus Sicht des Jobservers.

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

connect

Zweck

Zweck Das *connect* Statement wird eingesetzt um einen Job am Server zu authentifizieren.

Syntax

Syntax Die Syntax des *connect* Statements ist

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

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

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

Beschreibung

Beschreibung Das *connect* Kommando wird benutzt um den verbundenen Prozess am Server zu authentifizieren. Es kann wahlweise ein Kommunikationsprotokoll spezifiziert werden. Das Default-Protokoll ist **line**.
Das ausgewählte Protokoll definiert die Form des Outputs. Alle Protokolle, außer **serial**, liefern ASCII Output. Das Protokoll **serial** liefert ein Serialized Java Objekt zurück.
Beim *connect* Kommando kann auch gleich ein auszuführendes Kommando mitgegeben werden. Als Output des *connect* Kommandos wird in diesem Fall der Output des mitgegebenen Kommandos genutzt. Falls das Kommando fehlschlägt, der *connect* aber gültig war, bleibt die Connection bestehen.

Im Folgenden ist für alle Protokolle, außer für das **serial** Protokoll, ein Beispiel gegeben.

line protocol Das line protocol liefert nur einen ASCII-Text als Ergebnis von einem Kommando zurück.

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

Connect

CONNECT_TIME : 19 Jan 2005 11:12:43 GMT

Connected

SDMS>
```

XML protocol Das XML protocol liefert eine XML-Struktur als Ergebnis eines Kommandos zurück.

```
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>
```

python protocol Das python protocol liefert eine Python-Struktur, welche durch die *python eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

perl protocol Das perl protocol liefert eine Perl-Struktur, welche durch die *perl eval* Funktion ausgewertet werden kann, zurück.

```
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'
}
```

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

disconnect

Zweck

Das *disconnect* Statement wird eingesetzt um die Serververbindung zu beenden. *Zweck*

Syntax

Die Syntax des *disconnect* Statements ist *Syntax*

disconnect

Beschreibung

Mit dem *disconnect* Statement kann die Verbindung zum Server beendet werden. *Beschreibung*

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert. *Ausgabe*

get parameter

Zweck

Zweck Das *get parameter* Statement wird eingesetzt um den Wert des spezifizierten Parameters innerhalb des Kontext des anfordernden Jobs, entsprechend seiner Spezifikation, zu bekommen.

Syntax

Syntax Die Syntax des *get parameter* Statements ist

get parameter *parametername* [< **strict** | **warn** | **liberal** >]

get parameter of *jobid parametername* [< **strict** | **warn** | **liberal** >]

Beschreibung

Beschreibung Das *get parameter* Statement wird eingesetzt um den Wert des spezifierten Parameters innerhalb des Kontextes eines Jobs zu bekommen.
Die Zusatzoption hat dabei folgende Bedeutung:

Option	Bedeutung
strict	Der Server liefert einen Fehler, wenn der gefragte Parameter nicht explizit in der Job Definition deklariert ist
warn	Es wird eine Meldung ins Logfile des Server geschrieben, wenn versucht wird den Wert eines nicht deklarierten Parameters zu ermitteln.
liberal	Der Versuch nicht deklarierte Parameter abzufragen wird stillschweigend erlaubt.

Das Default-Verhalten hängt von der Serverkonfiguration ab.

Ausgabe

Ausgabe Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
VALUE	Wert des angeforderten Parameters

Tabelle 28.1: Beschreibung der Output-Struktur des get parameter Statements

get submittag

Zweck

Zweck

Das *get submittag* Statement wird eingesetzt um eine eindeutige Identifikation vom Server zu bekommen. Diese Identifikation kann benutzt werden, um *race conditions* zwischen Frontend und Backend während des Submits zu verhindern.

Syntax

Syntax

Die Syntax des *get submittag* Statements ist

get submittag

Beschreibung

Beschreibung

Mit dem *get submittag* Statement bekommt man eine Identifikation vom Server. Damit verhindert man Race Conditions zwischen Frontend und Backend wenn Jobs submitted werden.
Eine solche Situation entsteht, wenn aufgrund eines Fehlers die Rückmeldung des Submits nicht ins Frontend eintrifft. Durch Benutzung eines Submit Tags kann das Frontend gefahrlos einen zweiten Versuch starten. Der Server erkennt, ob der betreffende Job bereits submitted wurde und antwortet dementsprechend. Ein doppeltes Submitten des Jobs wird damit zuverlässig verhindert.

Ausgabe

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record.

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
VALUE	Das angeforderte Submit Tag

Tabelle 28.2: Beschreibung der Output-Struktur des get submittag Statements

multicommand

Zweck

Zweck Der Zweck des *multicommands* ist es mehrere SDMS-Kommandos als Einheit zuzuführen.

Syntax

Syntax Die Syntax des *multicommand* Statements ist

begin multicommand *commandlist* **end multicommand**

begin multicommand *commandlist* **end multicommand rollback**

Beschreibung

Beschreibung Mit den *multicommands* ist es möglich mehrere SDMS-Kommandos zusammen, d.h. in einer Transaktion auszuführen. Damit wird gewährleistet, dass entweder alle Statements fehlerfrei ausgeführt werden, oder nichts passiert. Des Weiteren wird die Transaktion nicht von anderen schreibenden Transaktionen unterbrochen. Wird das Keyword **rollback** spezifiziert, wird die Transaktion am Ende der Verarbeitung rückgängig gemacht. Auf diese Weise kann getestet werden, ob die Statements (technisch) korrekt verarbeitet werden können.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

set parameter

Zweck

Zweck

Das *set parameter* Statement wird eingesetzt um den Wert des spezifizierten Parameters innerhalb des Kontext eines Jobs zu setzen.

Syntax

Syntax

Die Syntax des *set parameter* Statements ist

set parameter *parametername* **=** *string* {**,** *parametername* **=** *string*}

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,**
parametername **=** *string*} [**with comment =** *string*]

set parameter < **on** | **of** > *jobid parametername* **=** *string* {**,**
parametername **=** *string*} **identified by** *string* [**with comment =** *string*]

Beschreibung

Beschreibung

Mittels des *set parameter* Statements können Jobs oder Benutzer Parameterwerte im Kontext des Jobs setzen.
Falls die **identified by** Option spezifiziert ist, wird der Parameter nur dann gesetzt, wenn das Paar *jobid* und *string* eine Anmeldung ermöglichen würden.

Ausgabe

Ausgabe

Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

set state

Zweck

Zweck Das *set state* Statement wird eingesetzt um den Exit State eines Jobs in einem pending Exit State zu setzen.

Syntax

Syntax Die Syntax des *set state* Statements ist

set state = *statename*

Beschreibung

Beschreibung Das *set state* Statement wird eingesetzt um den Exit State eines Jobs in einem pending Exit State zu setzen.

Ausgabe

Ausgabe Es wird die Bestätigung einer erfolgreichen Durchführung zurückgeliefert.

submit

Zweck

Das *submit* Statement wird eingesetzt um einen Master Batch oder Job, sowie alle definierten Children, auszuführen. *Zweck*

Syntax

Die Syntax des *submit* Statements ist *Syntax*

submit *folderpath* [**with** WITHITEM {**,** WITHITEM}]

submit *aliasname* [**with** WITHITEM {**,** WITHITEM}]

WITHITEM:
check only
| **childtag =** *string*
| < **enable** | **disable** >
| **master**
| **nicevalue =** *signed_integer*
| **parameter = none**
| **parameter = (** PARAM {**,** PARAM} **)**
| < **noresume** | **resume in** *period* | **resume at** *datetime* >
| **submittag =** *string*
| < **nosuspend** | **suspend** >
| **time zone =** *string*
| **unresolved =** JRQ_UNRESOLVED
| **group =** *groupname*

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

JRQ_UNRESOLVED:
defer
| **defer ignore**
| **error**
| **ignore**
| **suspend**

Beschreibung

Beschreibung Das *submit* Statement wird benutzt um einen Job oder Batch zu submitten. Es existieren zwei Formen des Submit-Kommandos.

- Die erste Form wird von Benutzern, welche auch Programme sein können und dem Time Scheduling Module genutzt. Diese Form submitted Master Jobs und Batches.
- Die zweite Form des Statements wird von Jobs genutzt, um dynamische Children zu submitten.

check only Die check only Option wird benutzt, um zu überprüfen, ob ein Master Submittable Batch oder Job submitted werden kann. Das bedeutet, es wird geprüft, ob alle Abhängigkeiten erfüllt werden können und alle referenzierten Parameter definiert sind.
Es wird nicht überprüft, ob die Jobs in irgendeinem Scope ausgeführt werden können oder nicht. Dies ist eine Situation die jederzeit zur Laufzeit auftreten kann.
Eine positive Rückmeldung bedeutet, dass der Job oder Batch aus Sicht des Systems submitted werden kann.
Die check only Option kann nicht in einem Job-Kontext benutzt werden.

childtag Die childtag Option wird von Jobs benutzt, um verschiedene Instanzen von demselben Scheduling Entity zu submitten und um zwischen ihnen unterscheiden zu können.
Es führt zu einem Fehler, wenn der gleiche Scheduling Entity doppelt submitted wird, wenn sich der childtag nicht unterscheidet. Der Inhalt des childtags hat keine weitere Bedeutung für das Scheduling System.
Die maximale Länge eines childtags beträgt 70 Zeichen. Die childtag Option wird im Falle eines Master Submits ignoriert.

group Die group Option wird benutzt um die Owner-Gruppe auf den spezifizierten Wert zu setzen. Der Benutzer muss zu dieser Gruppe gehören, es sei denn er gehört zu der priviligierten Gruppe ADMIN, in diesem Fall kann jede beliebige Gruppe spezifiziert werden.

nicevalue Die nicevalue Option definiert eine Korrektur die für die Berechnung der Prioritäten des Jobs und seiner Children benutzt wird. Es sind Werte von -100 bis 100 erlaubt.

parameter Die parameter Option wird benutzt um den Wert von Job Parametern beim Submit zu spezifizieren. Die Parameter werden im Scope des Master Batches oder Jobs gesetzt. Das bedeutet, wenn Parameter, die nicht in dem Master Batch oder Job definiert sind, spezifiziert werden, sind diese Parameter unsichtbar für Children.

submittag Wenn der submittag spezifiziert ist, muss er eine eindeutige Bezeichnung für den Submitted Entity haben. Dieser Tag wurde, um imstande zu sein Jobs und Batches programmatisch zu submitten und um den Job oder Batch, mit demselben Tag, nach einem Absturz von einem der Komponenten neu zu submitten. Wenn die Submission des Jobs das erste Mal erfolgreich war, wird der zweite Submit einen Fehler melden. Wenn nicht, wird der zweite Submit erfolgreich sein.

unresolved Die unresolved Option spezifiziert wie der Server bei nicht auflösbaren Abhängigkeiten reagieren sollte. Diese Option wird hauptsächlich benutzt, wenn Teile eines Batches nach Reparaturarbeiten submitted werden. Der fehlerhafte Teil wird normal gecancelt und dann als Master Run neu submittet. Die vorherigen Abhängigkeiten müssen in diesem Fall ignoriert werden, andernfalls wird der Submit scheitern.

suspend Die suspend Option wird benutzt um Jobs oder Batches zu submitten und sie zur selben Zeit zu suspenden. Wenn nichts festgelegt wurde, wird nicht suspended. Dies kann explizit zur Submit-Zeit spezifiziert werden.
Wenn ein Job oder Batch suspended wurde, wird er, sowie auch seine Children, nicht gestartet. Wenn ein Job bereits läuft, wird er keinen final State erreichen, wenn er suspended ist.

resume Die resume Option kann zusammen mit der suspend Option verwendet werden um eine verzögerte Ausführung zu bewirken. Es gibt dabei zwei Möglichkeiten. Entweder erreicht man eine Verzögerung dadurch, dass die Anzahl von Zeiteinheiten die gewartetet werden sollen, spezifiziert werden, oder aber man spezifiziert den Zeitpunkt zu dem der Job oder Batch aktiviert werden soll.
Mit dieser Option kann die `at`-Funktionalität ohne das Anlegen eines Schedules nachgebildet werden.

Ausgabe

Dieses Statement liefert eine Output-Struktur vom Typ Record. *Ausgabe*

Output-Beschreibung Die Datenelemente des Outputs werden in der nachfolgenden Tabelle beschrieben.

Feld	Beschreibung
ID	Id des Submitted Entities

Tabelle 28.3: Beschreibung der Output-Struktur des submit Statements

Teil V

Programming Examples

Kapitel 29

Programmierbeispiele

In diesem Abschnitt werden einige einfache Beispiele gegeben, die zeigen wie in einigen Programmiersprachen mit dem Scheduling Server kommuniziert werden kann.
In den Beispielen geht es darum die wesentliche Struktur zu zeigen. Die Fehlerbehandlung ist sehr rudimentär und auch die Verarbeitung der Server Responses ist minimal gehalten.
Für eine Anmeldung an den Scheduling Server werden bekanntlich einige Daten benötigt: Hostname oder IP Adresse des Systems auf dem der Scheduling Server läuft, der Port auf den er hört (im Normalfall die 2506), ein Username und ein Passwort. Diese Daten werden in den Beispielen als Konstanten definiert. Es mag klar sein, dass eine seriöse Implementierung eine andere Methode, wie zum Beispiel das Auswerten der `.sdmshrc`, benutzen sollte.
Unter `$BICSUITEHOME/examples` sind alle abgebildeten Programme als Source Code vorhanden.

Java

Java

Da schedulix selbst in Java geschrieben ist, kann bei der Entwicklung von Utilities in Java die `BICsuite.jar` genutzt werden.
Im unten stehenden Beispiel wird die `SDMSServerConnection` benutzt um die Verbindung zum Scheduling Server aufzubauen. Dazu wird zuerst mit Hilfe der Standardinformation ein Objekt angelegt. Anschließend wird mittels der Methode `connect()` die Verbindung aufgebaut. Die `finish()` Methode wird benutzt um die Verbindung zu terminieren.
So lange die Verbindung aktiv ist, können beliebige Statements mit Hilfe der Methode `execute()` ausgeführt werden. Im unten stehenden Beispiel wird der Befehl `list sessions;` ausgeführt.
Als Ergebnis wird ein Objekt vom Typ `SDMSOutput` zurückgegeben. Wenn die Member Variable `error` nicht `null` ist, trat während der Verarbeitung des Befehls ein Fehler auf. Die Member Variablen `error.code` sowie `error.message` geben

nähere Information zum vorliegenden Fehler.
Im Beispiel wird die Klasse `SDMSLineRenderer` benutzt um das formatierte Ergebnis des Befehls auf `stdout` auszugeben.
Die Fehlerbehandung ist sehr spartanisch gehalten. Wenn ein Fehler auftritt, wird das Programm mit einem Return Code 1 verlassen.

```
1  import de.independit.scheduler.shell.SDMSServerConnection;
2  import de.independit.scheduler.server.output.SDMSOutput;
3  import de.independit.scheduler.server.output.SDMSLineRenderer;
4  import java.io.IOException;
5
6  public class SimpleAccess
7  {
8
9      private static SDMSServerConnection sc = null;
10     private static SDMSLineRenderer lr = null;
11
12     public static void main(String argv[])
13     {
14         sc = new SDMSServerConnection(
15             "localhost",     /* host */
16             2506,            /* port */
17             "SYSTEM",        /* user */
18             "G0H0ME",        /* password */
19             0,               /* connection timeout disabled */
20             false            /* no TLS */
21         );
22         lr = new SDMSLineRenderer();
23
24         try {
25             SDMSOutput o = sc.connect(null /* no special options */);
26             if (o.error != null) {
27                 System.err.println("Connect Error: " +
28                                    o.error.code + ", " + o.error.message)
    ;
29                 System.exit(1);
30             }
31
32             o = sc.execute("LIST SESSIONS;");
33             try {
34                 lr.render(System.out, o);
35             } catch (Exception e) {
36                 System.err.println("Something went wrong: " +
37                                    e.toString());
38             }
39
40             sc.finish();
41         } catch (IOException ioe) {
42             System.err.println("Something went wrong : " +
43                                ioe.toString());
44             System.exit(1);
```

```
45          }
46
47          System.exit(0);
48      }
49  }
```

Um das Java Programm umzuwandeln sollte die `BICsuite.jar` im `CLASSPATH` aufgenommen werden. Unter Unix oder Linux könnte das folgendermaßen aussehen (die Outputzeilen wurden aus Darstellungsgründen gekürzt):

```
$ CLASSPATH=$CLASSPATH:$BICSUITEHOME/lib/BICsuite.jar
$ export CLASSPATH
$ javac SimpleAccess.java
$ java SimpleAccess

List of Sessions

THIS SESSIONID PORT START                          TYPE      USER           ...
---- --------- ---- ----------------------------- --------- ---------------...
          1001 2506 Mon Oct 12 11:25:47 CEST 2020 JOBSERVER GLOBAL.EXAMPLES...
          1002 2506 Mon Oct 12 11:25:47 CEST 2020 JOBSERVER GLOBAL.EXAMPLES...
          1003 2506 Mon Oct 12 11:25:47 CEST 2020 JOBSERVER GLOBAL.EXAMPLES...
 *        1043 2506 Wed Oct 21 15:00:12 CEST 2020 USER      SYSTEM         ...
       1234321    0 Mon Oct 12 11:25:22 CEST 2020 USER      SchedulingThrea...
       1234322    0 Mon Oct 12 11:25:22 CEST 2020 USER      GarbageThread  ...
       1234323    0 Mon Oct 12 11:25:22 CEST 2020 USER      TriggerThread  ...
       1234324    0 Mon Oct 12 11:25:22 CEST 2020 USER      PoolThread     ...
      19630127    0 Mon Oct 12 11:25:22 CEST 2020 USER      TimerThread    ...


9 Session(s) found
```

Ein zweites Beispiel zeigt wie Attribute aus der Output Struktur abgefragt werden können. In dem Beispiel werden, nach dem Verbindungsaufbau, zwei Befehle abgesetzt und aus den beiden Ergebnissen anschließend selektiv Daten extrahiert und ausgegeben.
Das Ergebnis eines `SHOW SYSTEM` Befehls ist immer ein Record mit einer Tabelle. In Zeile 41 wird die Versionsinformation aus den Recorddaten extrahiert. In Zeile 44 bis 47 werden die Namen der Worker Threads aus der Tabelle mit Namen `WORKER` ermittelt.
Das Ergebnis eines `LIST SESSIONS` Befehl ist immer eine reine Tabelle. In Zeile 58 bis 61 werden die Namen der angemeldeten Benutzer, Jobserver sowie internal Threads ermittelt und ausgegeben.

```
1 import java.io.IOException;
2 import de.independit.scheduler.shell.SDMSServerConnection;
3 import de.independit.scheduler.server.output.SDMSOutput;
4 import de.independit.scheduler.server.output.SDMSOutputUtil;
```

```
5
6  public class testJavaApi {
7
8      public static void main(String[] args) {
9
10         SDMSServerConnection sc = new SDMSServerConnection(
11             "localhost",     /* host */
12             2506,            /* port */
13             "SYSTEM",        /* user */
14             "G0H0ME",        /* password */
15             0,               /* connection timeout disabled */
16             false            /* no TLS */
17         );
18         SDMSOutput output = null;
19
20         try {
21             output = sc.connect(null);
22         } catch (IOException ioe) {
23             System.err.println("Error '" + ioe.toString() +
24                         "' establishing BICsuite server connection");
25             System.exit(1);
26         }
27         if (output.error != null) {
28             System.err.println("Error '" + output.error.code + ":" +
29                    output.error.message + "' connecting to BICsuite
       server");
30             System.exit(1);
31         }
32
33         String command = "SHOW SYSTEM";
34         output = sc.execute(command);
35         if (output.error != null) {
36             System.err.println("Error '" + output.error.code + ":" +
37                    output.error.message + "' executing command: " +
       command);
38             System.exit(1);
39         }
40
41         System.out.println("Version: " + SDMSOutputUtil.getFromRecord(
       output,"VERSION"));
42         int workers = SDMSOutputUtil.getTableLength(output,"WORKER");
43         System.out.println("Workers: " + workers);
44         for (int i = 0; i < workers; i ++) {
45             System.out.println("  Name: " +
46                    SDMSOutputUtil.getFromTable(output, "WORKER", i, "NAME
       "));
47         }
48
49         command = "LIST SESSIONS";
50         output = sc.execute(command);
51         if (output.error != null) {
52             System.err.println("Error '" + output.error.code + ":" +
53                    output.error.message + "' executing command: " +
```

```
    command);
54              System.exit(1);
55          }
56          int sessions = SDMSOutputUtil.getTableLength(output);
57          System.out.println("Sessions: " + sessions);
58          for (int i = 0; i < sessions; i ++) {
59              System.out.println("  User: " +
60                     SDMSOutputUtil.getFromTable(output, i, "USER"));
61          }
62
63          try {
64              sc.finish();
65          } catch (IOException ioe) {
66              System.err.println("Error '" + ioe.toString() +
67                         "' closing BICsuite server connection");
68              System.exit(1);
69          }
70      }
71  }
```

Das Umwandeln und Ausführen des Programms funktioniert natürlich genauso wie im ersten Beispiel. Selbstverständlich muss der `CLASSPATH` nicht vor jeder Umwandlung oder Ausführung erneut gesetzt werden.

```
$ CLASSPATH=$CLASSPATH:$BICSUITEHOME/lib/BICsuite.jar
$ export CLASSPATH
$ javac testJavaApi.java
$ java testJavaApi
Version: 2.10
Workers: 6
  Name: Worker0
  Name: Worker1
  Name: Worker2
  Name: Worker3
  Name: Worker4
  Name: Worker5
Sessions: 9
  User: GLOBAL.EXAMPLES.HOST_1.SERVER
  User: GLOBAL.EXAMPLES.LOCALHOST.SERVER
  User: GLOBAL.EXAMPLES.HOST_2.SERVER
  User: SYSTEM
  User: SchedulingThread
  User: GarbageThread
  User: TriggerThread
  User: PoolThread
  User: TimerThread
```

Python 2

Python 2

Auch der Zugriff mit Python 2 ist ziemlich einfach. Immerhin wurde der Zope Application Server in Python geschrieben und benutzt die Datei `sdms.py` als Extension um die Kommunikation mit dem Scheduling Server abzuwickeln.

Selbstverständlich kann diese Datei auch von jedem beliebigen anderen Python Script benutzt werden.
Mit Hilfe der `SDMSConnectionOpenV2()` Methode wird die Verbindung zum Scheduling Server aufgebaut. Diese Methode benötigt als ersten Parameter ein Dictionary mit der Spezifikation von Host und Port. Zwei weitere Parameter spezifizieren User und Passwort. Der letzte Parameter ist optional und dient nur dazu der Session einen sinnvollen Namen zu geben.
Wenn der Verbindungsaufbau fehlschlägt wird ein Dictionary anstelle eines Socket Objektes zurückgeliefert. Dies kann leicht mit Hilfe der `has_key` Methode in einem `try` - `except` Block geprüft werden. Im unten stehenden Code Fragment ist dies in den Zeilen 11 bis 16 abgebildet.
Sobald die Verbindung existiert, können mittels `SDMSCommandWithSoc` beliebige Befehle ausgeführt werden. Das Resultat ist immer eine `SDMSOutput` Datenstruktur. Falls ein Fehler aufgetreten ist, enthält diese einen `ERROR` Eintrag.
Die `close()` Methode terminiert die Verbindung.

```
1  import sdms
2
3  server = {'HOST' : 'localhost',
4            'PORT' : '2506',
5            'USER' : 'SYSTEM',
6            'PASSWORD' : 'G0H0ME' }
7  conn = sdms.SDMSConnectionOpenV2(server,
8                                   server['USER'],
9                                   server['PASSWORD'],
10                                  "Simple Access Example")
11 try:
12     if conn.has_key('ERROR'):
13         print str(conn)
14         exit(1)
15 except:
16         pass
17
18 stmt = "LIST SESSIONS;"
19 result = sdms.SDMSCommandWithSoc(conn, stmt)
20 if result.has_key('ERROR'):
21     print str(result['ERROR'])
22 else:
23     for row in result['DATA']['TABLE']:
24         print "{0:3} {1:8}  {2:32}  {3:9}  {4:15}  {5:>15}  {6}".format(\
25             row['THIS'], \
26             row['UID'], \
27             row['USER'], \
28             row['TYPE'], \
29             row['START'], \
30             row['IP'], \
31             row['INFORMATION'])
32
```

```
33 conn.close()
```

Um das Programm auszuführen muss nur der PYTHONPATH entsprechend gesetzt werden. Aus Darstellungsgründen wurde der Output gekürzt.

```
$ PYTHONPATH=$PYTHONPATH:$BICSUITEHOME/../schedulixweb/Extensions
$ export PYTHONPATH
$ python2 SimpleAccess.py
   1047  GLOBAL.EXAMPLES.HOST_1.SERVER    JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
   1037  GLOBAL.EXAMPLES.LOCALHOST.SERVER JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
   1057  GLOBAL.EXAMPLES.HOST_2.SERVER    JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
 * 0     SYSTEM                           USER       Wed Oct 21 14:20:40 CEST 20...
   2     SchedulingThread                 USER       Mon Oct 12 11:25:22 CEST 20...
   2     GarbageThread                    USER       Mon Oct 12 11:25:22 CEST 20...
   2     TriggerThread                    USER       Mon Oct 12 11:25:22 CEST 20...
   2     PoolThread                       USER       Mon Oct 12 11:25:22 CEST 20...
   2     TimerThread                      USER       Mon Oct 12 11:25:22 CEST 20...
```

Python 3

Python 3

In einer Python 3 Umgebung läuft alles genauso wie in der Python 2 Umgebung, natürlich abgesehen von den Differenzen in den beiden Sprachen. Das Python 3 Modul befindet sich im Zope4 Tree, auch unter Extensions.

```
1  import sdms
2
3  server = {'HOST' : 'localhost',
4        'PORT' : '2506',
5        'USER' : 'SYSTEM',
6        'PASSWORD' : 'G0H0ME' }
7  conn = sdms.SDMSConnectionOpenV2(server,
8                                   server['USER'],
9                                   server['PASSWORD'],
10                                  "Simple Access Example")
11 try:
12     if 'ERROR' in conn:
13         print(str(conn))
14         exit(1)
15 except:
16     pass
17
18 stmt = "LIST SESSIONS;"
19 result = sdms.SDMSCommandWithSoc(conn, stmt)
20 if 'ERROR' in result:
21     print(str(result['ERROR']))
22 else:
23     for row in result['DATA']['TABLE']:
24         print("{0:3} {1:8}  {2:32}  {3:9}  {4:15}  {5:>15}  {6}".format(\
25             str(row['THIS']), \
```

```
26              str(row['UID']), \
27              str(row['USER']), \
28              str(row['TYPE']), \
29              str(row['START']), \
30              str(row['IP']), \
31              str(row['INFORMATION'])))
32
33 conn.close()
```

Die Ausführung funktioniert genauso wie bei Python 2:

```
$ PYTHONPATH=$PYTHONPATH:/opt/schedulix/schedulix/../schedulixweb4/Extensions
$ export PYTHONPATH
$ python3 SimpleAccess3.py
   1047  GLOBAL.EXAMPLES.HOST_1.SERVER    JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
   1037  GLOBAL.EXAMPLES.LOCALHOST.SERVER JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
   1057  GLOBAL.EXAMPLES.HOST_2.SERVER    JOBSERVER  Mon Oct 12 11:25:47 CEST 20...
 * 0     SYSTEM                           USER       Wed Oct 21 15:33:31 CEST 20...
   2     SchedulingThread                 USER       Mon Oct 12 11:25:22 CEST 20...
   2     GarbageThread                    USER       Mon Oct 12 11:25:22 CEST 20...
   2     TriggerThread                    USER       Mon Oct 12 11:25:22 CEST 20...
   2     PoolThread                       USER       Mon Oct 12 11:25:22 CEST 20...
   2     TimerThread                      USER       Mon Oct 12 11:25:22 CEST 20...
```

C

C Für den Zugang aus einem C-Programm heraus wird unser C-API benutzt. Dieses findet man unter `$BICSUITEHOME/src/capi`. Bekanntlich ist C eine relativ hardwarenahe Programmiersprache, in der Themen wie etwa Speicherverwaltung weitgehend dem Entwickler überlassen werden. Dementsprechend ist auch das Handling mit den Outputstrukturen komplexer als in Java oder Python. Dennoch wurde versucht die ganze Bedienung so einfach wie möglich zu gestalten.
Die Prototypen der zur Verfügung stehenden Funktionen stehen im `sdms_api.h` Header File. Der relevante Teil der Datei ist unten stehend abgebildet.

```
1 extern int sdms_connection_open(SDMS_CONNECTION **connection, char *host,
      int port,
2                                 char *user, char *password);
3 extern int sdms_command(SDMS_OUTPUT **output, SDMS_CONNECTION *connection
    ,
4                         SDMS_STRING *command);
5 extern int sdms_connection_close(SDMS_CONNECTION **connection);
6
7 extern int sdms_string(SDMS_STRING **sdms_string, char *s);
8 extern int sdms_string_append(SDMS_STRING *string, char *text);
9 extern void sdms_string_clear(SDMS_STRING *string);
10 extern void sdms_string_free(SDMS_STRING **string);
11
```

```
12 extern int sdms_vector(SDMS_VECTOR **vector);
13 extern int sdms_vector_append(SDMS_VECTOR *vector, void *data);
14 extern void sdms_vector_free(SDMS_VECTOR **vector);
15
16 extern void sdms_output_free(SDMS_OUTPUT **output);
17
18 extern void sdms_error_print(char *text);
19 extern void sdms_error_clear(void);
20
21 extern int sdms_output_data_get_table_size(SDMS_OUTPUT_DATA *output_data,
       int *size);
22 extern int sdms_vector_find(SDMS_VECTOR *vector, char *name);
23 extern int sdms_output_data_get_by_name (SDMS_OUTPUT_DATA *output_data,
24                                          SDMS_OUTPUT_DATA **value, char *
      name);
25 extern int sdms_output_data_get_string(SDMS_OUTPUT_DATA *output_data,
      char **value);
26 extern int sdms_output_data_get_row(SDMS_OUTPUT_DATA *output_data,
27                                     SDMS_VECTOR **row, int index);
```

Die Funktionen `sdms_connection_open()` sowie `sdms_connection_close()` sprechen für sich. Die Funktion `sdms_command()` führt das im `command` spezifizierte Kommando aus. Das Ergebnis wird im Parameter `output` zurückgeliefert.

Da ein Parameter vom Typ `SDMS_STRING` zum Ausführen von Befehlen benötigt wird, stehen eine Anzahl Funktionen für den Umgang mit diesem Datentyp bereit. Mit Hilfe der Funktion `sdms_string()` kann ein normaler String in C in ein `SDMS_STRING` verwandelt werden. Die Funktion `sdms_string_append()` dient dazu, ein `SDMS_STRING` um den spezifizierten Text zu erweitern. Die Funktion `sdms_string_clear()` löscht den Inhalt des Strings. Da für die Arbeit mit Strings dynamisch allokierter Speicher benötigt wird, gibt es zum Schluss noch die `sdms_string_free()` Funktion um den Speicher auch wieder kontrolliert frei zu geben.

Vielfach werden Daten in Form einer Liste von Werten, oder gar eine Liste von Listen zurückgeliefert. In Java wird dazu ein Vector benutzt. In Anlehnung daran wird in der C-Schnittstelle ein `SDMS_VECTOR` bereitgestellt. Die Funktionen zur Manipulation dieser Datenstruktur sind in etwa vergleichbar mit den `SDMS_STRING` Funktionen. Normalerweise werden diese Funktionen in Anwendungen jedoch nicht genutzt, da die Vectoren nicht von der Anwendung, sondern vielmehr vom Interface aufgebaut werden. Viel interessanter sind dafür die Funktionen die elementare Daten aus den Vektoren extrahieren.

Die Datenstruktur `SDMS_OUTPUT` ist der umfassende Container in dem die Ergebnisse von Befehlen zurückgeliefert werden. Der Container ist aufgebaut aus verschiedenen Datentypen die im Normalfall in dynamisch allokierten Speichern abgelegt werden. Damit dieser Speicher wieder freigegeben werden kann, wird die

Funktion `sdms_output_free()` aufgerufen. Diese Funktion berücksichtigt auch korrekterweise die dynamische interne Datenstruktur.
Nach dem Motto "Ein Bild sagt mehr alt tausend Worte", werden im unten stehenden Programm nach dem Aufbau der Verbindung ein `SHOW USER;` sowie ein `LIST SESSIONS;` ausgeführt und die Ergebnisse auf dem Bildschirm ausgegeben.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #ifdef _WIN32
5  #include <winsock.h>
6  #endif
7
8  #include "sdms_api.h"
9
10 /* some constants / literals */
11 /* default values */
12 char * LOCALHOST = (char *) "localhost";
13 char * PORT      = (char *) "2506";
14 char * SYSTEM    = (char *) "SYSTEM";
15 char * PASSWD    = (char *) "G0H0ME";
16
17 /* column names */
18 char * NAME      = (char *) "NAME";
19 char * GROUPS    = (char *) "GROUPS";
20 char * SESSIONID = (char *) "SESSIONID";
21 char * USER      = (char *) "USER";
22
23 /* used commands */
24 char * SHOW_USER = (char *) "SHOW USER;";
25 char * LIST_SESSION = (char *) "LIST SESSIONS;";
26
27 void do_exit (int exit_code);
28
29 /* sdms_connection_open() requires initialized pointer */
30 SDMS_CONNECTION *sdms_connection = NULL;
31
32 int main (int argc, char *argv[])
33 {
34     char *host;
35     char *port;
36     char *user;
37         char *pass;
38     if (argc >= 2)
39         host = argv[1];
40     else
41         host = LOCALHOST;
42     if (argc >= 3)
43         port = argv[2];
44     else
45         port = PORT;
```

```
46      if (argc >= 4)
47          user = argv[3];
48      else
49          user = SYSTEM;
50      if (argc >= 5)
51          pass = argv[4];
52      else
53          pass = PASSWD;
54
55
56  #ifdef _WIN32
57      WSADATA wsaData;
58      if (WSAStartup (MAKEWORD(1, 1), &wsaData) != 0) {
59          fprintf (stderr, "WSAStartup(): Can't initialize Winsock.\n");
60          do_exit (1);
61      }
62  #endif
63
64      if (sdms_connection_open(&sdms_connection, host,
65                               atoi(port), user, pass) != SDMS_OK) {
66          sdms_error_print((char *) "Error opening sdms connection");
67          do_exit(1);
68      }
69
70      int size;
71      int i;
72
73      printf("-------------------------------------------------------\n");
74
75      /* sdms_string() requires initialized pointer */
76      SDMS_STRING *command = NULL;
77
78      /* sdms_command() requires initialized pointer */
79      SDMS_OUTPUT *sdms_output = NULL;
80
81      if (sdms_string (&command, SHOW_USER) != SDMS_OK) {
82          sdms_error_print((char *) "Error allocating command SDMS_STRING")
    ;
83          do_exit(1);
84      }
85
86      if (sdms_command (&sdms_output,
87                        sdms_connection, command) != SDMS_OK) {
88          sdms_error_print((char *) "Error executing command");
89          do_exit(1);
90      }
91
92      /* sdms_output_dump(sdms_output); */
93
94      SDMS_OUTPUT_DATA *name;
95      sdms_output_data_get_by_name(sdms_output->data, &name, NAME);
96      fprintf (stderr, "User %s is in the groups", (char *)(name->data));
97
```

```
98      SDMS_OUTPUT_DATA *groups;
99      sdms_output_data_get_by_name(sdms_output->data, &groups, GROUPS);
100     int groupname_idx = sdms_vector_find(groups->desc, NAME);
101     sdms_output_data_get_table_size(groups, &size);
102     char sep = ' ';
103     for (i = 0; i < size; i ++) {
104         SDMS_VECTOR *row;
105         sdms_output_data_get_row(groups, &row, i);
106         SDMS_OUTPUT_DATA *groupname =
107                         (SDMS_OUTPUT_DATA *)(row->buf[groupname_idx]);
108         fprintf (stderr, "%c%s", sep, (char *)(groupname->data));
109         sep = ',';
110     }
111     fprintf (stderr, "\n");
112
113     sdms_output_free(&sdms_output);
114
115     printf("------------------------------------------------------\n");
116
117     sdms_string_clear(command);
118     if (sdms_string_append(command, LIST_SESSION) != SDMS_OK) {
119         sdms_error_print((char *) "Error building command");
120         do_exit(1);
121     }
122     if (sdms_command (&sdms_output, sdms_connection, command) != SDMS_OK)
        {
123         sdms_error_print((char *) "Error executing command");
124         do_exit(1);
125     }
126     /* sdms_output_dump(sdms_output); */
127
128     SDMS_OUTPUT_DATA *data = sdms_output->data;
129     int sessionid_idx = sdms_vector_find(data->desc, SESSIONID);
130     int user_idx = sdms_vector_find(data->desc, USER);
131     sdms_output_data_get_table_size(data, &size);
132     for (i = 0; i < size; i ++) {
133         SDMS_VECTOR *row;
134         sdms_output_data_get_row(data, &row, i);
135         SDMS_OUTPUT_DATA *sessionid =
136                          (SDMS_OUTPUT_DATA *)(row->buf[sessionid_idx]);
137         SDMS_OUTPUT_DATA *data_user =
138                          (SDMS_OUTPUT_DATA *)(row->buf[user_idx]);
139         fprintf (stderr, "User %s connected with id %s\n",
140             (char *)(data_user->data), (char *)(sessionid->data));
141     }
142
143     sdms_output_free(&sdms_output);
144
145     printf("------------------------------------------------------\n");
146
147     sdms_string_free(&command);
148
149     if (sdms_connection_close(&sdms_connection) != SDMS_OK) {
```

```
150         sdms_error_print((char *) "Error closing sdms connection");
151         do_exit(1);
152     }
153
154     return 0;
155 }
156
157 void do_exit (int exit_code)
158 {
159     // Try to close connection
160     if (sdms_connection != NULL)
161         sdms_connection_close(&sdms_connection);
162 #ifdef _WIN32
163     WSACleanup();
164 #endif
165     exit(1);
166 }
```

Das Umwandeln und Ausführen des Programms ist vergleichsweise einfach. Es steht dazu ein Makefile bereit, was zumindest auf allen Linux Systemen problemlos funktionieren sollte. Die Zeilenumbrüche wurden aus Darstellungsgründen zugefügt.

```
$ cd $BICSUITEHOME/src/capi
$ make sdms_test
cc -g -fno-exceptions -Wall -Wshadow -Wpointer-arith -Wwrite-strings \
  -Wstrict-prototypes -Wmissing-declarations -Wnested-externs -DLINUX \
  -Winline -O3 -I . -c sdms_api.c
cc -g -fno-exceptions -Wall -Wshadow -Wpointer-arith -Wwrite-strings \
  -Wstrict-prototypes -Wmissing-declarations -Wnested-externs -DLINUX \
  -Winline -O3 -I . -c sdms_test.c
cc sdms_api.o sdms_test.o -o sdms_test
$ ./sdms_test localhost 2506 SYSTEM G0H0ME
------------------------------------------------------------------------
User SYSTEM is in the groups ADMIN,PUBLIC
------------------------------------------------------------------------
User GLOBAL.EXAMPLES.HOST_1.SERVER connected with session id 1001
User GLOBAL.EXAMPLES.LOCALHOST.SERVER connected with session id 1002
User GLOBAL.EXAMPLES.HOST_2.SERVER connected with session id 1003
User SYSTEM connected with session id 1059
User SchedulingThread connected with session id 1234321
User GarbageThread connected with session id 1234322
User TriggerThread connected with session id 1234323
User PoolThread connected with session id 1234324
User TimerThread connected with session id 19630127
------------------------------------------------------------------------
```

Wie in den vorherigen Beispielen verfolgt auch dieses Beispiel den einfachen Ansatz: Entweder es klappt, oder es terminiert mit exit code 1.
Es wurde auch bewusst auf eine Modularisierung verzichtet. Dass dies in größeren Projekten unerlässlich ist, sollte unbestritten sein. In diesem einfachen Beispiel würde es jedoch eher von dem ablenken, was gezeigt werden soll.

In den Zeilen 34 bis 54 werden die Command Line Parameters verarbeitet. Fehlende Parameter werden mit den Defaults ersetzt.
In Zeile 56 bis 62 wird die WinSock Library initialisiert (und damit sollte das Beispiel auch unter Windows funktionieren). Dazu muss das Symbol `_WIN32` gesetzt sein.
Anschließend wird dann in Zeile 64 bis 68 eine Verbindung mit dem Scheduling Server aufgebaut. Ab jetzt kann mit dem Server kommuniziert werden.
Das erste Kommando soll ein `SHOW USER` sein. Dementsprechend wird in Zeile 81 das Kommando in einen `SDMS_STRING` gepackt und diese Datenstruktur in Zeile 86 (und 87) an den Server geschickt.
Dieser liefert eine Datenstruktur vom Type `SDMS_OUTPUT` zurück.
In Zeile 94 bis 111 werden die erhaltenen Daten auf `stderr` ausgegeben. Dazu wird zuerst, in Zeile 95, das Data Item `NAME` aus dem Output extrahiert. Als nächstes wird, in Zeile 99, die Tabelle mit Gruppen geholt. Aus dieser Tabelle wird dann zuerst die Position des Gruppennamens ermittelt (Zeile 100), sowie die Größe der Tabelle abgefragt (Zeile 101).
Dann folgt eine einfache Schleife um die Gruppennamen auszugeben. Dabei wird der Name in Zeile 106 und 107 mittels des zuvor ermittelten Indexes extrahiert.
Damit ist die Verarbeitung dieser Outputstruktur abgeschlossen und der belegte Speicher wird in Zeile 113 freigegeben.
Da ein weiteres Kommando abgesetzt werden soll, wird auch der Speicher für das alte Kommando in Zeile 117 freigegeben.
Anschließend geht das Spiel von neuem los. Der Unterschied zwischen beiden Kommandos ist im Wesentlichen, dass ein Show Kommando immer ein Record mit eventuell einer oder mehreren Tabellen zurückliefert. Ein List Kommando liefert dagegen immer nur eine Tabelle zurück.
Andere Kommandos liefern bis auf wenige Ausnahmen keine Daten zurück. In dem Fall reicht es den Return Wert auf `SDMS_OK` zu prüfen. Wird ein `SDMS_OK` zurückgegeben, dann ist garantiert, dass das Kommando auch korrekt verarbeitet wurde.
Im Verzeichnis `$BICSUITEHOME/src/capi` gibt es noch einige weitere Dateien. Eine davon ist `jsstub.c`. Dabei handelt es sich um ein kleines C-Programm welches sich, aus Sicht des Scheduling Servers, als Jobserver benimmt. Es holt ganz brav neue Jobs ab und meldet sie nach 10 Sekunden auch wieder fertig mit exit code 0. Ausführen tut es aber nichts.
Dieses Prográmmchen wird in der Entwicklung für Stress-Testing benutzt. Es können problemlos viele solcher Dummy Jobserver gestartet werden, ohne dass dies den Rechner groß belastet. Allerdings muss der Scheduling Server hart arbeiten um diese Angeber an ihre Grenzen zu bringen.
Es ist allerdings eine in C geschriebene, in der Entwicklung produktiv genutzte, Anwendung. Mit dem Wissen aus dem Vorherigen ist es jetzt möglich zu sehen wie dieses Programm mit dem Server kommuniziert und anschließend Daten verarbeitet.